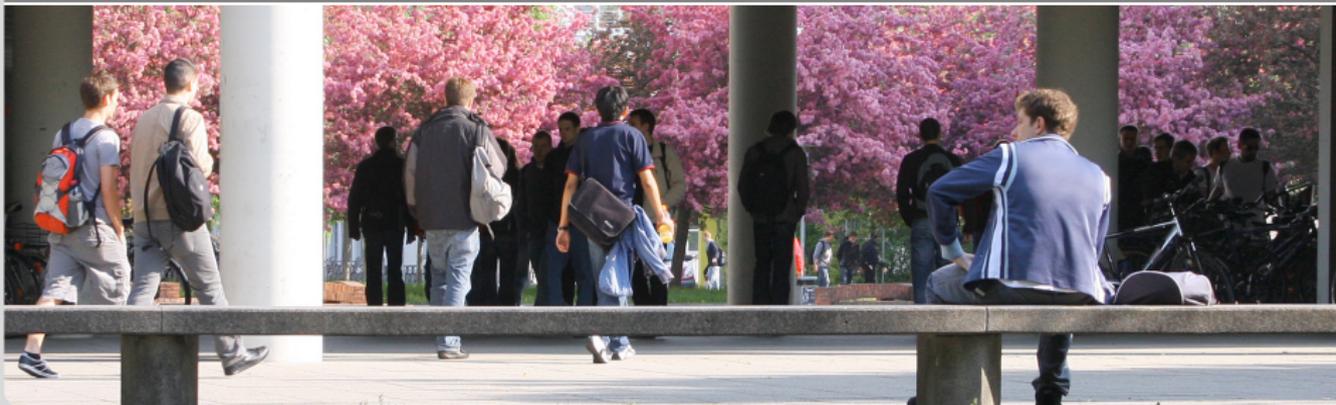


# Theoretische Grundlagen der Informatik

## Vorlesung am 15. November 2018

INSTITUT FÜR THEORETISCHE INFORMATIK



Bearbeitet eine TM  $\mathcal{M}$  eine Eingabe  $w$ , so gibt es drei Möglichkeiten:

1.  $\mathcal{M}$  "läuft" in einen Zustand in  $F$ .

$\rightsquigarrow \mathcal{M}$  akzeptiert  $w$

2.  $\mathcal{M}$  "läuft" in einen Übergang

$$\delta(q, a) = (q, a, N).$$

$\rightsquigarrow \mathcal{M}$  lehnt  $w$  ab

3.  $\mathcal{M}$  "läuft" unendlich lange.

$\rightsquigarrow \mathcal{M}$  stoppt nicht

Für eine Turing-Maschine  $\mathcal{M}$  und Sprache  $L$  definieren wir:

$\mathcal{M}$ hält	1. oder 2.	
$\mathcal{M}$ akzeptiert $L$ , ( $L$ semi-entscheidbar)	$\forall w \in L$ : 1.	$\forall w \notin L$ : 2. oder 3.
$\mathcal{M}$ entscheidet $L$ , ( $L$ entscheidbar)	$\forall w \in L$ : 1.	$\forall w \notin L$ : 2.

Die **universelle Sprache**  $L_U$  über  $\{0, 1\}$  ist definiert durch

$$L_U := \{wv \mid v \in L(T_w)\}.$$

$L_U$  ist also die Menge aller Wörter  $wv$  für die  $T_w$  bei der Eingabe  $v$  hält und  $v$  akzeptiert.

**Satz (Unentscheidbarkeit der Universellen Sprache):**  
Die Sprache  $L_U$  ist nicht entscheidbar.

## Satz (Unentscheidbarkeit der Universellen Sprache):

Die Sprache  $L_U := \{wv \mid v \in L(T_w)\}$  ist nicht entscheidbar.

### Beweis:

- Wir zeigen, dass  $L_U$  eine Verallgemeinerung von  $L_G^C$  ist.
- Wir nehmen an, dass es eine TM gibt, die  $L_U$  entscheidet.
- Dann zeigen wir, dass wir damit auch  $L_G^C$  entscheiden können.

Wir gehen wie folgt vor:

- Berechne das  $i$ , für das  $w = w_i$ .
- Betrachte die durch  $w_i$  kodierte Turing-Maschine  $\mathcal{M}_i$ .
- Wende die Turing-Maschine für  $L_U$  auf  $\langle \mathcal{M}_i \rangle w_i$  an.

Wäre  $L_U$  entscheidbar, so auch  $L_G^C$  im Widerspruch zum letzten Korollar.

**Satz (Semi-Entscheidbarkeit der Universellen Sprache):**  
Die Sprache  $L_U := \{wv \mid v \in L(T_w)\}$  ist semi-entscheidbar.

**Satz (Semi-Entscheidbarkeit der Universellen Sprache):**  
Die Sprache  $L_U := \{wv \mid v \in L(T_w)\}$  ist semi-entscheidbar.

## Beweis:

Wir benutzen die universelle Turing-Maschine, mit der Eingabe  $wv$ :

- Falls  $T_w$  die Eingabe  $v$  akzeptiert, geschieht dies nach endlich vielen Schritten und die universelle Turing-Maschine akzeptiert  $wv$ .
- Falls  $T_w$  die Eingabe  $v$  nicht akzeptiert, wird  $wv$  von der universellen Turing-Maschine ebenfalls nicht akzeptiert. Dies ist unabhängig davon, ob die Simulation stoppt oder nicht.

**Satz (Semi-Entscheidbarkeit der Universellen Sprache):**  
Die Sprache  $L_U := \{wv \mid v \in L(T_w)\}$  ist semi-entscheidbar.

## Beweis:

Wir benutzen die universelle Turing-Maschine, mit der Eingabe  $wv$ :

- Falls  $T_w$  die Eingabe  $v$  akzeptiert, geschieht dies nach endlich vielen Schritten und die universelle Turing-Maschine akzeptiert  $wv$ .
- Falls  $T_w$  die Eingabe  $v$  nicht akzeptiert, wird  $wv$  von der universellen Turing-Maschine ebenfalls nicht akzeptiert. Dies ist unabhängig davon, ob die Simulation stoppt oder nicht.

