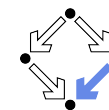
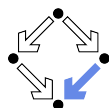


Limits of Computability

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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



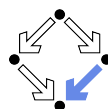
1. Decision Problems

2. The Halting Problem

3. Reduction Proofs

4. Rice's Theorem

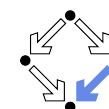
Decision Problems



- **Decision problem** P .
 - A set of words $P \subseteq \Sigma^*$.
 $w \in P \dots w$ has property P .
 - Interpretation as a property of words over Σ .
 $P(w) \dots w$ has property P .
- **Formal definition** by a formula:
 $P := \{w \in \Sigma^* \mid \dots\}$
 $P(w) := \Leftrightarrow \dots$
- **Informal definition** by a decision question:
Does word w have property \dots ?
- **Example problem**: Is the length of w a square number?
 $P := \{w \in \Sigma^* \mid \exists n \in \mathbb{N} : |w| = n^2\}$
 $P(w) := \Leftrightarrow \exists n \in \mathbb{N} : |w| = n^2$
 $P = \{\varepsilon, 0, 0000, 000000000, \dots\}$

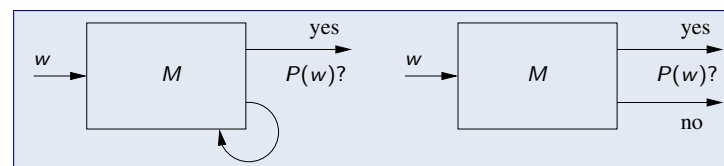
A decision problem is the set of all words for which the answer to a decision question is “yes”.

Semi-Decidability and Decidability

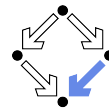


Problems can be the languages of Turing machines.

- A problem P is **semi-decidable**, if P is recursively enumerable.
 - There exists a Turing machine M that semi-decides P .
 - M must only terminate, if the answer to “ $P(w)$?” is “yes”.
- A problem P is **decidable** if P is recursive.
 - There exists a Turing machine M that decides P .
 - M must also terminate, if the answer to “ $P(w)$?” is “no”.



Decidability of Complement



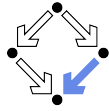
- **Theorem:** If P is decidable, also its complement \bar{P} is decidable.

The answer to “ $\bar{P}(w)$?” is “yes”, if and only if the answer to “ $P(w)$?” is “no” ($\bar{P}(w) \Leftrightarrow \neg P(w)$).

- Proof: If P is decidable, it is recursive, thus \bar{P} is recursive, thus \bar{P} is decidable.
- **Theorem:** P is decidable, if and only if both P and \bar{P} are semi-decidable.
 - Proof: If P and \bar{P} are semi-decidable, they are recursive enumerable. Thus P is recursive and therefore decidable. Analogous for the other direction.

Direct consequences of the previously established results about recursively enumerable and recursive languages.

Decidability and Computability



- **Theorem:** $P \subseteq \Sigma^*$ is semi-decidable, if and only if the partial characteristic function $1'_P : \Sigma^* \rightarrow_p \{1\}$ is Turing computable:

$$1'_P(w) := \begin{cases} 1 & \text{if } P(w) \\ \text{undefined} & \text{if } \neg P(w) \end{cases}$$

- Proof: if P is semi-decidable, there exists M such that, for every word $w \in P = \text{domain}(1'_P)$, M accepts w . We can then construct M' which calls M on w . If M accepts w , M' writes 1 on output tape. If $1'_P$ is Turing computable, there exists M such that, for every word $w \in P = \text{domain}(1'_P)$, M accepts w and writes 1 on the tape. We can then construct M' which takes w from the tape and calls M on w . If M writes 1, M' accepts w .
- **Theorem:** $P \subseteq \Sigma^*$ is decidable, if and only if the characteristic function $1_P : \Sigma^* \rightarrow \{0, 1\}$ is Turing computable:

$$1_P(w) := \begin{cases} 1 & \text{if } P(w) \\ 0 & \text{if } \neg P(w) \end{cases}$$

- Proof: analogous.

