Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik
Vorlesung am 22. November 2016

INSTITUT FÜR THEORETISCHE INFORMATIK

Die Universelle Sprache

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.

Die Universelle Sprache

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_d^c ist.
- Wir nehmen an, dass es eine TM gibt, die L_u entscheidet.
- Dann zeigen wir, dass wir damit auch L_d^c entscheiden können.

Wir gehen wie folgt vor:

- Berechne das i, für das $w = w_i$.
- Betrachte die durch w_i codierte 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_d^c im Widerspruch zum letzten Korollar.

Die Universelle Sprache

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

Die Universelle Sprache

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.

Die Universelle Sprache

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 von Rice

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 von Rice

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: $\mathcal{H}_\varepsilon := \{\langle \mathcal{M} \rangle \mid \mathcal{M}$ 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 Σ^* 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.

Das Post'sche Korrespondenzproblem

Post'sches Korrespondenzproblems
Gegeben ist endliche Folge von Wortpaaren

$$K = ((x_1, y_1), \ldots, (x_n, y_n))$$

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

Beispiele

- $K = ((1, 111), (10111, 10), (10, 0))$ hat die Lösung $(2, 1, 1, 3)$, denn es gilt:
$$x_2 x_1 x_1 x_3 = 101111110 = y_2 y_1 y_1 y_3$$
- $K = ((10, 101), (011, 11), (101, 011))$ hat keine Lösung
- $K = ((001, 0), (01, 011), (01, 101), (10, 001))$ hat eine Lösung der Länge 66.

Das Post'sche Korrespondenzproblem

Satz (Unentscheidbarkeit des PKP):
Das Post'sche Korrespondezproblem ist nicht entscheidbar

Beweis:
Beweis über Nicht-Entscheidbarkeit des Halteproblems.

Eigenschaften von (semi-)entscheibaren 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 gdw. L^c entscheidbar.
- L und L^c semi-entscheidbar gdw. L entscheidbar.

Beweis: Übung und Tutorien.

Kapitel

- **Komplexitätstheorie**

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.

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

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** Π ist gegeben durch:

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

Ein **Problembeispiel** / (**Instanz**) von Π erhalten wir, indem wir die Parameter von Π festlegen.

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 Problembeispiele

- Ein **Kodierungsschema** *s* ordnet jedem Problembeispiel eines Problems eine Zeichenkette oder *Kodierung* über einem Alphabet Σ zu.
- Die **Inputlänge** eines Problembeispiels ist die Anzahl des Symbole seiner Kodierung.

Kodierungsschema

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 eines Problembeispiels sollte keine überflüssigen Informationen enthalten.
- Zahlen sollen binär (oder k-är für $k \neq 1$) kodiert sein.

Kodierungsschema

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 $X = (x_1, \ldots, x_n)$ 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 Π, falls es Polynome p_1, p_2 gibt, so dass gilt:

$$(|s_1(I)| = n \Rightarrow |s_2(I)| \leq p_2(n)) \text{ und } (|s_2(I)| = m \Rightarrow |s_1(I)| \leq p_1(m))$$

für alle Problembeispiele I von Π.

Entscheidungsprobleme

- Ein Entscheidungsproblem Π können wir als Klasse von Problembeispielen D_Π auffassen.
- Eine Teilmenge dieser Klasse ist $J_\Pi \subseteq D_\Pi$, die Klasse der **Ja–Beispiele**, d.h. die Problembeispiele deren Antwort Ja ist.
- Der Rest der Klasse $N_\Pi \subseteq D_\Pi$ ist die Klasse der **Nein–Beispiele**.

Korrespondenz von Entscheidungsproblemen und Sprachen

Ein Problem Π und ein Kodierungsschema $s\colon D_\Pi \to \Sigma^*$ zerlegen Σ^* in drei Klassen:

- Wörter aus Σ^*, die *nicht* Kodierung eines Beispiels aus D_Π sind,
- Wörter aus Σ^*, die Kodierung eines Beispiels $I \in N_\Pi$ sind,
- Wörter aus Σ^*, die Kodierung eines Beispiels $I \in J_\Pi$ sind.

Die dritte Klasse ist die Sprache, die zu Π im Kodierungsschema s korrespondiert.