## Bemerkung:

Die Begriffe entscheidbar und semi-entscheidbar unterscheiden sich tatsächlich.

# Satz von Rice – Motivation

- Wir haben bisher gezeigt, dass wir kein Programm schreiben können, das für ein Turing-Maschinen-Programm  $\langle \mathcal{M} \rangle$  und eine Eingabe  $w$  entscheidet, ob  $\mathcal{M}$  auf der Eingabe  $w$  hält.
- Wir werden im Folgenden sehen, dass wir aus einem Programm im Allgemeinen keine nicht-trivialen Eigenschaften der von dem Programm realisierten Funktion ableiten können.

## Satz (Satz von Rice):

Sei  $R$  die Menge der von Turing-Maschinen berechenbaren Funktionen und  $S$  eine nicht-triviale Teilmenge von  $R$  ( $\emptyset \neq S \neq R$ ). Dann ist die Sprache

$$L(S) := \{ \langle \mathcal{M} \rangle \mid \mathcal{M} \text{ berechnet eine Funktion aus } S \}$$

nicht entscheidbar.

## Satz (Satz von Rice):

Sei  $R$  die Menge der von Turing-Maschinen berechenbaren Funktionen und  $S$  eine nicht-triviale Teilmenge von  $R$  ( $\emptyset \neq S \neq R$ ). Dann ist die Sprache

$$L(S) := \{ \langle \mathcal{M} \rangle \mid \mathcal{M} \text{ berechnet eine Funktion aus } S \}$$

nicht entscheidbar.

## Beweisskizze:

- Zeige:  $\mathcal{H}_\varepsilon := \{ \langle \mathcal{M} \rangle \mid \mathcal{M} \text{ hält auf der Eingabe } \varepsilon \}$  ist unentscheidbar
- Zeige:  $\mathcal{H}_\varepsilon^c$  ist unentscheidbar
- Führe den Widerspruchsbeweis für die Unentscheidbarkeit von  $L(S)$ :
- Konstruiere TM für  $\mathcal{H}_\varepsilon^c$  unter Benutzung von TM  $\mathcal{M}'$  für  $L(S)$

# Bemerkungen zum Satz von Rice

Der Satz von Rice hat weitreichende Konsequenzen:

Es ist für Programme nicht entscheidbar, ob die durch sie definierte Sprache endlich, leer, unendlich oder ganz  $\Sigma^*$  ist.

Wir haben hier nur die Unentscheidbarkeit von  $L_d$  direkt bewiesen. Die anderen Beweise folgten dem folgenden Schema:

Um zu zeigen, dass ein Problem  $A$  unentscheidbar ist, zeigen wir, wie man mit einem Entscheidungsverfahren für  $A$  ein bekanntermaßen unentscheidbares Problem  $B$  entscheiden kann. Dies liefert den gewünschten Widerspruch.

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(1, 111), (10111, 10), (10, 0)\}$  hat die Lösung  $(2, 1, 1, 3)$ , denn es gilt:  $x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$

$$K = \begin{array}{|c|c|c|} \hline 1 & 10111 & 10 \\ \hline \dots & \dots & \dots \\ \hline 111 & 10 & 0 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \end{array}$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(1, 111), (10111, 10), (10, 0)\}$  hat die Lösung  $(2, 1, 1, 3)$ , denn es gilt:  $x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$

$$K = \begin{array}{|c|c|c|} \hline \begin{array}{c} 1 \\ \dots \\ 1 \ 1 \ 1 \end{array} & \begin{array}{c} 1 \ 0 \ 1 \ 1 \ 1 \\ \dots \\ 1 \ 0 \end{array} & \begin{array}{c} 1 \ 0 \\ \dots \\ 0 \end{array} \\ \hline \end{array} \quad \begin{array}{c} \begin{array}{c} 1 \ 0 \ 1 \ 1 \ 1 \\ \vdots \\ 1 \ 0 \end{array} \\ 2 \end{array}$$

$(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3)$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

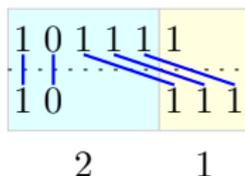
$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(1, 111), (10111, 10), (10, 0)\}$  hat die Lösung  $(2, 1, 1, 3)$ , denn es gilt:  $x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$

$$K = \begin{array}{|c|c|c|} \hline 1 & 10111 & 10 \\ \hline \dots & \dots & \dots \\ \hline 111 & 10 & 0 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \end{array}$$



$$\begin{array}{|c|c|} \hline 101111 & 1 \\ \hline \dots & \dots \\ \hline 10 & 111 \\ \hline \end{array} \begin{array}{l} 2 \\ 1 \end{array}$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

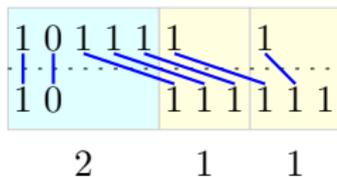
$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(1, 111), (10111, 10), (10, 0)\}$  hat die Lösung  $(2, 1, 1, 3)$ , denn es gilt:  $x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$

$$K = \begin{array}{|c|c|c|} \hline 1 & 10111 & 10 \\ \hline \dots & \dots & \dots \\ \hline 111 & 10 & 0 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \end{array}$$



## Post'sches Korrespondenzproblems

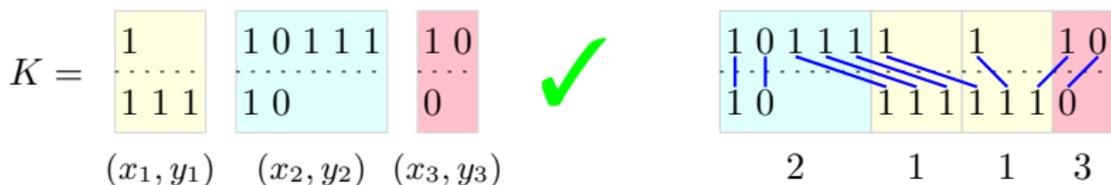
Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(1, 111), (10111, 10), (10, 0)\}$  hat die Lösung  $(2, 1, 1, 3)$ , denn es gilt:  $x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$



## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 10 & 011 & 101 \\ \hline 101 & 11 & 011 \\ \hline \end{array} \\ (x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3)$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 10 & 011 & 101 \\ \hline 101 & 11 & 011 \\ \hline \end{array} \quad \begin{array}{|c|} \hline 10 \\ \hline 101 \\ \hline \end{array}$$

$(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \quad 1$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 10 & 011 & 101 \\ \hline \dots & \dots & \dots \\ \hline 101 & 11 & 011 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \end{array}$$

$$\begin{array}{|c|c|} \hline 10 & 101 \\ \hline \dots & \dots \\ \hline 101 & 011 \\ \hline \end{array} \begin{array}{l} 1 \\ 3 \end{array}$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 10 & 011 & 101 \\ \hline 101 & 11 & 011 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 10 & 101 & 101 & 101 \\ \hline 101 & 011 & 101 & 11 \\ \hline \end{array} \begin{array}{l} 1 \\ 3 \\ 3 \end{array}$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 10 & 011 & 101 \\ \hline 101 & 11 & 011 \\ \hline \end{array} \quad \times \quad \begin{array}{|c|c|c|c|} \hline 10 & 101 & 101 & 101 \\ \hline 101 & 011 & 101 & 101 \\ \hline \end{array} \dots$$

$(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \qquad \qquad \qquad 1 \quad 3 \quad 3 \quad 3$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(10, 101), (011, 11), (101, 011)\}$  hat keine Lösung.

$$K = \begin{array}{|c|c|c|} \hline 1 & 0 & \\ \hline \hline 1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 0 & 1 & 1 \\ \hline \hline 1 & 1 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline \hline 0 & 1 & 1 \\ \hline \end{array} \quad \times$$

$(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3)$

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 1 \\ \hline \hline 1 & 1 & \\ \hline \end{array} \quad 2$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

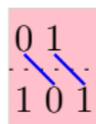
$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline \dots & \dots & \dots & \dots \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{c} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$



0	1
1	0

3

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

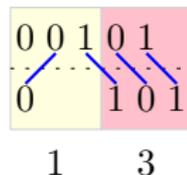
$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$



## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

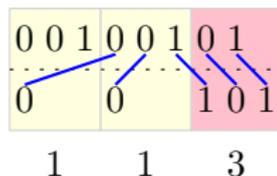
$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$



## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 10 & 001 & 001 & 01 \\ \hline 0010 & 0 & 101 & 101 \\ \hline \end{array} \begin{array}{l} 4 \\ 1 \\ 1 \\ 3 \end{array}$$

*(Note: Blue arrows in the original image point from the top row to the bottom row, indicating a sequence of indices: 4, 1, 1, 3.)*

## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

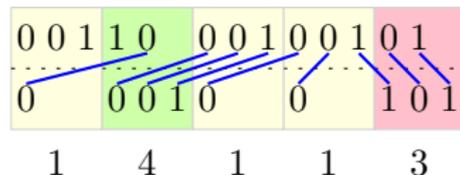
über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array}$$

$(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \quad (x_4, y_4)$



## Post'sches Korrespondenzproblems

Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

■  $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$

$$K = \begin{array}{|c|c|c|c|} \hline 001 & 01 & 01 & 10 \\ \hline 0 & 011 & 101 & 001 \\ \hline \end{array} \begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ (x_3, y_3) \\ (x_4, y_4) \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|} \hline 01 & 001 & 10 & 001 & 001 & 01 \\ \hline 0110 & 0010 & 0 & 101 & 101 & 101 \\ \hline \end{array} \begin{array}{l} 2 \\ 1 \\ 4 \\ 1 \\ 1 \\ 3 \end{array}$$

*(Note: Blue lines in the original image connect the top row to the bottom row, showing a sequence of indices 2, 1, 4, 1, 1, 3 that forms a valid solution.)*

## Post'sches Korrespondenzproblems

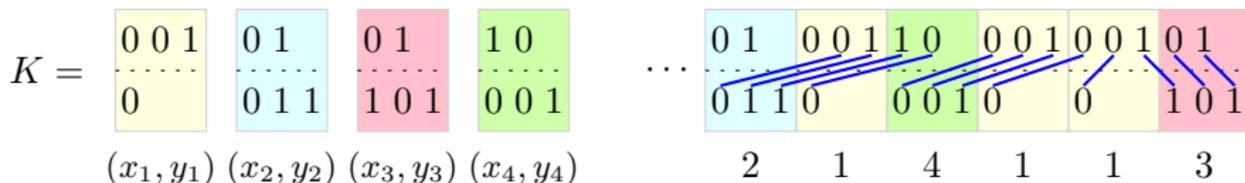
Gegeben ist endliche Menge von Wortpaaren

$$K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

über einem endlichen Alphabet  $\Sigma$ . Es gilt  $x_i \neq \varepsilon$  und  $y_i \neq \varepsilon$ . Gefragt ist, ob es eine endliche Folge von Indizes  $i_1, \dots, i_k \in \{1, \dots, n\}$  gibt, so dass  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  gilt.

## Beispiele

- $K = \{(001, 0), (01, 011), (01, 101), (10, 001)\}$   
 $\rightsquigarrow$  Die kürzeste Lösung hat Länge 66.



## **Satz (Unentscheidbarkeit des PKP):**

Das Post'sche Korrespondenzproblem ist nicht entscheidbar.

## **Beweisidee:**

Dies kann über die Nicht-Entscheidbarkeit des Halteproblems bewiesen werden.

# Eigenschaften von (semi-)entscheidbaren Sprachen

- Die entscheidbaren Sprachen sind abgeschlossen unter Komplementbildung, Schnitt und Vereinigung.
- Die semi-entscheidbaren Sprachen sind abgeschlossen unter Schnitt und Vereinigung, aber nicht unter Komplementbildung.