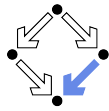
1. Decision Problems

2. The Halting Problem

3. Reduction Proofs

4. Rice's Theorem

Turing Machine Codes



Theorem: for every Turing machine M , there exists a bit string $\langle M \rangle$,

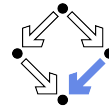
- the **Turing machine code** of M

such that

1. different Turing machines have different codes
 - if $M \neq M'$, then $\langle M \rangle \neq \langle M' \rangle$;
 2. we can recognize valid Turing-machine codes
 - $w \in \text{range}(\langle \cdot \rangle)$ is decidable
- Core idea: assign to all machine states, alphabet symbols, and tape directions unique natural numbers and encode every transition $\delta(q_i, a_j) = (q_k, a_l, d_r)$ by the tuple (i, j, k, l, r) in binary form.

A Turing machine code is also called a “Gödel number”.

The Halting Problem



The most famous undecidable problem in computer science.

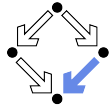
- The **halting problem** HP is to decide, for given Turing machine code $\langle M \rangle$ and word w , whether M halts on input w :

$$HP := \{(\langle M \rangle, w) \mid \text{Turing machine } M \text{ halts on input word } w\}$$

- (w_1, w_2) : a bit string that reversibly encodes the pair w_1, w_2 .
- **Theorem:** The halting problem is undecidable.
 - There is no Turing machine that always halts and says “yes”, if its input is of form $(\langle M \rangle, w)$ such that M halts on input w , respectively says “no”, if this is not the case.

The remainder of this section is dedicated to the proof of this theorem.

Enumeration of Words and Turing Machines



- **Theorem:** There exists an enumeration w of all words over Σ .

$$w = (w_0, w_1, \dots)$$

- For every word $w' \in \Sigma^*$, there exists $i \in \mathbb{N}$ such that $w' = w_i$.
- The enumeration w starts with the empty word, then lists the all words of length 1, then lists all the words of length 2, and so on. Thus every word eventually appears in w .

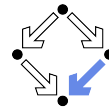
- **Theorem:** There exists an enumeration M of all Turing machines.

$$M = (M_0, M_1, \dots)$$

- For every Turing machine M' there exists $i \in \mathbb{N}$ such that $M' = M_i$.
- Let $C = (C_0, C_1, \dots)$ be the enumeration of all Turing machine codes in bit-alphabetic word order. We define M_i as the unique Turing machine denoted by C_i . Since every Turing machine has a code and C enumerates all codes, M is the enumeration of all Turing machines.

There are countably many words and countably many Turing machines.

Undecidability of the Halting Problem



Proof: define $h : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}$ as

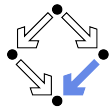
$$h(i, j) := \begin{cases} 1 & \text{if Turing machine } M_i \text{ halts on input word } w_j \\ 0 & \text{otherwise} \end{cases}$$

If the halting problem were decidable, then h were computable.

- Let M be a Turing machine that decides the halting problem.
- We construct a Turing machine M_h which computes h .
- M_h takes input (i, j) and computes $\langle M_i \rangle$ and w_j .
 - M_h enumerates codes $\langle M_0 \rangle, \dots, \langle M_i \rangle$ and words w_0, \dots, w_j .
- M_h passes $(\langle M_i \rangle, w_j)$ to M which eventually halts.
- If M accepts its input, M_h returns 1, else it returns 0.

It thus suffices to show that h is not computable by a Turing machine.

Undecidability of the Halting Problem



We assume that h is computable and derive a contradiction.

- Define $d : \mathbb{N} \rightarrow \{0, 1\}$ as

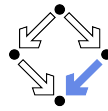
$$d(i) := h(i, i)$$

- $d(i) = 1$: M_i terminates on input word w_i .
- Diagonalization: $d(0), d(1), d(2), \dots$ is diagonal of value table for h .

