Kapitel: Berechenbarkeit

Worum geht's in diesem Kapitel?

- Welche Art von Problemen können wir prinzipiell mit Computern lösen? Welche nicht?
- Dabei wollen wir uns nicht die Details real existierender Computer anschauen, sondern eine technologie-unabhängige Formalisierung des Begriffs der Berechenbarkeit einer Funktion, beziehungsweise der Entscheidbarkeit einer Sprache L entwickeln. Denn:
  - Wir wissen nicht, wie Rechner zukünftiger Technologien aufgebaut sind.
  - Unsere Begriffsbildung sollte für heutige von Neumann Rechner genau so gelten wie z.B. für Quanten- oder DNA-Rechner!
- Welche Sprachen oder Probleme sind zu schwierig, können also durch Rechner auch in zukünftigen Technologien nicht gelöst werden?
  - Wir untersuchen entscheidbare und unentscheidbare Sprachen.

Das Halteproblem

Automatische Verifikation

- Wäre es nicht schön, wenn Computer richtig funktionieren würden?
- Wenigstens sollten sie nicht ständig abstürzen.
- Wir könnten ein Programm schreiben, das automatisch prüft, ob andere Programme zum Absturz (z.B. in eine Endlosschleife) führen.

- Wir verwenden hier zunächst einen intuitiven Begriff von “Algorithmus” und “Programm” (hier können wir uns z.B. JAVA-Programme oder C++-Programme vorstellen).

  Später in diesem Kapitel werden wir den intuitiven Begriff präzisieren.

Das Halteproblem (1/2)

HALTEPROBLEM

Eingabe: Ein Programm P und eine Eingabe E.

Ausgabe: $\begin{cases} \text{„Hält“} & \text{wenn P bei Eingabe E anhält} \\ \text{„Hält nicht“} & \text{sonst.} \end{cases}$

Frage: Gibt es ein Programm, das das Halteproblem löst?

Nehmen wir mal an, STOP wäre ein Programm, das das Halteproblem löst.

Dann konstruieren wir ein neues Programm POTS wie folgt:

Programm POTS: Die Eingabe besteht aus einem Programm P

(1) STOP wird auf

Programm P und Eingabe P

angesetzt.

(2) Bei Ausgabe „Hält nicht“ hält das Programm POTS an.

(3) Bei Ausgabe „Hält“ läuft das Programm POTS in eine Endlosschleife und hält nie an.

Das Halteproblem (2/2)

Dann gilt:

POTS angesetzt auf P hält an

$\Longleftrightarrow$ STOP angesetzt auf
Programm P und Eingabe P
gibt „Hält nicht“ aus

$\Longleftrightarrow$ P angesetzt auf P hält nicht an

Setzen wir jetzt POTS auf sich selbst an:

POTS angesetzt auf POTS hält an

$\Longleftrightarrow$ POTS angesetzt auf POTS hält nicht an

Da stimmt was nicht! Aber was?
Unsere Annahme, dass es ein Programm STOP für das Halteproblem gibt, muss falsch gewesen sein.
Also gibt es kein Programm, das das Halteproblem löst!

Der Satz von Rice

Okay, dann können wir das Halteproblem eben nicht lösen.

Aber es gibt ja noch andere interessante Probleme, z.B.:

- Sind zwei Programme äquivalent?
- Gibt ein Programm stets die Zahl “42” aus?
- Berechnet ein Programm bei Eingabe einer Zahl x den Wert $f(x)$?

Aus dem folgenden Satz von Rice folgt, dass keine dieser Fragen durch ein Computerprogramm entschieden werden kann.

Der Satz von Rice

Keine (nicht-triviale) Eigenschaft von Programmen kann von einem Computerprogramm “erkannt” werden.

Genauer: Sei B die Menge aller berechenbaren Funktionen. Sei $F \subseteq B$ mit $\emptyset \neq F \neq B$. Dann gibt es keinen Algorithmus, der bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion zur Menge F gehört.

Später in diesem Kapitel werden wir den Satz von Rice präzise formulieren und beweisen. Aber zunächst sollten wir uns anschauen, was die Aussage “. . . kann nicht durch ein Computerprogramm entschieden werden” genau bedeuten soll.

Entscheidbarkeit, Semi-Entscheidbarkeit rekursive Aufzählbarkeit und Berechenbarkeit

Entscheidungsprobleme

Wir betrachten im Folgenden meistens Entscheidungsprobleme, d.h. Probleme, die mit "ja" oder "nein" beantwortet werden können:

- Sei M eine abzählbar unendliche Menge, zum Beispiel
  - die Menge Σ^* aller Worte über einem endlichen Alphabet Σ, oder
  - die Menge aller Graphen, deren Knotenmenge eine endliche Teilmenge der natürlichen Zahlen ist.

- Das Entscheidungsproblem für eine Menge $L \subseteq M$ ist das folgende Berechnungsproblem:

> Das Entscheidungsproblem für $L \subseteq M$
>
> *Eingabe:* Ein Element $m \in M$.
>
> *Frage:* Ist $m \in L$?

Beispiele für Entscheidungsprobleme

- Das Halteproblem ist das Entscheidungsproblem für $L \subseteq M$, wobei
  - M die Menge aller Tupel (P, E) von Programmen P und Eingaben E ist und
  - L die Menge aller (P, E) ist, so dass P bei Eingabe E anhält.

- Graphzusammenhang ist das Entscheidungsproblem für $L \subseteq M$, wobei
  - M die Menge aller ungerichteten Graphen ist, deren Knotenmenge eine endliche Teilmenge von $\mathbb{N}$ ist und
  - L die Menge aller zusammenhängenden Graphen aus M ist.

Entscheidbarkeit und Semi-Entscheidbarkeit

Definition:

Sei M eine abzählbar unendliche Menge.

(a) Eine Menge $L \subseteq M$ heißt entscheidbar, falls es einen Algorithmus gibt, der bei Eingabe eines $m \in M$ nach endlich vielen Schritten anhält und

- "ja" ausgibt, falls $m \in L$
- "nein" ausgibt, falls $m \notin L$.

(b) $L \subseteq M$ heißt semi-entscheidbar, falls es einen Algorithmus gibt, der bei Eingabe eines $m \in M$

- nach endlich vielen Schritten anhält und "ja" ausgibt, falls $m \in L$
- nie anhält, falls $m \notin L$.

Beispiele:

- Graphzusammenhang ist entscheidbar (z.B. durch Tiefen- oder Breitensuche).
- Das Halteproblem ist semi-entscheidbar (bei Eingabe von P und E lasse einfach Programm P mit Eingabe E laufen).
  Ist es auch entscheidbar? Nein! Ganz am Anfang dieses Kapitels haben wir aus der Annahme, dass das Halteproblem durch ein Programm STOP entschieden wird, einen Widerspruch hergeleitet.

Einfache Beobachtungen

- Jede entscheidbare Menge $L \subseteq M$ ist auch semi-entscheidbar (anstatt "nein" auszugeben und anzuhalten, gehen wir einfach in eine Endlosschleife).

- Für jede entscheidbare Menge $L \subseteq M$ ist auch die Menge $\overline{L} := (M \setminus L) \subseteq M$ entscheidbar (vertausche einfach die Antworten "ja" und "nein").

Entscheidbarkeit vs. Semi-Entscheidbarkeit

Satz: Sei M eine abzählbare Menge.

Wenn sowohl $L \subseteq M$ als auch $\overline{L} := (M \setminus L) \subseteq M$ semi-entscheidbar sind, dann sind $L \subseteq M$ und $\overline{L} \subseteq M$ sogar entscheidbar.

Beweis: Sei A ein Algorithmus, der $L \subseteq M$ semi-entscheidet, und sei B ein Algorithmus, der $\overline{L} \subseteq M$ semi-entscheidet.

Um $L \subseteq M$ zu entscheiden, bauen einen Algorithmus C, der bei Eingabe von $m \in M$ wie folgt vorgeht:

(1) Für $i = 1, 2, 3, \ldots$ tue folgendes:

(2) Führe den i-ten Berechnungsschritt von Algorithmus A bei Eingabe m aus. Falls A in diesem Schritt anhält, so gib "ja" aus und halte an.

(3) Führe den i-ten Berechnungsschritt von Algorithmus B bei Eingabe m aus. Falls B in diesem Schritt anhält, so gib "nein" aus und halte an.

Falls $m \in L$, so wird A nach endlich vielen Schritten anhalten — und dann hält auch C mit der korrekten Ausgabe "ja".
Falls $m \notin L$, so ist $m \in \overline{L}$. Daher wird B nach endlich vielen Schritten anhalten — und dann hält auch C mit der korrekten Ausgabe "nein". □

Beobachtungen

- Es gibt entscheidbare Probleme.
  Beispiele: Graphzusammenhang, Hamiltonkreis, 3-Färbbarkeit, das aussagenlogische Erfüllbarkeitsproblem, ...

- Es gibt Probleme, die semi-entscheidbar sind, aber nicht entscheidbar.
  Beispiel: das Halteproblem

- **Frage:** Gibt es auch Probleme, die **nicht** semi-entscheidbar sind?

  Antwort: Ja!

  Sei $L \subseteq M$ ein semi-entscheidbares, aber nicht entscheidbares Problem. Wenn $\overline{L} := (M \setminus L) \subseteq M$ semi-entscheidbar wäre, dann wäre gemäß dem Satz auf der vorherigen Folie $L \subseteq M$ sogar entscheidbar. WIDERSPRUCH!

  Somit gilt: Für jedes semi-entscheidbare, aber nicht entscheidbare Problem $L \subseteq M$ ist $\overline{L} := (M \setminus L) \subseteq M$ ist nicht semi-entscheidbar.

  Folgerung: Das Komplement des Halteproblems ist nicht semi-entscheidbar.

Rekursive Aufzählbarkeit

Manchmal wollen wir keine Algorithmen, die eine ja/nein-Antwort liefern, sondern solche, die nach und nach alle Elemente einer Menge ausgeben.

Schön wäre z.B. ein Algorithmus, der nach und nach alle korrekten mathematischen Sätze (etwa der Zahlen- oder Gruppentheorie) ausgibt.