## Satz:

Sei  $L \subseteq \Sigma^*$  und  $L^c = \Sigma^* \setminus L$ . Dann gilt

- $L$  entscheidbar  $\iff L^c$  entscheidbar.
- $L$  entscheidbar  $\iff L$  und  $L^c$  semi-entscheidbar.

Beweis: Übung und Tutorien.

## ■ Komplexitätstheorie

Fragestellung bisher:

- Ist eine Sprache  $L$  entscheidbar oder nicht?
- Ist eine Funktion berechenbar oder nicht?
- Benutzung von deterministischen Turing-Maschinen.

In diesem Kapitel:

- Wie effizient kann ein Problem gelöst werden?
- Betrachtung von nichtdeterministischen Turing-Maschinen.

## Frage (P vs. NP):

Gibt es einen wesentlichen Effizienzgewinn beim Übergang von der deterministischen Turing-Maschine zur nichtdeterministischen Turing-Maschine?

# Wie sieht ein Problem aus?

## Beispiel: Traveling Salesman Problem (TSP)

Gegeben sei ein vollständiger Graph  $G = (V, E, c)$ , d.h.

- $V := \{1, \dots, n\}$
- $E := \{\{u, v\} \mid u, v \in V, u \neq v\}$
- $c: E \rightarrow \mathbb{Z}^+$ .

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**  
Gesucht ist eine Tour (Rundreise), die alle Elemente aus  $V$  enthält und minimale Gesamtlänge unter allen solchen Touren hat.
- **Optimalwertproblem:**  
Gesucht ist die Länge einer minimalen Tour.
- **Entscheidungsproblem:**  
Gegeben sei zusätzlich auch ein Parameter  $k \in \mathbb{Z}^+$ . Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens  $k$  ist?

# Wie sieht ein Problem aus?

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**

Gesucht ist eine Tour (Rundreise), die alle Elemente aus  $V$  enthält und minimale Gesamtlänge unter allen solchen Touren hat.

- **Optimalwertproblem:**

Gesucht ist die Länge einer minimalen Tour.

- **Entscheidungsproblem:**

Gegeben sei zusätzlich auch ein Parameter  $k \in \mathbb{Z}^+$ . Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens  $k$  ist?

Bemerkung:

- Mit einer Lösung des Optimierungsproblems kann man leicht auch das Optimalwertproblem und das Entscheidungsproblem lösen.
- Mit einer Lösung des Optimalwertproblems kann man leicht auch das Entscheidungsproblem lösen.

# Definition: Problem

Ein **Problem**  $\Pi$  ist gegeben durch:

- eine allgemeine Beschreibung aller vorkommenden Parameter;
- eine genaue Beschreibung der Eigenschaften, die die Lösung haben soll.

Eingabe: Graph  $G = (V, E)$ , Kantengewichtung  $c: E \rightarrow \mathbb{Z}^+$ , Zahl  $k$

Lösung: zykl. Permutation  $x_1 x_2 \cdots x_n$  von  $V$  mit  $\{x_i, x_{i+1}\} \in E$  für  $i = 1, \dots, n-1$  und  $\sum_{i=1}^{n-1} c(\{x_i, x_{i+1}\}) \leq k$

Eine **Instanz**  $I$  von  $\Pi$  erhalten wir, indem wir die Parameter von  $\Pi$  festlegen. (**Problembispiel**)

$$V = \{a, b, c, d\}$$

$$E = \{\{a, b\}, \{a, c\}, \{c, d\}, \{b, d\}, \{b, c\}\}$$

$$c(\{a, b\}) = c(\{b, d\}) = c(\{b, c\}) = 1, c(\{a, c\}) = 2, c(\{c, d\}) = 4$$

# Definition: Kodierungsschema

- Wir interessieren uns für die **Laufzeit** von Algorithmen.
- Diese wird in der Größe des Problems gemessen.

Die Größe eines Problems ist abhängig von der Beschreibung oder Kodierung der Instanzen.

- Ein **Kodierungsschema**  $s$  ordnet jeder Instanz  $I$  eines Problems ein Wort oder *Kodierung*  $s(I)$  über einem Alphabet  $\Sigma$  zu.

$$V = \{a, b, c, d\}$$

$$E = \{\{a, b\}, \{a, c\}, \{c, d\}, \{b, d\}, \{b, c\}\}$$

$$c(\{a, b\}) = c(\{b, d\}) = c(\{b, c\}) = 1, c(\{a, c\}) = 2, c(\{c, d\}) = 4$$

$$s(I) = 00|01|10|11 \sqcup 00 * 01|00 * 10|10 * 11|01 * 11|01 * 10 \sqcup 1|2|4|1|1$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, |, \sqcup, *\}$$

# Definition: Kodierungsschema

- Wir interessieren uns für die **Laufzeit** von Algorithmen.
- Diese wird in der Größe des Problems gemessen.

Die Größe eines Problems ist abhängig von der Beschreibung oder Kodierung der Instanzen.

- Ein **Kodierungsschema**  $s$  ordnet jeder Instanz  $I$  eines Problems ein Wort oder *Kodierung*  $s(I)$  über einem Alphabet  $\Sigma$  zu.

$$s(I) = 00|01|10|11 \sqcup 00 * 01|00 * 10|10 * 11|01 * 11|01 * 10 \sqcup 1|2|4|1|1$$
$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, |, \sqcup, *\}$$

- Die **Inputlänge** einer Instanz ist die Anzahl der Symbole ihrer Kodierung.

$$\text{hier: } |s(I)| = 51$$

Es gibt verschiedene Kodierungsschemata für ein bestimmtes Problem.

## Beispiel:

- Zahlen können dezimal, binär, unär, usw. kodiert werden.
- Die Inputlänge von 5127 beträgt dann 4 für dezimal, 13 für binär und 5127 für unär.

Wir werden uns auf vernünftige Schemata festlegen:

- Die Kodierung einer Instanz soll keine überflüssigen Informationen enthalten.
- Zahlen sollen binär (oder  $k$ -är für  $k \neq 1$ ) kodiert sein.

