Vorlesung Berechenbarkeit und Logik

Markus Lohrey

Universität Siegen

Wintersemester 2025/2026

Organisatorisches zur Vorlesung

Unter `https://www.eti.uni-siegen.de/ti/lehre/ws2526/bul/index.html?lang=de` gibt es

- aktuelle Versionen der Folien,
- Übungsblätter,
- aktuelle Informationen, etc.

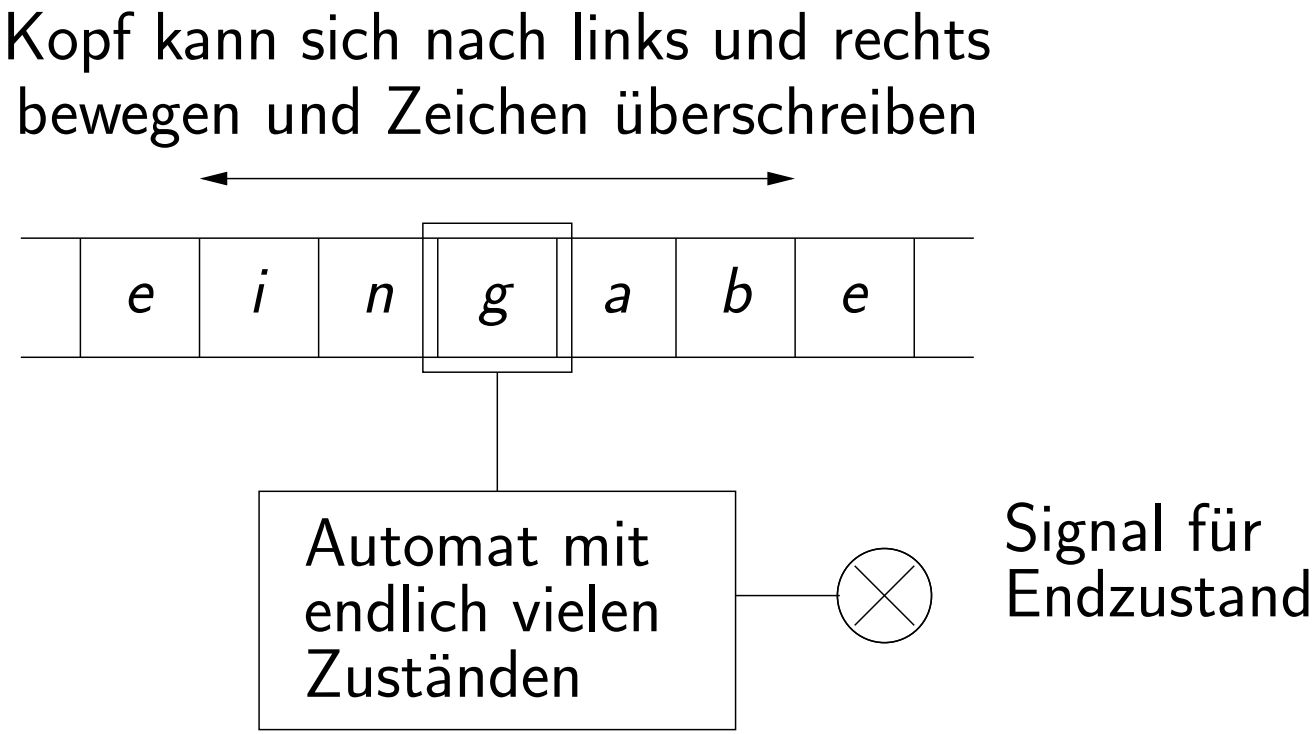
Literaturempfehlungen:

- Uwe Schöning, Theoretische Informatik – kurz gefasst, Spektrum Akademischer Verlag (5. Auflage): Der Teil zu Berechenbarkeit folgt inhaltlich sehr eng diesem Buch.
- Uwe Schöning, Logik für Informatiker, Spektrum Akademischer Verlag: Der Teil zu Logik folgt inhaltlich sehr eng diesem Buch.
- Alexander Asteroth, Christel Baier, Theoretische Informatik, Pearson Studium: Dieses Buch ist vom Aufbau etwas anders strukturiert als die Vorlesung, stellt aber dennoch eine sehr gute Ergänzung für die Vorlesungsteile Berechenbarkeit und Aussagenlogik dar.

Alexander Thumm organisiert die **Übungen**.

Berechenbarkeit: was wir aus FSA benötigen

- In der Vorlesung *Formale Sprachen und Automaten* haben wir am Ende das Model der Turingmaschinen kennengelernt.
- Deterministische als auch nichtdeterministische Turingmaschinen akzeptieren genau die Chomsk-Typ-0 Sprachen, siehe FSA, Folie 348 und 352.



Berechenbarkeit: Motivation

Nach der Beantwortung der Frage, welche Sprachen maschinell akzeptierbar sind, beschäftigen wir uns mit der Frage, welche Funktionen berechenbar sind.

Wir betrachten folgende Typen von Funktionen:

- (mehrstellige) Funktionen auf natürlichen Zahlen (die Null ist eingeschlossen):

$$f : \mathbb{N}^k \to \mathbb{N}$$

- Funktionen auf Wörtern:

$$f : \Sigma^* \to \Sigma^*$$

Erlaubt sind auch partielle Funktionen, die nicht notwendigerweise überall definiert sind.

Formal kann man eine partielle Funktion $f : A \to B$ als eine Funktion $f : A \to B \cup \{\perp\}$ definieren, wobei $\perp \notin B$ ein spezielles Element ist, und $f(a) = \perp$ bedeutet, dass f an der Stelle a undefiniert ist.

Berechenbarkeit: Motivation

Intuitiver Berechenbarkeitsbegriff

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ soll als berechenbar angesehen werden, wenn es ein Rechenverfahren/einen Algorithmus/ein Programm gibt, das f berechnet, d.h. für eine Eingabe $(n_1, \ldots, n_k)$ verhält sich das Programm wie folgt:

- Falls $f(n_1, \ldots, n_k)$ definiert ist, terminiert das Programm nach endlich vielen Schritten und gibt $f(n_1, \ldots, n_k)$ aus.
- Falls die Funktion auf $(n_1, \ldots, n_k)$ nicht definiert ist, so soll das Programm nicht stoppen (z.B., durch eine unendliche Schleife).

Berechenbarkeit: Motivation

Die Äquivalenz vieler Berechnungsmodelle (das wird noch gezeigt) und das intuitive Verständnis des Begriffs der Berechenbarkeit führen zu folgender (nicht beweisbaren) These.

Churchsche These

Die durch die formale Definition der Turingmaschinen-Berechenbarkeit (äquivalent: WHILE-Berechenbarkeit, GOTO-Berechenbarkeit, μ-Rekursivität) erfasste Klasse von Funktionen stimmt genau mit der Klasse der im intuitiven Sinne berechenbaren Funktionen überein.

Bemerkungen: Ein Berechnungsmodell, das äquivalent zu Turingmaschinen ist, nennt man auch Turing-mächtig. Der entsprechende Berechenbarkeitsbegriff heißt Turing-Berechenbarkeit.

Fast alle Programmiersprachen sind Turing-mächtig.

Berechenbarkeit: Motivation

Nicht-berechenbare Funktionen

Es gibt Funktionen der Form $f : \mathbb{N} \to \mathbb{N}$, die nicht berechenbar sind.

Beweisidee: wir wählen ein beliebiges Berechnungsmodell und stellen nur eine Anforderung:

> Programme bzw. Maschinen in diesem Berechnungsmodell, können als Wörter über einem endlichen Alphabet kodiert werden.

Dann gilt: es gibt höchstens abzählbar viele Maschinen/Programme.

Berechenbarkeit: Motivation

Aber: es gibt überabzählbar viele (totale) Funktionen.

Wir zeigen dies durch einen Widerspruchsbeweis: angenommen die Menge aller Funktionen $\mathcal{F}$ auf natürlichen Zahlen ist abzählbar. Das heißt, es gibt eine surjektive Abbildung $F : \mathbb{N} \to \mathcal{F}$.

Wir konstruieren die Funktion $g : \mathbb{N} \to \mathbb{N}$ mit

$$g(n) = f_n(n) + 1, \qquad \text{wobei } f_n = F(n).$$

Da F surjektiv ist, muss es eine natürliche Zahl i geben mit $F(i) = g$. Für dieses i gilt dann: $g(i) = f_i(i)$. Aber das ist ein Widerspruch zur Definition von g mit $g(i) = f_i(i) + 1$.

Berechenbarkeit: Motivation

Veranschaulichung: wir stellen F dadurch dar, indem wir zu jedem n die Funktion f_n als Folge $f_n(0)$, $f_n(1)$, $f_n(2)$, ... notieren.

Zum Beispiel:

n	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	$f_n(4)$	...
0	7	20	33	0	12	
1	12	33	94	2	17	
2	99	101	16	11	22	
3	2	0	14	99	42	
4	17	5	77	7	11	
...						

Berechenbarkeit: Motivation

Veranschaulichung: wir stellen F dadurch dar, indem wir zu jedem n die Funktion f_n als Folge $f_n(0)$, $f_n(1)$, $f_n(2)$, ... notieren.

Alle Zahlen auf der Diagonale verwenden ...

n	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	$f_n(4)$	...
0	7	20	33	0	12	
1	12	33	94	2	17	
2	99	101	16	11	22	
3	2	0	14	99	42	
4	17	5	77	7	11	
...						

Berechenbarkeit: Motivation

Veranschaulichung: wir stellen F dadurch dar, indem wir zu jedem n die Funktion f_n als Folge $f_n(0)$, $f_n(1)$, $f_n(2)$, ... notieren.

... und um eins erhöhen. Dadurch erhält man g.

n	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	$f_n(4)$	...
0	8	20	33	0	12	
1	12	34	94	2	17	
2	99	101	17	11	22	
3	2	0	14	100	42	
4	17	5	77	7	12	
...						

Berechenbarkeit: Motivation

Veranschaulichung: wir stellen F dadurch dar, indem wir zu jedem n die Funktion f_n als Folge $f_n(0)$, $f_n(1)$, $f_n(2)$, ... notieren.

Die Funktion g kann aber aufgrund dieser Konstruktion mit keiner der anderen Funktionen übereinstimmen.

n	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	$f_n(4)$	...
0	8	20	33	0	12	
1	12	34	94	2	17	
2	99	101	17	11	22	
3	2	0	14	100	42	
4	17	5	77	7	12	
...						

Berechenbarkeit: Motivation

Veranschaulichung: wir stellen F dadurch dar, indem wir zu jedem n die Funktion f_n als Folge $f_n(0)$, $f_n(1)$, $f_n(2)$, ... notieren.

Die Funktion g kann aber aufgrund dieser Konstruktion mit keiner der anderen Funktionen übereinstimmen.

n	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	$f_n(4)$	...
0	8	20	33	0	12	
1	12	34	94	2	17	
2	99	101	17	11	22	
3	2	0	14	100	42	
4	17	5	77	7	12	
...						

Diese Art von "selbstbezüglichen" Beweisen nennt man aufgrund ihrer Veranschaulichung durch solche Diagramme oft Diagonalisierungsbeweise.

Turing-Berechenbarkeit

Nach dem intuitiven Berechenbarkeitsbegriff beschäftigen wir uns nun mit dem formalen Berechenbarkeitsbegriff, zunächst basierend auf Turingmaschinen.

Wir wissen bereits, was es bedeutet, dass eine Turingmaschine eine Sprache akzeptiert. Nun definieren wir, was es bedeutet, dass eine Turingmaschine eine Funktion berechnet.

