Perlen der Informatik 1

Erik Kynast, Florian Bodlée, Jan Schuchardt, Tobias Holl,
Luca Sinn, Benedikt Seidl, Felix Opolka, Sabine Rieder
David Schneller, Sarah Tilscher, Tobias Nipkow

29. Januar 2016

Kapitel 1

Mathematische Grundlagen

Sind A und B Mengen, so ist $A \to B$ die Menge der Funktionen von A nach B. Eine Funktion $f \in A \to B$ ist eine Menge von Paaren (a, b) mit $a \in A, b \in B$, so dass $(a, b) \in f \wedge (a, b') \in f \Rightarrow b = b'$. Statt $(a, b) \in f$ schreiben wir $f(a) = b$.

Definition 1.1 *Eine Funktion $f \in A \to B$ ist*

- total *gdw.* $\forall a \in A. \exists b \in B.\ f(a) = b$.
- (echt) partiell *gdw. f nicht total.*
- injektiv *gdw.* $\forall a, a'.\ f(a) = f(a') \Rightarrow a = a'$.
- surjektiv *gdw.* $\forall b \in B. \exists a \in A.\ f(a) = b$.
- bijektiv *gdw. f total, injektiv und surjektiv ist.*

$dom(f) = \{a \in A \mid \exists b \in B.\ f(a) = b\}$

Konvention: $f(a) = \perp$ gdw. $a \notin dom(f)$

Definition 1.2 *M ist* abzählbar *gdw. es eine totale, injektive Funktion von M nach $\mathbb{N}$ gibt.*

Fakt 1.1 *Jede endliche Menge ist abzählbar.*

Fakt 1.2 *Jede Teilmenge einer abzählbaren Menge ist abzählbar.*

Definition 1.3 *Eine Menge M ist* abzählbar unendlich *gdw. es eine Bijektion zwischen M und $\mathbb{N}$ gibt.*

Damit ist $\mathbb{N}$ trivialerweise abzählbar unendlich.

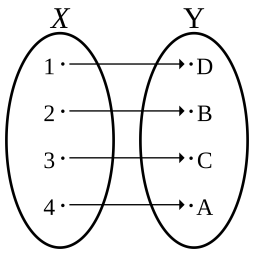


Abbildung 1.1: Beispiel einer Bijektion

Fakt 1.3 *Die geraden Zahlen sind abzählbar unendlich.*

Lemma 1.1 $\mathbb{N} \times \mathbb{N}$ *ist abzählbar unendlich.*

Lemma 1.2 $\mathbb{Q}$ *ist abzählbar unendlich.*

Theorem 1.1 (Cantor) $\mathbb{R}$ *ist nicht abzählbar unendlich.*

Definition 1.4 *Ist* Σ *eine Menge, so ist* Σ^* *die Menge aller endlichen Folgen/Listen/Sequenzen von Elementen aus* Σ*.*

Konventionen:

- Σ wird oft *Alphabet* genannt, Σ^* die Menge der *Wörter* über Σ.
- Wörter werden einfach durch Ananeinanderreihung von Buchstaben von Σ gebildet. Beispiel: Falls $a, b \in \Sigma$ dann ist $aba \in \Sigma^*$.
- Die leere Liste wird mit $\epsilon \in \Sigma^*$ bezeichnet.

Lemma 1.3 *Ist* Σ *endlich, so ist* Σ^* *abzählbar unendlich.*

Kapitel 2

Berechenbarkeit

2.1 Algorithmen und berechenbare Funktionen

Informelle Definition des Begriffs *Algorithmus*:

> Endliche, eindeutige Beschreibung eines (Berechenungs-)Verfahrens, in dem jeder Einzelschritt effektiv, d.h. mit den gegebenen Mitteln in endlicher Zeit ausgef"uhrt werden kann.

Beispiele:

- Kochrezepte (solange sie detailliert genug sind)
- Javaprogramme

Gegenbeispiele:

- Unendliche Tabellen
- "Wenn in der Dezimalexpansion von π die Folge 987654321 enthalten ist, gib 1 aus, sonst 0"

Auf jeden Fall ist ein Algorithmus ein Text über einem endlichen Alphabet. Wir nehmen an Algorithmen berechnen Ein-/Ausgabe-Funktionen, insbesondere Funktionen von $\mathbb{N}$ nach $\mathbb{N}$.

Definition 2.1 *Ein Algorithmus A berechnet eine Funktion $f : \mathbb{N} \to \mathbb{N}$ gdw. für alle $n \in \mathbb{N}$ gilt:*

- *Falls A mit der Eingabe n die Ausgabe m liefert, dann $f(n) = m$*
- *Falls A mit Eingabe n nicht terminiert, dann $f(n) = \perp$*

Eine Funktion ist *berechenbar* gdw. es einen Algorithmus gibt, der sie berechnet.

Theorem 2.1 *Es gibt in* $\mathbb{N} \to \mathbb{N}$ *nicht-berechenbare Funktionen.*

Beweis Betrachte die folgende Tabelle, in der das Ein-/Ausgabe-Verhalten aller Algorithmen verzeichnet ist (mit fiktiven Werten), da die Menge der Algorithmen abzählbar unendlich ist. Nun können wir eine Funktion aus

	Eingabe					
	0	1	2	...	n	...
$Algorithmus_0$	**5**	21	42	...	67	...
$Algorithmus_1$	$\perp$	**73**	$\perp$	...	$\perp$	...
$Algorithmus_2$	0	0	**0**	...	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	...
$Algorithmus_n$	1	2	$\perp$	...	**86**	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	...	$\vdots$	$\ddots$

$\mathbb{N} \to \mathbb{N}$ konstruieren, die nicht in der Tabelle enthalten ist, und die damit nicht berechenbar ist. Nicht enthalten ist die "Diagonale + 1", wobei $\perp+1 := 0$. Formal sei $F(p, n)$ die von $Algorithmus_p$ berechnete Funktion angewandt auf n. Dann ist $f(n) := F(n, n)+1$ eine durch keinen Algorithmus berechnete Funktion.

Die Beweisidee ist die Gleiche wie beim Cantorschen Diagonalschluss: Wir haben gezeigt, dass $\mathbb{N} \to \mathbb{N}$ überabzählbar ist, und damit nich von einer abzählbaren Menge (hier: die Algorithmen) ausgeschöpft werden kann.

2.2 Entscheidbarkeit

Wir identifizieren: Programme = Daten = Bitfolgen = $\mathbb{N}$. Programme werden als Zahlen betrachtet (Idee von Kurt Gödel (1906 - 1978), ursprünglich auf Formeln - analoge Anwendung auf Programme). Man spricht deshalb auch oft von *Gödelisierung.*

Definition 2.2 *Die vom Programm* $p \in \mathbb{N}$ *berechnete Funktion bezeichnen wir mit* φ_p

Theorem 2.2 *Spezielles Halteproblem: Die Funktion*

$$T(x) = \begin{cases} 1 & \textit{falls } \varphi_{n(n)} \neq \perp \\ 0 & \textit{falls } \varphi_{n(n)} = \perp \end{cases}$$

ist nicht berechenbar.

BEWEIS Wir stellen eine Tabelle auf, die alle Algorithmen vertikal, alle Eingaben horizontal und im Schnittpunkt das Ergebnis enthält:

	0	1	2	...
0	$\perp$	3	5	...
1	2	7	$\perp$	
2				
$\vdots$				

Wir konstruieren durch Modifikation der Diagonale eine Funktion, die nicht in der Tabelle ist:

$$T'(x) = \begin{cases} 0 & \text{falls } \varphi_{n(n)} = \perp \\ \perp & \text{falls } \varphi_{n(n)} \neq \perp \end{cases}$$

Damit ist T' nicht berechenbar. Falls T berechenbar wäre, dann wäre auch T' berechenbar, etwa wie folgt:

$$T'(n) = \text{if } T(n) = 0 \text{ then } 0 \text{ else } T'(n)$$

Definition 2.3 *Eine Menge M ist* entscheidbar*, gdw. die charakteristische Funktion*

$$\chi_M(n) = \begin{cases} 1 & \textit{falls } n \in M \\ 0 & \textit{falls } n \notin M \end{cases}$$

berechenbar ist.