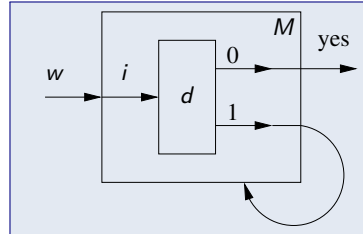
h	$j = 0$	$j = 1$	$j = 2$	\dots
$i = 0$	$\mathbf{h(0,0)}$	$h(0,1)$	$h(0,2)$	\dots
$i = 1$	$h(1,0)$	$\mathbf{h(1,1)}$	$h(1,2)$	\dots
$i = 2$	$h(2,0)$	$h(2,1)$	$\mathbf{h(2,2)}$	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

Since h is computable, also d is computable.

Undecidability of the Halting Problem



```
function M(w):
  let i ∈ ℕ such that w = wi
  case d(i) of
    0: return yes
    1: loop end loop
  end case
end function
```



- Construct M which takes w and determines $i \in \mathbb{N}$ with $w = w_i$.
 - $M(w)$ halts, if and only if $d(i) = 0$.
- Let i be such that $M = M_i$ and compute $M(w_i)$.
 - $M(w_i)$ halts, if and only if $d(i) = 0$.
 - $M(w_i)$ halts, if and only if $M_i(w_i)$ does not halt.
 - $M(w_i)$ halts, if and only if $M(w_i)$ does not halt.

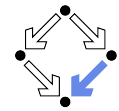
By letting M reason about its own behavior, we derive a contradiction.

1. Decision Problems

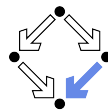
2. The Halting Problem

3. Reduction Proofs

4. Rice's Theorem



Reduction Proofs



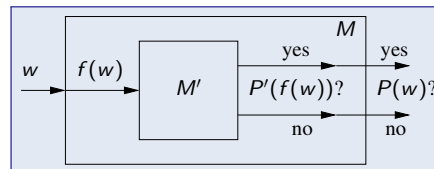
We can construct a partial order on decision problems.

- Decision problem $P \subseteq \Sigma^*$ is **reducible** to $P' \subseteq \Gamma^*$ ($P \leq P'$), if there is a computable function $f : \Sigma^* \rightarrow \Gamma^*$ such that

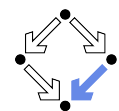
$$P(w) \Leftrightarrow P'(f(w))$$

- w has property P if and only if $f(w)$ has property P' .
- Theorem:** For all decision problems P and P' with $P \leq P'$, it holds that, if P is not decidable, then also P' is not decidable.
 - Proof: we assume that P' is decidable and show that P is decidable. Since P' is decidable, there is a Turing machine M' that decides P' . We construct M that decides P :

```
function M(w):
  w' ← f(w)
  return M'(w')
end function
```



Undecidability of Restricted Halting Problem



To show that some problem P is not decidable, it thus suffices to show $HP \leq P$, i.e., that if P were decidable, then also the halting problem HP would be decidable.

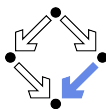
- Theorem:** the restricted halting problem RHP is not decidable.

$$RHP := \{ \langle M \rangle \mid \text{Turing machine } M \text{ halts on input word } \varepsilon \}$$

- Decide, for given $\langle M \rangle$, whether M halts for input word ε .

Pattern for many undecidability proofs.

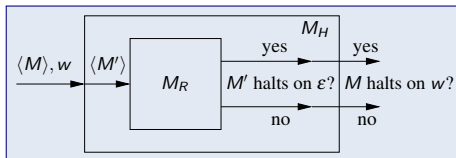
Undecidability of Restricted Halting Problem



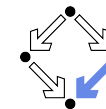
We assume that RHP is decidable and show that HP is decidable.