Für eine Zahl $n \in \mathbb{N}$ sei $bin(n)$ die Binärdarstellung von n:

- $bin(n) \in 1\{0,1\}^* \cup \{0\}$
- Falls $bin(n) = b_k b_{k-1} \cdots b_0$, so gilt $n = \sum_{i=0}^{k} b_i 2^i$.

Beispiel: $bin(5) = 101$, $bin(6) = 110$, $bin(7) = 111$, $bin(8) = 1000$.

Turing-Berechenbarkeit

Turing-berechenbare Funktionen auf natürlichen Zahlen

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist Turing-berechenbar, falls es eine deterministische Turingmaschine $M = (Z, \{0, 1, \#\}, \Gamma, \delta, z_0, \Box, E)$ gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$ gilt:

- Falls $f(n_1, \ldots, n_k)$ undefiniert ist, so terminiert M auf der Startkonfiguration $z_0 bin(n_1)\#bin(n_2)\# \ldots bin(n_k)\#$ nicht, d. h. es gibt keine Konfiguration $c \in \Gamma^* E \Gamma^+$ mit

$$z_0 bin(n_1)\#bin(n_2)\# \ldots bin(n_k)\# \vdash_M^* c.$$

- Falls $f(n_1, \ldots, n_k)$ definiert ist, und $f(n_1, \ldots, n_k) = m$, dann existiert ein Endzustand $z_e \in E$ mit

$$z_0 bin(n_1)\#bin(n_2)\# \ldots bin(n_k)\# \vdash_M^* \Box \cdots \Box z_e bin(m) \Box \cdots \Box.$$

Turing-Berechenbarkeit

Intuition:

- Wenn man die Zahlen $n_1, \ldots, n_k$ in Binärdarstellung – voneinander getrennt durch $\#$ – aufs Band schreibt, so terminiert M genau dann, wenn $f(n_1, \ldots, n_k)$ definiert ist.
- Falls $f(n_1, \ldots, n_k) = m$, so berechnet M aus $n_1, \ldots, n_k$ die Zahl $f(n_1, \ldots, n_k)$ (möglicherweise umgeben von Leerzeichen) und geht in einen Endzustand über.

Turing-Berechenbarkeit

Turing-berechenbare Funktionen auf Wörtern

Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist Turing-berechenbar, falls es eine deterministische Turingmaschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ gibt, so dass für alle $x \in \Sigma^*$ gilt:

- Falls $f(x)$ undefiniert ist, so terminiert M auf der Startkonfiguration $z_0x\Box$ nicht, d. h. es gibt kein $c \in \Gamma^* E \Gamma^+$ mit $z_0x\Box \vdash_M^* c$.
- Falls $f(x)$ definiert ist, und $f(x) = y$, dann existiert ein Endzustand $z_e \in E$ mit $z_0x\Box \vdash_M^* \Box \cdots \Box z_e y \Box \cdots \Box$.

Intuition:

- Wenn man das Wort x aufs Band schreibt, so terminiert M genau dann, wenn $f(x)$ definiert ist.
- Wenn $f(x) = y$, so berechnet M aus x das Wort y (möglicherweise umgeben von Leerzeichen) und geht in einen Endzustand über.

Turing-Berechenbarkeit

Beispiele für Turing-berechenbare Funktionen:

Beispiel 1: die Nachfolgerfunktion $n \mapsto n+1$ ist Turing-berechenbar (siehe FSA, Folie 324 und 325).

Beispiel 2: Sei Ω die überall undefinierte Funktion. Diese ist auch Turing-berechenbar, etwa durch eine Turingmaschine, die keinen Endzustand hat. Beispielsweise durch eine Turingmaschine mit der Übergangsregel

$$\delta(z_0, a) = (z_0, a, N) \qquad \text{für alle } a \in \Gamma.$$

Beispiel 3: gegeben sei eine Typ-0-Sprache $L \subseteq \Sigma^*$. Wir betrachten die sogenannte "halbe" charakteristische Funktion von L:

$$\begin{aligned} \chi_L \colon \Sigma^* &\to \{1\} \\ \chi_L(w) &= \begin{cases} 1 & \text{falls } w \in L \\ \text{undefiniert} & \text{sonst} \end{cases} \end{aligned}$$

Turing-Berechenbarkeit

Idee für eine Turingmaschine M, die χ_L berechnet:

- Wir verwenden die Transformation "Grammatik $\rightarrow$ Turingmaschine" (FSA, Folie 348), und erhalten eine Turingmaschine M' mit $T(M') = L$.
- Die Maschine M' ist nicht-deterministisch. Wegen der Äquivalenz von deterministischen und nicht-deterministischen Turingmaschinen (FSA, Folie 352) kann man M' in eine deterministische Turingmaschine M'' mit $T(M'') = T(M') = L$ umwandeln.
- Aus M'' erhalten wir leicht eine deterministische Turingmaschine M, die sich wie M'' verhält, außer: Wenn M'' in einen Endzustand übergehen sollte (und damit akzeptiert), dann überschreibt M den kompletten Bandinhalt mit 1 und geht in einen Endzustand über.
- Beachte: Wenn die Eingabe x nicht zu L gehört, wird M nicht terminieren.

Mehrband-Turingmaschine

Wir führen jetzt mehrere neue Berechnungsmodelle ein und zeigen, dass sie alle äquivalent zu Turingmaschinen sind. Das erste davon ist die sogenannte Mehrband-Turingmaschine.

Mehrband-Turingmaschine

- Eine Mehrband-Turingmaschine besitzt k ($k \geq 1$) Bänder mit k unabhängigen Köpfen, aber nur einen Zustand.
- Die Übergangsfunktion hat die Form

$$\delta \colon (Z \setminus E) \times \Gamma^k \to Z \times \Gamma^k \times \{L, R, N\}^k$$

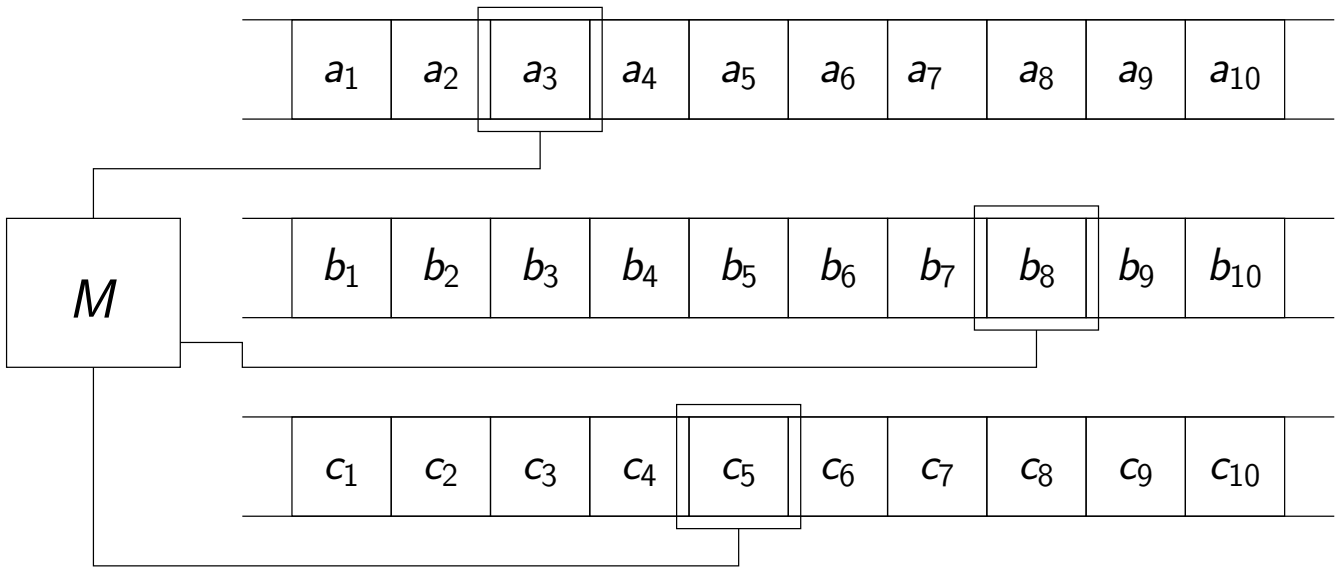
  (ein Zustand, k Bandsymbole, k Bewegungen).
- Die Ein- und Ausgabe stehen jeweils auf dem ersten Band. Zu Beginn sind die restlichen Bänder leer.

Mehrband-Turingmaschine

Satz 1 (Mehrband-Turingmaschinen $\rightarrow$ Turingmaschinen)

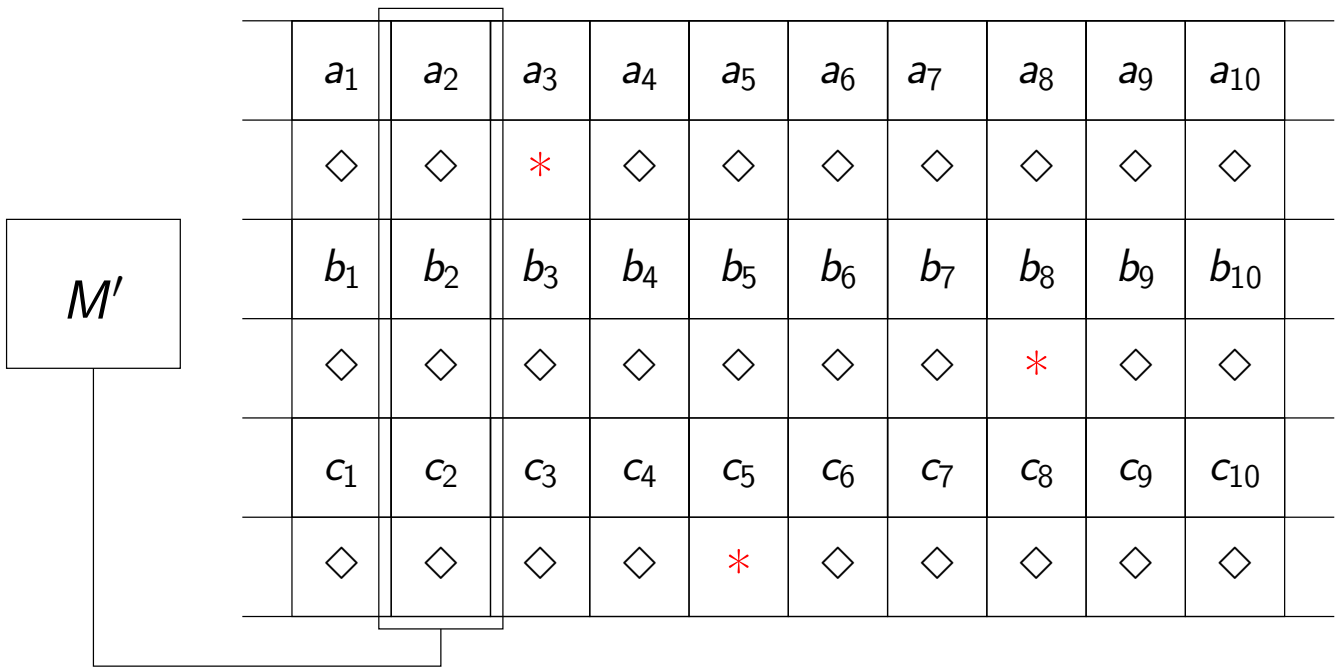
Zu jeder Mehrband-Turingmaschine M gibt es eine (Einband-)Turingmaschine M', die dieselbe Sprache akzeptiert bzw. dieselbe Funktion berechnet.