Korollar 2.1 *Die Menge $\{p \mid p \in dom(\varphi_p\}$ ist nicht entscheidbar.*

BEWEIS Die charakteristische Funktion von $\{p \mid p \in dom(\varphi_p\}$ ist T.

In Worten: Es ist (für ein beliebiges p) nicht entscheidbar, ob $p \in dom(\varphi_p)$.

Fakt 2.1 *Für $p \leq 42$ ist $p \in dom(\varphi_p)$ entscheidbar.*

Fakt 2.2 *Jede endliche Menge ist entscheidbar.*

Achtung: Beide Fakten sind nicht-konstruktiv.

Korollar 2.2 (Allgemeines Halteproblem) *Es ist für beliebige p und n nicht entscheidbar, ob $n \in dom(\varphi_p)$ ist, d.h., ob Algorithmus p mit Eingabe n terminiert.*

Es ist eine empirische Tatsache, dass in jeder allgemeinen Programmiersprache (d.h. ohne künstliche Einschränkungen), ein Interpreter, für die Programmiersprache programmiert werden kann, der p und n als Eingabe nimmt und $\varphi_p(n)$ berechnet. Diesen Interpreter nennen wir U (universelles Programm).

Theorem 2.3 (Rice) *Sei F eine Menge berechenbarer Funktionen mit $\varnothing \neq F \neq$ Menge aller berechenbaren Funktionen, dann ist es für beliebige p unentscheidbar, ob $\varphi_p \in F$. [Bsp. zu F: F = Menge aller total berechenbaren Funktionen]*

Beweis Sei Ω die total (überall) undefinierte Funktion. Wir machen nun eine Fallunterscheidung danach ob $\Omega \in F$.

Im Fall $\Omega \notin F$ sei $f \in F$ beliebig. Also gibt es $k_f \in \mathbb{N}$ mit $f = \varphi_{k_f}$. Wir beschreiben einen Algorithmus, der ein $n \in \mathbb{N}$ auf einen Algorithmus $g(n) \in \mathbb{N}$ abbildet:

> Nimm die Eingabe n und liefere als Ausgabe den folgenden Algorithmus: Mit Eingabe m führe $U(n, n)$ aus. Terminiert dies, führe k_f mit Eingabe m aus.

Dann gilt:

$$\varphi_n(n) = \begin{cases} f & \text{falls } \varphi_n(n) \neq \perp \\ \Omega & \text{falls } \varphi_n(n) = \perp \end{cases}$$

Daraus folgt:

$$\varphi_n(n) \neq \perp \Leftrightarrow \ \varphi_{g(n)} = f \ \Leftrightarrow \ \varphi_{g(n)} \in F$$

Damit haben wir das Problem $\varphi_n(n) \neq \perp$ reduziert auf $\varphi_{g(n)} \in F$. D.h., wir könnten $\varphi_n(n) \neq \perp$ entscheiden, falls wir (für ein beliebiges p) $\varphi_p \in F$ entscheiden könnten. Da das spezielle Halteproblem nicht entscheidbar ist, so ist auch $\varphi_p \in F$ nicht entscheidbar.

Korollar 2.3 *Es ist für einen beliebigen Algorithmus unentscheidbar, ob der Algorithmus:*

- *überall terminiert (F = Menge aller total berechenbaren Funktionen)*
- *irgendwo terminiert ($F = \{p \mid \varphi_p \neq \Omega\}$)*

- *mit Eingabe 42 Ausgabe 42 liefert. (*$F = \{p \mid \varphi_p(42) = 42\}$*)*

Fazit des Satzes von Rice:

> Jede nicht-triviale Eigenschaft der von einem Programm berechneten Funktion ist unentscheidbar.

Kapitel 3

Zufallswege und PageRank

3.1 Zufallswege in ungerichteten Graphen

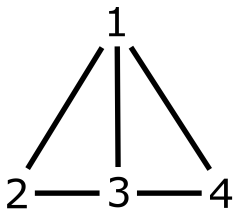


Abbildung 3.1: Beispiel für einen ungerichteten Graphen

Modell: Menschen laufen zufällig durch Graphen, halten sich auf den Knoten auf und springen über die Kanten von Knoten zu Knoten. Es handelt sich dabei um einen synchronen Prozess, alle springen zur gleichen Zeit.

> *Frage:* Wie bestimmt man, wie viele Menschen sich durchschnittlich auf einem Knoten aufhalten, beziehungsweise: Wie bestimmt man den *Rang* eines Knotens?
>
> *Antwort:* Das Problem kann als lineares Gleichungssystem beschrieben werden. Eine Lösung weist jedem Knoten einen Rang zu.

Das zu Abbildung 3.1 gehörige Gleichungssystem:

$$\begin{aligned} r_1 &= \frac{1}{2}r_2 + \frac{1}{3}r_3 + \frac{1}{2}r_4 \\ r_2 &= \frac{1}{3}r_1 + \frac{1}{3}r_2 \\ r_3 &= \frac{1}{3}r_1 + \frac{1}{2}r_2 + \frac{1}{2}r_4 \\ r_4 &= \frac{1}{3}r_1 + \frac{1}{3}r_3 \end{aligned}$$

Eine Lösung ist: $r_1 = r_3 = 3; r_2 = r_4 = 2$. Die allgemeine Lösung in Abhängigkeit eines Parameters $t \in \mathbb{R}$ ist

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = t * \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \end{pmatrix}$$

Beobachtung: Rang = Anzahl der der Nachbarknoten =: Grad

Sei d_k der Grad des Knotens K und N_k die Menge der Nachbarn von K. Dann ergibt sich für den Grad eines Knotens $r_i, i \in N$:

$$r_i = \sum_{j \in Ni} \frac{r_j}{d_j}$$

Eine Lösung ist $r_i = d_i$, denn:

$$r_i = \sum_{j \in N_i} \frac{r_j}{d_j} = \sum_{j \in N_i} 1 = |Ni| = d_i = r_i$$

3.2 Zufallswege in gerichteten Graphen

Modell: Das Modell aus 3.1 bleibt erhalten, allerdings sind jetzt nur noch Sprünge entlang der Pfeilrichtungen möglich. Das zu Abbildung 3.2 gehörige Gleichungssystem ist

$$\begin{aligned} r_1 &= r_4 \\ r_2 &= \frac{1}{2}r_1 \\ r_3 &= r_2 \ + \frac{1}{2}r_1 \\ r_4 &= r_3 \end{aligned}$$

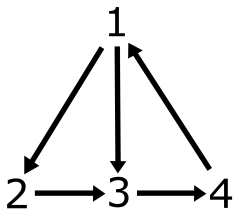


Abbildung 3.2: Beispiel für einen gerichteten Graphen

Die Lösung in Abhängigkeit von einem Paramter $t \in \mathbb{R}$ ist

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = t * \begin{pmatrix} 2 \\ 1 \\ 2 \\ 2 \end{pmatrix}$$

Dieses Modell ist die Grundlage von PageRank. Sei V_i die Menge der Vorgänger von r_i und N_i die Menge seiner Nachfolger. Der Rang eines Knotens berechnet sich dann nach:

$$r_i = \sum_{j \in V_i} \frac{r_j}{|N_j|}$$

Problem: Senken wie z.B. Knoten 2 im Graphen $1 \to 2$.