Definition:

Sei M eine abzählbare Menge.

Eine Menge $L \subseteq M$ heißt rekursiv aufzählbar (kurz: r.e., für "recursively enumerable"), falls es einen Algorithmus A gibt, der nach und nach sämtliche Elemente aus L ausgibt.

Genauer:
Sind $m_1, m_2, m_3, \ldots$ die nacheinander von A ausgegebenen Elemente, so gilt:

- Für jedes $i \in \mathbb{N}_{>0}$ ist $m_i \in L$.
- Für jedes $\ell \in L$ gibt es mindestens ein $i \in \mathbb{N}$, so dass $m_i = \ell$.

Für den Fall, dass $M = \Sigma^*$ ist (für ein endliches Alphabet Σ), stellt sich heraus, dass rekursive Aufzählbarkeit derselbe Begriff ist wie Semi-Entscheidbarkeit:

Rekursiv aufzählbar $\iff$ semi-entscheidbar

Satz: Sei Σ ein endliches Alphabet und sei $L \subseteq \Sigma^*$. Es gilt:

L ist genau dann rekursiv aufzählbar, wenn $L \subseteq \Sigma^*$ semi-entscheidbar ist.

Beweis:

"Rekursiv aufzählbar $\Longrightarrow$ semi-entscheidbar": Übungsaufgabe!

"Semi-entscheidbar $\Longrightarrow$ rekursiv aufzählbar": Sei A ein Algorithmus, der $L \subseteq \Sigma^*$ semi-entscheidet. D.h. bei Eingabe eines Worts $w \in \Sigma^*$ wird A

- nach endlich vielen Schritten anhalten (und "ja" ausgeben), falls $w \in L$ ist,
- nie anhalten, falls $w \notin L$ ist.

Wir bauen folgenden Algorithmus B: *(Methode: "Verzahnung", engl.: "dove tailing")*

(1) Für $i = 1, 2, 3, \ldots$ tue folgendes:
(2) Für jedes Wort $w \in (\Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^i)$ tue folgendes:
(3) Führe die ersten i Berechnungsschritte von Algorithmus A bei Eingabe w aus.
(4) Falls A innerhalb dieser Schritte anhält, gib w aus.

Beachte: Jedes Wort, das B ausgibt, gehört zur Menge L. Jedes Wort $w \in L$ wird irgendwann von B ausgegeben. Somit wird L von B rekursiv aufgezählt. □

Berechenbare Funktionen

Manchmal wollen wir nicht einfach nur Mengen aufzählen oder ja/nein-Antworten erhalten, sondern Funktionen berechnen.

Definition: Seien M und M' abzählbare Mengen

Eine partielle Funktion f von M nach M' heißt berechenbar, falls es einen Algorithmus gibt, der bei Eingabe eines $m \in M$

- nach endlich vielen Schritten anhält und $f(m)$ ausgibt, falls m im Definitionsbereich von f liegt
- nie anhält, falls m nicht im Definitionsbereich von f liegt.

Beispiele:

- Die Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) := 2^n$ für alle $n \in \mathbb{N}$ ist berechenbar.
- Die Funktion $f_\perp$ mit leerem Definitionsbereich ist berechenbar (durch einen Algorithmus, der nur aus einer Endlosschleife besteht).
- Die Funktion g, die jedem ungerichteten Graphen, dessen Knotenmenge eine endliche Teilmenge von $\mathbb{N}$ ist, die Anzahl seiner Zusammenhangskomponenten zuordnet, ist berechenbar.
- Die Funktion h, die jedem Programm P und jeder Eingabe E den Wert 1 (bzw. 0) zuordnet, wenn P bei Eingabe E anhält (bzw. nicht anhält), ist nicht berechenbar.

Berechenbarkeit vs. Entscheidbarkeit

Anstatt das Entscheidungsproblem für $L \subseteq M$ können wir auch die charakteristische Funktion $\chi_L : M \to \{0, 1\}$ betrachten, die wie folgt definiert ist:

$$\chi_L(m) := \begin{cases} 1 & \text{falls } m \in L \\ 0 & \text{falls } m \notin L \end{cases}$$

Beobachtung: Sei M eine abzählbare Menge und sei $L \subseteq M$.

$L \subseteq M$ ist genau dann entscheidbar, wenn die Funktion χ_L berechenbar ist.

Beweis:

"$\Longrightarrow$": Sei A ein Algorithmus, der $L \subseteq M$ entscheidet. Anstatt "ja" (bzw. "nein") auszugeben, gib einfach 1 (bzw. 0) aus. Dies ist dann ein Algorithmus, der χ_L berechnet.

"$\Longleftarrow$": Sei B ein Algorithmus, der χ_L berechnet. Anstatt 1 (bzw. 0) auszugeben, gib einfach "ja" (bzw. "nein") aus. Dies ist dann ein Algorithmus, der $L \subseteq M$ entscheidet. □

Berechenbarkeit vs. Semi-Entscheidbarkeit

Anstatt das Entscheidungsproblem für $L \subseteq M$ können wir auch die partielle Funktion χ_L^* von M nach $\{1\}$ betrachten, deren Definitionsbereich die Menge L ist, und die jedes $m \in L$ auf die Zahl $\chi_L^*(m) := 1$ abbildet.

Beobachtung: Sei M eine abzählbare Menge und sei $L \subseteq M$.

$L \subseteq M$ ist genau dann semi-entscheidbar, wenn die partielle Funktion χ_L^* berechenbar ist.

Beweis:

"$\Longrightarrow$": Sei A ein Algorithmus, der $L \subseteq M$ semi-entscheidet. Anstatt "ja" auszugeben, gib einfach 1 aus. Dies ist dann ein Algorithmus, der χ_L^* berechnet.

"$\Longleftarrow$": Sei B ein Algorithmus, der χ_L^* berechnet. Anstatt anzuhalten und $\chi_L^*(m)$ auszugeben, halte an und gib "ja" aus. Dies ist dann ein Algorithmus, der $L \subseteq M$ semi-entscheidet. □

Der Satz von Rice

Der Satz von Rice

Sei Σ ein endliches Alphabet und sei B die Menge aller berechenbaren partiellen Funktionen von Σ^* nach Σ^*. Sei $F \subseteq B$ mit $\emptyset \neq F \neq B$.
Dann gibt es keinen Algorithmus, der bei Eingabe eines Programms P entscheidet, ob die von P berechnete partielle Funktion zur Menge F gehört.

Beweis: Durch Widerspruch.

Angenommen, A_F ist ein Algorithmus, der bei Eingabe eines Programms Q nach endlich vielen Schritten anhält und "ja" (bzw. "nein") ausgibt, falls die von Q berechnete partielle Funktion zur Menge F gehört (bzw. nicht zur Menge F gehört).

Ziel: Konstruiere einen Algorithmus STOP, der das Halteproblem löst, d.h. der bei Eingabe eines Programms P und einer Eingabe E nach endlich vielen Schritten anhält und "ja" (bzw. "nein") ausgibt, falls P bei Eingabe E anhält (bzw. nicht anhält).

Beachte:

Wir wissen bereits, dass es einen solchen Algorithmus STOP nicht geben kann!

Der Satz von Rice (2/3)

Sei $f_\perp$ die überall undefinierte Funktion (d.h. die partielle Funktion mit leerem Definitionsbereich).

Fall 1: $f_\perp \notin F$:
Da $F \neq \emptyset$ ist, gibt es eine berechenbare partielle Funktion $g \in F$. Sei A_g ein Algorithmus, der g berechnet.

Sei STOP ein Algorithmus, der bei Eingabe von P und E folgendes tut:

(1) Konstruiere ein Programm Q, das bei Eingabe eines Worts $x \in \Sigma^*$ folgendes tut:
  (1) Lasse P mit Eingabe E laufen.
  (2) Lasse A_g mit Eingabe x laufen.

(2) Lasse A_F mit Eingabe Q laufen.

Gemäß dieser Konstruktion gilt:

- Falls P bei Eingabe E hält, so berechnet Q die partielle Funktion $g \in F$, und daher gibt A_F bei Eingabe von Q die Antwort “ja” aus.
- Falls P bei Eingabe E nicht hält, so berechnet Q die partielle Funktion $f_\perp \notin F$, und daher gibt A_F bei Eingabe von Q die Antwort “nein” aus.

Somit entscheidet der Algorithmus STOP das Halteproblem. WIDERSPRUCH!

Der Satz von Rice (3/3)

Fall 2: $f_\perp \in F$:
Da $F \neq B$ ist, gibt es eine berechenbare partielle Funktion g mit $g \notin F$. Sei A_g ein Algorithmus, der g berechnet.

Sei STOP ein Algorithmus, der bei Eingabe von P und E folgendes tut:

(1) Konstruiere ein Programm Q, das bei Eingabe eines Worts $x \in \Sigma^*$ folgendes tut:
  (1) Lasse P mit Eingabe E laufen.
  (2) Lasse A_g mit Eingabe x laufen.

(2) Lasse A_F mit Eingabe Q laufen.

(3) Falls A_F "nein" ausgibt, so gib "ja" aus. Falls A_F "ja" ausgibt, so gib "nein" aus.

Gemäß dieser Konstruktion gilt:

- Falls P bei Eingabe E hält, so berechnet Q die partielle Funktion $g \notin F$, und daher gibt A_F bei Eingabe von Q die Antwort "nein" aus.
- Falls P bei Eingabe E nicht hält, so berechnet Q die partielle Funktion $f_\perp \in F$, und daher gibt A_F bei Eingabe von Q die Antwort "ja" aus.

Somit entscheidet der Algorithmus STOP das Halteproblem. WIDERSPRUCH! □

Folgerungen aus dem Satz von Rice (1/2)

Der Satz von Rice

Sei Σ ein endliches Alphabet und sei B die Menge aller berechenbaren partiellen Funktionen von Σ^* nach Σ^*. Sei $F \subseteq B$ mit $\emptyset \neq F \neq B$.
Dann gibt es keinen Algorithmus, der bei Eingabe eines Programms P entscheidet, ob die von P berechnete partielle Funktion zur Menge F gehört.