Beweisidee: wir beginnen mit der Darstellung einer typischen Konfiguration einer Mehrband-Turingmaschine



Mehrband-Turingmaschine

Simulation mittels Einband-Turingmaschine durch Erweiterung des Alphabets: Wir fassen die übereinanderliegenden Bandeinträge zu einem Feld zusammen. Mit den Symbolen $*$ und $\diamond$ wird kodiert, wo die Köpfe der Mehrband-Turingmaschine stehen.



Mehrband-Turingmaschine

Sei k die Anzahl der Bänder von M, Γ ihr Bandalphabet und Σ das Eingabealphabet.

Bandalphabet von M': $\Gamma' = \Sigma \cup (\Gamma \times \{*, \diamond\})^k$

- Bedeutung eines Bandsymbols aus $(\Gamma \times \{*, \diamond\})^k$ am Beispiel von $(a, \diamond, b, *, c, \diamond)$, d.h. $k = 3$:

  Von jedem der 3 Bänder von M tastest M' (die Einband-TM) gerade eine Zelle ab. In der von Band 1 (bzw., 2, 3) abgetasteten M-Zelle steht gerade a (bzw. b, c).

  Der M-Kopf von Band 2 befindet sich gerade auf der von M' abgetasteten Zelle von Band 2 (wegen dem $*$).

  Die M-Köpfe von Band 1 und 3 befinden sich hingegen auf Zellen, die von M' gerade nicht abgetastet werden (wegen den beiden $\diamond$).

Mehrband-Turingmaschine

- Da M' und M das gleiche Eingabealphabet haben, muss Σ zu Γ' gehören. Bekommt M' die Eingabe $w \in \Sigma^*$, so läuft M' in einer ersten Phase einmal komplett über w drüber, und wandelt dabei w in die obige "Mehrband-Kodierung" um.

Problem: Eine Einband-Turingmaschine hat nur einen Kopf und der kann nur an einer Stelle stehen $\rightsquigarrow$ Simulation eines Übergangs der Mehrband-Turingmaschine in mehreren Schritten.

Simulation eines Übergangs der Mehrband-Turingmaschine:

- Zu Beginn der Simulation eines Schritts steht der Kopf der Einband-Turingmaschine M' links von allen $*$-Markierungen.
- Dann läuft der Kopf nach rechts, überschreitet alle k vielen $*$-Markierungen und merkt sich die jeweils anzuwendenden Fälle der δ-Funktion. (Dazu benötigt man viele Zustände.)
- Dann läuft der Kopf wieder zurück nach links und führt alle notwendigen Änderungen aus.

□

Loop-, While-, Goto-Berechenbarkeit

Wir betrachten nun ein weiteres Berechnungsmodell, das im wesentlichen eine einfache Programmiersprache mit verschiedenen Konstrukten darstellt.

- Diese Programme haben Variablen, die mit natürlichen Zahlen belegt sind. Diesen Variablen dürfen arithmetische Ausdrücke (mit Konstanten, Variablen und Operatoren $+$, $-$) zugewiesen werden.
- Außerdem enthalten die Programme verschiedene Schleifenkonstrukte.

Loop-, While-, Goto-Berechenbarkeit

Insbesondere betrachten wir folgende Typen von Programmen:

Loop-Programme

Enthalten nur Loop- bzw. For-Schleifen, bei denen bereits bei Eintritt feststeht, wie oft sie durchlaufen werden.

While-Programme

Enthalten nur While-Schleifen mit einer immer wieder zu evaluierenden Abbruchbedingung.

Goto-Programme

Enthalten Gotos (unbedingte Sprünge) und If-Then-Else-Anweisungen.

Wir interessieren uns vor allem für die Funktionen, die von solchen Programmen berechnet werden.

LOOP-Programme

Syntaktische Komponenten für LOOP-Programme

- Variablen: $x_1, x_2, x_3, \ldots$
- Konstanten: $0, 1, 2 \ldots$
- Trennsymbole: ; und $:=$
- Operatorsymbole: $+$ und $-$
- Schlüsselwörter: LOOP, DO, END

Induktive Syntax-Definition von LOOP-Programmen

Ein LOOP-Programm ist entweder von der Form

- $x_i := x_j + c$ oder $x_i := x_j - c$ mit $c \in \mathbb{N}$ und $i, j \geq 1$ (Wertzuweisung) oder
- $P_1; P_2$, wobei P_1 und P_2 bereits LOOP-Programme sind (sequentielle Komposition) oder
- LOOP x_i DO P END, wobei P ein LOOP-Programm ist und $i \geq 1$.

LOOP-Programme

Informelle Beschreibung der Semantik

- Ein LOOP-Programm, das eine k-stellige Funktion $f : \mathbb{N}^k \to \mathbb{N}$ berechnen soll, startet mit dem Eingabewerten $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$. Alle anderen Variablen haben den Startwert 0. Das Ergebnis $f(n_1, \ldots, n_k)$ liegt bei Termination in x_1.
- Interpretation der Wertzuweisungen:
  - $x_i := x_j + c$ – wie üblich
  - $x_i := x_j - c$ – modifizierte Subtraktion, falls $c > x_j$, so ist das Resultat gleich 0
- Sequentielle Komposition $P_1; P_2$: erst P_1, dann P_2 ausführen.
- LOOP x_i DO P END: das Programm P wird so oft ausgeführt, wie die Variable x_i zu Beginn angibt.

LOOP-Programme

Beachte: Wird in einem Program $\text{LOOP } x_i \text{ DO } P \text{ END}$ der Wert von x_i innerhalb von P verändert, so hat dies keinen Einfluss auf die Anzahl der Ausführungen des Schleifenkörpers P.

Hat x_i den Wert n bevor P das erste mal ausgeführt wird, so wird P genau n mal ausgeführt.

Beispiel: $x_1 := x_1 + 3; \text{LOOP } x_1 \text{ DO } x_1 := x_1 + 1 \text{ END}$

Angenommen x_1 hat zu Beginn den Wert 0.

Dann wird der Schleifenkörper 3 mal ausgeführt.

An Ende hat also x_1 den Wert 6.

LOOP-Programme

Formale Beschreibung der Semantik

Für jedes LOOP-Programm P, in dem keine Variable x_i mit $i > k$ vorkommt (d.h. nur die Variablen $x_1, \ldots, x_k$ dürfen in P vorkommen), definieren wir zunächst eine Funktion

$$[P]_k : \mathbb{N}^k \to \mathbb{N}^k$$

wie folgt durch Induktion über den Aufbau von P:

- $[x_i := x_j + c]_k(n_1, \ldots, n_k) = (m_1, \ldots, m_k)$ genau dann, wenn (i) $m_\ell = n_\ell$ für $\ell \neq i$ und (ii) $m_i = n_j + c$.
- $[x_i := x_j - c]_k(n_1, \ldots, n_k) = (m_1, \ldots, m_k)$ genau dann, wenn (i) $m_\ell = n_\ell$ für $\ell \neq i$ und (ii) $m_i = \max\{0, n_j - c\}$.
- $[P_1; P_2]_k(n_1, \ldots, n_k) = [P_2]_k([P_1]_k(n_1, \ldots, n_k))$
- $[\text{LOOP } x_i \text{ DO } P \text{ END}]_k(n_1, \ldots, n_k) = [P]_k^{n_i}(n_1, \ldots, n_k)$

LOOP-Programme

Intuition: $[P]_k(n_1, \ldots, n_k)$ berechnet sich wie folgt:

- Setze zu Beginn jede Variable x_i mit $1 \leq i \leq k$ auf den Wert n_i.

  Die initialen Werte von Variablen x_i mit $i > k$ spielen keine Rolle, da solche Variablen in P nicht vorkommen.

- Führe nun das Programm P aus.

- Angenommen nach Ausführung von P hat die Variable x_i $(1 \leq i \leq k)$ den Wert m_i.

  Dann gilt $[P]_k(n_1, \ldots, n_k) = (m_1, \ldots, m_k)$.

LOOP-Programme

Sei im folgenden $\pi_i(n_1, \ldots, n_k) = n_i$ (Projektion auf i-te Komponente).

LOOP-Berechenbarkeit (Definition)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt LOOP-berechenbar, falls es ein $\ell \geq k$ und ein LOOP-Programm P, in dem nur die Variablen $x_1, \ldots, x_\ell$ vorkommen, gibt mit:

$$\forall n_1, \ldots, n_k \in \mathbb{N} : f(n_1, \ldots, n_k) = \pi_1([P]_\ell(n_1, \ldots, n_k, \underbrace{0, \ldots, 0}_{\ell - k \text{ viele}})).$$

Zur Berechnung von $f(n_1, \ldots, n_k)$ werden also zu Beginn die Variablen $x_1, \ldots, x_k$ auf die Eingabezahlen $n_1, \ldots, n_k$ gesetzt.

Die Hilfsvariablen $x_{k+1}, \ldots, x_\ell$ werden mit 0 initialisiert.

Dann wird P gestartet.

Nach Ausführung von P ist der Wert der Variablen x_1 der Funktionswert $f(n_1, \ldots, n_k)$.

LOOP-Programme

Bemerkungen:

- Alle LOOP-Programme stoppen nach endlicher Zeit (LOOP-Schleifen terminieren immer).
- Daher sind alle LOOP-berechenbaren Funktionen total (d.h. überall definiert).
- Es gibt also Turing-berechenbare Funktionen, die nicht LOOP-berechenbar sind (z.B. die überall undefinierte Funktion Ω von Folie 14).
- Wir werden später sehen, dass es sogar totale Turing-berechenbare Funktionen gibt, die nicht LOOP-berechenbar sind.

LOOP-Programme

LOOP-Programme können gewisse Programmkonstrukte simulieren, die in der Syntax nicht enthalten sind.

If-Then

Simulation von IF $x_1 = 0$ THEN A END

$x_2 := 1;$ LOOP x_1 DO $x_2 := 0$ END; LOOP x_2 DO A END

Addition

Simulation von $x_i := x_j + x_k$ (sei $i \neq k$)

$x_i := x_j;$ LOOP x_k DO $x_i := x_i + 1$ END

Multiplikation

Simulation von $x_i := x_j \cdot x_k$ (sei $k \neq i \neq j$)
$x_i := 0;$ LOOP x_k DO $x_i := x_i + x_j$ END

LOOP-Programme

Wir werden im folgenden solche Konstrukte auch in While-Programmen verwenden. Wir nehmen dann an, dass sie – wie oben – geeignet simuliert werden.

Analog: Ganzzahlige Division (x DIV y) und Divisionsrest (x MOD y).

WHILE-Programme

Wir erweitern nun die Syntax von LOOP-Programmen zu WHILE-Programmen, indem wir neben LOOP-Schleifen noch ein weiteres Schleifenkonstrukt erlauben.

Syntax von WHILE-Programmen

Wenn P ein WHILE-Programm ist und $i \geq 1$ gilt, so ist auch

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

ein WHILE-Programm.

Alle in LOOP-Programmen erlaubten Konstrukte (siehe Folie 23) sind auch in WHILE-Programmen erlaubt.

Intuition: Programm P wird so lange ausgeführt bis der Wert von x_i gleich 0 ist.

WHILE-Programme

Semantik von WHILE-Programmen

Wie bei LOOP-Programmen definieren wir zunächst für jedes WHILE-Programm P, in dem keine Variable x_i mit $i > k$ vorkommt, eine (diesmal partielle) Abbildung $[P]_k : \mathbb{N}^k \to \mathbb{N}^k$ induktiv.

Für die in LOOP-Programmen verfügbaren Konstrukte übernehmen wir die Definitionen von Folie 26.

Sei nun $P = \text{WHILE } x_i \neq 0 \text{ DO } A \text{ END}$ $(i \leq k)$ und $(n_1, \ldots, n_k) \in \mathbb{N}^k$.