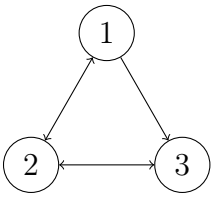
Lösung: Mit einer gewissen kleinen Wahrscheinlichkeit s (z.B. $0,15$) springt ein Surfer nicht zu einer Nachfolgeseite, sondern zu einer beliebigen Seite.

$$r_i = (1-s) * (\sum_{j \in Vi} \frac{r_j}{|N_j|}) + \frac{s}{n}$$

Approximative, iterative Lösung: Da der Rechenaufwand zur Lösung eines Gleichungssystems mit Millionen von Parametern zu groß ist, wird von Google ein approximativer Ansatz gewählt:

$$r_i^{l+1} = (1-s) * (\sum \frac{r_j^l}{|N_j|}) + \frac{s}{n}$$

Beispiel



Ohne Berücksichtigung von s führt das zu folgendem Gleichungssystem:

$$\begin{aligned} r_1^{l+1} &= \frac{1}{2} \cdot r_2^l \\ r_2^{l+1} &= \frac{1}{2} \cdot r_1^l + r_3^l \\ r_3^{l+1} &= \frac{1}{2} \cdot (r_1^l + r_2^l) \end{aligned}$$

Wenn wir mit 8 Surfern auf jedem Knoten beginnen, verläuft die Iteration wie folgt:

$$\begin{matrix} 1 \\ \begin{pmatrix} 8 \\ 8 \\ 8 \end{pmatrix} \end{matrix} \quad \begin{matrix} 2 \\ \begin{pmatrix} 4 \\ 12 \\ 8 \end{pmatrix} \end{matrix} \quad \begin{matrix} 3 \\ \begin{pmatrix} 6 \\ 10 \\ 8 \end{pmatrix} \end{matrix} \quad \begin{matrix} 4 \\ \begin{pmatrix} 5 \\ 11 \\ 8 \end{pmatrix} \end{matrix} \quad \begin{matrix} 5 \\ \begin{pmatrix} 5,5 \\ 10,5 \\ 8 \end{pmatrix} \end{matrix} \quad \rightsquigarrow \quad \begin{pmatrix} 5\frac{1}{3} \\ 10\frac{2}{3} \\ 8 \end{pmatrix}$$

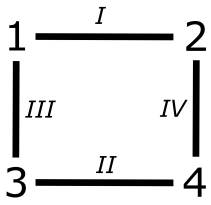
Kapitel 4

Das Telefon-Problem

Eine Gruppe von n Freunden geht jeden Samstag zu n verschiedenen Bundesligaspielen, d.h. je eine Person zu einem Spiel. Danach telefonieren sie miteinander und tauschen in jedem Gespräch alle Informationen aus.

> **Frage:** Was ist die minimale Anzahl an Anrufen, damit jeder alle Ergebnisse kennt?

Eine naive Lösung ist, jeder ruft jeden an ($\frac{n*(n-1)}{n}$ Anrufe). Eine bessere Lösung ist, wenn zuerst $n-1$ Freunde einen fixen Freund k anrufen und dann k $n-2$ Freunde anruft. Insgesamt sind $2n-3$ Anrufe nötig. Allerdings findet sich bereits ab $n=4$ eine noch bessere Lösung:



Optimale Lösung des Telefonproblems für $n \geq 4$ als Algorithmus mit drei Phasen:

					$\sum$	
1.	1-5,	1-6,	...,	1-n,	$n-4$	1 weiß alles über 1, 5, ..., n
2.	1-2,	3-4,	1-3,	2-4,	4	1, 2, 3, 4 wissen alles
3.	1-5,	1-6,	...,	1-n	$n-4$	alle wissen alles
					$2n-4$	

Für den Beweis der Optimaliät siehe z.B. den Artikel von Hurkes [Hur00].

Kapitel 5

Modelle des Berechenbarkeitsbegriffs

5.1 Turingmaschinen

Definition 5.1 *Eine* Turingmaschine *besteht aus einem Speicherband, dessen Speicherzellen alle Werte des (endlichen) Bandalphabets Σ annehmen können, sowie einer* Kontrollinstanz *mit einem Zustand $q \in Q$ (mit endlicher Zustandsmenge Q) und einem Programm δ. In jedem Schritt wird die Speicherzelle, an der sich der* Lesekopf *der Turingmaschine befindet, ausgelesen (a). Das Programm bestimmt dann anhand von a und des Zustandes q den Folgezustand q', einen Wert a', mit dem a ersetzt wird, und ob der Kopf nach links oder rechts bewegt werden soll.*

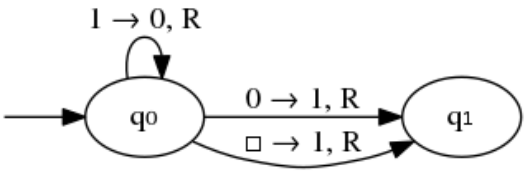
Das Programm δ ist also eine eventuell partielle Abbildung vom Typ $Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$. Ist die Turingmaschine in Zustand q, liest den Wert a und es gilt $\delta(q, a) = (q', a', L/R)$, dann ist der Folgezustand q', a wird durch a' ersetzt und der Kopf bewegt sich nach links oder rechts. Bewegt sich der Kopf über das Ende des Bandes hinaus, wird automatisch eine Zelle mit Symbol $\square \in \Sigma$ angefügt. Das Band wird mit der Programmeingabe initialisiert, die Ausführung startet o.B.d.A. mit dem Lesekopf am linken Ende des Speicherbands und im Startzustand q_0. Ist $\delta(q, a)$ nicht definiert, terminiert das Programm.

Eine *Konfiguration* einer Turingmaschine ist ein Tripel, das den Gesamtzustand einer Turingmaschine zu einem bestimmten Zeitpunkt beschreibt. Sie enthält das Band links des Lesekopfes, den Zustand q und das Band rechts des Lesekopfes (inkl. der Speicherzelle, an der der Kopf gerade positioniert ist) enthält, ist also ein Element aus $\Sigma^* \times Q \times \Sigma^*$.

Das Verhalten einer Turingmaschine kann man dann als binäre Relation auf Konfigurationen beschreiben.

Beispiel *Binärer Inkrementierer*
Es soll das niederwertigste Bit zuerst (links) im Speicher liegen.

$$\begin{aligned} \delta: \quad & (q_0, 1) \to (q_0, 0, R) \\ & (q_0, 0) \to (q_1, 1, R) \\ & (q_0, \square) \to (q_1, 1, R) \end{aligned}$$



Beispielabläufe mit Anfangskonfigurationen $(\varepsilon, q_0, 011)$ und $(\varepsilon, q_0, 11)$ (Inkrementierung von 6 und 3):

$$\begin{aligned} & (\varepsilon, q_0, 011) \rightsquigarrow (1, q_1, 11) \\ & (\varepsilon, q_0, 11) \rightsquigarrow (0, q_0, 1) \rightsquigarrow (00, q_0, \square) \rightsquigarrow (001, q_1, \square) \end{aligned}$$

Definition 5.2 *Eine Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann* Turingmaschinen-berechenbar*, wenn es eine Turingmaschine gibt, die bei Eingabe $w \in \Sigma^*$ mit Ausgabe $f(w)$ anhält, falls $f(w) \neq \perp$ und nicht terminiert, falls $f(w) = \perp$.*

Church-Turing These Die Turingmaschinen-berechenbaren Funktionen sind genau die im intuitiven Sinne berechenbaren.