Die zu einem Problem Π und einem Kodierungsschema s **zugehörige Sprache** ist

$$L[\Pi, s] := \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} \Sigma \text{ ist das Alphabet zu } s \text{ und } x \text{ ist Kodierung} \\ \text{eines Ja–Beispiels } I \text{ von } \Pi \text{ unter } s, \text{ d.h. } I \in J_\Pi \end{array} \right\}$$

- 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 Π 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.

Zeitkomplexität

Für eine deterministische Turing-Maschine $\mathcal{M}$, die für alle Eingaben über dem Eingabe-Alphabet Σ hält, ist die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ definiert durch

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

Die Klasse $\mathcal{P}$

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

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

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 $i, j \in V := \{1, \ldots, n\}$, Algorithmus $\mathcal{A}$

Output: d_{ij} $(1 \leq i, j \leq n)$, so dass alle bis auf n der d_{ij}-Werte den Wert $\left(\max_{i,j} c_{ij}\right) + 1$ 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)

- berechne $m := \max_{1 \le i,j \le n} c_{ij}$;
  setze L(ow) := 0 und H(igh) := $n \cdot m$;
- solange $H - L > 1$ gilt, führe aus:

$$\left.\begin{array}{l} \text{falls } \mathcal{A}\left(n, c, \left\lceil \frac{1}{2}(H+L) \right\rceil\right) = \text{nein ist,} \\ \qquad \text{setze } L := \left\lceil \frac{1}{2}(H+L) \right\rceil + 1; \\ \text{sonst} \\ \qquad \text{setze } H := \left\lceil \frac{1}{2}(H+L) \right\rceil; \end{array}\right\} (* \text{ binäre Suche } *)$$

- falls $\mathcal{A}(n, c, L) =$„nein“ ist,setze $OPT := H$; sonst setze $OPT := L$;
- für $i = 1 \ldots n$ führe aus
- für $j = 1 \ldots n$ führe aus
- setze $R := c_{ij}$; $c_{ij} := m + 1$;
  falls $\mathcal{A}(n, c, OPT) =$ nein ist, setze $c_{ij} := R$;
- setze $d_{ij} = c_{ij}$;

Bemerkungen zum Algorithmus

Die Schleife in Schritt 2 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 $L \neq \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$ und $H \neq \left\lceil \frac{1}{2}(H + L) \right\rceil$ ist.
- Die Differenz verkleinert sich also mit jedem Durchlauf
- Da H und L ganzzahlig sind, tritt der Fall $H - L \leq 1$ ein.

- Nach Abbruch der Schleife gilt $H - L \geq 0$:
  - Eine Differenz zwischen H und L von 0 kann genau durch Erhöhen von L bei einer aktuellen Differenz von 2 bzw. 3 erreicht werden
  - Eine Differenz zwischen H und L von 0 ist minimal (bei einer Differenz von weniger als 2 wird die Schleife nicht mehr betreten).

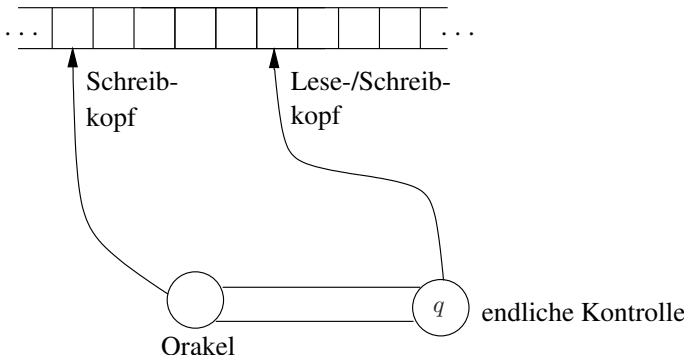
Laufzeit des Algorithmus

- In 2. wird $\mathcal{A}(n, c, k)$ etwa $\log(n \cdot m)$–mal aufgerufen
- In 4. wird $\mathcal{A}(n, c, OPT)$ etwa n^2–mal aufgerufen.
- Es finden also $\mathcal{O}(n^2 + \log(nm))$ Aufrufe von $\mathcal{A}$ statt.
- Die Inputlänge ist $\mathcal{O}(n^2 \cdot \log(\max c_{ij}))$.
- Da $\mathcal{A}$ polynomial ist, ist dies also auch OPT-TOUR.