Falls eine Zahl τ mit $\pi_i([A]_k^\tau(n_1, \ldots, n_k)) = 0$ existiert, so sei t die kleinste Zahl mit dieser Eigenschaft. Ansonsten sei t undefiniert.

Dann sei

$$[P]_k(n_1, \ldots, n_k) = \begin{cases} [A]_k^t(n_1, \ldots, n_k) & \text{falls } t \text{ definiert ist} \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

WHILE-Programme

Zur Erklärung: $[A]_k^\tau$ steht für die τ-fache Komposition von $[A]_k$:

$$[A]_k^\tau(n_1, \ldots, n_k) = \underbrace{[A]_k([A]_k([A]_k(\cdots [A]_k}_{\tau \text{ viele}}(n_1, \ldots, n_k)\cdots)))$$

Somit ist t die kleinste Zahl τ, so dass nach τ vielen Ausführungen von A (gestartet mit den initialen Werten $n_1, \ldots, n_k$ für die Variablen $x_1, \ldots, x_k$) die Variable x_i den Wert 0 hat.

Falls x_i niemals den Wert 0 annimmt, terminiert die WHILE-Schleife nicht, und $[P]_k(n_1, \ldots, n_k)$ ist undefiniert.

Beachte: Eine LOOP-Schleife LOOP x DO P END kann simuliert werden durch

$y := x;$
WHILE $y \neq 0$ DO $y := y - 1; P$ END

Wichtig: y ist eine neue Variable, die insbesondere in P nicht vorkommt.

WHILE-Programme

WHILE-Berechenbarkeit (Definition)

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt WHILE-berechenbar, falls es ein $\ell \geq k$ und ein WHILE-Programm P, in dem nur die Variablen $x_1, \ldots, x_\ell$ vorkommen, gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$ gilt:

- $f(n_1, \ldots, n_k)$ definiert $\iff$ $[P]_\ell(n_1, \ldots, n_k, \underbrace{0, \ldots, 0}_{\ell - k \text{ viele}})$ definiert
- Falls $f(n_1, \ldots, n_k)$ definiert ist, gilt $f(n_1, \ldots, n_k) = \pi_1([P]_\ell(n_1, \ldots, n_k, \underbrace{0, \ldots, 0}_{\ell - k \text{ viele}}))$.

WHILE-Programme

Satz 2 (WHILE-Programme $\rightarrow$ Turingmaschinen)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Anders ausgedrückt: Turingmaschinen können WHILE-Programme simulieren.

Beweisidee:

- Wir verwenden eine Mehrband-Turingmaschine, bei der auf jedem Band eine andere Variable des WHILE-Programms in Binärdarstellung gespeichert wird.
  k Variablen $\rightsquigarrow$ k Bänder
- $x_i := x_j + c$ kann von der Turingmaschine durchgeführt werden, indem die Inkrementierungsfunktion $(+1)$ c-mal ausgeführt wird.
- $x_i := x_j - c$ funktioniert ähnlich.

WHILE-Programme

- Sequentielle Komposition $P_1; P_2$:

  Wir bestimmen induktiv Turingmaschinen M_1, M_2 für P_1, P_2.

  Diese machen wir wie folgt zu einer Turingmaschine für $P_1; P_2$:

  - Vereinigung der Zustandsmengen, Bandalphabete und Übergangsfunktionen
  - Anfangszustand ist Anfangszustand von M_1. Endzustände sind Endzustände von M_2.
  - Statt in einen Endzustand von M_1 wird ein Übergang in den Anfangszustand von M_2 gemacht.

    (Vergleiche mit der Konkatenationskonstruktion für endliche Automaten (FSA, Folien 98 und 99).

WHILE-Programme

- WHILE-Schleife WHILE $x_i \neq 0$ DO P END:

  Bestimme zunächst eine Turingmaschine M für P.

  Modifiziere M wie folgt:
  Im neuen Anfangszustand wird zunächst überprüft, ob 0 auf dem i-ten Band steht.
  - Falls ja: Übergang in Endzustand
  - Falls nein: M wird ausgeführt

  Statt Übergang in Endzustand: Übergang in den neuen Anfangszustand.

□

GOTO-Programme

Syntax von GOTO-Programmen

Mögliche Anweisungen für GOTO-Programme:

Wertzuweisung: $x_i := x_j + c$ bzw. $x_i := x_j - c$ (mit $c \in \mathbb{N}$)

Unbedingter Sprung: GOTO M_i

Bedingter Sprung: IF $x_i = c$ THEN GOTO M_j

Stopanweisung: HALT

Ein GOTO-Programm besteht aus einer Folge von Anweisungen A_i, vor denen sich jeweils eine (Sprung-)Marke M_i befindet.

$$M_1 : A_1; M_2 : A_2; \ldots; M_k : A_k$$

(Wenn Marken nicht angesprungen werden, werden wir sie manchmal einfach weglassen.)

GOTO-Programme

Intuitive Semantik von GOTO-Programmen

- Die Anweisungen eines GOTO-Programms werden der Reihe nach ausgeführt.
- Ausnahme: GOTO *M* springt zur Anweisung mit der Marke *M*.
- IF-Anweisungen werden wie üblich interpretiert.
- HALT-Anweisungen beenden GOTO-Programme. (Die letzte Anweisung eines Programms sollte ein HALT oder ein unbedingter Sprung sein.)

Dies ist keine wirklich formale Semantikdefinition. Schreiben Sie als Übung eine formale Semantikdefinition (analog zu WHILE-Programmen) auf.

Wie WHILE-Programme können auch GOTO-Programme in unendliche Schleifen geraten.

GOTO-Programme

GOTO-berechenbare Funktionen sind analog zu WHILE-berechenbaren Funktionen definiert.

Beispiel: ein GOTO-Programm zur Berechnung von $x_1 := x_1 + x_2$

M_1 : IF $x_2 = 0$ THEN GOTO M_2;
$x_1 := x_1 + 1$;
$x_2 := x_2 - 1$;
GOTO M_1;
M_2 : HALT

GOTO-Programme

Satz 3 (WHILE-Programme $\rightarrow$ GOTO-Programme)

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden, d. h. jede WHILE-berechenbare Funktion ist GOTO-berechenbar.

Beweis:

Eine WHILE-Schleife

WHILE $x \neq 0$ DO P END

kann simuliert werden durch

M_1: IF $x = 0$ THEN GOTO M_2;
P;
GOTO M_1;
M_2: ...

□

GOTO-Programme

Auch die nicht ganz so offensichtliche Umkehrung gilt:

Satz 4 (GOTO-Programme → WHILE-Programme)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden, d. h. jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Das ist einer der Gründe dafür, warum in modernen Programmiersprachen im allgemeinen keine GOTOS verwendet werden.

Weitere Gründe: Spaghetti-Code bei Verwendung von GOTOS, siehe auch Edsger W. Dijkstra: "Go To Statement Considered Harmful" (1968), `http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF`.

GOTO-Programme

Beweis (Goto-Programme $\rightarrow$ While-Programme):

Sei $P = (M_1 : A_1; M_2 : A_2; \ldots M_k : A_k)$ ein GOTO-Programm.

Wir simulieren P durch das folgende WHILE-Programm Q mit nur einer WHILE-Schleife:

count := 1;
WHILE count $\neq 0$ DO
IF count = 1 THEN A'_1 END;
IF count = 2 THEN A'_2 END;
$\vdots$
IF count = k THEN A'_k END
END

GOTO-Programme

Hierbei ist A'_i das folgende WHILE-Programm.

$$A'_i = \begin{cases} x_j := x_\ell \pm c;\ \text{count} := \text{count} + 1 & \text{falls } A_i = (x_j := x_\ell \pm c) \\ \text{count} := n & \text{falls } A_i = (\textsc{Goto}\ M_n) \\ \textsc{If}\ x_j = c\ \textsc{Then}\ \text{count} := n & \text{falls } A_i = (\textsc{If}\ x_j = c\ \textsc{Then} \\ \quad \textsc{Else}\ \text{count} := \text{count} + 1\ \textsc{End} & \qquad \textsc{Goto}\ M_n) \\ \text{count} := 0 & \text{falls } A_i = \textsc{Halt} \end{cases}$$

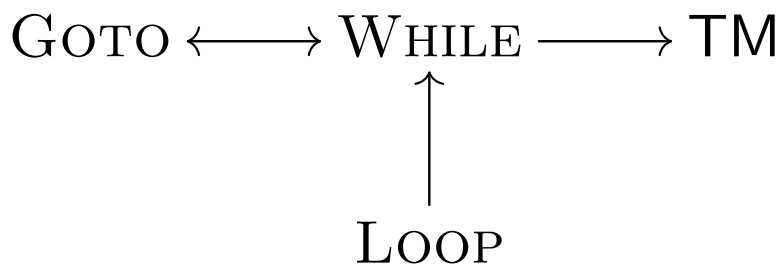
□

Die Simulation von GOTO-Programmen durch WHILE-Programme verwendet nur eine WHILE-Schleife (falls man IF THEN ELSE als elementares Konstrukt erlaubt).

Das bedeutet: Ein WHILE-Programm kann durch Umwandlung in ein GOTO-Programm und Zurückumwandlung in ein WHILE-Programm in ein äquivalentes WHILE-Programm mit einer WHILE-Schleife umgewandelt werden (Kleenesche Normalform für WHILE-Programme).

Goto-Programme

Welche Transformationen haben wir bisher durchgeführt?



Um die Äquivalenz von Goto-, While- und Turing-Berechenbarkeit zu zeigen, fehlt uns noch die Richtung

$$\text{TM} \to \text{Goto}$$

GOTO-Programme

Satz 5 (TM $\rightarrow$ GOTO-Programme)

Jede Turingmaschine kann durch ein GOTO-Programm simuliert werden. Das heißt, jede Turing-berechenbare Funktion ist GOTO-berechenbar.

Beweis:

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, E, \square)$ eine deterministische Turingmaschine, welche eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ berechnet.

O.B.d.A. sei $\Gamma = \{0, \ldots, m-1\}$, $\square = 0$ und $Z = \{0, \ldots, n-1\}$.

Für $a_1 a_2 \cdots a_p \in \Gamma^*$ sei

$$(a_1 a_2 \cdots a_p)_m = \sum_{i=1}^{p} a_i \cdot m^{i-1}$$

der Wert von $a_1 a_2 \cdots a_p$ in der Darstellung zur Basis m (niederwertigste Ziffer ganz links).

GOTO-Programme

Eine Konfiguration $a_1 \cdots a_p\, z\, b_1 \cdots b_q$ wird durch das Tripel

$$\big((a_p \ldots a_1)_m, z, (b_1 \cdots b_q)_m\big) \in \mathbb{N} \times \{0, \ldots, n-1\} \times \mathbb{N}$$

repräsentiert.

Solch ein Tripel wird im folgenden mit den drei Variablen x, z, y gespeichert:

- $z =$ aktueller Zustand
- $x =$ Kodierung des Bandinhalts links vom Kopf
- $y =$ Kodierung des Bandinhalts rechts vom Kopf inklusive der gerade gelesenen Zelle

Beachte: $(a_p \ldots a_1 \square)_m = (a_p \ldots a_1)_m$ und $(b_1 \cdots b_q \square)_m = (b_1 \cdots b_q)_m$, da $\square = 0$ und die höchstwertige Ziffer ganz rechts steht.

Das ist gut so, denn $a_1 \cdots a_p\, z\, b_1 \cdots b_q$ und $\square a_1 \cdots a_p\, z\, b_1 \cdots b_q \square$ repräsentieren die gleiche Konfiguration.