Dies bedeutet, die Kodierungslänge

- einer ganzen Zahl  $n$  ist  $\lfloor \log_k |n| + 1 \rfloor + 1 =: \langle n \rangle$   
(eine 1 benötigt man für das Vorzeichen);
- einer rationalen Zahl  $r = \frac{p}{q}$  ist  $\langle r \rangle = \langle p \rangle + \langle q \rangle$ ;
- eines Vektors  $X = (x_1, \dots, x_n)$  ist  $\langle X \rangle := \sum_{i=1}^n \langle x_i \rangle$ ;
- einer Matrix  $A \in \mathbb{Q}^{m \times n}$  ist  $\langle A \rangle := \sum_{i=1}^m \sum_{j=1}^n \langle a_{ij} \rangle$ .
- eines Graphen  $G = (V, E)$  kann zum Beispiel durch die Kodierung seiner *Adjazenzmatrix*, die eines gewichteten Graphen durch die Kodierung der *Gewichtsmatrix* beschrieben werden.

# Äquivalenz von Kodierungsschemata

Zwei Kodierungsschemata  $s_1, s_2$  heißen **äquivalent** bezüglich eines Problems  $\Pi$ , falls es Polynome  $p_1, p_2$  gibt, so dass gilt:

$$|s_1(I)| = n \implies |s_2(I)| \leq p_2(n)$$

und

$$|s_2(I)| = m \implies |s_1(I)| \leq p_1(m)$$

für alle Instanzen  $I$  von  $\Pi$ .

- Ein Entscheidungsproblem  $\Pi$  können wir als Familie / Klasse  $D_{\Pi}$  von Instanzen auffassen.
  - Mit festem Kodierungsschema  $s$  ist das eine Menge von Wörtern über  $\Sigma$   
 $\rightsquigarrow$  unsere Eingaben
- Eine Teilmenge dieser Klasse ist  $J_{\Pi} \subseteq D_{\Pi}$ , die Klasse der **Ja-Instanzen**, d.h. die Instanzen deren Antwort Ja ist.
- Der Rest der Klasse  $N_{\Pi} \subseteq D_{\Pi}$  ist die Klasse der **Nein-Instanzen**.

# Korrespondenz von Entscheidungsproblemen und Sprachen

Ein Problem  $\Pi$  und ein Kodierungsschema  $s: D_{\Pi} \rightarrow \Sigma^*$  zerlegen  $\Sigma^*$  in drei Klassen:

- Wörter aus  $\Sigma^*$ , die *nicht* Kodierung eines Beispiels aus  $D_{\Pi}$  sind,
- Wörter aus  $\Sigma^*$ , die Kodierung einer Instanz  $I \in N_{\Pi}$  sind,
- Wörter aus  $\Sigma^*$ , die Kodierung einer Instanz  $I \in J_{\Pi}$  sind.

Die dritte Klasse ist die Sprache, die zu  $\Pi$  im Kodierungsschema  $s$  **korrespondiert**.

Die zu einem Problem  $\Pi$  und einem Kodierungsschema  $s$  **zugehörige Sprache** ist

$$L[\Pi, s] := \left\{ x \in \Sigma^* \mid \begin{array}{l} \Sigma \text{ ist das Alphabet zu } s \text{ und } x \text{ ist Kodierung} \\ \text{einer Ja-Instanz } I \text{ von } \Pi \text{ unter } s, \text{ d.h. } I \in J_{\Pi} \end{array} \right\}$$

- Wir betrachten im folgenden deterministische Turing-Maschinen mit zwei Endzuständen  $q_J, q_N$ , wobei  $q_J$  akzeptierender Endzustand ist.
- Dann wird die Sprache  $L_{\mathcal{M}}$  akzeptiert von der Turing-Maschine  $\mathcal{M}$ , falls

$$L_{\mathcal{M}} = \{x \in \Sigma^* \mid \mathcal{M} \text{ akzeptiert } x\} .$$

- Eine deterministische Turing-Maschine  $\mathcal{M}$  **löst** ein Entscheidungsproblem  $\Pi$  unter einem Kodierungsschema  $s$ , falls  $\mathcal{M}$  bei jeder Eingabe über dem Eingabe-Alphabet in einem Endzustand endet und  $L_{\mathcal{M}} = L[\Pi, s]$  ist.
  - D.h. die Turing-Maschine  $\mathcal{M}$  **entscheidet**  $L[\Pi, s]$ .

Für eine deterministische Turing-Maschine  $\mathcal{M}$ , die für alle Eingaben über dem Eingabe-Alphabet  $\Sigma$  hält, ist die **Zeitkomplexitätsfunktion**

$T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  definiert durch

$$T_{\mathcal{M}}(n) = \max \left\{ m \mid \begin{array}{l} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n, \text{ so} \\ \text{dass die Berechnung von } \mathcal{M} \text{ bei Eingabe } x \\ m \text{ Berechnungsschritte (Übergänge) benötigt,} \\ \text{bis ein Endzustand erreicht wird} \end{array} \right\}$$

Für eine deterministische Turing-Maschine  $\mathcal{M}$ , die für alle Eingaben über dem Eingabe-Alphabet  $\Sigma$  hält, ist die **Zeitkomplexitätsfunktion**

$T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  definiert durch

$$T_{\mathcal{M}}(n) = \max \left\{ m \mid \begin{array}{l} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n, \text{ so} \\ \text{dass die Berechnung von } \mathcal{M} \text{ bei Eingabe } x \\ m \text{ Berechnungsschritte (Übergänge) benötigt,} \\ \text{bis ein Endzustand erreicht wird} \end{array} \right\}$$

## Bemerkungen

- Wenn Eingabe  $x$  Länge  $n$  hat, so braucht  $\mathcal{M}$  höchstens  $T_{\mathcal{M}}(n)$  Berechnungsschritte.