5.2 Registermaschinen

Definition 5.3 *Eine* Registermaschine *besteht aus einer Anzahl von Registern $R_1, \ldots R_n$. Jedes Register kann eine beliebige natürliche Zahl $\in \mathbb{N}$ enthalten. Ein Registermaschinen-Programm besteht aus einer endlichen Sequenz der Instruktionen*

`INC` R_i	inkrementiert R_i um 1
`DEC` R_i	dekrementiert R_i um 1 (wobei $0 \dot{-} 1 = 0$)
`GOTO` n	fahre mit Instruktion n fort
`IF` $R_i = 0$ `GOTO` n	falls $R_i = 0$, fahre mit Instruktion n fort

Beispiel

Setze R_i auf 0 ($R_i := 0$)

```
0: IF R_i = 0 GOTO 3
1: DEC R_i
2: GOTO 0
```

Setze R_i auf $n = const.$ ($R_i := n$)

```
0: R_i := 0
1: INC R_i
⋮
n: INC R_i
```

Setze R_i auf $R_i + R_j$ mit $i \neq j$ ($R_i := R_i + R_j$). Dabei wird $R_j := 0$.

```
0: IF R_j = 0 GOTO 4
1: DEC R_j
2: INC R_i
3: GOTO 0
```

Setze R_i auf R_j mit $i \neq j$ ($R_i := R_j$). Dabei wird $R_j := 0$.

```
0: R_i := 0
1: R_i := R_i + R_j
```

Definition 5.4 *Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ ist genau dann* Registermaschinen-berechenbar*, wenn es eine Registermaschine gibt, die, wenn zu Beginn $R_0 = m$, $R_1, \ldots R_k = 0$, mit $R_0 = f(m)$ anhält, falls $f(m) \neq \perp$ und nicht terminiert, wenn $f(m) = \perp$.*

Theorem 5.1 *Jede Turingmaschinen-berechenbare Funktion ist Registermaschinenberechenbar und umgekehrt (wobei wir geeignet zwischen $\mathbb{N}$ und Σ^* konvertieren).*

Beweis Jede Registermaschinen-berechenbare Funktion ist turingmaschinenberechenbar

1. Wir repräsentieren die Registerinhalte in unärer Kodierung auf dem Band, getrennt durch #:

$$\underbrace{\texttt{II}\ldots\texttt{III}}_{R_0}\#\underbrace{\texttt{I}\ldots\texttt{I}}_{R_1}\#\ldots\#\underbrace{\texttt{I}\ldots\texttt{I}}_{R_k}$$

2. Wir repräsentieren den Programmzähler durch Zustände $q_0, \ldots, q_l$ mit $l =$ Länge des Programmcodes.

3. Wir simulieren die Befehle `INC`, `DEC` und `IF` der Registermaschine durch Turingmaschinenprogramme

4. Gesamtprogramm:

   - Start ganz links
   - Falls der Zustand q_m ist und Instruktion m des Registermaschinenprogramms benutzt R_i, bewege den Kopf zum linken Ende der Kodierung von R_i und führe das zur Instruktion m passende Turingmaschinen-Programm aus.
   - Bewege den Kopf wieder ganz nach links.

Kapitel 6

Hilberts 10. Problem

Eine *diophantische Gleichung* ist eine Gleichung der Form $f(x_1, ..., x_n) = 0$ wobei $f(x_1, ..., x_n)$ ein Polynom in $x_1, ..., x_n$ mit Koeffizienten in $\mathbb{Z}$ ist.

Hilberts 10. Problem: Entwickle eine Methode, um in endlich vielen Schritten festzustellen, ob eine diophantische Gleichung eine Lösung in $\mathbb{Z}$ hat.

Theorem 6.1 (Matiyasevich 1970) *Die Lösbarkeit von diophantischen Gleichungen in $\mathbb{Z}$ ist unentscheidbar.*

Beispiel

$5x + 3y + 1 = 0$	hat z.B. die Lösungen $x = 1$ und $y = -2$
$4x + 6y + 1 = 0$	hat keine Lösung
$2x^2y^5 + 3xyz^2177x^{12}z^3 + 42 = 0$	???

Lemma 6.1 *Ob ein System diophantischer Gleichungen eine Lösung in $\mathbb{N}$ hat ist reduzierbar auf die Frage, ob eine einzelne Gleichung eine Lösung in $\mathbb{Z}$ hat.*

Wir reduzieren nun das Halteproblem für Registermaschinen auf die Lösbarkeit eines Systems von diophantischen Gleichungen in $\mathbb{N}$. Daraus folgt die Unentscheidbarkeit der Lösbarkeit einer diophantischen Gleichung in $\mathbb{Z}$.

Idee 1 Eine Nullstelle kodiert die Sequenz der Inhalte eines Registers im Verlauf einer terminierenden Berechnung.

Idee 2 Das Gleichungssystem beschreibt Anfangs- und Endzustand und das Verhalten jeder Instruktion.

Eine Sequenz $(n_0, ..., n_S)$ wird kodiert als $\sum_{i=n}^{S} n_i \cdot B^i$, wobei B eine hinreichend große Basis seien muss (für alle i muss $B > n_i$ gelten).

Es sei ein Registermaschinenprogramm $A_1, ..., A_m$ mit Registern $R_1, ..., R_k$ gegeben. Wir konstruieren ein System erweiterter diophantischer Gleichungen, so dass die Registermaschine gestartet mit $R_0 = R_2 = ... = R_k = 0$ genau dann terminiert, wenn das Gleichungssystem eine Lösung in $\mathbb{N}$ hat.

Zur Vereinfachung nehmen wir an, dass es eine `HALT` Instruktion gibt und dass das Registermaischienenprogramm folgende Bedingungen erfüllt:

1. $A_m = \texttt{HALT}, A_i \neq \texttt{HALT}$ für $i < m$
2. Alle Sprungziele liegen zwischen 1 und m
3. Bei Terminierung gilt $R_1 = ... = R_k = 0$
4. $\texttt{DEC}R_j$ wird nur ausgeführt, wenn $R_j > 0$

Alle Bedingungen können durch Transformationen des Programms erreicht werden.

Die (wichtigsten) Variablen des Gleichungssystems:

- B (Basis)
- S (Anzahl der Schritte bis zur Terminierung)
- $W_j, 1 \leq j \leq k$ (Kodierung der Sequenz der Inhalte von R_j zu den Zeitpunkten $0, ..., s$)
- $N_i, 1 \leq i \leq m$ (Kodierung der Sequenz $(a_0, ..., a_S), al \in \{0, 1\}$ wobei $a_l = 1$ genau dann, wenn die Instruktion A_i zum Zeitpunkt l ausgeführt wird)

Beispiel Es seien $B = 10$ und $S = 5$ und durchlaufe R_1 die Inhalte $0, 1, 2, 1, 1, 0$, dann sollte $W_1 = 11210$ gelten.

Erweiterung der diophantischen Gleichungen durch *Exponentiation* und den *Dominanzoperator*: $x \trianglelefteq y$ genau dann, wenn alle Bit in der Binärdarstellung von x kleiner oder gleich den entsprechenden Bits in y sind

Beispiel $1101 \trianglelefteq 11101$

Ungleichungen wie $x < y$ sind darstellbar als $y = x + z + 1$, wobei z eine neue Hilfsvariable ist.