GOTO-Programme

Zur Simulation von Turingmaschinenoperationen verwenden wir die arithmetischen Operationen DIV und MOD (ganzzahlige Division und Divisionsrest), wobei wir immer durch unsere Basis m teilen. Diese Operationen kann man leicht mittels GOTO-Programmen realisieren.

Beispiel: In der Basis $m = 10$ gilt (hier schreiben wir Zahlen wieder wie gewohnt, d.h. die niederwertigste Ziffer steht rechts)

$$\begin{aligned} 5634 \text{ DIV } 10 &= 563 \\ 5634 \text{ MOD } 10 &= 4 \end{aligned}$$

Simulation von Turingmaschinenoperationen:

Kopf liest Zeichen: $a := y \text{ MOD } m$

Zeichen b aufs Band schreiben: $y := (y \text{ DIV } m) \cdot m + b$

Kopf nach links: $y := y \cdot m + (x \text{ MOD } m);\ x := x \text{ DIV } m$

Kopf nach rechts: $x := x \cdot m + (y \text{ MOD } m);\ y := y \text{ DIV } m$

GOTO-Programme

Erläuterung: Sei $a_1 \cdots a_p\, z\, b_1 \cdots b_q$ die aktuelle Konfiguration und damit

$$x = (a_p \ldots a_1)_m, \quad y = (b_1 \cdots b_q)_m \quad (\text{und } z = z)$$

Dann gilt $y \text{ MOD } m = (b_1 + m \cdot (b_2 \cdots b_q)_m) \text{ MOD } m = b_1$.

Also wird die Variable a auf das aktuell gelesene Symbol gesetzt.

Weiter gilt

$$\begin{aligned}
(y \text{ DIV } m) \cdot m + b &= ((b_1 \cdots b_q)_m \text{ DIV } m) \cdot m + b \\
&= (b_2 \cdots b_q)_m \cdot m + b \\
&= (0 b_2 \cdots b_q)_m + b \\
&= (b b_2 \cdots b_q)_m
\end{aligned}$$

Also schreibt $y := (y \text{ DIV } m) \cdot m + b$ tatsächlich b in die aktuelle gescannte Bandzelle.

GOTO-Programme

Es gilt

$$x \text{ DIV } m = (a_p a_{p-1} \dots a_1)_m \text{ DIV } m = (a_{p-1} \dots a_1)_m$$

sowie

$$\begin{aligned} y \cdot m + (x \text{ MOD } m) &= (b_1 \cdots b_q)_m \cdot m + ((a_p a_{p-1} \dots a_1)_m \text{ MOD } m) \\ &= (0 b_1 \cdots b_q)_m + a_p \\ &= (a_p b_1 \cdots b_q)_m \end{aligned}$$

Daher implementiert

$$y := y \cdot m + (x \text{ MOD } m);\ x := x \text{ DIV } m$$

korrekt eine Linksbewegung des Kopfes.

Analog zeigt man, dass

$$x := x \cdot m + (y \text{ MOD } m);\ y := y \text{ DIV } m$$

korrekt eine Rechtsbewegung des Kopfes implementiert.

GOTO-Programme

Sei nun $(n_1, \ldots, n_k) \in \mathbb{N}^k$ eine Eingabetupel.

Das zu erstellende GOTO-Programm besteht aus folgenden drei Programmteilen:

Teil 1: Initialisierung der Variablen x, y, z mit der Anfangskonfiguration.

Die zum Eingabetupel $(n_1, \ldots, n_k)$ gehörende Anfangskonfiguration ist

$$z_0 \,\mathrm{bin}(n_1) \# \,\mathrm{bin}(n_2) \# \cdots \# \,\mathrm{bin}(n_k) \#$$

Also initialisieren wir x, y, z mit

$$x := 0;\ \ z = z_0;\ \ y := (\mathrm{bin}(n_1)\#\mathrm{bin}(n_2)\#\cdots\#\mathrm{bin}(n_k)\#)_m.$$

Beachte: Da $\square$ das Bandsymbol $0 \in \Gamma = \{0, 1, 2, \ldots, m-1\}$ ist, muss das Bit 0 in the den Binärdarstellungen $\mathrm{bin}(n_i)$ durch ein anderes Bandsymbol aus Γ repräsentiert werden. Wir repräsentieren im folgenden das Bit 0 durch $1 \in \Gamma$ und das Bit 1 durch $2 \in \Gamma$.

GOTO-Programme

Die Zahl $(\text{bin}(n_1)\#\text{bin}(n_2)\#\cdots\#\text{bin}(n_k)\#)_m$ muss man zunächst aus den Eingabezahlen $n_1, \ldots, n_k$ (n_i steht zu Beginn in der Variablen x_i) mittels eines GOTO-Programms berechnen und in der Variablen y abspeichern.

Dies kann durch folgenden code gemacht werden (der sich wieder leicht in ein GOTO-Programm umwandeln lässt):

$p := 1$; $y := 0$;
FOR $i := 1$ TO k DO
  WHILE $x_i \neq 0$ DO
    $b_i := (x_i \text{ MOD } 2) + 1$;
    $x_i := x_i \text{ DIV } 2$;
    $y := y + p \cdot b_i$;
    $p := p \cdot m$;
  END
  $y := y + p \cdot \#$;
  $p := p \cdot m$
END

GOTO-Programme

Teil 2: Die Turingmaschinenberechnung wird durch Manipulation von x, z, y simuliert bis schließlich $z \in E$ gilt.

Siehe nächste Folie.

Teil 3: Die in y gespeicherte Zahl wird in den eigentlichen Ausgabewert überführt:

Hat y den Wert $(a_1 a_2 \cdots a_n)_m$ mit $a_1, \ldots, a_n \in \{1, 2\}$, so muss die eindeutige Zahl n mit $\text{bin}(n) = (a_1 - 1)(a_2 - 1) \cdots (a_n - 1)$ berechnet werden.

Diese arithmetische Transformation kann man wieder durch ein GOTO-Programm realisieren (ähnlich zu Teil 1).

Bemerkung: Nur der 2. Teil hängt von der Überführungsfunktion δ ab.

GOTO-Programme

Schema für Teil 2:

M_2: IF $(z \in E)$ THEN GOTO M_3;
$a := y$ MOD m; (Zeichen einlesen)
IF $(z = 0)$ AND $(a = 0)$ THEN GOTO $M_{0,0}$;
IF $(z = 0)$ AND $(a = 1)$ THEN GOTO $M_{0,1}$;
$\vdots$

$M_{0,0}$: $P_{0,0}$; (Aktion $\delta(0,0)$ ausführen)
GOTO M_2;

$M_{0,1}$: $P_{0,1}$; (Aktion $\delta(0,1)$ ausführen)
GOTO M_2;
$\vdots$

M_3: Führe Teil 3 aus

Im Programmteil $P_{i,j}$ wird die durch $\delta(i,j)$ beschriebene Aktion so wie auf Folie 49 unten simuliert. □

Primitiv rekursive und μ-rekursive Funktionen

LOOP-, WHILE- und GOTO-Programme sind vereinfachte imperative Programme und stehen für imperative Programmiersprachen, bei denen Programme als Folgen von Befehlen aufgefaßt werden.

Parallel dazu gibt es jedoch auch funktionale Programme, deren Hauptbestandteil die Definition rekursiver Funktionen ist. Es gibt auch Berechnungsbegriffe, die sich eher an funktionalen Programmen orientieren.

Zum Beispiel:

- λ-Kalkül (Alonzo Church, 1932)
- μ-rekursive und primitiv rekursive Funktionen (werden hier in der Vorlesung betrachtet)

Primitiv rekursive und μ-rekursive Funktionen

Wir definieren nun Klassen von Funktionen der Form $f : \mathbb{N}^k \to \mathbb{N}$.

Primitiv rekursive Funktionen (Definition)

Die Klasse der primitiv rekursive Funktionen ist induktiv wie folgt definiert:

- Alle konstanten Funktionen der Form $k_m : \mathbb{N}^0 \to \mathbb{N}$ mit $k_m() = m$ (für ein festes $m \in \mathbb{N}$) sind primitiv rekursiv.
- Alle Projektionen der Form $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$ mit $\pi_i^k(n_1, \ldots, n_k) = n_i$ für $1 \leq i \leq k$ sind primitiv rekursiv.
- Die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N}$ mit $s(n) = n + 1$ ist primitiv rekursiv.
- Wenn $g : \mathbb{N}^k \to \mathbb{N}$ und $f_1, \ldots, f_k : \mathbb{N}^r \to \mathbb{N}$ $(k \geq 0)$ primitiv rekursiv sind, dann ist auch die Abbildung $f : \mathbb{N}^r \to \mathbb{N}$ mit

$$f(n_1, \ldots, n_r) = g(f_1(n_1, \ldots, n_r), \ldots, f_k(n_1, \ldots, n_r))$$

primitiv rekursiv (Einsetzung/Komposition)

Primitiv rekursive und μ-rekursive Funktionen

Primitiv rekursive Funktionen (Definition, Fortsetzung)

- Jede Funktion f, die durch primitive Rekursion aus primitiv rekursiven Funktionen entsteht ist primitiv rekursiv.

  Das heißt $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ muss folgende Gleichungen erfüllen (für primitiv rekursive Funktionen $g\colon \mathbb{N}^k \to \mathbb{N}$, $h\colon \mathbb{N}^{k+2} \to \mathbb{N}$):

$$\begin{aligned} f(0, n_1, \ldots, n_k) &= g(n_1, \ldots, n_k) \\ f(n+1, n_1, \ldots, n_k) &= h(f(n, n_1, \ldots, n_k), n, n_1, \ldots, n_k) \end{aligned}$$

Anschaulich: bei primitiver Rekursion wird die Definition von $f(n+1, \ldots)$ zurückgeführt auf $f(n, \ldots)$. Das bedeutet, dass primitive Rekursion immer terminiert.

Es handelt sich also um ein Berechnungsmodell analog zu LOOP-Programmen.

Primitiv rekursive und μ-rekursive Funktionen

Beispiele für primitiv rekursive Funktionen:

Additionsfunktion

Die Funktion $add : \mathbb{N}^2 \to \mathbb{N}$ mit $add(n, m) = n + m$ ist primitiv rekursiv.

$$\begin{aligned} add(0, m) &= m \\ add(n+1, m) &= s(add(n, m)) \end{aligned}$$

Streng genommen müssten wir schreiben:

$$\begin{aligned} add(0, m) &= g(m) \\ add(n+1, m) &= h(add(n, m), n, m) \end{aligned}$$

wobei $g = \pi_1^1$ und $h : \mathbb{N}^3 \to \mathbb{N}$ folgende primitiv rekursive Funktion ist:

$$h(x, y, z) = s(\pi_1^3(x, y, z))$$

Im folgenden werden wir aber meistens nicht so genau sein.

Primitiv rekursive und μ-rekursive Funktionen

Multiplikationsfunktion

Die Funktion $mult\colon \mathbb{N}^2 \to \mathbb{N}$ mit $mult(n, m) = n \cdot m$ ist primitiv rekursiv.

$$\begin{aligned} mult(0, m) &= 0 \\ mult(n+1, m) &= add(mult(n, m), m) \end{aligned}$$

Statt $mult(0, m) = 0$ müssten wir streng genommen $mult(0, m) = k_0()$ schreiben, was erlaubt ist, da wir bei der Komposition auf Folie 57 den Fall $k = 0$ erlauben.