Sei Σ ein endliches Alphabet mit $2, 4 \in \Sigma$.

Behauptung: Es gibt keinen Algorithmus A_{42}, der bei Eingabe eines Programms P entscheidet, ob P unabhängig von seiner Eingabe stets "42" ausgibt.

Beweis: Sei f_{42} die Funktion von Σ^* nach Σ^* mit $f_{42}(w) = 42$ für alle $w \in \Sigma^*$.
Klar: f_{42} ist berechenbar. Und für $F := \{f_{42}\}$ ist $F \subseteq B$ mit $\emptyset \neq F \neq B$.
Aus dem Satz von Rice folgt, dass es keinen Algorithmus gibt, der bei Eingabe eines Programms P entscheidet, ob die von P berechnete Funktion die Funktion f_{42} ist. □

Folgerungen aus dem Satz von Rice (2/2)

Sei Σ ein endliches Alphabet mit $0, 1 \in \Sigma$.

- Es gibt keinen Algorithmus A_{konstant}, der bei Eingabe eines Programms P entscheidet, ob P unabhängig von seiner Eingabe immer dieselbe Ausgabe produziert. (**Beweis:** Übung!)

- Sei $L \subseteq \Sigma^*$ eine beliebige entscheidbare Sprache.

  Es gibt keinen Algorithmus A_L, der bei Eingabe eines Programms P entscheidet, ob P die charakteristische Funktion der Sprache L berechnet. (**Beweis:** Übung!)

- Es gibt keinen Algorithmus $A_{\text{regulär}}$, der bei Eingabe eines Programms P entscheidet, ob P die charakteristische Funktion einer regulären Sprache berechnet. (**Beweis:** Übung!)

Ein formales Berechnungsmodell: Turingmaschinen

Ein formales Rechnermodell

- Bisher haben wir abstrakt von “Algorithmen” bzw. “Programmen” gesprochen und uns dabei JAVA- oder C++-Programme vorgestellt.

  Die Begriffe und Beweise, die wir uns bisher in diesem Kapitel angeschaut haben, waren aber nicht speziell auf “JAVA” oder “C++” zugeschnitten, sondern funktionieren für jedes sinnvolle Berechnungsmodell.

- Im Folgenden wollen wir ein konkretes Rechnermodell einführen. **Ziele:**
  - Unser Rechnermodell sollte die wesentlichen Eigenschaften heutiger und zukünftiger, leistungsstarker Computer beinhalten.
  - Ein solcher Rechner soll jederzeit auf die Eingabe zugreifen können und Berechnungen ohne Beschränkung der Rechenzeit und des Speicherplatzes ausführen können.
  - Unser Rechnermodell sollte so einfach wie möglich sein, damit wir präzise Beweise führen können.
  - Insbesondere wir wollen von den konkreten technischen Details (Speicherchips etc.) heutiger Rechner abstrahieren.
  - Wir interessieren uns hier zunächst nicht für Effizienz, sondern nur für Berechenbarkeit bzw. (Semi-)Entscheidbarkeit.

Turingmaschinen (kurz: TM)

Eine Turingmaschine besitzt

- eine endliche Zustandsmenge Q und
- ein nach links und nach rechts unbeschränktes **Band**, das in Zellen unterteilt ist.
- Die Zellen speichern Buchstaben aus einem Arbeitsalphabet Γ und besitzen Zahlen aus $\mathbb{Z}$ als Adressen.

Die Turingmaschine manipuliert ihr Band mit Hilfe eines Lese-/Schreibkopfes.

Der Kopf kann den in einer Zelle gespeicherten Buchstaben lesen, ihn überschreiben, und dann in einen neuen Zustand gehen und zur linken oder rechten Nachbarzelle wandern oder auf der aktuellen Zelle verbleiben. **Skizze:** siehe Tafel!

Turingmaschinen in Aktion: (zuletzt besucht am 23.06.2012)

- "The LEGO Turing Machine" auf
  `http://www.youtube.com/watch?v=cYw2ewoO6c4`
- "A Turing Machine in the Classic Style" auf
  `http://www.youtube.com/watch?v=E3keLeMwfHY`

Die Startkonfiguration einer TM

Zu Beginn einer Berechnung gilt Folgendes:

- Die Eingabe $w = w_1 \cdots w_n$ ist in den Zellen $1 \ldots, n$ gespeichert, wobei Zelle i (mit $1 \leqslant i \leqslant n$) den Buchstaben w_i enthält.
- In allen anderen Zellen steht das Leersymbol (engl.: Blanksymbol), das wir mit $\square$ bezeichnen.
- Der Kopf steht auf Zelle 1.
- Der aktuelle Zustand ist der Startzustand q_0.

Skizze: siehe Tafel!

Die Berechnung einer TM

- **Ein einzelner Rechenschritt:**
  - Das Programm einer Turingmaschine wird durch eine **Transitionsfunktion**

$$\delta : \; Q \times \Gamma \; \rightarrow \; Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$$

    beschrieben.
  - Wenn die Maschine sich im Zustand q befindet und den Buchstaben $a \in \Gamma$ liest, dann schaut sie im Programm nach, was $\delta(q, a)$ ist.

    Wenn $\delta(q, a) = (q', a', \texttt{Richtung})$ ist, dann
    - überschreibt die Maschine den Buchstaben a mit a',
    - wechselt in den Zustand q' und
    - wandert zu der durch `Richtung` vorgeschriebenen Nachbarzelle.

- Die Maschine hält, wenn die Maschine im Zustand $q \in Q$ den Buchstaben $\gamma \in \Gamma$ liest und $\delta(q, \gamma) = (q, \gamma, \downarrow)$ gilt.

- $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.
  Die Maschine **akzeptiert** Eingabe w, wenn sie in einem Zustand aus F hält.

Alle Komponenten einer Turingmaschine

Eine Turingmaschine $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ besteht aus

- einem endlichen Eingabealphabet Σ, in dem das Leersymbol $\square$ **nicht** enthalten ist,
- einer endlichen Zustandsmenge Q,
- einem endlichen Arbeitsalphabet Γ mit $\Gamma \supseteq \Sigma \cup \{\square\}$.
- einem Programm $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$,
- einem Startzustand $q_0 \in Q$,
- einer Menge $F \subseteq Q$ von akzeptierenden Zuständen.

Beispiel: Eine TM für die Sprache $L := \{a^n b^n : n \in \mathbb{N}_{>0}\}$ (1/2)

Wir bauen eine TM $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$, die stets anhält und genau diejenigen Eingabeworte akzeptiert, die in der Sprache L liegen. Wähle $\Gamma = \{a, b, \square, 1, 2\}$.

Unsere TM geht wie folgt vor: **Skizze:** siehe Tafel!

- Überprüfe, dass das Eingabewort von der Form $a^+ b^+$ ist. Falls nicht, halte in einem nicht-akzeptierenden Zustand. Ansonsten setze den Kopf wieder an die erste Position des Worts, überschreibe das aktuell gelesene a mit 1, geh in den Zustand q_r und geh einen Schritt nach rechts.
- Bleib im Zustand q_r und geh, solange du a oder 2 liest, immer weiter nach rechts. Falls du dabei irgendwann ein Leerzeichen $\square$ liest, halte in einem nicht-akzeptierenden Zustand. Falls du stattdessen irgendwann ein b liest, überschreib dieses durch 2, geh in den Zustand q_ℓ und geh einen Schritt nach links.
- Bleib im Zustand q_ℓ und geh, solange du 2 oder a liest, immer weiter nach links. Sobald du eine 1 liest, geh in den Zustand q_1 und geh einen Schritt nach rechts. Wenn Du dann ein a liest, überschreib es mit 1, geh in den Zustand q_r und geh einen Schritt nach rechts. Wenn Du stattdessen eine 2 liest, geh in den Zustand q_2 und geh einen Schritt nach rechts.
- Bleib im Zustand q_2 und geh, solange du 2 liest, immer weiter nach rechts. Wenn du dabei irgendwann ein Leerzeichen $\square$ liest, halte in einem akzeptierenden Zustand. Wenn Du stattdessen irgendwann ein Zeichen $\notin \{2, \square\}$ liest, halte in einem nicht-akzeptierenden Zustand.

Beispiel: Eine TM für die Sprache $L := \{a^n b^n : n \in \mathbb{N}_{>0}\}$ (2/2)

Somit hat unsere TM $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ die folgenden Komponenten:

- $\Sigma = \{a, b\}, \quad \Gamma = \{a, b, \square, 1, 2\},$
- $Q = \{q_0, q_0', q_a, q_b, q_\leftarrow, q_r, q_\ell, q_1, q_2, q_{\text{accept}}, q_{\text{reject}}\}, \quad F = \{q_{\text{accept}}\}$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$ mit

$$
\begin{array}{lcll}
\delta(q, \gamma) & = & (q, \gamma, \downarrow) & \text{für alle } q \in \{q_{\text{accept}}, q_{\text{reject}}\} \text{ und alle } \gamma \in \Gamma \\
\delta(q_0, a) & = & (q_a, a, \rightarrow) & \\
\delta(q_\gamma, \gamma) & = & (q_\gamma, \gamma, \rightarrow) & \text{für alle } \gamma \in \{a, b\} \\
\delta(q_a, b) & = & (q_b, b, \rightarrow) & \delta(q_b, \square) = (q_\leftarrow, \square, \leftarrow) \\
\delta(q_\leftarrow, \gamma) & = & (q_\leftarrow, \gamma, \leftarrow) & \text{für alle } \gamma \in \{a, b\} \\
\delta(q_\leftarrow, \square) & = & (q_0', \square, \rightarrow) & \delta(q_0', a) = (q_r, 1, \rightarrow) \\
\delta(q_r, \gamma) & = & (q_r, \gamma, \rightarrow) & \text{für alle } \gamma \in \{a, 2\} \\
\delta(q_r, \square) & = & (q_{\text{reject}}, \square, \downarrow) & \delta(q_r, b) = (q_\ell, 2, \leftarrow) \\
\delta(q_\ell, \gamma) & = & (q_\ell, \gamma, \leftarrow) & \text{für alle } \gamma \in \{2, a\} \\
\delta(q_\ell, 1) & = & (q_1, 1, \rightarrow) & \\
\delta(q_1, a) & = & (q_r, 1, \rightarrow) & \delta(q_1, 2) = (q_2, 2, \rightarrow) \\
\delta(q_2, 2) & = & (q_2, 2, \rightarrow) & \delta(q_2, \square) = (q_{\text{accept}}, \square, \downarrow) \\
\delta(q_2, \gamma) & = & (q_{\text{reject}}, \gamma, \downarrow) & \text{für alle } \gamma \in \Gamma \setminus \{2, \square\} \\
\delta(q, \gamma) & = & (q_{\text{reject}}, \gamma, \downarrow) & \text{für alle bisher noch nicht genannten } (q, \gamma) \in Q \times \Gamma.
\end{array}
$$

Man kann sich leicht davon überzeugen, dass diese TM bei jeder Eingabe $w \in \Sigma^*$ anhält und genau die Worte aus L akzeptiert.