Nun konstruieren wir ein zu dem Programm passendes Gleichungssystem. Zunächst die Gleichungen für B:

$$B > 2S, B > m, B > k, B = 2^c$$

Die Gleichungen für die Randbedingungen für N_i:

$$N_i \trianglelefteq T, 1 + (B-1)T = B^{S+1}$$

Die Gleichungen für die Startbedingung:

$$1 \trianglelefteq N_1$$

Die Gleichungen für die Endbedingung:

$$B^S \trianglelefteq N_m$$

Die Gleichungen für die Startbedingung für W_j:

$$B^{S+1} - B$$

Die Gleichung für die Instruktion $A_i = \texttt{GOTO}j$:

$$B \cdot N_i \trianglelefteq N_j$$

Die Gleichung für die Instruktion$A_i = \texttt{INC}R_j/\texttt{DEC}R_j$:

$$B \cdot N_i \trianglelefteq N_{i+1}, W_j = B \cdot \left(W_j + \sum_{i \in I} N_i - \sum_{i \in D} N_i \right)$$

mit $I = \{1 \leq i \leq m \mid A_i = \texttt{INC}R_j\}$ und $D = \{1 \leq i \leq m \mid A_i = \texttt{DEC}R_j\}$

Beispiel

$$\begin{array}{lcl} W_j & = & 0122110 \\ \sum_{i \in I} N_i & = & 0000101 \\ \sum_{i \in I} N_i & = & \underline{0110000} \\ & & 0012211 \end{array}$$

Zuletzt noch die Gleichung für die Instruktion $A_i = \texttt{IF}R_j = 0\ \texttt{GOTO}\ l$:

$$B \cdot N_i \trianglelefteq N_{i+1} + N_l$$

$$B \cdot N_i \trianglelefteq N_{i+1} + B \cdot T - 2W_j$$

Theorem 6.2 *Das Halteproblem für Registermaschinen ist reduzierbar auf die Lösbarkeit eines Systems erweiterter diophantischer Gleichungen über* $\mathbb{N}$.

Da $\trianglelefteq$ und Exponentiation eliminierbar sind (Matiyasevich) gilt:

Theorem 6.3 *Hilberts 10. Problem ist unentscheidbar.*

Theorem 6.4 (Tarski) *Es ist entscheidbar ob eine Formel mit* $0, 1, +, -, \cdot, <, =, \wedge, \vee, \neg, \exists, \forall$ *über* $\mathbb{R}$ *gilt.*

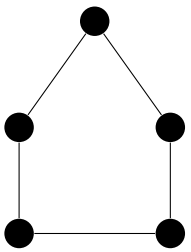
Kapitel 7

Ramsey und Hall

7.1 Der Satz von Ramsey

Für jedes n gibt es in k, so dass auf jeder Party mit mindestens k Personen es entweder n Personen gibt, von denen jeder jeden kennt, oder n Personen, von denen keiner keinen kennt.

Beispiel Für $n = 3$ und $k = 5$ gilt die Aussage nicht.



Definition 7.1 *In einem Graph ist eine* Clique (Anticlique) *eine Teilmenge von Knoten, wo jeder mit jedem (keiner mit keinem) verbunden ist.*

Im Folgenden sind alle Zahlen aus $\mathbb{N}^+$.

Theorem 7.1 (Ramsey 1930) *Für alle m, n gibt es eine kleinste Zahl $R(m, n)$, so dass jeder ungerichtete Graph mit mindestens $R(m, n)$ Knoten entweder eine Clique der Größe m oder eine Anticlique der Größe n hat.*

Beweis mit Induktion über $m + n$.

Basis: $m = 1$ oder $n = 1 \Rightarrow R(1, \_) = R(\_, 1) = 1$

Induktionsschritt: Sei $m, n > 1$. Wir zeigen nun, dass $R(m, n)$ existiert und dass $R(m, n) \leq R(m - 1, n) + R(m, n - 1)$.

Sei G ein Graph mit mindestens $R(m-1,n)+R(m,n-1)$ Kanten. Sei v ein beliebiger Knoten aus G. Wir definieren folgende Mengen:

$$M := \{w \mid v \text{ und } w \text{ benachbart}\}$$
$$N := \{w \mid v \text{ und } w \text{ nicht benachbart}\}\backslash\{v\}$$

Es gilt:

$$|M| + |N| + 1 \geq R(m-1,n) + R(m,n-1)$$

Daraus folgt, dass entweder $|M| \geq R(m-1,n)$ oder $|N| \geq R(m,n-1)$ gilt und wir kommen zu folgender Fallunterscheidung:

1. $|M| \geq R(m-1,n)$

   Entweder hat M eine Anticlique der Größe n (qed) oder eine Clique der Größe $m-1$. Damit hat G die Clique $C \cup \{v\}$.

2. $|N| \geq R(m,n-1)$ symmetrisch.

qed.

Abschätzung $\bar{R}$ für R (obere Grenze des Minimums):

$$\bar{R}(1,\_) = \bar{R}(\_,1) = 1$$
$$\bar{R}(m,n) = \bar{R}(m-1,n) + \bar{R}(m,n-1)$$

Beispiel

$$\begin{aligned}\bar{R}(3,3) &= \bar{R}(2,3) + \bar{R}(3,2)\\ &= \bar{R}(1,3) + \bar{R}(2,2) + \bar{R}(2,2) + \bar{R}(3,1)\\ &= 1 + 2(\bar{R}(1,2) + \bar{R}(2,1)) + 1\\ &= 6\end{aligned}$$

Die Version für mehr als 2 Farben:

Theorem 7.2 (Ramsey) *Sei c die Anzahl der Farben für Kanten. Für alle $n_1,\ldots,n_c$ gibt es eine Zahl $R(n_1,\ldots,n_c)$, so dass es für jeden vollständigen Graphen G mit mindestens $R(n_1,\ldots,n_c)$ Knoten eine Farbe i gibt, so dass G einen Teilgraphen der Größe n_i enthält, in dem alle Kanten mit der Farbe i eingefärbt sind.*

7.2 Der Heiratssatz von Hall

Gegeben ist eine Reihe von $1, \ldots, m$ Männern, die Heiratskandidatinnen $A_1, \ldots, A_m$ haben, die bereit sind, die entsprechenden Männer zu heiraten. Wann gibt es eine Auswahl von Frauen $x_i \in A_i$, so dass $i \neq j \rightarrow x_i \neq x_j$, d.h. die Auswahl ist injektiv.

Beispiel

$$\begin{aligned} A_1 &= \{1, 2\} & x_1 &= 2 \\ A_2 &= \{1, 3\} & x_2 &= 3 \\ A_3 &= \{1, 4\} & x_3 &= 1 \\ A_4 &= \{3, 4\} & x_4 &= 4 \end{aligned}$$

Das Beispiel funktioniert nicht, falls $A_1 = \{1\}$.

Was passiert, wenn es $A_{i_1}, \ldots, A_{i_k}$ gibt, so dass $|A_{i_1} \cup \cdots \cup A_{i_k}| < k$? Dann gibt es keine Auswahl.

Theorem 7.3 (Hall 1935) *Eine Liste endlicher Mengen $A_1, \ldots, A_m$ besitzt eine injektive Auswahl gdw. für alle $I \subseteq \{1, \ldots, m\}$ die Heiratsbedingung H gilt:*

$$\left| \bigcup_{i \in I} A_i \right| \geq |I|$$

BEWEIS mit Induktion über m.
Basis: $m = 1$ ✓(da $A_1 \neq \emptyset$ wegen H)
Induktionsschritt: Sei $m > 1$.
Wir nennen $K \subset \{1, \ldots, m\}$ eine *kritische Familie* gdw. $\left| \bigcup_{i \in I} A_i \right| = |K|$
Fallunterscheidung:

1. Es gibt keine kritische Familie:

   Wähle ein beliebiges $x_m \in A_m$. $A_i^{'} = A_i \backslash \{x_m\}$ $(i = 1, \ldots, m-1)$ erfüllt die Heiratsbedingung, weil es keine kritische Familie gab. Also gibt es nach der Induktionshypothese eine injektive Auswahl $x_1 \in A_1^{'}, \ldots, x_{m-1} \in A_{m-1}^{'}$ und damit auch eine injektive Auswahl $x_1, \ldots, x_m$.