Primitiv rekursive und μ-rekursive Funktionen

Dekrementierung

Die Funktion $dec\colon \mathbb{N} \to \mathbb{N}$ mit $dec(n) = n - 1$ ist primitiv rekursiv.

$$\begin{aligned} dec(0) &= 0 \\ dec(n+1) &= n \end{aligned}$$

Subtraktion

Die Funktion $sub : \mathbb{N}^2 \to \mathbb{N}$ mit $sub(n, m) = \max\{0, n - m\}$ ist primitiv rekursiv.

$$\begin{aligned} sub(n, 0) &= n \\ sub(n, m+1) &= dec(sub(n, m)) \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Erinnerung: Der Binomialkoeffizient $\binom{n}{k}$ ist für $n \geq 0, k \geq 1$ definiert durch

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k \cdot (k-1) \cdots 1}.$$

Lemma 6

Die einstellige Funktion $n \mapsto \binom{n}{2} = \frac{(n-1)n}{2}$ is primitiv rekursiv.

$$\binom{0}{2} = 0$$

$$\binom{n+1}{2} = \frac{n(n+1)}{2} = \frac{(n-1)n}{2} + n = \binom{n}{2} + n$$

Durch Komposition folgt, dass auch die folgende Abbildung $c : \mathbb{N}^2 \to \mathbb{N}$ primitiv rekursiv ist:

$$c(n,m) = n + \binom{n+m+1}{2}$$

Primitiv rekursive und μ-rekursive Funktionen

Die Abbildung c ist eine Bijektion von $\mathbb{N}^2$ nach $\mathbb{N}$ (Paarungsfunktion).

	$n = 0$	1	2	3	4
$m = 0$	0	2	5	9	14
1	1	4	8	13	19
2	3	7	12	18	25
3	6	11	17	24	32
4	10	16	23	31	40

Beachte: $c(n, m) = n + \sum_{i=1}^{n+m} i$.

Die Funktion c kann verwendet werden, um beliebige k-Tupel ($k \geq 2$) von natürlichen Zahlen durch eine Zahl zu kodieren:

$$\langle n_1, n_2, \ldots, n_k \rangle = c(n_1, c(n_2, \ldots, c(n_{k-1}, n_k) \ldots))$$

Die Abbildung $(n_1, n_2, \ldots, n_k) \mapsto \langle n_1, n_2, \ldots, n_k \rangle$ ist dann auch primitiv rekursiv (für jedes feste k).

Primitiv rekursive und μ-rekursive Funktionen

Es seien $\ell : \mathbb{N} \to \mathbb{N}$ und $r : \mathbb{N} \to \mathbb{N}$ die eindeutigen Funktionen mit:

$$\forall n, m \in \mathbb{N} : \ell(c(n, m)) = n \text{ und } r(c(n, m)) = m$$

Wir werden nun zeigen, dass die Funktionen ℓ und r ebenfalls primitiv rekursiv sind. Mit diesen lassen sich dann auch primitiv rekursive Dekodierfunktionen für kodierte k-Tupel definieren:

$$\begin{aligned}
d_1(n) &= \ell(n) \\
d_2(n) &= \ell(r(n)) \\
&\vdots \\
d_{k-1}(n) &= \ell(\underbrace{r(r(\cdots r}_{k-2 \text{ mal}}(n)\cdots)) \\
d_k(n) &= \underbrace{r(r(\cdots r}_{k-1 \text{ mal}}(n)\cdots))
\end{aligned}$$

Dann gilt: $d_i(\langle n_1, n_2, \ldots, n_k \rangle) = n_i$.

Primitiv rekursive und μ-rekursive Funktionen

Sei $P : \mathbb{N}^{k+1} \to \{0,1\}$ ein Prädikat (eine Funktion mit Wertebereich $\{0,1\}$). Dann wird die Funktion $q : \mathbb{N}^{k+1} \to \mathbb{N}$ mit

$$q(n, n_1, \ldots, n_k) = \begin{cases} 0 \text{ falls } P(x, n_1, \ldots, n_k) = 0 \text{ für alle } x \in \{0, \ldots, n\} \\ \max\{x \leq n \mid P(x, n_1, \ldots, n_k) = 1\} \text{ sonst} \end{cases}$$

durch Anwendung des beschränkten max-Operators auf P definiert.

Lemma 7

Wenn P primitiv rekursiv ist, dann ist auch q primitiv rekursiv.

$$\begin{aligned} q(0, n_1, \ldots, n_k) &= 0 \\ q(n+1, n_1, \ldots, n_k) &= \begin{cases} n+1 & \text{falls } P(n+1, n_1, \ldots, n_k) = 1 \\ q(n, n_1, \ldots, n_k) & \text{sonst} \end{cases} \\ &= q(n, n_1, \ldots, n_k) + \\ &\quad P(n+1, n_1, \ldots, n_k) * (n+1 - q(n, n_1, \ldots, n_k)) \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Sei $P : \mathbb{N}^{k+1} \to \{0,1\}$ ein Prädikat. Dann wird das Prädikat $Q : \mathbb{N}^{k+1} \to \{0,1\}$ mit

$$Q(n, n_1, \ldots, n_k) = \begin{cases} 1 & \text{falls } \exists x \leq n : P(x, n_1, \ldots, n_k) = 1 \\ 0 & \text{sonst} \end{cases}$$

durch Anwendung des beschränkten Existenzquantors auf P definiert.

Lemma 8

Wenn P primitiv rekursiv ist, dann ist auch Q primitiv rekursiv.

$$\begin{aligned} Q(0, n_1, \ldots, n_k) &= P(0, n_1, \ldots, n_k) \\ Q(n+1, n_1, \ldots, n_k) &= P(n+1, n_1, \ldots, n_k) + Q(n, n_1, \ldots, n_k) \\ &\quad - P(n+1, n_1, \ldots, n_k) * Q(n, n_1, \ldots, n_k) \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Nun können wir zeigen, dass auch die Umkehrfunktionen ℓ und r von $c : \mathbb{N}^2 \to \mathbb{N}$ primitiv rekursiv sind.

Das Prädikat $C : \mathbb{N}^3 \to \{0, 1\}$ mit

$$C(x, y, z) = \begin{cases} 1 & \text{falls } c(x, y) = z \\ 0 & \text{falls } c(x, y) \neq z \end{cases}$$

ist primitiv rekursiv:

$$C(x, y, z) = \Big(1 - (c(x, y) - z)\Big) * \Big(1 - (z - c(x, y))\Big).$$

Deshalb sind auch folgende Funktionen ℓ' und r' primitiv rekursiv:

$$\begin{aligned} \ell'(k, m, n) &= \max\{x \leq k \mid \exists y \leq m : C(x, y, n) = 1\} \\ r'(k, m, n) &= \max\{y \leq k \mid \exists x \leq m : C(x, y, n) = 1\} \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Schließlich gilt $\ell(n) = \ell'(n, n, n)$ und $r(n) = r'(n, n, n)$:

Beachte hierzu, dass $x \leq c(x, y)$ und $y \leq c(x, y)$ für alle $x, y \in \mathbb{N}$.

Weiter gilt:

$$
\begin{aligned}
& \ell'(c(x,y), c(x,y), c(x,y)) \\
= \; & \max\{x' \leq c(x,y) \mid \exists y' \leq c(x,y) : C(x', y', c(x,y)) = 1\} \\
= \; & \max\{x' \leq c(x,y) \mid \exists y' \leq c(x,y) : c(x', y') = c(x,y))\} \\
= \; & \max\{x' \leq c(x,y) \mid \exists y' \leq c(x,y) : x' = x \text{ und } y' = y\} \\
= \; & \max\{x' \leq c(x,y) \mid x' = x\} = x
\end{aligned}
$$

und ananlog $r'(c(x, y), c(x, y), c(x, y)) = y$.

Nach Definition der Funktionen ℓ und r (Folie 64) folgt hieraus $\ell(n) = \ell'(n, n, n)$ und $r(n) = r'(n, n, n)$.

Primitiv rekursive und μ-rekursive Funktionen

Satz 9 (Primitiv rekursiv $\Longleftrightarrow$ LOOP-berechenbar)

Die Klasse der primitiv rekursiven Funktionen stimmt genau mit der Klasse der LOOP-berechenbaren Funktionen überein.

Beweis: Sei zunächst $f : \mathbb{N}^r \to \mathbb{N}$ LOOP-berechenbar.

Also gibt es ein LOOP-Programm P, in dem nur die Variablen $x_1, \ldots, x_k$ ($k \geq r$) vorkommen, mit (siehe Folie 28)

$$\forall n_1, \ldots, n_r \in \mathbb{N} : f(n_1, \ldots, n_r) = \pi_1([P]_k(n_1, \ldots, n_r, 0, \ldots, 0)).$$

Durch Induktion über den Aufbau von P definieren wir eine primitiv rekursive Funktion $g_P : \mathbb{N} \to \mathbb{N}$ mit

$$\forall n_1, \ldots, n_k \in \mathbb{N} : g_P(\langle n_1, \ldots, n_k \rangle) = \langle [P]_k(n_1, \ldots, n_k) \rangle.$$

Wegen $f(n_1, \ldots, n_r) = d_1(g_P(\langle n_1, \ldots, n_r, 0, \ldots, 0 \rangle))$ ist dann auch f primitiv rekursiv.

Primitiv rekursive und μ-rekursive Funktionen

Fall 1. $P = (x_i := x_j \pm c)$: Definiere

$$g_P(n) := \langle d_1(n), \ldots, d_{i-1}(n), d_j(n) \pm c, d_{i+1}(n), \ldots, d_k(n) \rangle.$$

Dann gilt

$$\begin{aligned}
& g_P(\langle n_1, \ldots, n_k \rangle) \\
= \quad & \langle d_1(\langle n_1, \ldots, n_k \rangle), \ldots, d_{i-1}(\langle n_1, \ldots, n_k \rangle), d_j(\langle n_1, \ldots, n_k \rangle) \pm c, \\
& \quad d_{i+1}(\langle n_1, \ldots, n_k \rangle), \ldots, d_k(\langle n_1, \ldots, n_k \rangle) \rangle \\
= \quad & \langle n_1, \ldots, n_{i-1}, n_j \pm c, n_{i+1}, \ldots, n_k \rangle \\
\overset{\text{Folie 26}}{=} \quad & \langle [P]_k(n_1, \ldots, n_k) \rangle.
\end{aligned}$$

Außerdem: Da alle Funktionen $d_1, \ldots, d_k$ sowie $n \mapsto n \pm c$ und $(n_1, \ldots, n_k) \mapsto \langle n_1, \ldots, n_k \rangle$ primitiv rekursiv sind, ist nach obiger Definition auch g_P primitiv rekursiv.

Primitiv rekursive und μ-rekursive Funktionen

Fall 2. $P = (Q; R)$: Definiere

$$g_P(n) := g_R(g_Q(n)).$$

Nach Induktion gilt für alle $n_1, \ldots, n_k \in \mathbb{N}$:

$$\begin{aligned} g_Q(\langle n_1, \ldots, n_k \rangle) &= \langle [Q]_k(n_1, \ldots, n_k) \rangle \\ g_R(\langle n_1, \ldots, n_k \rangle) &= \langle [R]_k(n_1, \ldots, n_k) \rangle \end{aligned}$$

Also gilt

$$\begin{aligned} g_P(\langle n_1, \ldots, n_k \rangle) &= g_R(g_Q(\langle n_1, \ldots, n_k \rangle)) \\ &= g_R(\langle [Q]_k(n_1, \ldots, n_k) \rangle) \\ &= \langle [R]_k([Q]_k(n_1, \ldots, n_k)) \rangle \\ &\overset{\text{Folie 26}}{=} \langle [P]_k(n_1, \ldots, n_k) \rangle. \end{aligned}$$

Außerdem: Da g_Q und g_R nach Induktion primitiv rekursiv sind, ist nach obiger Definition auch g_P primitiv rekursiv.