- Für ein Entscheidungsproblem  $\Pi$  mit Kodierungsschema  $s$ :

Instanz  $I \in D_{\Pi}$  von  $\Pi$   $\rightsquigarrow$  Kodierung  $s(I) \in \Sigma^*$

Kodierungslänge  $|s(I)|$   $\rightsquigarrow$  Länge des Wortes

Zeit zum Lösen von  $I$   $\rightsquigarrow$  Anzahl Berechnungsschritte von  $\mathcal{M}$

Die Klasse  $\mathcal{P}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine deterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom  $p$  mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

- Zum Beispiel  $T_{\mathcal{M}}(n) \leq 4n^2 + 42$ .
- Sprachen / Entscheidungsprobleme in  $\mathcal{P}$  sind “effizient lösbar”.

# Schwierigkeit von Entscheidungs- und Optimierungsproblem

## Satz:

Falls es einen Algorithmus  $\mathcal{A}$  gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

# Schwierigkeit von Entscheidungs- und Optimierungsproblem

## Satz:

Falls es einen Algorithmus  $\mathcal{A}$  gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

**Beweis:** Algorithmus, der das Optimierungsproblem löst.

Input:  $G = (V, E)$ ,  $c_{ij} = c(\{i, j\})$  für  $i, j \in V := \{1, \dots, n\}$ ,  
Algorithmus  $\mathcal{A}$

Output:  $d_{ij}$  ( $1 \leq i, j \leq n$ ), so dass alle bis auf  $n$  der  $d_{ij}$ -Werte den Wert  $1 + \max\{c_{ij} \mid 1 \leq i, j \leq n\}$  haben. Die restlichen  $n$   $d_{ij}$ -Werte haben den Wert  $c_{ij}$  und geben genau die Kanten einer optimalen Tour an.

# Algorithmus OPT-TOUR (als Beweis) 1/2

- 1 berechne  $m := \max\{c_{ij} \mid 1 \leq i, j \leq n\}$ ;
- 2 setze  $L(\text{ow}) := 0$  und  $H(\text{igh}) := n \cdot m$ ;  $(L \leq OPT \leq H)$
- 3 **Solange**  $H - L > 1$  gilt, führe aus: (binäre Suche nach  $OPT$ )
- 4 **Falls**  $\mathcal{A}(G, c, \lceil \frac{1}{2}(H + L) \rceil) = \text{“nein”}$ ,  $(OPT > \lceil \frac{1}{2}(H + L) \rceil)$
- 5 setze  $L := \lceil \frac{1}{2}(H + L) \rceil + 1$ ;
- 6 **Sonst**  $(OPT \leq \lceil \frac{1}{2}(H + L) \rceil)$
- 7 setze  $H := \lceil \frac{1}{2}(H + L) \rceil$ ;
- 8 **Falls**  $\mathcal{A}(G, c, L) = \text{“nein”}$  (hier gilt  $H - L \leq 1$ )
- 9 setze  $OPT := H$ ;
- 10 **Sonst**
- 11 setze  $OPT := L$ ;



Die Schleife der binären Suche bricht ab, und danach ist die Differenz  $H - L$  gleich 1 oder 0, denn:

- Solange  $H - L > 1$ , ändert sich bei jedem Schleifendurchlauf einer der Werte  $H, L$ :
  - Für  $H - L > 1$  gilt, dass  $L \neq \lceil \frac{1}{2}(H + L) \rceil + 1$  und  $H \neq \lceil \frac{1}{2}(H + L) \rceil$  ist.
- Die Differenz  $H - L$  verkleinert sich mit jedem Durchlauf
- Da  $H$  und  $L$  ganzzahlig sind, tritt der Fall  $H - L \leq 1$  ein.
- Zu jedem Zeitpunkt gilt  $H - L \geq 0$ :
  - $H - L = 0$  ist möglich, wenn zum Beispiel  $L$  auf  $\lceil \frac{1}{2}(H + L) \rceil + 1$  erhöht wird und vorher  $H - L = 2$  oder  $H - L = 3$  war.

- $\mathcal{A}(G, c, k)$  wird (für verschiedene  $k$ ) etwa  $\log(n \cdot m)$ -mal aufgerufen.
- $\mathcal{A}(G, c, OPT)$  wird etwa  $n^2$ -mal aufgerufen.
- Es finden also  $\mathcal{O}(n^2 + \log(nm))$  Aufrufe von  $\mathcal{A}$  statt.
- Die Inputlänge ist  $\mathcal{O}(n^2 \cdot \max \langle c_{ij} \rangle) = \mathcal{O}(n^2 \cdot \max \log c_{ij})$ .
- Da  $\mathcal{A}$  polynomiell ist, ist dies also auch OPT-TOUR.

## ■ Probleme

- Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
- Problem  $\Pi$  ist Klasse  $D_{\Pi}$  von Instanzen  $I$ .

## ■ Eingabegrößen

- Kodierungsschema  $s: D_{\Pi} \rightarrow \Sigma^*$  über Alphabet  $\Sigma^*$ .
- Kodierung  $s(I)$  einer Instanz  $I$  ist ein Wort aus  $\Sigma^*$ .
- Inputlänge  $|s(I)|$  ist Länge des Wortes.

## ■ Entscheidungsprobleme

- Ja-Instanzen  $J_{\Pi}$  und Nein-Instanzen  $N_{\Pi}$
- Sprache  $L[\Pi, s]$  der Kodierungen aller Ja-Instanzen
- TM  $\mathcal{M}$  löst  $\Pi$  wenn  $\mathcal{M}$  Sprache  $L[\Pi, s]$  entscheidet.

## ■ Laufzeiten

- Zeitkomplexitätsfunktion  $T_{\mathcal{M}}(n)$  von  $\mathcal{M}$  bei Eingaben der Länge  $n$
- Die Klasse  $\mathcal{P}$ : Sprachen von  $\mathcal{M}$  mit  $T_{\mathcal{M}}(n)$  polynomiell in  $n$ .

## ■ Am Beispiel TSP: Entscheidung $\rightarrow$ Optimalwert $\rightarrow$ Optimierung

## ■ Probleme

- Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
- Problem  $\Pi$  ist Klasse  $D_{\Pi}$  von Instanzen  $I$ .