2. Es gibt eine kritische Familie $K \Rightarrow |K| < m$:

   Es gibt nach der Induktionshypothese eine injektive A mit $x_i \in A_i, i \in K$. Es gelte:

$$X := \bigcup_{i \in K} \{x_i\} = \bigcup_{i \in K} A_i$$

Wir zeigen, dass $A_i \backslash X, i \in \bar{K}$ (wobei $\bar{K} = \{1, \ldots, m\} \backslash K$) die Bedingung H erfüllt. Dann gibt es auch hier eine injektive Auswahl nach Induktionshypothese.

Da $A_1, \ldots, A_m$ die Heiratsbedingung erfüllt, gilt:

$$\Big| \bigcup_{i \in I \cup K} A_i \Big| \geq |I| + |K|$$

Daraus folgt, dass $\bigcup_{i \in I} A_i$ mindestens $|I|$ Elemente enthält, die nicht in X sind:

$$\Big| \bigcup_{i \in I} A_i \backslash X \Big| \geq |I|$$

qed.

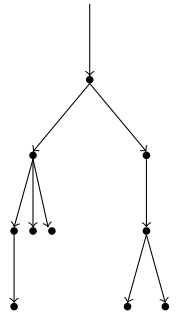
Kapitel 8

Komplexität

8.1 Nichtdeterminismus

Definition 8.1 *Eine* nichtdeterministische Turingmaschine *(*NDTM*) ist definiert wie eine Turingmaschine, aber mit* $\delta : Q \times \Sigma \to 2^{Q \times \Sigma \times \{L,R\}}$*. In jedem Berechnungsschritt wird nichtdeterministisch ein Element aus* $\delta(q, a)$ *ausgewählt.*

Die Berechnungen einer nichtdeterministischen Turingmaschine bilden einen Baum von Konfigurationen.



Jeder Pfad in diesem Baum ist eine mögliche Berechnungsfolge.

Ab jetzt ist eine *deterministische* Turingmaschine (*DTM*) eine NDTM mit $|\delta(q, a)| \leq 1$ für alle $q \in Q$, $a \in \Sigma$.

Ab jetzt hat eine Turingmaschine zusätzlich eine Menge $F \subseteq Q$ der *akzeptierenden* Zustände.

Definition 8.2 *Eine NDTM* akzeptiert $w \in \Sigma^*$ *genau dann, wenn es bei Eingabe von* w *einen Berechnungspfad gibt, der in einer Konfiguration terminiert, in der der Zustand* $\in F$ *ist.*

Beispiel Hat eine aussagenlogische Formel eine erfüllende Belegung?
Eingabe: Eine logische Formel $F(x_1, \ldots, x_k)$, die nur aus (,), $\wedge$, $\vee$, $\neg$, x_1, $\ldots$, x_k besteht.
Ein nichtdeterministischer Algorithmus:
for $i = 1 \ldots k$ *do* $v_i = 0$ oder $v_i = 1$
if $F(v_1, \ldots, v_k) = 1$ *then* gehe in akzeptierenden Zustand.
else gehe in nicht akzeptierenden Zustand.

Definition 8.3 *Sei M eine NDTM. Die von M* akzeptierte Sprache *ist* $\mathcal{L}(M) = \{w \mid M \text{ akzeptiert } w\}$.

Theorem 8.1 *Eine Sprache wird von einer NDTM akzeptiert genau dann, wenn sie von einer DTM akzeptiert wird.*

Beweis Zu jeder NDTM existiert eine DTM, welche die gleiche Sprache akzeptiert, indem sie den Berechnungsbaum der NDTM in Breitensuche nach einer terminierenden Konfiguration mit akzeptierenden Zustand durchsucht.

8.2 Die Komplexitätsklassen *TIME* und *NTIME*

Im Folgenden steht $M[w]$ für "M mit Eingabe w".

Definition 8.4 *Sei $f : \mathbb{N} \to \mathbb{N}$.*

$$\begin{aligned} TIME(f(n)) = \{A \subseteq \Sigma^* \mid \exists DTM\ M.\ \mathcal{L}(M) = A \wedge \\ \forall w \in \Sigma^*.\ M[w] \text{ terminiert in} \leq f(|w|)\ Schritten\} \\ NTIME(f(n)) = \{A \subseteq \Sigma^* \mid \exists NDTM\ M.\ \mathcal{L}(M) = A \wedge \\ \forall w \in \Sigma^*.\ Jeder\ Berechnungspfad\ von\ M[w] \\ terminiert\ in \leq f(|w|)\ Schritten\} \end{aligned}$$

NB: M ist in beiden Fällen eine Entscheidungsprozedur für A.

Beispiel Menge aller Palindrome $\in TIME(c \cdot n^2)$

Beispiel Sei SAT die Menge der erfüllbaren aussagenlogischen Formeln. Dann gilt $SAT \in NTIME(p(n))$ für ein Polynom $p(n)$.

8.3 Die Komplexitätsklassen P und NP

$$P = \bigcup_{p \text{ Polynom}} TIME(p(n))$$
$$NP = \bigcup_{p \text{ Polynom}} NTIME(p(n))$$

Fakt 8.1 *$SAT \in NP$*

Definition 8.5 *$A \subseteq \Sigma^*$ ist* polynomiell reduzierbar *auf $B \subseteq \Gamma^*$ gdw. es eine totale, in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit $w \in A \iff f(w) \in B$. Dann schreiben wir $A \leq_p B$.*

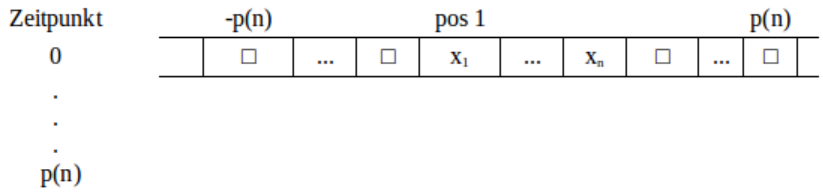
Definition 8.6 *Eine Sprache A ist NP-*hart *gdw. für alle $L \in NP$ gilt $L \leq_p A$. A ist NP-*vollständig *gdw. A NP-hart ist und $A \in NP$.*

Lemma 8.1 *Ist A NP-vollständig, so gilt $A \in P \iff P = NP$.*

Beweis Nach Definition gilt $P \subseteq NP$. Um $NP \subseteq P$ zu zeigen, nehmen wir an $A \in P$. Sei $L \in NP$. Dann gilt $L \leq_p A$ und daraus folgt $L \in P$.

Theorem 8.2 *SAT ist NP-vollständig.*

Beweis Wir müssen zeigen, dass SAT NP-hart ist. Sei $L \in NP$, dann ist zu zeigen $L \leq_p SAT$. Es gibt also eine NDTM M und ein Polynom p, so dass $L(M) = L$ und $M[w]$ terminiert in max. $p(|w|)$ Schritten. Gegeben sei die Eingabe $x_1 \dots x_n \in \Sigma^*$. Nun soll in polynomieller Zeit eine aussagenlogische Formel F mit $x_1 \dots x_n \in L \iff F \in SAT$ konstruiert werden. Die Verhalten von $M[w]$ wird im Folgenden Bild verdeutlicht. Der Kopf beginnt bei x_1. Er kann sich auf Grund der maximalen Zeit nur $p(n)$ Schritte nach rechts oder links bewegen, deshalb sind nur diese Felder von Bedeutung.