Die Nichtdeterministische Turingmaschine

- Bei der nichtdeterministischen Turing-Maschine wird die Übergangsfunktion δ zu einer Relation erweitert.
- Dies ermöglicht Wahlmöglichkeiten und ε–Übergänge (vergleiche endliche Automaten).
- Wir betrachten ein äquivalentes Modell einer nichtdeterministischen Turing-Maschine (NTM), die auf einem **Orakel** basiert.
- Dies kommt der Intuition näher.

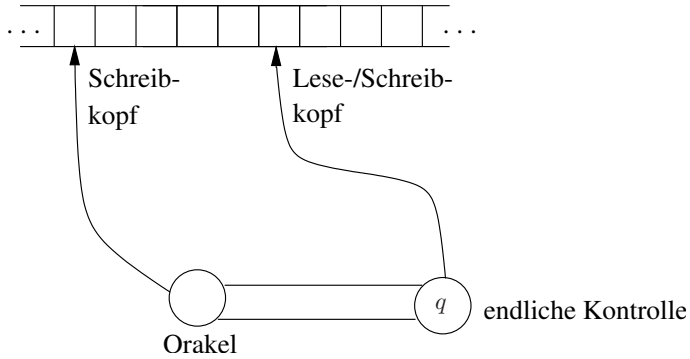
Die Nichtdeterministische Turingmaschine



Nichtdeterministische Turingmaschinen (NTM)

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

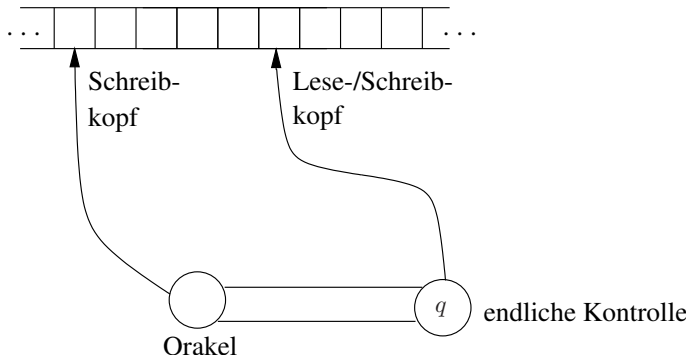
Die Nichtdeterministische Turingmaschine



Berechnung bei 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.

Die Nichtdeterministische Turingmaschine



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

Übertragung auf Entscheidungsprobleme Π

Die Eingabe ist ein Wort aus Σ^*, zum Beispiel eine Kodierung eines Problembeispiels $I \in D_\Pi$.

- **1. Stufe:** Es wird ein Orakel aus Γ^* berechnet, zum Beispiel ein Lösungsbeispiel für I, also ein Indikator, ob $I \in J_\Pi$ oder $I \in N_\Pi$ gilt.
- **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 Permutation σ auf der Knotenmenge V vorgeschlagen. D.h. $(\sigma(1), \ldots, \sigma(n))$, $G = (V, E)$, c und k bilden die Eingabe.
- **2. Stufe:** Es wird nun überprüft, ob $\sigma(V)$ eine Tour in G enthält, deren Länge bezüglich c nicht größer als k ist.

Bemerkungen zur NTM

- Das Orakel kann ein beliebiges Wort aus Γ^* sein.
- Darum muss in der Überprüfungsphase (2.Stufe) zunächst geprüft werden, ob das Orakel ein zulässiges Lösungsbeispiel ist.
- Ist dies nicht der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort „Nein“ beendet werden.

- Jede NTM $\mathcal{M}$ hat zu einer gegebenen Eingabe x eine unendliche Anzahl möglicher Berechnungen, eine zu jedem Orakel aus Γ^*.
- Endet mindestens eine in q_J, so wird x akzeptiert.

Zeitkomplexität für NTM

- 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_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

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

Zeitkomplexität für NTM

- 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_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

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

Bemerkung 1

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

Zeitkomplexität für NTM

- 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_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine $\mathcal{M}$ ist definiert durch

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

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 NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

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

Die Klasse NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

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

Bemerkungen

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

Die Klasse NP

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L, für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

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

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

Denn zu gegebenem $G = (V, E), c, k$ und einer festen Permutation σ auf V kann man in $O(|V| \cdot \log C)$, wobei C die größte vorkommende Zahl ist, überprüft werden, ob

$$\sum_{i=1}^{n-1} c(\{\sigma(i), \sigma(i+1)\}) + c(\{\sigma(n), \sigma(1)\}) \leq k$$

gilt.