## ■ Eingabegrößen

- Kodierungsschema  $s: D_{\Pi} \rightarrow \Sigma^*$  über Alphabet  $\Sigma^*$ .
- Kodierung  $s(I)$  einer Instanz  $I$  ist ein Wort aus  $\Sigma^*$ .
- Inputlänge  $|s(I)|$  ist Länge des Wortes.

## ■ Entscheidungsprobleme

Testen Sie sich!

- Ja-Instanzen  $J_{\Pi}$  und Nein-Instanzen  $N_{\Pi}$
- Sprache  $L[\Pi, s]$  der Kodierungen aller Ja-Instanzen
- TM  $\mathcal{M}$  löst  $\Pi$  wenn  $\mathcal{M}$  Sprache  $L[\Pi, s]$  entscheidet.

## ■ Laufzeiten

- Zeitkomplexitätsfunktion  $T_{\mathcal{M}}(n)$  von  $\mathcal{M}$  bei Eingaben der Länge  $n$
- Die Klasse  $\mathcal{P}$ : Sprachen von  $\mathcal{M}$  mit  $T_{\mathcal{M}}(n)$  polynomiell in  $n$ .

## ■ Am Beispiel TSP: Entscheidung $\rightarrow$ Optimalwert $\rightarrow$ Optimierung

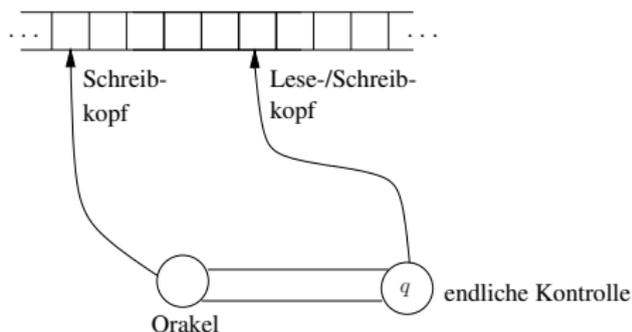
Die “klassische” nichtdeterministische Turing-Maschine:

- Bei der nichtdeterministischen Turing-Maschine wird die Übergangsfunktion  $\delta$  zu einer Relation erweitert.
- Dies ermöglicht Wahlmöglichkeiten und  $\varepsilon$ -Übergänge  
 $\rightsquigarrow$  vergleiche endliche Automaten.

Die **Orakel-Turing-Maschine**:

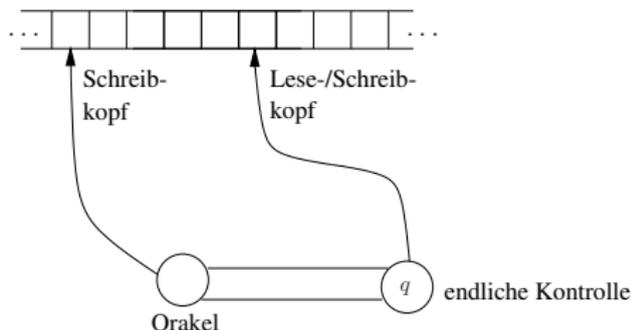
- Wir betrachten ein äquivalentes Modell einer nichtdeterministischen Turing-Maschine (NTM), die auf einem **Orakel** basiert.
- Dies kommt der Intuition näher.

## Nichtdeterministische Turing-Maschinen



- werden analog zu DTMs durch ein Oktupel  $(Q, \Sigma, \sqcup, \Gamma, s, \delta, q_J, q_N)$  beschrieben.
- haben genau zwei Endzustände  $q_J$  und  $q_N$ , wobei  $q_J$  der akzeptierende Endzustand ist.
- haben zusätzlich zu der endlichen Kontrolle mit dem Lese-/Schreibkopf ein **Orakelmodul** mit einem eigenen Schreibkopf.

Berechnung einer nichtdeterministischen Turing-Maschine:

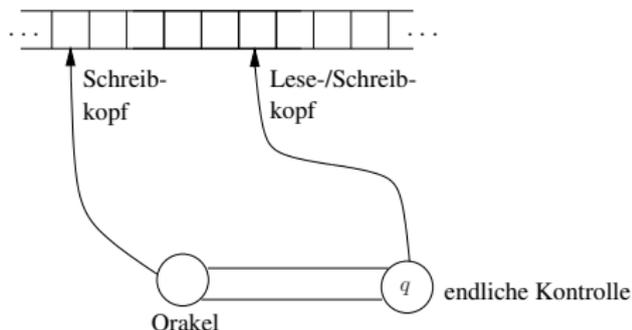


## ■ 1. Stufe:

- Das Orakelmodul schreibt zunächst ein Trennzeichen in die Zelle direkt vor der Eingabe.
- Dann weist es seinen Schreibkopf an, Schritt für Schritt entweder ein Symbol zu schreiben und nach links zu gehen oder anzuhalten.
- Falls der Schreibkopf anhält, wird das Orakelmodul inaktiv, und die endliche Zustandskontrolle wird aktiv.

## ■ 2. Stufe:

- Ab jetzt genau wie bei DTM.
- Das Orakelmodul und sein Schreibkopf sind nicht weiter beteiligt.



Akzeptanz einer Eingabe:

- Eine NTM  $\mathcal{M}$  **akzeptiert** ein Wort  $x \in \Sigma^*$  genau dann, wenn **es eine Berechnung gibt**, die in  $q_A$  endet.
- Eine NTM  $\mathcal{M}$  akzeptiert die Sprache  $L \subseteq \Sigma^*$  genau dann, wenn sie gerade die Wörter aus  $L$  akzeptiert.

Die **Eingabe** ist ein Wort aus  $\Sigma^*$ , zum Beispiel eine Kodierung einer Instanz  $I \in D_{\Pi}$  des Entscheidungsproblems  $\Pi$ .

- **1. Stufe:** Es wird ein Orakel aus  $\Gamma^*$  berechnet, zum Beispiel ein **Lösungsbeispiel** für  $I$ , also ein Indikator, warum  $I \in J_{\Pi}$  gelten sollte.
- **2. Stufe:** Hier wird nun dieser **Lösungsvorschlag überprüft**, d.h. es wird geprüft ob  $I \in J_{\Pi}$ .