Primitiv rekursive und μ-rekursive Funktionen

Fall 3. $P = (\text{LOOP } x_i \text{ DO } Q \text{ END})$:

Definiere zunächst die primitiv rekursive Funktion h durch

$$\begin{aligned} h(0, m) &= m \\ h(n+1, m) &= g_Q(h(n, m)) \end{aligned}$$

Dann gilt also $h(n, m) = g_Q^n(m)$ (n-fache Anwendung von g_Q auf m).

Definiere schließlich $g_P(x) = h(d_i(x), x)$.

Dann gilt:

$$\begin{aligned} g_P(\langle n_1, \ldots, n_k \rangle) &= h(d_i(\langle n_1, \ldots, n_k \rangle), \langle n_1, \ldots, n_k \rangle) \\ &= h(n_i, \langle n_1, \ldots, n_k \rangle) \\ &= g_Q^{n_i}(\langle n_1, \ldots, n_k \rangle) \\ &\overset{\text{Ind.hyp.}}{=} \langle [Q]_k^{n_i}(n_1, \ldots, n_k) \rangle \\ &\overset{\text{Folie 26}}{=} \langle [P]_k(n_1, \ldots, n_k) \rangle. \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Sei nun $f : \mathbb{N}^r \to \mathbb{N}$ primitiv rekursiv.

Durch Induktion über den Aufbau von f zeigen wir, dass f LOOP-berechenbar ist.

Fall 1: f ist eine der Basisfunktionen (konstante Funktionen, Projektionen, Nachfolgerfunktion).

Dann ist f offensichtlich LOOP-berechenbar.

Fall 2: Es gibt primitiv rekursive Funktionen $g, f_1, \ldots, f_k$ mit

$$f(n_1, \ldots, n_r) = g(f_1(n_1, \ldots, n_r), \ldots, f_k(n_1, \ldots, n_r)).$$

Nach Induktion sind $g, f_1, \ldots, f_k$ LOOP-berechenbar.

Seien $G, F_1, \ldots, F_k$ LOOP-Programme zur Berechnung von $g, f_1, \ldots, f_k$.

Dann berechnet das folgende LOOP-Programm f:

Primitiv rekursive und μ-rekursive Funktionen

$y_1 := x_1; \ldots; y_r := x_r;$
$F_1;$
$z_1 := x_1;$
$x_1 := y_1; \ldots; x_r := y_r;$
$F_2;$
$z_2 := x_1;$
$x_1 := y_1; \ldots; x_r := y_r;$
$F_3;$
$z_3 := x_1;$
$\vdots$
$x_1 := y_1; \ldots; x_r := y_r;$
$F_k;$
$z_k := x_1;$
$x_1 := z_1; \ldots; x_k := z_k;$
G

Primitiv rekursive und μ-rekursive Funktionen

Erläuterung:

- Die Eingabezahlen $n_1, \ldots, n_r$ stehen zu Beginn in den Variablen $x_1, \ldots, x_r$. Diese sichern wir mittels $y_1 := x_1; \ldots; y_r := x_r$ in den Variablen $y_1, \ldots, y_r$.
- Mittels des Programms F_i wird $f_i(n_1, \ldots, n_r)$ berechnet und dieser Wert befindet sich nach Ablauf von F_i in der Variablen x_1.
- Mittels $z_i := x_1$ sichern wir den Wert $f_i(n_1, \ldots, n_r)$ in der Variablen z_i.
- Bevor F_{i+1} gestartet wird, müssen mittels $x_1 := y_1; \ldots; x_r := y_r$ die Variablen $x_1, \ldots, x_r$ wieder auf die Eingabezahlen $n_1, \ldots, n_r$ setzen.
- Schließlich hat jede Variablen z_i $(1 \leq i \leq k)$ den Wert $f_i(n_1, \ldots, n_r)$.
- Mittels $x_1 := z_1; \ldots; x_k := z_k$ werden diese Werte nun in die Variablen $x_1, \ldots, x_k$ kopiert. Danach wird G gestartet.
- Nach Ablauf von G befindet sich in der Variablen x_1 der gesuchte Wert $g(f_1(n_1, \ldots, n_r), \ldots, f_k(n_1, \ldots, n_r)) = f(n_1, \ldots, n_r)$.

Primitiv rekursive und μ-rekursive Funktionen

Fall 3: f entsteht aus g und h durch primitive Rekursion.

Es gibt also primitiv rekursive Funktionen $g\colon \mathbb{N}^{r-1} \to \mathbb{N}$ und $h\colon \mathbb{N}^{r+1} \to \mathbb{N}$ mit

$$\begin{aligned} f(0, n_2, \ldots, n_r) &= g(n_2, \ldots, n_r) \\ f(n_1 + 1, n_2, \ldots, n_r) &= h(f(n_1, n_2, \ldots, n_r), n_1, n_2, \ldots, n_r) \end{aligned}$$

Die Funktion f lässt sich dann durch das folgende (Pseudocode-) LOOP-Programm berechnen:

$y := g(x_2, \ldots, x_r);\ k := 0;$
LOOP x_1 DO
    $y := h(y, k, x_2, \ldots, x_r); k := k + 1$
END

Unter Verwendung von LOOP-Programmen für g und h sowie Zwischenspeicherung ähnlich zu Fall 2 kann dieser Pseudocode in ein LOOP-Programm für f umgesetzt werden. □

Primitiv rekursive und μ-rekursive Funktionen

Wir definieren nun ein weitere Klasse, die gleichmächtig zu WHILE-Programmen, GOTO-Programmen und Turingmaschinen ist.

μ-rekursive Funktionen (Definition)

Die μ-rekursiven Funktionen verwenden die gleichen Basisfunktionen (konstante Funktionen, Projektionen, Nachfolgerfunktion) und Operatoren (Einsetzung, primitive Rekursion) wie primitiv rekursive Funktionen.

Zusätzlich darf noch der μ-Operator verwendet werden.

Der μ-Operator verwandelt eine Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ in eine Funktion $\mu f : \mathbb{N}^k \to \mathbb{N}$ mit

$$\mu f(n_1, \ldots, n_k) = \min\{n \mid f(n, n_1, \ldots, n_k) = 0 \text{ und } \forall m < n : f(m, n_1, \ldots, n_k) \text{ ist definiert}\}$$

Dabei gilt $\min \emptyset =$ *undefiniert*.

Primitiv rekursive und μ-rekursive Funktionen

Intuitive Berechnung von $\mu f(n_1, \ldots, n_k)$:

- Berechne $f(0, n_1, \ldots, n_k)$, $f(1, n_1, \ldots, n_k)$, $\ldots$
- Falls für ein n gilt $f(n, n_1, \ldots, n_k) = 0$, so gib n als Funktionswert zurück.
- Falls $f(m, n_1, \ldots, n_k)$ undefiniert ist (ohne, dass vorher einmal der Funktionswert gleich 0 war), oder der Funktionswert 0 nie erreicht wird: die intuitive Berechnung terminiert nicht.
  In diesem Fall gilt auch $\mu f(n_1, \ldots, n_k) = \min \emptyset = \textit{undefiniert}$.

Analogie zu WHILE-Programmen: es ist nicht klar, ob die Abbruchbedingung jemals erfüllt wird.

Primitiv rekursive und μ-rekursive Funktionen

Durch den μ-Operator können nun auch echt partielle Funktionen erzeugt werden.

Überall undefinierte Funktion

Die Funktion $\Omega\colon \mathbb{N} \to \mathbb{N}$ mit $\Omega(n) = \textit{undefiniert}$ für alle $n \in \mathbb{N}$ ist μ-rekursiv.

Verwende z.B. die 2-stellige Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(n, m) = n + 1$ für alle n, m.

Es gilt $f(n, m) = s(\pi_1^2(n, m))$, wobei s die Nachfolgerfunktion ist.

Dann gilt $\Omega = \mu f$.

Primitiv rekursive und μ-rekursive Funktionen

Weiteres Beispiel:

Wurzelfunktion

Die Funktion $sqrt : \mathbb{N} \to \mathbb{N}$ mit $sqrt(n) = \lceil \sqrt{n} \rceil$ ist μ-rekursiv.

Dabei rundet $\lceil \dots \rceil$ eine reelle Zahl auf die nächstgrößere (oder gleiche) ganze Zahl auf.

Sei $f(m, n) = n - m \cdot m$. (beachte: die Multiplikationsfunktion und Subtraktionsfunktion sind primitiv rekursiv).

Dann gilt $sqrt = \mu f$, denn: $\mu f(n) = \min\{m \in \mathbb{N} \mid n - m \cdot m = 0\} = \lceil \sqrt{n} \rceil$

Diese Funktion ist jedoch auch primitiv rekursiv.

Intuition: bei Berechnung von $sqrt(n)$ sind immer höchstens n Iterationen notwendig.

Primitiv rekursive und μ-rekursive Funktionen

Satz 10

Die Klasse der μ-rekursiven Funktionen stimmt genau mit der Klasse der WHILE-(GOTO-, Turing-)berechenbaren Funktionen überein.

Beweis:

Wir zeigen, dass die Klasse der μ-rekursiven Funktionen mit der Klasse der WHILE-berechenbaren Funktionen übereinstimmt.

Hierfür genügt es den Beweis für den Satz von Folie 69 (primitiv rekursive Funktionen $=$ LOOP-berechenbare Funktionen) um den μ-Operator sowie die WHILE-Schleife zu erweitern.

Sei zunächst $P = (\text{WHILE } x_i \neq 0 \text{ DO } Q \text{ End})$ ein WHILE-Programm.

Wir müssen zeigen, dass die auf Folie 69 definierte Funktion $g_P : \mathbb{N} \to \mathbb{N}$ μ-rekursiv ist.

Nach Induktion ist dies für g_Q bereits der Fall.

Primitiv rekursive und μ-rekursive Funktionen

Wie im Beweis des Satzes von Folie 69 (Fall 3 auf Folie 72) können wir eine μ-rekursive Funktion $h : \mathbb{N}^2 \to \mathbb{N}$ mit $h(n, m) = g_Q^n(m)$ definieren.

Definiere $j : \mathbb{N}^2 \to \mathbb{N}$ durch $j(n, m) = d_i(h(n, m))$ und

$$g_P(x) := h((\mu j)(x), x).$$

Beachte: $(\mu j)(\langle n_1, \ldots, n_k \rangle)$ ist definiert genau dann, wenn eine Zahl t existiert, so dass

$$\begin{aligned} 0 &= d_i(h(t, \langle n_1, \ldots, n_k \rangle)) \\ &= d_i(g_Q^t(\langle n_1, \ldots, n_k \rangle)) \\ &= d_i(\langle [Q]_k^t(n_1, \ldots, n_k) \rangle) \\ &= \pi_i^k([Q]_k^t(n_1, \ldots, n_k)) \end{aligned}$$

und $[Q]_k^s(n_1, \ldots, n_k)$ für alle $s < t$ definiert ist.

Primitiv rekursive und μ-rekursive Funktionen

In diesem Fall ist $(\mu j)(\langle n_1, \ldots, n_k \rangle)$ die kleinste solche Zahl t.

Daher ist $(\mu j)(\langle n_1, \ldots, n_k \rangle)$ genau dann definiert, wenn das Programm $P = (\text{WHILE } x_i \neq 0 \text{ DO } Q \text{ End})$ bei Eingabe $n_1, \ldots, n_k$ terminiert.

Also: $g_P(\langle n_1, \ldots, n_k \rangle)$ definiert $\iff$ $\langle [P]_k(n_1, \ldots, n_k) \rangle$ definiert.

Ausserdem: Wenn $(\mu j)(\langle n_1, \ldots, n_k \rangle)$ definiert ist und gleich $t \in \mathbb{N}$ ist, so ist t die Anzahl der Durchläufe durch die WHILE-Schleife bei Eingabe $n_1, \ldots, n_k$.