Die von einer TM T akzeptierte Sprache $L(T)$ und die von ihr berechnete partielle Funktion f_T

Sei $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ eine Turingmaschine.

Die von T akzeptierte Sprache ist

$$L(T) := \left\{ w \in \Sigma^* : \begin{array}{l} T \text{ akzeptiert } w, \text{ d.h. bei Eingabe } w \text{ hält } T \text{ nach endlich} \\ \text{vielen Schritten an und ist dann in einem Zustand aus } F \end{array} \right\}.$$

Die von T berechnete Funktion

ist die partielle Funktion f_T von Σ^* nach Σ^*, so dass für alle $w \in \Sigma^*$ gilt:

- $f_T(w)$ ist undefiniert, falls T bei Eingabe w nicht anhält.
- $f_T(w) = u$, falls T bei Eingabe w nach endlich vielen Schritten anhält und dabei u dasjenige Wort maximaler Länge aus Σ^* ist, das beginnend mit der aktuellen Kopfposition auf dem Band steht.

TM-Berechenbarkeit und TM-Entscheidbarkeit

Sei Σ ein endliches Alphabet und sei $M := \Sigma^*$.

Definition:

(a) Eine partielle Funktion f von M nach M ist TM-berechenbar, falls es eine Turingmaschine T mit $f_T = f$ gibt.

(b) Eine Sprache $L \subseteq M$ heißt TM-semi-entscheidbar, falls es eine Turingmaschine T mit $L(T) = L$ gibt. *Beachte:* Dies ist genau dann der Fall, wenn es eine TM T' gibt, die bei Eingabe eines $m \in M$

- nach endlich vielen Schritten anhält und akzeptiert, falls $m \in L$
- nie anhält, falls $m \notin L$.

(c) Eine Sprache $L \subseteq M$ heißt TM-entscheidbar, falls es eine Turingmaschine gibt, die bei Eingabe eines $m \in M$ nach endlich vielen Schritten anhält und

- akzeptiert, falls $m \in L$
- nicht akzeptiert, falls $m \notin L$.

Die Church-Turing-These

Die Church-Turing-These besagt, dass Turingmaschinen genauso mächtig sind wie jedes andere sinnvolle Berechnungsmodell:

Church-Turing-These:

Sei Σ ein endliches Alphabet und sei $M := \Sigma^*$.

- Eine Sprache $L \subseteq M$ ist genau dann entscheidbar, wenn sie TM-entscheidbar ist.
- Eine Sprache $L \subseteq M$ ist genau dann semi-entscheidbar, wenn sie TM-semi-entscheidbar ist.
- Eine partielle Funktion f von M nach M ist genau dann berechenbar, wenn sie TM-berechenbar ist.

Beachte: Die Church-Turing-These, so wie sie hier formuliert ist, kann man nicht beweisen, da die Begriffe der “Enscheidbarkeit”, “Semi-Entscheidbarkeit” und “Berechenbarkeit”, die wir am Anfang dieses Kapitels eingeführt haben, nicht präzise definiert sind (wir hatten dort kein konkretes Berechnungsmodell angegeben). Aber für jedes bisher betrachtete konkrete Berechnungsmodell konnte die Church-Turing-These tatsächlich bewiesen werden:

Church-Turing-These für konkrete Berechnungsmodelle

Satz:

Sei Σ ein endliches Alphabet und sei f eine partielle Funktion von Σ^* nach Σ^*. Dann sind die folgenden Aussagen äquivalent:

(1) f ist TM-berechenbar.

(2) f kann durch ein WHILE-Programm berechnet werden.

(3) f kann durch ein GOTO-Programm berechnet werden.

(4) f ist eine μ-rekursive Funktion.

(5) f kann durch eine Registermaschine (kurz: RAM, für "random access machine") berechnet werden.

(6) f kann von einer probabilistischen Turingmaschine berechnet werden.

(7) f kann von einer **Mehrband-Turingmaschine** berechnet werden.

(8) f kann von einem Quantencomputer berechnet werden.

(9) f kann durch ein JAVA-Programm berechnet werden.

Hier ohne Beweis.

Ein Beweis der Äquivalenz von (1)–(4) findet sich z.B. im Buch von Schöning. Ein Beweis der Äquivalenz von (1), (7) und (5) findet sich in Prof. Schnitgers Skript zur Vorlesung "Formale Sprachen und Berechenbarkeit" (SoSe 2011).

Mehrband-Turingmaschinen

Mehrband-Turingmaschinen

Sei $k \in \mathbb{N}_{>0}$. Eine k-Band Turingmaschine $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ besitzt k nach links und rechts unbeschränkte Bänder und für jedes Band einen Lese-/Schreibkopf.

- Σ ist das endliche Eingabealphabet, in dem das Leersymbol $\square$ nicht enthalten ist.
- Q ist die endliche Zustandsmenge.
- Γ ist das endliche Arbeitsalphabet mit $\Gamma \supseteq \Sigma \cup \{\square\}$.
- $q_0 \in Q$ ist der Startzustand.
- $F \subseteq Q$ ist die Menge von akzeptierenden Zuständen.
- Das Programm δ ist eine Funktion

$$\delta \; : \; Q \times \Gamma^k \; \to \; Q \times \Gamma^k \times \{\leftarrow, \downarrow, \rightarrow\}^k$$

Wenn sich die Maschine im Zustand q befindet und auf den k Bändern die Symbole $a_1, \ldots, a_k$ liest, dann gibt $\delta(q, a_1, \ldots, a_k) = (q', a'_1, \ldots, a'_k, b_1, \ldots, b_k)$ an, was die Maschine im nächsten Schritt tut: Sie geht in den Zustand q', schreibt die Symbole $a'_1, \ldots, a'_k$ an die aktuellen Kopfpositionen auf den k Bändern und bewegt, für jedes $i \in \{1, \ldots, k\}$, den Kopf auf Band i in Richtung b_i.

Start- und Endkonfiguration einer Mehrband-TM

Sei $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ eine k-Band-TM.

- Die Startkonfiguration von T bei Eingabe $w \in \Sigma^*$ ist wie folgt festgelegt:
  - T ist im Startzustand q_0.
  - Auf jedem Band steht der Lese-/Schreibkopf auf Zelle 1.
  - Auf Band 1 steht die Eingabe w, wobei für jedes $j \in \{1, \ldots, |w|\}$ der j-te Buchstabe von w in Zelle j steht. In allen anderen Zellen steht das Leersymbol $\square$.
  - Auf jedem der Bänder $2, \ldots, k$ steht in jeder Zelle das Leersymbol $\square$.

- Die Maschine T hält, wenn sie in einem Zustand $q \in Q$ auf den k Bändern die Buchstaben $a_1, \ldots, a_k$ liest und $\delta(q, a_1, \ldots, a_k) = (q, a_1, \ldots, a_k, \downarrow, \ldots, \downarrow)$ gilt.

- Die Maschine T akzeptiert w, wenn sie in einem Zustand aus F hält.
- Die von T berechnete Funktion ist die partielle Funktion f_T von Σ^* nach Σ^*, so dass für alle $w \in \Sigma^*$ gilt:
  - $f_T(w)$ ist undefiniert, falls T bei Eingabe w nicht anhält.
  - $f_T(w) = u$, falls T bei Eingabe w nach endlich vielen Schritten anhält und dabei u dasjenige Wort maximaler Länge aus Σ^* ist, das beginnend mit der aktuellen Kopfposition auf dem k-ten Band steht.

Beispiel: Eine 2-Band-TM für die Sprache $L := \{a^n b^n : n \in \mathbb{N}_{>0}\}$

Wir bauen eine 2-Band-TM $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$, die stets anhält und genau diejenigen Eingabeworte akzeptiert, die in L liegen. Eingabealphabet: $\Sigma = \{a, b\}$. Arbeitsalphabet: $\Gamma = \{a, b, \square\}$. Unsere 2-Band-TM geht wie folgt vor:

- Solange du auf Band 1 a's liest, geh auf Band 1 und 2 von links nach rechts und kopiere jedes a auf Band 2.
- Solange du auf Band 1 dann b's liest, geh auf Band 1 weiter nach rechts, während du auf Band 2 rückwärts gehst.
- Akzeptiere, falls du auf beiden Bändern gleichzeitig das Symbol $\square$ liest. Verwerfe, falls du auf Band 1 nochmal ein a liest oder auf einem der beiden Bänder ein $\square$ liest, während auf dem anderen Band ein Symbol $\neq \square$ gelesen wird.