## Beispiel TSP

- **1. Stufe:** Es wird zum Beispiel eine zykl. Permutation  $x_1 x_2 \dots x_n$  der Knotenmenge  $V$  vorgeschlagen.  
D.h.  $(x_1, x_2, \dots, x_n), G = (V, E), c, k$  ist die Eingabe für 2. Stufe.
- **2. Stufe:** Es wird nun überprüft, ob  $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n$  eine Tour in  $G = (V, E)$  darstellt, deren Länge bezüglich  $c$  nicht größer als  $k$  ist.

- Das Orakel kann ein beliebiges Wort aus  $\Gamma^*$  sein.
- Darum muss in der Überprüfungsphase (2.Stufe) geprüft werden, ob das Orakel ein zulässiges Lösungsbeispiel für die gegebenen Eingabe ist.
- Ist dies der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort “Ja” beendet werden.  
     $\rightsquigarrow$  gehe in Zustand  $q_J$
- Ist dies nicht der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort “Nein” beendet werden.  
     $\rightsquigarrow$  gehe in Zustand  $q_N$
  
- Jede NTM  $\mathcal{M}$  hat zu einer gegebenen Eingabe  $x$  eine unendliche Anzahl möglicher Berechnungen, eine zu jedem Orakel aus  $\Gamma^*$ .
- Endet **mindestens eine** in  $q_J$ , so wird  $x$  akzeptiert.

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine  $\mathcal{M}$  benötigt, um ein Wort  $x \in L_{\mathcal{M}}$  zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die  $\mathcal{M}$  in den Zustand  $q_f$  überführt.
- Die **Zeitkomplexitätsfunktion**  $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  einer nichtdeterministischen Turing-Maschine  $\mathcal{M}$  ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left( \left\{ 1 \right\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right. \right)$$

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine  $\mathcal{M}$  benötigt, um ein Wort  $x \in L_{\mathcal{M}}$  zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die  $\mathcal{M}$  in den Zustand  $q_f$  überführt.
- Die **Zeitkomplexitätsfunktion**  $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  einer nichtdeterministischen Turing-Maschine  $\mathcal{M}$  ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left( \{1\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right\} \right)$$

## Bemerkung 1

- Zur Berechnung von  $T_{\mathcal{M}}(n)$  wird für jedes  $x \in L_{\mathcal{M}}$  mit  $|x| = n$  eine kürzeste akzeptierende Berechnung betrachtet.
- Anschließend wird von diesen kürzesten die längste bestimmt.
- Somit ergibt sich eine *worst-case* Abschätzung.

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine  $\mathcal{M}$  benötigt, um ein Wort  $x \in L_{\mathcal{M}}$  zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die  $\mathcal{M}$  in den Zustand  $q_f$  überführt.
- Die **Zeitkomplexitätsfunktion**  $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$  einer nichtdeterministischen Turing-Maschine  $\mathcal{M}$  ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left( \{1\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right\} \right)$$

## Bemerkung 2

- Die Zeitkomplexität hängt nur von der Anzahl der Schritte ab, die bei einer akzeptierenden Berechnung auftreten.
- Hierbei umfasst die Anzahl der Schritte auch die Schritte der Orakelphase.
- Per Konvention ist  $T_{\mathcal{M}}(n) = 1$ , falls es keine Eingabe  $x$  der Länge  $n$  gibt, die von  $\mathcal{M}$  akzeptiert wird.

# Die Klasse $\mathcal{NP}$

Die Klasse  $\mathcal{NP}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom  $p$  mit

$$T_M(n) \leq p(n).$$

( $\mathcal{NP}$  steht für **nichtdeterministisch polynomial**.)

Die Klasse  $\mathcal{NP}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom  $p$  mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

( $\mathcal{NP}$  steht für **nichtdeterministisch polynomial**.)

## Bemerkungen

- Alle Sprachen in  $\mathcal{NP}$  sind entscheidbar.
- Informell ausgedrückt gehört  $\Pi$  zu  $\mathcal{NP}$ , falls  $\Pi$  folgende Eigenschaft hat:  
Ist die Antwort bei Eingabe eines Beispiels  $I$  von  $\Pi$  Ja, dann kann die Korrektheit der Antwort in polynomialer Zeit überprüft werden.

Die Klasse  $\mathcal{NP}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom  $p$  mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

( $\mathcal{NP}$  steht für **nichtdeterministisch polynomial**.)

**Beispiel:**  $\text{TSP} \in \mathcal{NP}$ :

Denn zu gegebenem  $G = (V, E)$ ,  $c, k$  und einer festen zykl. Permutation  $x_1 x_2 \cdots x_n$  von  $V$  kann in  $O(|V| \cdot \log C)$  (wobei  $C$  die größte vorkommende Zahl ist) Schritten überprüft werden, ob

$$\{x_i, x_{i+1}\} \in E \text{ für } i = 1, \dots, n-1 \quad \text{und} \quad \sum_{i=1}^{n-1} c(\{x_i, x_{i+1}\}) \leq k$$

gilt.

Die Klasse  $\mathcal{P}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine **deterministische** Turing-Maschine existiert, deren Zeitkomplexitätsfunktion **polynomial** beschränkt ist.

- ↪ Bei Eingabe einer Instanz  $I$  von  $\Pi$  kann die Existenz einer Lösung in polynomialer Zeit überprüft werden.

Die Klasse  $\mathcal{NP}$  ist die Menge aller Sprachen  $L$  (Entscheidungsprobleme), für die eine **nichtdeterministische** Turing-Maschine existiert, deren Zeitkomplexitätsfunktion **polynomial** beschränkt ist.

- ↪ Existiert für die Eingabe einer Instanz  $I$  von  $\Pi$  eine Lösung, dann kann die Korrektheit einer Lösung in polynomialer Zeit überprüft werden.

Große Frage: Ist  $\mathcal{P} = \mathcal{NP}$ ?