- Since RHP is decidable, there exists M_R such that M_R accepts input c , if and only if c is the code of some M which halts on input ε .
- We can then define M_H , which accepts input (c, w) , if and only if c is the code of some M that terminates on input w :
 - M_H constructs from (c, w) the code of some M' which first prints w on its tape and then behaves like M .
 - M' terminates for input ε (which is ignored and overwritten by w) if and only if M terminates on input w .
 - M_H accepts its input, if and only if M_R accepts $\langle M' \rangle$.

```
function  $M_H(\langle M \rangle, w)$ :
   $\langle M' \rangle := \text{compute}(\langle M \rangle, w)$ 
  return  $M_R(\langle M' \rangle)$ 
end function
```



Undecidability of the Acceptance Problem

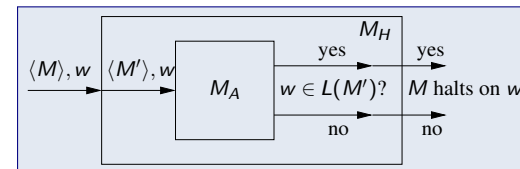


- Theorem:** the acceptance problem AP is not decidable.

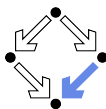
$$AP := \{ \langle M \rangle, w \mid w \in L(M) \}$$

- Decide, for given M and w , whether M accepts w .
- Proof:** we assume AP is decidable and show HP is decidable.
 - Since AP is decidable, there exists M_A such that M_A accepts (c, w) , if and only if c is the code of some M which accepts w .
 - We define M_H , which accepts input (c, w) , if and only if c is the code of some M that halts on input w .
 - M_H modifies $\langle M \rangle$ to $\langle M' \rangle$ where M' behaves as M , except that, if M halts and does not accept, M' halts and accepts.
 - M' thus accepts input w , if and only if M halts on input w .
 - M_H accepts its input, if M_A accepts $(\langle M' \rangle, w)$.

```
function  $M_H(\langle M \rangle, w)$ :
   $\langle M' \rangle := \text{compute}(\langle M \rangle)$ 
  return  $M_A(\langle M' \rangle, w)$ 
end function
```



Semi-Decidability of Acceptance Problem

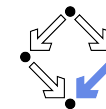


An undecidable problem may be semi-decidable.

- Theorem:** the acceptance problem AP is semi-decidable.
 - There is some Turing Machine that halts and says “yes”, if its input is of form $(\langle M \rangle, w)$ with $w \in L(M)$ (and does not halt or says “no”, else).
- Proof:** we construct a “universal Turing machine” M_u with language AP which acts as an “interpreter” for Turing machine codes: given input $(\langle M \rangle, w)$, M_u simulates the execution of M for input w :
 - If the real execution of M halts for input w with/without acceptance, then also the simulated execution halts with/without acceptance; thus M_u accepts its input (c, w) , if in the simulation M has accepted w .
 - If the real execution of M does not halt for input w , then also the simulated execution does not halt; thus M_u does not accept its input.

Turing machines can be “interpreted/simulated” by other Turing machines (which was already assumed in the previous reduction proofs).

Halting versus Acceptance

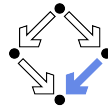


We know that the halting problem is reducible to the acceptance problem.

- Theorem:** the acceptance problem is reducible to the halting prob.
 - $HP \leq AP$ and $AP \leq HP$.
- Proof:** assume that there exists M_H which decides the halting problem. Then we can construct M_A which decides acceptance:
 - From input (c, w) , M_A constructs machine M_{cw} and invokes M_H with input $(\langle M_{cw} \rangle, \varepsilon)$; thus M_H must accept this input if and only if the Turing machine with code c accepts input w .
 - Since M_H decides the halting problem, M_{cw} must thus halt on input ε if and only if the Turing machine with code c accepts input w :
 - M_{cw} invokes M_u with input (c, w) ; if M_u halts and accepts this input, then also M_{cw} halts and accepts its input.
 - If M_u does not accept its input (because it does not halt or because it halts in a non-accepting state), then M_{cw} does not halt.
 - Thus M_{cw} halts if and only if M_u accepts input (c, w) .

The halting problem and the acceptance problem are “equivalent”.