Wähle $Q = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$, $F = \{q_{\text{accept}}\}$, $\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{\leftarrow, \downarrow, \rightarrow\}^2$ mit

$$\begin{aligned}
\delta(q, \gamma_1, \gamma_2) &= (q, \gamma_1, \gamma_2, \downarrow, \downarrow) && \text{für alle } q \in \{q_{\text{accept}}, q_{\text{reject}}\} \text{ und } \gamma_1, \gamma_2 \in \Gamma \\
\delta(q_0, a, \square) &= (q_0, a, a, \rightarrow, \rightarrow) \\
\delta(q_0, b, \square) &= (q_1, b, \square, \downarrow, \leftarrow) \\
\delta(q_1, b, a) &= (q_1, b, a, \rightarrow, \leftarrow) \\
\delta(q_1, \square, \square) &= (q_{\text{accept}}, \square, \square, \downarrow, \downarrow) \\
\delta(q, \gamma_1, \gamma_2) &= (q_{\text{reject}}, \gamma_1, \gamma_2, \downarrow, \downarrow) && \text{für alle bisher noch nicht genannten } (q, \gamma_1, \gamma_2) \in Q \times \Gamma^2.
\end{aligned}$$

Man kann sich leicht davon überzeugen, dass diese 2-Band-TM bei jeder Eingabe $w \in \Sigma^*$ anhält und genau die Worte aus L akzeptiert.

Simulation von Mehrband-TM durch 1-Band-TM (1/4)

Satz:

Sei $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ eine k-Band-TM, für $k \geqslant 2$. Dann gibt es eine 1-Band-TM $T' = (\Sigma, Q', \Gamma', \delta', q_0', F')$, die dieselbe partielle Funktion berechnet und dieselben Worte $w \in \Sigma^*$ akzeptiert wie T.

Beweisidee:

- T' simuliert die k Bänder von T auf einem einzigen Band, das in k Spuren aufgeteilt ist. Skizze: siehe Tafel!
- Für jedes $i \in \{1, \ldots, k\}$ soll die i-te Spur den Inhalt des i-ten Bandes sowie die akutelle Kopfposition auf dem i-ten Band enthalten.

  Als Arbeitsalphabet wählen wir daher $\Gamma' := (\Gamma \times \{\star, -\})^k \cup \Sigma \cup \{\Box\}$.
  Das Lesen des Symbols

  $$\begin{bmatrix} (a_1, -) \\ (a_2, \star) \\ (a_3, \star) \\ \vdots \\ (a_k, -) \end{bmatrix}$$

  an der akutellen Kopfposition von T' bedeutet, dass T zum entsprechenden Zeitpunkt auf ihren k Bändern die Symbole $a_1, \ldots, a_k$ liest und die Köpfe auf Band 2 und Band 3 an der aktuellen Position stehen, während die Köpfe auf den Bändern 1 und k nicht an der akutellen Position stehen.

Bei Eingabe von $w = w_1 \cdots w_n \in \Sigma^*$ geht T' wie folgt vor:

- Initialisierung:

  Ersetze w durch die k-Spur-Darstellung des Bandinhalts von T bei Eingabe von w.

  D.h.: Geh 1x von links nach rechts über das Band und ersetze $w_1 w_2 \cdots w_n$ durch

$$\begin{bmatrix} (w_1, \star) \\ (\square, \star) \\ \vdots \\ (\square, \star) \end{bmatrix} \begin{bmatrix} (w_2, -) \\ (\square, -) \\ \vdots \\ (\square, -) \end{bmatrix} \begin{bmatrix} (w_3, -) \\ (\square, -) \\ \vdots \\ (\square, -) \end{bmatrix} \begin{matrix} \cdots \\ \cdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} (w_n, -) \\ (\square, -) \\ \vdots \\ (\square, -) \end{bmatrix}$$

  Geh dann mit dem Kopf wieder ganz nach links zum ersten Symbol, das kein $\square$ ist.

Simulation von Mehrband-TM durch 1-Band-TM (3/4)

- Simuliere jeden einzelnen Berechnungsschritt von T wie folgt:
  - Zu Beginn der Simulation steht der Kopf auf dem ersten nicht-leeren Symbol des Bandes, und T' ist im aktuellen Zustand q von T.
  - Lies das Band 1x von links nach rechts und speichere im aktuellen Zustand sowohl q als auch die k Symbole $a_1, \ldots, a_k$ in den k Spuren, die mit einem $\star$ versehen sind. Sei $\delta(q, a_1, \ldots, a_k) = (q', a'_1, \ldots, a'_k, b_1, \ldots, b_k)$.
  - Lies das Band 1x von rechts nach links und aktualisiere dabei die mit einem $\star$ versehenen Symbole in den k Spuren durch die Symbole $a'_1, \ldots, a'_k$. Für jedes $i \in \{1, \ldots, k\}$ mit $b_i = \leftarrow$ ersetze $\star$ in der i-ten Spur durch $-$ und ersetze das links daneben stehende Symbol $(\gamma, -)$ der i-ten Spur durch $(\gamma, \star)$.
  - Lies das Band 1x von links nach rechts. Für jedes $i \in \{1, \ldots, k\}$ mit $b_i = \rightarrow$ ersetze $\star$ in der i-ten Spur durch $-$ und ersetze das rechts daneben stehende Symbol $(\gamma, -)$ der i-ten Spur durch $(\gamma, \star)$.
  - Geh dann mit dem Kopf wieder ganz nach links zum ersten nicht-leeren Symbol des Bandes und geh in den Zustand q'.

Simulation von Mehrband-TM durch 1-Band-TM (4/4)

- Erzeugen der korrekten Ausgabe:

  Wenn T anhält, so ersetze den aktuellen Bandinhalt durch den Inhalt der k-ten Spur, setze den Kopf auf die mit $\star$ markierte Position der k-ten Spur und geh in den Zustand, in dem T anhält.

- Insgesamt erhalten wir so eine 1-Band-TM T' die dieselben Worte akzeptiert wie T und die dieselbe partielle Funktion berechnet wie T.

□

Frage: Ist t die Anzahl der Schritte, die T bei Eingabe von w macht, so macht T' bei Eingabe von w wie viele Schritte?

Antwort: $O\big(t \cdot (|w| + t)\big)$.

Die Größe des Arbeitsalphabets

Satz: Sei Σ ein Alphabet mit $0, 1 \in \Sigma$.

Für jede 1-Band-TM $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ gibt es eine 1-Band-TM T' mit Eingabealphabet Σ und Arbeitsalphabet $\Sigma \cup \{\square\}$, die dieselbe partielle Funktion berechnet und dieselben Worte $w \in \Sigma^*$ akzeptiert wie T.

Beweisidee:

- Repräsentiere jedes $\gamma \in \Gamma$ durch einen Bitstring $\mathit{rep}(\gamma)$ der Länge $\ell := \lceil \log_2 |\Gamma| \rceil$. Bei Eingabe von $w = w_1 \cdots w_n \in \Sigma^*$ arbeitet unsere TM T' wie folgt:
- Phase 1: Initialisierung: Ersetze $w = w_1 \cdots w_n$ durch das Wort $\mathit{rep}(w_1) \cdots \mathit{rep}(w_n)$.
- Phase 2: Simulation von T: Jeder einzelne Schritt von T wird simuliert durch
  (1) "Lesen": ℓ Schritte nach rechts zum Lesen von $\mathit{rep}(\gamma)$, um das aktuell von T gelesene Symbol γ zu ermitteln. Sei $\delta(q, \gamma) = (q', \gamma', b)$.
  (2) "Schreiben": ℓ Schritte nach links zum Schreiben von $\mathit{rep}(\gamma')$.
  (3) "Kopf positionieren:" Falls $b = \leftarrow$ (bzw. $\rightarrow$), so mache k Schritte nach links (bzw. rechts); falls $b = \downarrow$, so tue nichts weiter. Geh dann in den Zustand q'.
- Phase 3: Erzeugen der korrekten Ausgabe: Wenn T anhält, so ermittle das Ausgabewort $u \in \Sigma^*$ von T (das steht als $\mathit{rep}(u_1) \cdots \mathit{rep}(u_\ell)$ auf dem Band), schreib $u\square$ aufs Band, geh mit dem Kopf an die erste Position von u und halte in dem Zustand an, in dem T anhält.

Repräsentation von Turingmaschinen durch Bitstrings (sog. Gödelnummern)

und

eine universelle Turingmaschine $\mathcal{U}$

Kodierung einer TM T durch einen Bitstring $\langle T \rangle$ (1/2)

Im Folgenden betrachten wir nur noch 1-Band-TM mit

- Eingabealphabet $\Sigma = \{0, 1\}$,
- Arbeitsalphabet $\Gamma = \{0, 1, \square\}$,
- Zustandsmenge $Q = \{1, 2 \ldots, r\}$ mit $r \geqslant 2$, wobei
  1 der Startzustand und 2 der einzige akzeptierende Zustand ist.

Frage: Warum ist dies keine wesentliche Einschränkung?
Antwort: Wegen dem Satz auf der vorherigen Folie. Und weil ein einziger akzeptierender Zustand ausreicht (Warum?).

Frage:

Wie können wir die Maschine T durch einen Bitstring $\langle T \rangle$ kodieren, so dass das Dekodieren einfach ist?

Kodierung einer TM T durch einen Bitstring $\langle T \rangle$ (2/2)

- Wir setzen $a_1 := 0$, $a_2 := 1$, $a_3 := \square$ und $b_1 := \leftarrow$, $b_2 := \downarrow$, $b_3 := \rightarrow$.
- Jede Programmzeile z der Form $\delta(q, a_i) = (p, a_j, b_k)$ (mit $q, p \in Q = \{1, 2, \ldots, r\}$ und $i, j, k \in \{1, 2, 3\}$) repräsentieren wir durch den Bitstring

$$\langle z \rangle \quad := \quad 0^q 1 0^i 1 0^p 1 0^j 1 0^k.$$

- Sind $z_1, z_2, \ldots, z_m$ sämtliche Zeilen des Programms δ, so repräsentieren wir δ durch den Bitstring $\langle \delta \rangle := \langle z_1 \rangle\ 11\ \langle z_2 \rangle\ 11\ \cdots\ 11\ \langle z_m \rangle$.
- Die TM T repräsentieren wir durch den Bitstring $\langle T \rangle := 111\ \langle \delta \rangle\ 111$.