Sei $T = \{0, ..., p(n)\}$ und $I = \{-p(n), ..., p(n)\}$, die Mengen der relevanten Zeitpunkte und Felder des Bandes. Die Formel F enthält die folgenden boolschen Variablen:

Boolsche Variablen	Bedeutung
$zust_{t,q}$ $(t \in T, q \in Q)$	$zust_{t,q} = 1$ gdw. $M[x_1, ..., x_n]$ ist nach t Schritten in Zustand q (auf einem der möglichen Berechnungspfade).
$pos_{t,i}$ $(t \in T, i \in I)$	$pos_{t,i} = 1$ gdw. nach t Schritten der Kopf in Position i sein kann.
$band_{t,i,a}$ $(t \in T, i \in I, a \in A)$	$band_{t,i,a} = 1$ gdw. nach t Schritten auf dem Band auf Position i a stehen kann.

Nun folgt noch eine Ergänzung zur Notation:

$$\bigwedge_{t \in T} F_t = F_0 \wedge F_1 \wedge ... \wedge F_{p(n)}$$

und analog für $\bigvee$ und für andere Indexmengen.

Formel F hat folgende Gestalt

$$F = R \wedge A \wedge \ddot{\mathrm{U}}_1 \wedge \ddot{\mathrm{U}}_2 \wedge E$$

wobei die Teilformeln wie folgt konstruiert werden. Dabei verwenden wir eine Hilfsformel G (mit variabler Anzahl von Argumenten) mit der Eingeschaft

$$G(v_1, ...v_r) \Longleftrightarrow \text{genau einer der } v_1, ..., v_r \text{ ist } 1$$

Übung: Konstruiere ein G, das quadratisch groß in r ist.

Randbedingung R:

$$\bigwedge_{t \in T} (G(zust_{t,q_0}, ..., zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, ..., pos_{t,p(n)}) \wedge \bigwedge_{i \in I} G(band_{t,i,a_1}, ..., band_{t,i,a_l}))$$

wobei $Q = \{q_0, ..., q_k\}$ und $\Sigma = \{a_1, ..., a_l\}$.

Anfangsbedingung A:

$$zust_{0,q_0} \wedge pos_{0,1} \wedge \bigwedge_{i=1}^{n} band_{0,i,x_i} \wedge \bigwedge_{i=-p(n)}^{0} band_{0,i,\square} \wedge \bigwedge_{i=n+1}^{p(n)} band_{0,i,\square}$$

Übergangsbedingung $\ddot{\mathrm{U}}_1$:

$$\bigwedge_{(t,i,q,a) \in T \times I \times Q \times \Sigma} ((zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a}) \longrightarrow \bigvee_{(q',a',m) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+d(m)} \wedge band_{t+1,i,a'}))$$

wobei $d(L) = -1$ und $d(R) = 1$. Falls $\delta(q, a) = \varnothing$ ersetze $\bigvee$ ... durch einen Stotterschritt: $zust_{t+1,q} \wedge pos_{t+1,i} \wedge band_{t,i,a}$

Übergangsbedingung $\ddot{U}_2$:

$$\bigwedge_{(t,i,a) \in T \times I \times \Sigma} (\neg pos_{t,i} \wedge band_{t,i,a}) \longrightarrow band_{t+1,i,a}$$

Endbedingung E:

$$\bigwedge_{q \in F} zust_{p(n),q}$$

wobei $F \subseteq Q$ die Menge der akzeptierenden Zustände ist.

Es sollte nun gelten: Jede erfüllende Belegung von F beschreibt einen möglichen akzeptierenden Berechnungspfad von $M[x_1 \dots x_n]$ und jeder akzeptierende Berechnungspfad von $M[x_1 \dots x_n]$ der Länge $\leq p(n)$ korrespondiert zu einer erfüllenden Belegung.

Außerdem kann man F in polynomieller Zeit (in n) konstruieren. Größe der Formeln:

- $|G(v_1, ...v_r)| = O(r^2)$
- $|R| = O(p(n)^3)$
- usw. für alle anderen Formeln

Kapitel 9

Prolog

9.1 Syntax und Semantik von reinem Prolog

Definition 9.1 *Sei Σ eine Menge von Funktionssymbolen und V eine Menge von Variablen. Die Menge der Terme $T(\Sigma, V)$ ist induktiv definiert:*

- *Jede Variable ist ein Term.*
- *Falls $t_1, ..., t_n \in T(\Sigma, V)$ und $f \in \Sigma$, dann $f(t_1, ..., t_n) \in T(\Sigma, V)$.*

Definition 9.2 *Ein Prolog-Programm ist eine Menge von Implikationen („Hornklauseln“)*

$$A \leftarrow A_1 \wedge ... \wedge A_n$$

wobei $A, A_1, ..., A_n$ von der Form $p(t_1, ..., t_k)$ sind („Literale“), wobei p ein Prädikatensymbol ist und $t_1, ..., t_k$ sind Terme.

Definition 9.3 *Eine Substitution $\sigma = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$ ist eine endliche Abbildung von Variablen auf Terme (wobei $x_i \neq x_j$, falls $i \neq j$), so dass $\sigma(x_i) = t_i$ und $\sigma(y) = y$, falls $y \notin \{x_1, ..., x_n\}$. Erweiterung von Substitutionen von Variablen auf Terme: $\sigma(f(t_1, ..., t_n)) = f(\sigma(t_1), ..., \sigma(t_n))$.*

Die Semantik eines Prolog-Programmes P ist induktiv definiert als die Menge der aus P mit folgender Regel herleitbaren Literale. Falls $(A \leftarrow A_1 \wedge ... \wedge A_n) \in P$, so gilt:

$$\frac{P \vdash \sigma(A_1) \qquad ... \qquad P \vdash \sigma(A_n)}{P \vdash \sigma(A)}$$

wobei $P \vdash A$ gelesen wird als „Aus P ist A herleitbar“.

Beispiel

$$P = \{ \quad add(0,N,N), \quad add(s(M),N,s(A)) \leftarrow add(M,N,A), \\ mult(0,N,0), \quad mult(s(M),N,A) \leftarrow mult(M,N,P) \wedge add(P,N,A) \quad \}$$

$$\frac{P \vdash mult(0,s(0),0) \qquad \dfrac{P \vdash add(0,0,0)}{P \vdash add(s(0),0,s(0))}}{P \vdash mult(s(0),s(0),s(0))}$$

9.2 Unifikation

Definition 9.4 *Ein Unifikator von $s = t$ ist eine Substitution σ, so dass $\sigma(s) = \sigma(t)$, das heißt, $\sigma(s)$ und $\sigma(t)$ sind syntaktisch identisch.*

Beispiel

- $X = a$ hat den Unifikator $\{X \mapsto a\}$.
- $X = g(Y)$ hat unendlich viele Unifikatoren: $\{X \mapsto g(Y)\}, \{X \mapsto g(a), Y \mapsto a\}, \{X \mapsto g(g(Z)), Y \mapsto g(Z)\}, ...$
- $f(X) = g(Y)$ hat keinen Unifikator.
- $f(X) = X$ hat keinen Unifikator.

Definition 9.5 *σ ist allgemeinster Unifikator von $s = t$ gdw. $\sigma(s) = \sigma(t)$ und für jeden Unifikator σ' von $s = t$ gibt es eine Substitution δ, so dass $\sigma' = \delta \circ \sigma$.*

Theorem 9.1 *Hat $s = t$ einen Unifikator, dann hat es auch einen allgemeinsten Unifikator.*

9.3 Ein Prolog-Interpreter

Definition 9.6 *Sei P ein Prolog-Programm. Die Relation $\overset{\sigma}{\to}_P$ auf Listen von Literalen ist wie folgt definiert:*

$$[B_1, ..., B_m] \overset{\sigma}{\to}_P \sigma[A_1, ..., A_n, B_2, ...B_n]$$

wobei

- *$(A \leftarrow A_1 \wedge ... \wedge A_n)$ ist eine Umbenennung einer Klausel aus P, sodass die Variablen in $A, A_1, ..., A_n$ disjunkt von denen in $B_1, ..., B_m$ sind.*

- σ *ist allgemeinster Unifikator von* $B_1 = A$.