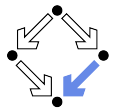
Semi-Decidability of Other Problems



- **Theorem:** the halting problem HP is semi-decidable.
 - **Proof:** we construct Turing machine M' which takes $(\langle M \rangle, w)$ and simulates the execution of M on input w . If (the simulation of) M halts, M' accepts its input. If (the simulation of) M does not halt, M' does not halt (and thus not accept its input).
- **Theorem:** the non-acceptance problem NAP and the non-halting problem NHP are *not* semi-decidable.
 - **Proof:** if both a problem and its complement were semi-decidable, they would be complementary recursively enumerable languages; thus they would be recursive and the problem and its complement decidable.

Problem	semi-decidable	decidable
Halting	yes	no
Non-Halting	no	no
Acceptance	yes	no
Non-Acceptance	no	no

There exist problems that are not even semi-decidable.



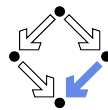
1. Decision Problems

2. The Halting Problem

3. Reduction Proofs

4. Rice's Theorem

Properties of Recurs. Enumerable Languages



- **Property S** of recursively enumerable languages:
 - A set of recursively enumerable languages.
- **S is non-trivial:**
 - there is at least one r.e. language in S , and
 - there is at least one r.e. language not in S .

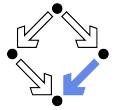
Some r.e. languages have the property and some do not.
- **S is decidable:** P_S is decidable.

$$P_S := \{ \langle M \rangle \mid L(M) \in S \}$$

- Given $\langle M \rangle$, it is decidable whether the language of M has property S .

Decision questions about the semantics of Turing machines.

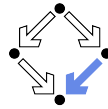
Rice's Theorem



- **Rice's Theorem:** every non-trivial property of recursively enumerable languages is undecidable (proof: see lecture notes).
 - There is no Turing machine which for every possible Turing machine M can decide whether the language of M has a non-trivial property.
- **Relevance:** all non-trivial questions about the **input/output behavior** of Turing machines are undecidable.
 - Also for Turing computable functions.
 - Also for other Turing complete computational models.
- Nevertheless, for **some** machines a decision may be possible.
 - For some machines, it is possible to decide termination.
- However, no method can perform such a decision for **all** machines.
 - No method can exist to decide termination for every possible machine.
- Not applicable to **arbitrary questions** about Turing machines.
 - Form/syntax: does Turing machine M have more than n states?
 - Non-functional property: does M stop in less than n steps?
- Not applicable to **trivial questions**.
 - Is the language of Turing machine M recursively enumerable?

Fundamental limit to automated reasoning about Turing complete models.

Undecidable Turing Machine Problems

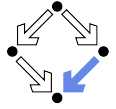


Many interesting problems about Turing machines are undecidable:

- The **halting problem** (also in its restricted form).
- The **acceptance problem** $w \in L(M)$ (also restricted to $\varepsilon \in L(M)$).
- The **emptiness problem**: is $L(M)$ empty?
- The problem of **language finiteness**: is $L(M)$ finite?
- The problem of **language equivalence**: $L(M_1) = L(M_2)$?
- The problem of **language inclusion**: $L(M_1) \subseteq L(M_2)$?
- The problem whether $L(M)$ is regular, context-free, context-sensitive.

Also the complements of these problems are not decidable; however, some of these problems (respectively their complements) may be semi-decidable.

Undecidable Problems from Other Domains



- The **Entscheidungsproblem**: given a formula and a finite set of axioms, all in first order predicate logic, decide whether the formula is valid in every structure that satisfies the axioms.
- **Post's correspondence problem**: given pairs $(x_1, y_1), \dots, (x_n, y_n)$ of non-empty words x_i and y_i , find a sequence i_1, \dots, i_k such that

$$x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}?$$

- The **word problem** for groups: given a group with finitely many generators g_1, \dots, g_n find two sequences $i_1, \dots, i_k, j_1, \dots, j_l$ such that

$$g_{i_1} \circ \dots \circ g_{i_k} = g_{j_1} \circ \dots \circ g_{j_l}$$

- The **ambiguity problem** for context-free grammars: are there two different derivations for the same sentence?

Theory of decidability/undecidability has profound impact on many areas in computer science, mathematics, and logic.