Beachte: Wir haben die Kodierung $\langle T \rangle$ so gewählt, dass gilt:

- Aus dem Bitstring $\langle T \rangle$ lässt sich ganz T rekonstruieren.
- Gegeben ein Bitstring w, ist es leicht, zu überprüfen, ob es eine TM T mit $w = \langle T \rangle$ gibt. (**Übung:** Es gibt sogar eine TM, die diese Überprüfung durchführen kann.)
- Nur ganz am Anfang und ganz am Ende von $\langle T \rangle$ stehen drei aufeinanderfolgende Einsen.
- Je nach Wahl der Reihenfolge der Programmzeilen gibt es verschiedene Bitstrings, die dieselbe TM kodieren. Um die Kodierung eindeutig zu machen, legen wir eine Reihenfolge fest (z.B. lexikographisch nach (q, i) bei $\delta(q, a_i)$).

Die meisten Sprachen sind unentscheidbar!

Der Bitstring $\langle T \rangle$, der eine TM T kodiert, wird auch Gödelnummer von T genannt.

Beobachtung:

- Jede TM T wird durch ihre Gödelnummer $\langle T \rangle \in \{0,1\}^*$ eindeutig beschrieben.
- Es gibt nur äbzählbar viele Bitstrings $w \in \{0,1\}^*$ (da für jedes endliche Alphabet A die Menge A^* nur abzählbar groß ist: Die Elemente können gemäß der lexikographischen Reihenfolge mit Zahlen $1, 2, \ldots$ durchnummeriert werden).
- Also gibt es auch nur abzählbar viele Turingmaschinen.
- Aber es gibt überabzählbar viele Sprachen $L \subseteq \{0,1\}^*$, (denn: $\{L : L \subseteq \{0,1\}^*\} = \mathcal{P}(\{0,1\}^*)$, und es gibt keine surjektive Funktion von einer Menge M auf ihre Potenzmenge $\mathcal{P}(M)$).

Somit gibt es viel mehr Sprachen $L \subseteq \{0,1\}^*$ als Turingmaschinen T. Daher sind die allermeisten Sprachen weder entscheidbar noch semi-entscheidbar!

Universelle Turingmaschine $\mathcal{U}$ (1/2)

- Die Gödelnummer $\langle T \rangle$ einer TM T ist ein Wort aus $\{0,1\}^*$ und kann daher als Eingabe für andere Turingmaschinen benutzt werden.

- In der Gödelnummer $\langle T \rangle$ taucht das Wort 111 nur ganz am Anfang und ganz am Ende auf.
  - Die Gödelnummer ist „selbstbegrenzend“.
  - Bei gegebenem Bitstring $\langle T \rangle w$ kann das Suffix w leicht rekonstruiert werden: Das zweite Vorkommen von 111 hat die Funktion eines „Kommas“.

- Wir können eine universelle Turingmaschine $\mathcal{U}$ bauen, die die folgenden Eigenschaften hat:
  - Eingabealphabet $\Sigma = \{0,1\}$
    Arbeitsalphabet $\Gamma = \{0,1,\square\}$
    Zustandsmenge $Q = \{1,2,\ldots,r\}$ für ein geeignetes $r \geqslant 2$, und Startzustand 1 und einzigem akzeptierenden Zustand 2.
  - Bei Eingabe von $\langle T \rangle w$, wobei T eine TM ist und $w \in \{0,1\}^*$, tut $\mathcal{U}$ folgendes:
    - $\mathcal{U}$ simuliert T bei Eingabe w und
    - akzeptiert genau dann, wenn T die Eingabe w akzeptiert und
    - hält genau dann mit Ausgabe $u \in \{0,1\}^*$, wenn T bei Eingabe w mit Ausgabe u anhält.

Universelle Turingmaschine $\mathcal{U}$ (2/2)

Konstruktion von $\mathcal{U}$:

- Bei Eingabe $\langle T \rangle w$ soll $\mathcal{U}$ die Berechnung der TM T bei Eingabe w simulieren.
- Es genügt, eine 3-Band-TM $\mathcal{U}$ zu konstruieren. Wir wissen bereits, dass diese durch eine 1-Band-TM simuliert werden kann.
- $\mathcal{U}$ benutzt
  - Band 1, um die Kodierung $\langle T \rangle$ von T zu speichern.
  - Band 2, um den aktuellen Bandinhalt von T zu speichern.
  - Band 3, um den aktuellen Zustand von T zu speichern (kodiert als Liste von Nullen).
- Details: Übung.

Unentscheidbare Probleme

und

Reduktionen

Die Diagonalsprache

Als erstes betrachten wir die so genannte Diagonalsprache $D \subseteq \{0,1\}^*$:

$$D := \{ \langle T \rangle : T \text{ ist eine TM, die das Eingabewort } \langle T \rangle \underline{\textbf{nicht}} \text{ akzeptiert} \}$$

Beachte:

- D besteht nur aus Gödelnummern von Turingmaschinen T.
- $\langle T \rangle$ gehört genau dann zu D, wenn die Turingmaschine T die Eingabe $\langle T \rangle$ nicht akzeptiert:

  Eine Turingmaschine akzeptiert nicht, wenn verworfen wird oder die Maschine **nicht hält**.

Genau so wie wir am Anfang dieses Kapitels die Unentscheidbarkeit des Halteproblems gezeigt haben, können wir nun zeigen, dass die Sprache D unentscheidbar ist:

Die Diagonalsprache D ist nicht TM-entscheidbar

$D := \{ \langle T \rangle : T$ ist eine TM, die das Eingabewort $\langle T \rangle$ **nicht** akzeptiert $\}$

Behauptung: $D \subseteq \{0,1\}^*$ ist nicht entscheidbar.

Beweis:

- Angenommen, D ist doch TM-entscheidbar. Dann gibt es eine stets haltende Turingmaschine T_D, die die Sprache D entscheidet.
  D.h für alle Turingmaschinen T gilt:

$$\begin{aligned} T_D \text{ akzeptiert } \langle T \rangle &\iff \langle T \rangle \in D \\ &\iff T \text{ akzeptiert } \langle T \rangle \text{ nicht.} \end{aligned}$$

- Und was macht T_D mit seiner eigenen Gödelnummer?

$$T_D \text{ akzeptiert } \langle T_D \rangle \iff T_D \text{ akzeptiert } \langle T_D \rangle \text{ nicht.}$$

Dies ist ein Widerspruch. Somit kann es die TM T_D, die $D \subseteq \{0,1\}^*$ entscheidet, nicht geben. □

Die Diagonalisierungsmethode

- Bilde eine unendliche Tabelle A,
  - die für jede Turingmaschine T eine Zeile und
  - für jede Gödelnummer $\langle T' \rangle$ eine Spalte besitzt.

  Der Eintrag in Zeile T und Spalte $\langle T' \rangle$ ist der Wert

$$A(T, \langle T' \rangle) = \begin{cases} 1 & T \text{ akzeptiert } \langle T' \rangle, \\ 0 & \text{sonst.} \end{cases}$$

- Die Diagonalsprache D „flippt“ die Diagonale von A und erzwingt damit, dass D von keiner Turingmaschine T entschieden werden kann:
  - T wird auf Eingabe $\langle T \rangle$ die Anwort $A(T, \langle T \rangle)$ geben,
  - während D die geflippte Antwort verlangt.

Das Komplement der Diagonalsprache D ist nicht entscheidbar

Wir wissen bereits, dass folgendes gilt:

Wenn $L \subseteq \{0,1\}^*$ unentscheidbar ist, dann ist auch das Komplement $\overline{L} := \{0,1\}^* \setminus L$ unentscheidbar.

Beweis:

- Angenommen, $\overline{L} \subseteq \{0,1\}^*$ ist entscheidbar.
- Dann gibt es eine stets haltende Turingmaschine T, die die Sprache $\overline{L} \subseteq \{0,1\}^*$ entscheidet.
- Mache verwerfende Zustände akzeptierend und umgekehrt.

  Die dadurch entstehende TM T' entscheidet $L \subseteq \{0,1\}^*$.

Folgerung:

Das Komplement $\overline{D} := \{0,1\}^* \setminus D$ der Diagonalsprache D ist unentscheidbar.

Reduktionen

Wir erhalten weitere Unenstscheidbarkeitsergebnisse mit Hilfe von Reduktionen:

Definition: Seien Σ_1 und Σ_2 endliche Alphabete.

Eine Sprache $L_1 \subseteq \Sigma_1^*$ ist auf eine Sprache $L_2 \subseteq \Sigma_2^*$ reduzierbar (kurz: $L_1 \leqslant L_2$), wenn es eine berechenbare totale Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \iff f(w) \in L_2.$$

Unentscheidbarkeit "vererbt" sich mittels Reduktionen:

Satz: Seien $L \subseteq \Sigma_1^*$ und $K \subseteq \Sigma_2^*$.

Ist $L \subseteq \Sigma_1^*$ unentscheidbar und ist $L \leqslant K$, so ist auch $K \subseteq \Sigma_2^*$ unentscheidbar.

Beweis: Angenommen, K ist entscheidbar durch einen Algorithmus (oder eine TM) A_K. Dann können wir auch L entscheiden, indem wir bei Eingabe $w \in \Sigma_1^*$ zunächst das Wort $f(w) \in \Sigma_2^*$ berechnen (das geht, da f berechenbar ist) und dann Algorithmus A_K mit Eingabe $f(w)$ starten. Dann gilt:

$$\begin{array}{lcll} A_K \text{ gibt "ja" aus} & \iff & f(w) \in K & \iff & w \in L \\ A_K \text{ gibt "nein" aus} & \iff & f(w) \notin K & \iff & w \notin L. \end{array}$$

Somit ist auch L entscheidbar. □

Ist die Unentscheidbarkeit von D interessant?

- Die Diagonalsprache

  $$D := \{ \langle T \rangle : T \text{ ist eine TM, die das Eingabewort } \langle T \rangle \ \underline{\textbf{nicht}} \ \text{akzeptiert} \}$$

  scheint „künstlich“ zu sein: Sie „redet“ über Turingmaschinen, die ihre eigene Gödelnummer (als Eingabe) nicht akzeptieren.

- Die Komplementsprache $\overline{D} := \{0,1\}^* \setminus D$ redet dann natürlich über Turingmaschinen, die ihre Gödelnummer als Eingabe akzeptieren.
  - Na, und? Immer noch nicht interessant!
  - Aber die Frage,

    ob eine Turingmaschine ihre Gödelnummer als Eingabe akzeptiert,

    ist schon unentscheidbar und leichter als die Frage,

    ob eine Turingmaschine eine beliebige Eingabe akzeptiert.