Beispiel

$$
\begin{array}{ll}
[mult(s(0), s(0), X)] & \\
\xrightarrow{\sigma_1}_P [mult(0, s(0), P_0), add(s(0), P_0, A_0)] & \sigma_1 = \{X \mapsto A_0\} \\
\xrightarrow{\sigma_2}_P [add(s(0), 0, A_0)] & \sigma_2 = \{P_0 \mapsto 0\} \\
\xrightarrow{\sigma_3}_P [add(0, 0, A_1)] & \sigma_3 = \{A_0 \mapsto s(A_1)\} \\
\xrightarrow{\sigma_4}_P [] & \sigma_4 = \{A_0 \mapsto 0\}
\end{array}
$$

Gesamtsubstitution:

$$\sigma := \sigma_4 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1$$

$$\sigma(X) = s(0)$$

Theorem 9.2 *(Korrektheit des Interpreters bezüglich der Semantik) Falls* $[A] \xrightarrow{\sigma_1}_P ... \xrightarrow{\sigma_n}_P []$*,dann gilt* $P \vdash \sigma_n(...\sigma_1(A)...)$.

Theorem 9.3 *(Vollständigkeit des Interpreters bezüglich der Semantik) Falls* $P \vdash \sigma(A)$*, dann gibt es* $[A] \xrightarrow{\sigma_1}_P ... \xrightarrow{\sigma_n}_P []$ *und* δ*, sodass* $\sigma = \delta \circ \sigma_n \circ ... \circ \sigma_1$.

Kapitel 10

Fourier-Motzkin Elimination

Motivation: Gegeben das Programmfragment:

$$\begin{array}{l}\text{if } x_1 + x_2 \leq -2 \text{ then} \\ \quad \text{if } -x_1 - 2x_2 \leq 1 \text{ then} \\ \quad\quad \text{if } -x_1 + x_2 \leq 2 \text{ then (P) ...}\end{array}$$

ist der Programmpunkt (P) erreichbar?

Äquivalente Frage: Ist das linke System linearer Ungleichungen erfüllbar?

$$\begin{array}{rcl} x_1 + x_2 & \leq -2 & \\ -x_1 - 2x_2 & \leq 1 & \Leftrightarrow \\ -x_1 + x_2 & \leq 2 & \end{array} \quad \begin{array}{rcl} -x_2 & \leq -1 & \\ x_2 & \leq 0 & \Leftrightarrow \\ & & \end{array} \quad \begin{array}{rcl} 0 & \leq -1 \\ & & \end{array}$$

Fourier-Motzkin Elimination: Gegeben eine Menge linearer Ungleichungen der Form $\sum a_i x_i \leq b$ wobei $a_i, b \in \mathbb{R}$. Entscheide durch sukzessive Variablenelimination, ob sie über $\mathbb{R}$ erfüllbar sind.

- Setze jedes $a_k \neq 0$ auf 1 oder -1, indem die Ungleichung durch $|a_k|$ dividiert wird.
- Sind alle a_k positiv oder alle a_k negativ, so ist das System erfüllbar: Setze alle Variablen auf 0, nur x_k auf das Minimum/-Minimum der b's.
- Sonst: Die neuen Ungleichungen sind die Menge aller Additionen

  $$\sum (a_i + a'_i) x_i \leq b + b'$$

  von zwei Ungleichungen $\sum a_i x_i \leq b$ und $\sum a'_i x_i \leq b'$ mit $a_k = 1$ und $a'_k = -1$. Ungleichungen mit $a_k = 0$ werden unverändert übernommen.

Sind alle Variablen eliminiert, dann ist das System erfüllt, gdw alle $b \geq 0$.

Wir verallgemeinern dieses Verfahren nun auf allgemeine logische Formeln: *Atomare Formeln* (lineare (Un)gleichung $s < t$, $s = t$) können mit $\wedge$,$\vee$,$\neg$,$\forall$,$\exists$ verknüpft werden. Abkürzung: $\forall x.\Phi \equiv \neg\exists x.\neg\Phi$.

Fakt 10.1 *Falls man für jede Formel $\exists x.\Phi$, Φ quantorenfrei, eine quantorenfreie Formel Φ' berechnen kann mit $(\exists x.\Phi) \Leftrightarrow \Phi'$ und die freien Variablen auf beiden Seiten die gleichen sind, dann sind alle Formeln ohne freie Variablen entscheidbar.*

Prinzip: Eliminiere alle Quantoren (von innen nach außen). Werte variablen- und quantorenfreie Formeln aus.

Was noch bleibt: Transformation von $\exists x.\Phi$, Φ quantorenfrei, in die Form $\exists x. \bigwedge s_i < t_i$ (*)

1. Bringe die Formel in Negationsnormalform (NNF)

$$\begin{array}{lcl} \neg(\Phi_1 \wedge \Phi_2) & \Leftrightarrow & \neg\Phi_1 \vee \neg\Phi_2 \\ \neg(\Phi_1 \vee \Phi_2) & \Leftrightarrow & \neg\Phi_1 \wedge \neg\Phi_2 \\ \neg\neg\Phi & \Leftrightarrow & \Phi \\ \neg s < t & \Leftrightarrow & (t < s) \vee (t = s) \\ \neg s = t & \Leftrightarrow & (s < t) \vee (t < s) \end{array}$$

2. Bringe die Formel DNF:

$$\Phi_1 \wedge (\Phi_2 \vee \Phi_3) \Leftrightarrow (\Phi_1 \wedge \Phi_2) \vee (\Phi_1 \wedge \Phi_3)$$

   Resultat: $\exists x. \bigvee_i \bigwedge_j atom_{ij}$

3. Neue Formel: $\bigvee_i(\exists x. \bigwedge_j atom_{ij})$ weil $(\exists x.\Phi_1 \vee \Phi_2) \Leftrightarrow (\exists x.\Phi_1) \vee (\exists x.\Phi_2)$. Jetzt muss jede $\exists$-Formel getrennt weiterbehandelt werden.

4. $\exists x. \bigwedge A \Leftrightarrow (\exists x. \bigwedge A_x) \wedge B$, wobei $A_x = \{\Phi \in A \mid \Phi \text{ enthält } x\}$ und $B = A \setminus A_x$.

5. Entweder ist $\exists x. \bigwedge A_x$ schon von der Form (*) oder sie enthält noch Gleichungen. Eliminiere Gleichungen $(s = t) \in A_x$: löse nach x auf und setze im Rest von A_x ein, was direkt eine quantorenfreie Formel liefert.

Beispiel

$$
\begin{aligned}
& \forall x.2x < 3 \rightarrow 3x \leq 5 \\
\Leftrightarrow \quad & \neg\exists x.\neg(\neg 2x < 3 \vee 3x \leq 5) \\
\Leftrightarrow \quad & \neg\exists x.(2x < 3 \wedge 3x > 5) \\
\Leftrightarrow \quad & \neg\exists x.(x < \frac{3}{2} \wedge \frac{5}{3} < x) \\
\Leftrightarrow \quad & \neg(\frac{5}{3} < \frac{3}{2}) \\
\Leftrightarrow \quad & True
\end{aligned}
$$

Literaturverzeichnis

[Hur00] C.A.J. Hurkes. Spreadig gossip efficiently. *Nieuw Arch. Wiskd.*, 5/1(2):208–210, 2000.