- Die Frage nach der Akzeptanz einer beliebigen Eingabe ist also unentscheidbar?

Die universelle Sprache ist unentscheidbar

Die universelle Sprache $U \subseteq \{0,1\}^*$ ist definiert durch

$U := \{ \langle T \rangle w : T \text{ ist eine TM, die das Eingabewort } w \in \{0,1\}^* \text{ akzeptiert } \}.$

Behauptung: $U \subseteq \{0,1\}^*$ ist nicht entscheidbar.

Beweis:

- Wir zeigen, dass $\overline{D} \leqslant U$ ist, d.h. wir konstruieren eine Reduktion von $\overline{D}$ auf U.
- Aus der Unentscheidbarkeit von $\overline{D}$ folgt dann, dass auch U unentscheidbar ist.
- Die Reduktion von $\overline{D}$ auf U wird durch eine Abbildung $f : \{0,1\}^* \to \{0,1\}^*$ realisiert, bei der für jedes $w \in \{0,1\}^*$ das Wort $f(w)$ wie folgt definiert ist:
  - Falls es eine TM T gibt, so dass $w = \langle T \rangle$, so ist $f(w) := \langle T \rangle \langle T \rangle$.

    Dann gilt: $w \in \overline{D} \iff \langle T \rangle \in \overline{D} \iff T \text{ akzeptiert } \langle T \rangle$
    $\iff \langle T \rangle \langle T \rangle \in U \iff f(w) \in U.$
  - Sonst setze $f(w) := \langle T_0 \rangle \varepsilon$, wobei T_0 eine TM ist, die bei Eingabe ε sofort anhält und akzeptiert. Dann gilt: $w \in \overline{D}$ und $f(w) \in U$.
- Man sieht leicht, dass die Funktion f berechenbar ist (denn: Bei gegebenem Bitstring w kann man leicht überprüfen, ob es eine TM T mit $w = \langle T \rangle$ gibt. Und die TM T_0 können wir leicht konstruieren.). Somit ist $\overline{D} \leqslant U$ mittels f. □

Das Halteproblem für Turingmaschinen

Das Halteproblem für Turingmaschinen ist definiert als die Sprache

$H := \{ \langle T \rangle w : T \text{ ist eine TM, die bei Eingabe } w \in \{0,1\}^* \text{ hält } \}.$

Behauptung: $H \subseteq \{0,1\}^*$ ist nicht entscheidbar.

Beweis:

- Wir zeigen, dass $U \leqslant H$ ist, d.h. wir konstruieren eine Reduktion von U auf H.
- Aus der Unentscheidbarkeit von U folgt dann, dass auch H unentscheidbar ist.
- Die Reduktion von U auf H wird durch die Abbildung $f : \{0,1\}^* \to \{0,1\}^*$ realisiert, bei der für alle $u \in \{0,1\}^*$ das Wort $f(u)$ wie folgt definiert ist:
  - Falls es eine TM T und ein Wort $w \in \{0,1\}^*$ gibt, so dass $u = \langle T \rangle w$, so ist $f(u) := \langle T' \rangle w$, wobei T' die TM ist, die T simuliert und immer dann, wenn T in einem nicht-akzeptierenden Zustand hält, in eine Endlosschleife geht.

    Dann gilt: $u \in U \iff \langle T \rangle w \in U \iff T \text{ akzeptiert } w$
    $\iff T' \text{ hält bei Eingabe } w \iff \langle T' \rangle w \in H \iff f(u) \in H.$
  - Sonst setze $f(u) := \langle T_0 \rangle \varepsilon$, wobei T_0 eine TM ist, die bei Eingabe ε sofort in eine Endlosschleife geht. Dann gilt: $u \notin U$ und $f(u) \notin H$.
- Man sieht leicht, dass die Funktion f berechenbar ist. Somit ist $U \leqslant H$ mittels f.

□

Das spezielle Halteproblem

Das spezielle Halteproblem H_ε für Turingmaschinen ist definiert als die Sprache

$H_\varepsilon := \{ \langle T \rangle : T \text{ ist eine TM, die bei Eingabe des leeren Worts } \varepsilon \text{ hält } \}$.

Behauptung: $H_\varepsilon \subseteq \{0,1\}^*$ ist nicht entscheidbar.

Beweis:

- Wir zeigen, dass $H \leqslant H_\varepsilon$ ist, d.h. wir konstruieren eine Reduktion von H auf H_ε.
- Aus der Unentscheidbarkeit von H folgt dann, dass auch H_ε unentscheidbar ist.
- Die Reduktion von H auf H_ε wird durch die Abbildung $f : \{0,1\}^* \to \{0,1\}^*$ realisiert, bei der für alle $u \in \{0,1\}^*$ das Wort $f(u)$ wie folgt definiert ist:
  - Falls es eine TM T und ein Wort w gibt, so dass $u = \langle T \rangle w$, so ist $f(u) := \langle T' \rangle$, wobei T' die TM ist, die bei leerer Eingabe zunächst das Wort w aufs Band schreibt und dann T simuliert. Dann gilt:

    $$u \in H \iff \langle T \rangle w \in H \iff T \text{ hält bei Eingabe } w$$
    $$\iff T' \text{ hält bei Eingabe } \varepsilon \iff \langle T' \rangle \in H_\varepsilon \iff f(u) \in H_\varepsilon .$$

  - Sonst setze $f(u) := \langle T_0 \rangle$, wobei T_0 eine TM ist, die bei Eingabe ε sofort in eine Endlosschleife geht. Dann gilt: $u \notin H$ und $f(u) \notin H_\varepsilon$.
- Man sieht leicht, dass die Funktion f berechenbar ist. Somit ist $H \leqslant H_\varepsilon$ mittels f.

□

Zusammenfassung

Die folgenden Sprachen über dem Alphabet $\{0,1\}$ sind nicht entscheidbar:

- Diagonalsprache
  $D = \{\langle T \rangle : T \text{ ist eine TM, die das Eingabewort } \langle T \rangle \text{ nicht akzeptiert}\}$
- Universelle Sprache
  $U = \{\langle T \rangle w : T \text{ ist eine TM, die das Eingabewort } w \in \{0,1\}^* \text{ akzeptiert}\}$
- Halteproblem
  $H = \{\langle T \rangle w : T \text{ ist eine TM, die bei Eingabe } w \in \{0,1\}^* \text{ hält}\}$
- spezielles Halteproblem
  $H_\varepsilon = \{\langle T \rangle : T \text{ ist eine TM, die bei Eingabe des leeren Worts } \varepsilon \text{ hält}\}$

Die folgenden Sprachen sind semi-entscheidbar, aber nicht entscheidbar:

- $H_\varepsilon, \quad H, \quad U, \quad \overline{D} = \{0,1\}^* \setminus D$

Die folgenden Sprachen sind nicht semi-entscheidbar:

- $\overline{H_\varepsilon}, \quad \overline{H}, \quad \overline{U}, \quad D.$

Frage: Gibt es auch unentscheidbare Probleme, die nichts mit Turingmaschinen oder Eigenschaften von Programmen zu tun haben?

Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PKP)

Eingabe: Ein endliches Alphabet Σ, eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

Frage: Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$?

Beispiel:

Das PKP mit Eingabe $\Sigma = \{0, 1\}$, $k = 3$ und

$$(x_1, y_1) = (1, 111), \qquad (x_2, y_2) = (10111, 10), \qquad (x_3, y_3) = (10, 0).$$

hat eine Lösung mit $n = 4$ und $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, denn:

$$\begin{aligned} x_2 x_1 x_1 x_3 &= 10111\ \ 1\ \ 1\ \ 10 \\ y_2 y_1 y_1 y_3 &= 10\ \ 111\ \ 111\ \ 0. \end{aligned}$$

Beachte: Das PKP ist semi-entscheidbar.

Ziel: Wir wollen zeigen, dass das PKP unentscheidbar ist.

Varianten des PKP

- Fester Startindex $i_1 = 1$:

Das Modifizierte PKP (MPKP)

Eingabe: Ein endliches Alphabet Σ, eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

Frage: Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $i_1 = 1$ und $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$?

Beispiel: Betrachtet als Eingabe für's MPKP hat das auf der vorherigen Folie gegebene Beispiel keine Lösung.

- Festes Alphabet Σ:

Das PKP über Alphabet Σ (PKP_Σ)

Eingabe: Eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

Frage: Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$?

Unentscheidbarkeit des PKP

Satz: $H_\varepsilon \leqslant \text{MPKP} \leqslant \text{PKP} \leqslant \text{PKP}_{\{0,1\}}$.

Insbesondere sind das Postsche Korrespondenzproblem PKP sowie seine Varianten MPKP und $\text{PKP}_{\{0,1\}}$ nicht entscheidbar.

Beweis:

$\text{PKP} \leqslant \text{PKP}_{\{0,1\}}$:

- Für ein endliches Alphabet $\Sigma = \{a_1, \ldots, a_m\}$ sei $h_\Sigma : \Sigma^* \to \{0,1\}^*$ der Homomorphismus mit $h_\Sigma(a_j) := 0\,1^j$ für alle $j \in \{1, \ldots, m\}$.
- Die Reduktion von PKP auf $\text{PKP}_{\{0,1\}}$ wird durch die Abbildung f realisiert, die einer Eingabe
$$\Sigma,\ k,\ (x_1, y_1), \ldots, (x_k, y_k)$$
für's PKP die folgende Eingabe für's $\text{PKP}_{\{0,1\}}$ zuordnet:
$$k,\ \big(h_\Sigma(x_1),\ h_\Sigma(y_1)\big),\ \ldots,\ \big(h_\Sigma(x_k),\ h_\Sigma(y_k)\big).$$
- Für alle $n \in \mathbb{N}_{>0}$ und alle $i_1, \ldots, i_n \in \{1, \ldots, k\}$ gilt:
$$x_{i_1} x_{i_2} \cdots x_{i_n} \;=\; y_{i_1} y_{i_2} \cdots y_{i_n} \quad\Longleftrightarrow\quad h_\Sigma(x_{i_1}) h_\Sigma(x_{i_2}) \cdots h_\Sigma(x_{i_n}) \;=\; h_\Sigma(y_{i_1}) h_\Sigma(y_{i_2}) \cdots h_\Sigma(y_{i_n}).$$
- f ist berechenbar. Somit ist f eine Reduktion von PKP auf $\text{PKP}_{\{0,1\}}$.

$\text{MPKP} \leqslant \text{PKP}$: Übung! (siehe z.B. die Bücher von Wegener oder Schöning)

Unentscheidbarkeit des PKP: $H_\varepsilon \leqslant$ MPKP (1/2)

Gesucht: Eine berechenbare Funktion f, die jedem Wort $w \in \{0,1\}^*$ eine Eingabe $f(w)$ für's MPKP zuordnet, so dass gilt:

$w \in H_\varepsilon$, d.h. $w = \langle T \rangle$ für eine TM T, die bei leerer Eingabe ε hält $\iff$ das MPKP $f(w)$ besitzt eine Lösung

Leichter Fall: w ist keine Repräsentation einer TM.

Dann sei $f(w)$ eine Eingabe für's MPKP, die keine Lösung besitzt, z.B., $\Sigma = \{0,1\}$, $k = 1$, $x_1 = 0$, $y_1 = 1$.

Schwieriger Fall: $w = \langle T \rangle$ für eine TM T. **Idee:**

- Repräsentiere Konfigurationen einer TM $T = (\Sigma, Q, \Gamma, \delta, q_0, F)$ durch Worte über dem Alphabet $\Gamma \dot\cup Q$ wie folgt:

  uqv repräsentiert die Situation, bei der die TM in Zustand q ist, die Bandinschrift uv ist, und der Kopf auf dem ersten Symbol von v steht.
- Startkonfiguration bei Eingabe des leeren Worts ε: $q_0\square$
- O.B.d.A. betrachten wir nur Turingmaschinen, die nur dann in einen Zustand aus F gehen, wenn sie unmittelbar danach anhalten.
- Konstruiere eine Eingabe $f(\langle T \rangle)$ für's MPKP, die aufeinander folgende Konfigurationen von T erzeugt.
  Alphabet: $\Gamma \dot\cup Q \dot\cup \{\#\}$. Das Symbol $\#$ dient als Trennsymbol zwischen einzelnen Konfigurationen.

Unentscheidbarkeit des PKP: $H_\varepsilon \leqslant$ MPKP (2/2)

Für eine gegebene TM T werden die Wortpaare $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ wie folgt gewählt (für ein geeignetes $k \in \mathbb{N}_{>0}$):

- Starttupel: (x_1, y_1) mit $x_1 := \#$ und $y_1 := \#q_0\square\#$.
- Überführungsregeln:
  - $(qa, q'a')$, falls $\delta(q, a) = (q', a', \downarrow)$
  - $(qa, a'q')$, falls $\delta(q, a) = (q', a', \rightarrow)$
  - $(bqa, q'ba')$, falls $\delta(q, a) = (q', a', \leftarrow)$, für $b \in \Gamma$
  - $(\#qa, \#q'\square a')$, falls $\delta(q, a) = (q', a', \leftarrow)$
  - $(q\#, q'a'\#)$, falls $\delta(q, \square) = (q', a', \downarrow)$
  - $(q\#, a'q'\#)$, falls $\delta(q, \square) = (q', a', \rightarrow)$
  - $(bq\#, q'ba'\#)$, falls $\delta(q, \square) = (q', a', \leftarrow)$, für $b \in \Gamma$
- Kopierregeln: (a, a) mit $a \in \Gamma \cup \{\#\}$.
- Löschregeln: (aq, q) sowie (qa, q) mit $q \in F$ und $a \in \Gamma$
- Abschlussregeln: $(q\#\#, \#)$ mit $q \in F$.

Behauptung:
T hält bei Eingabe $\varepsilon \iff f(\langle T \rangle)$ besitzt eine Lösung mit Starttupel (x_1, y_1).

Begründung: siehe Tafel. □

Unentscheidbare Grammatik-Probleme

Zwei Kontextfreie Grammatiken

Durch Reduktionen vom Postschen Korrespondenzproblem PKP können wir zeigen, dass viele Fragestellungen zu kontextfreien Grammatiken nicht entscheidbar sind:

Satz:

Folgende Probleme, bei denen die Eingabe aus zwei kontextfreien Grammatiken G und G' besteht, sind unentscheidbar:

(a) Ist $L(G) \cap L(G') = \emptyset$? ..... (Leerer Schnitt)

(b) Ist $|L(G) \cap L(G')| = \infty$? ..... (Unendlicher Schnitt)

(c) Ist $L(G) \cap L(G')$ kontextfrei? ..... (Kontextfreier Schnitt)

(d) Ist $L(G) \subseteq L(G')$? ..... (Subsumption)

(e) Ist $L(G) = L(G')$? ..... (Äquivalenz)

Beweis:

(a)&(b): siehe Tafel

(c)–(e): Übung (siehe auch die Bücher von Wegener und Schöning)

Zwei Deterministisch Kontextfreie Sprachen

Auf ähnliche Art erhalten wir auch einen Beweis für folgenden Satz:

Satz:

Folgende Probleme, bei denen die Eingabe aus zwei deterministischen Kellerautomaten K und K' besteht, sind unentscheidbar:

(a) Ist $L(K) \cap L(K') = \emptyset$? ..... (Leerer Schnitt)

(b) Ist $|L(K) \cap L(K')| = \infty$? ..... (Unendlicher Schnitt)

(c) Ist $L(K) \cap L(K')$ kontextfrei? ..... (Kontextfreier Schnitt)

(d) Ist $L(K) \subseteq L(K')$? ..... (Subsumption)

Beweis: Ähnlich wie der Beweis des entsprechenden Resultate für KFGs:
Z.B. beim Beweis der Unentscheidbarkeit des "Leerer Schnitt"-Problems für KFGs hatten wir zwei Grammatiken G und G' konstruiert, deren Sprachen $L(G)$ und $L(G')$ sogar deterministisch kontextfrei sind.

Satz von Sénizergues (1997): (hier ohne Beweis)

Das Äquivalenzproblem für deterministische Kellerautomaten ist entscheidbar.

D.h. es gibt einen Algorithmus, der bei Eingabe zweier DPDAs K und K' entscheidet, ob $L(K) = L(K')$ ist.

Eine Kontextfreie Grammatik

Satz:

Folgende Probleme, bei denen die Eingabe aus einer kontextfreien Grammatik G besteht, sind nicht entscheidbar:

(a) Ist G mehrdeutig?

(b) Ist $\overline{L(G)}$ kontextfrei?

(c) Ist $L(G)$ regulär?

(d) Ist $L(G)$ deterministisch kontextfrei?

Beweis: (a): Wir reduzieren das (unentscheidbare) "Leerer Schnitt"-Problem für DPDAs auf das Mehrdeutigkeitsproblem für KFGs:

- Seien K_1 und K_2 zwei DPDAs.
- Konstruiere mit Hilfe der Tripelkonstruktion zwei eindeutige KFGs $G_1 = (\Sigma, V_1, S_1, P_1)$ und $G_2 = (\Sigma, V_2, S_2, P_2)$ mit $L(G_1) = L(K_1)$ und $L(G_2) = L(K_2)$. O.B.d.A. sind die Variablenmengen von G_1 und G_2 disjunkt.
- Sei $G = (\Sigma, V, S, P)$ mit $V := \{S\} \dot\cup V_1 \dot\cup V_2$ und $P := P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$.
- Dann ist $L(G) = L(G_1) \cup L(G_2)$. Und G ist genau dann mehrdeutig, wenn $L(G_1) \cap L(G_2) = \emptyset$.

(b)–(d): Übung (siehe auch die Bücher von Wegener und Schöning) □

Zusammenfassung

Zusammenfassung (1/2)

- Wir haben folgende Begriffe kennengelernt:
  - entscheidbare Probleme
  - semi-entscheidbare Probleme
    (äquivalent zu: rekursiv aufzählbare Probleme)
  - berechenbare partielle Funktionen
- Wenn die Church-Turing-These korrekt ist, sind diese Begriffe unabhängig von der Wahl des konkreten Berechnungsmodells.
- Als konkrete Berechnungsmodelle haben wir Turingmaschinen und Mehrband-Turingmaschinen kennengelernt.
- Aus dem Satz von Rice folgt, dass fast alle semantischen Eigenschaften von Programmen oder Turingmaschinen unentscheidbar sind.
- Mittels Diagonalisierung haben wir gezeigt, dass die Diagonalsprache

$$D = \{\langle T \rangle : T \text{ ist eine TM, die die Eingabe } \langle T \rangle \underline{\text{nicht}} \text{ akzeptiert}\} \subseteq \{0,1\}^*$$

  nicht entscheidbar ist.

Zusammenfassung (2/2)

- Reduktionen lieferten die Unenscheidbarkeit vieler weiterer Sprachen (**Merke:** Wenn L unentscheidbar und $L \leqslant K$, dann ist auch K unentscheidbar):
  - die universelle Sprache
    $U := \{\langle T \rangle w : T \text{ ist eine TM, die das Eingabewort } w \text{ akzeptiert}\} \subseteq \{0,1\}^*$
  - das Halteproblem
    $H := \{\langle T \rangle w : T \text{ ist eine TM, die bei Eingabe } w \text{ hält}\} \subseteq \{0,1\}^*$
  - das spezielle Halteproblem
    $H_\varepsilon := \{\langle T \rangle : T \text{ ist eine TM, die bei Eingabe des leeren Worts hält}\} \subseteq \{0,1\}^*$
  - das Postsche Korrespondenzproblem PKP
    und seine Varianten MPKP und $\text{PKP}_{\{0,1\}}$
  - viele Probleme, die kontextfreie Grammatiken oder deterministische Kellerautomaten betreffen,
    z.B. das “Leerer Schnitt”-Problem.