In diesem Fall gilt:

$$\begin{aligned} g_P(\langle n_1, \ldots, n_k \rangle) &= h((\mu j)(\langle n_1, \ldots, n_k \rangle), \langle n_1, \ldots, n_k \rangle) \\ &= h(t, \langle n_1, \ldots, n_k \rangle) \\ &= g_Q^t(\langle n_1, \ldots, n_k \rangle) \\ &\overset{\text{Ind.hyp.}}{=} \langle [Q]_k^t(n_1, \ldots, n_k) \rangle \\ &\overset{\text{Folie 33}}{=} \langle [P]_k(n_1, \ldots, n_k) \rangle. \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Sei nun $f = \mu g : \mathbb{N}^r \to \mathbb{N}$ für eine μ-rekursive Funktion $g : \mathbb{N}^{r+1} \to \mathbb{N}$.

Dann kann f durch das folgende (Pseudocode-) WHILE-Programm berechnet werden:

$y := 0;\ z := g(0, x_1, \ldots, x_r);$
WHILE $z \neq 0$ DO
    $y := y + 1;\ z := g(y, x_1, \ldots, x_r);$
END;
$x_1 := y$

□

Primitiv rekursive und μ-rekursive Funktionen

Wir werden nun zeigen, dass es totale Turing-berechenbare/μ-rekursive Funktionen gibt, die nicht primitiv rekursiv sind.

Ein klassisches Beispiel hierfür ist die sogenannte Ackermannfunktion.

Ackermannfunktion $a\colon \mathbb{N}^2 \to \mathbb{N}$ (Ackermann 1928)

$$
\begin{aligned}
a(0,y) &= y+1 && (1)\\
a(x,0) &= a(x-1,1), \text{ falls } x>0 && (2)\\
a(x,y) &= a(x-1,a(x,y-1)), \text{ falls } x,y>0 && (3)
\end{aligned}
$$

Lemma 11

Die Ackermannfunktion ist eine totale Funktion.

Primitiv rekursive und μ-rekursive Funktionen

Beweis: Durch Induktion über das erste Argument x.

Seien $x, y \in \mathbb{N}$.

Falls $x = 0$, gilt $a(x, y) \overset{(1)}{=} y + 1$.

Falls $x > 0$, gilt

$$\begin{aligned} a(x, y) &\overset{(3)}{=} a(x-1, a(x, y-1)) \\ &\overset{(3)}{=} a(x-1, a(x-1, a(x, y-2))) = \cdots \\ &= \underbrace{a(x-1, a(x-1, \ldots a(x-1}_{(y+1)\text{-mal}}, 1) \ldots)). \end{aligned}$$

Da nach Induktion alle Werte $a(x-1, z)$ (für $z \in \mathbb{N}$) definiert sind, ist auch $a(x, y)$ definiert. □

Primitiv rekursive und μ-rekursive Funktionen

Wertetabelle der Ackermannfunktion für kleine Werte:

$y =$	0	1	2	3	4	...	$a(x, y)$
$x = 0$	1	2	3	4	5	...	$y + 1$
$x = 1$	2	3	4	5	6	...	$y + 2$
$x = 2$	3	5	7	9	11	...	$2y + 3$
$x = 3$	5	13	29	61	125	...	$2^{y+3} - 3$
$x = 4$	13	65533	$> 10^{19727}$				$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3 \text{ Zweier}} - 3$
...							

Satz 12

Die Ackermannfunktion ist WHILE-berechenbar, aber nicht primitiv rekursiv bzw. nicht LOOP-berechenbar.

Primitiv rekursive und μ-rekursive Funktionen

Beweis:

Wir zeigen zunächst, dass die Ackermannfunktion $\textsc{While}$-berechenbar ist (die Ackermannfunktion ist sicherlich berechenbar im intuitiven Sinne).

Hierfür ist es sinnvoll, Stacks (Keller) von natürlichen Zahlen einzuführen.

Eine Folge $(n_0, n_1, \ldots, n_k)$ von natürlichen Zahlen kann durch die Kodierungsfunktion $(n_0, n_1, \ldots, n_k) \mapsto \langle n_0, n_1, \ldots, n_k \rangle$ (siehe Folie 63) in einer Zahl abgespeichert werden.

Sei stack eine Integer-Variable, die einen Stack von natürlichen Zahlen repräsentiert. Wir definieren die folgenden Operationen:

- $\textsc{Init}$(stack): stack $:= 0$
- $\textsc{Push}(n, \text{stack})$ für $n \in \mathbb{N}$: stack $:= c(n+1, \text{stack})$
- $x :=$ $\textsc{Pop}$(stack): $x := \ell(\text{stack}) - 1$; stack $:= r(\text{stack})$

Ausserdem verwenden wir size(stack) $\neq 1$ als Abkürzung für $r(\text{stack}) \neq 0$.

Primitiv rekursive und μ-rekursive Funktionen

Beachte: Bei einer PUSH-Operation wird das auf den Keller zu legende Argument um 1 inkrementiert, bei einer POP-Operation wird es dann wieder dekrementiert.

Dies ist notwendig, um diese Argumente von dem untersten Kellersymbol 0 zu unterscheiden.

Würden wir diese Inkrementierung nicht vornehmen, so würden alle Keller der Form $(0, 0, \ldots, 0)$ durch die Zahl 0 kodiert werden.

Mit diesen Operationen kann die Ackermannfunktion durch das folgende WHILE-Programm berechnet werden:

Primitiv rekursive und μ-rekursive Funktionen

```
INIT(stack);
PUSH(x_1, stack);
PUSH(x_2, stack);
WHILE size(stack) ≠ 1 DO
    y := POP(stack);
    x := POP(stack);
    IF x = 0 THEN PUSH(y + 1, stack);
    ELSEIF y = 0 THEN PUSH(x − 1, stack); PUSH(1, stack);
    ELSE PUSH(x − 1, stack); PUSH(x, stack); PUSH(y − 1, stack);
    END (if)
END (while)
x_1 := POP(stack);
```

Wir zeigen nun, dass die Ackermannfunktion stärker als jede primitiv rekursive Funktion wächst.

Hierfür beweisen wir eine Reihe von Lemmata.

Primitiv rekursive und μ-rekursive Funktionen

Lemma 13

$\forall x, y \in \mathbb{N} : y < a(x, y)$

Beweis: Induktion über x.

IA: $x = 0$.

Es gilt $y < y + 1 \overset{(1)}{=} a(0, y)$ für alle $y \in \mathbb{N}$.

IS: Gelte $\forall y \in \mathbb{N} : y < a(x, y)$ (IH 1).

Wir zeigen nun durch Induktion über $y \in \mathbb{N}$, dass $\forall y \in \mathbb{N} : y < a(x+1, y)$.

IA: $y = 0$.

Es gilt $1 < a(x, 1)$ (nach (IH 1)) und damit $0 < 1 < a(x, 1) \overset{(2)}{=} a(x + 1, 0)$.

IS: Gelte $y < a(x + 1, y)$ (IH 2).

Wir zeigen nun $y + 1 < a(x + 1, y + 1)$.

Primitiv rekursive und μ-rekursive Funktionen

Zunächst gilt nach (IH 1): $a(x+1,y) < a(x,a(x+1,y))$
(ersetze y in (IH 1) durch $a(x+1,y)$).

Also gilt: $y+1 \overset{\text{(IH 2)}}{\leq} a(x+1,y) < a(x,a(x+1,y)) \overset{(3)}{=} a(x+1,y+1)$. □

Lemma 14

$\forall x,y \in \mathbb{N} : a(x,y) < a(x,y+1)$

Beweis:

Für $x = 0$ gilt $a(0,y) \overset{(1)}{=} y+1 < y+2 \overset{(1)}{=} a(0,y+1)$ für alle $y \in \mathbb{N}$.

Für $x > 0$ gilt $a(x,y) < a(x-1,a(x,y))$ nach Lemma 13
(ersetze in Lemma 13 x durch $x-1$ und y durch $a(x,y)$).

Dies ergibt $a(x,y) < a(x-1,a(x,y)) \overset{(3)}{=} a(x,y+1)$. □

Primitiv rekursive und μ-rekursive Funktionen

Lemma 15

$\forall x, y \in \mathbb{N} : a(x, y+1) \leq a(x+1, y)$

Beweis: Induktion über y.

IA: $y = 0$.

Es gilt $a(x, 1) \overset{(2)}{=} a(x+1, 0)$ für alle $x \in \mathbb{N}$.

IS: Gelte $\forall x \in \mathbb{N} : a(x, y+1) \leq a(x+1, y)$ (IH).

Wir zeigen $a(x, y+2) \leq a(x+1, y+1)$ für alle $x \in \mathbb{N}$.

Nach Lemma 13 gilt $y + 1 < a(x, y+1)$.

Also gilt $y + 2 \leq a(x, y+1) \overset{\text{(IH)}}{\leq} a(x+1, y)$.

Lemma 14 impliziert $a(x, y+2) \leq a(x, a(x+1, y))$.

Also ergibt sich $a(x, y+2) \leq a(x, a(x+1, y)) \overset{(3)}{=} a(x+1, y+1)$. □

Primitiv rekursive und μ-rekursive Funktionen

Lemma 16

$\forall x, y \in \mathbb{N} : a(x, y) < a(x + 1, y)$

Beweis:

Lemma 14 $\rightsquigarrow$ $a(x, y) < a(x, y + 1)$.

Lemma 15 $\rightsquigarrow$ $a(x, y + 1) \leq a(x + 1, y)$. □

Aus Lemma 14 und Lemma 16 folgt, dass die Funktion a monoton ist:

Wenn $x \leq x'$ und $y \leq y'$ dann $a(x, y) \leq a(x', y')$. Gilt ausserdem noch $x < x'$ oder $y < y'$ so folgt $a(x, y) < a(x', y')$.

Primitiv rekursive und μ-rekursive Funktionen

Sei nun P ein LOOP Programm, in dem keine Variable x_i mit $i > k$ vorkommt.

Für ein Tupel $(n_1, \ldots, n_k) \in \mathbb{N}^k$ schreiben wir im folgenden

$$\sum(n_1, \ldots, n_k) = n_1 + \cdots + n_k.$$

Dann definieren wir

$$f_P(n) = \max\{\sum[P]_k(n_1, \ldots, n_k) \mid n_1, \ldots, n_k \in \mathbb{N}, \sum(n_1, \ldots, n_k) \leq n\}.$$

Lemma 17

Für jedes LOOP Programm P gibt es eine Zahl ℓ, so dass für alle $n \in \mathbb{N}$ gilt: $f_P(n) < a(\ell, n)$.

Beweis: Induktion über den Aufbau von P.

Primitiv rekursive und μ-rekursive Funktionen

O.B.d.A. erfülle P die folgenden Eigenschaften:

- Für jedes Teilprogramm $x_i := x_j \pm c$ in P gilt $c = 1$.

  Z. B. kann $x_i := x_j + 2$ ersetzt werden durch $x_i := x_j + 1; x_i := x_i + 1$.

- Für jedes Teilprogramm $\text{LOOP } x_i \text{ DO } Q \text{ END}$ in P gilt:
  x_i kommt in Q nicht vor.

  Sollte x_i in Q vorkommen, so ersetzen wir $\text{LOOP } x_i \text{ DO } Q \text{ END}$ durch $y := x_i;\ \text{LOOP } y \text{ DO } Q \text{ END}$, wobei y eine neue Variable ist.

Fall 1: $P = (x_i := x_j \pm 1)$

Dann gilt $f_P(n) \leq 2n + 1$.

Mit Induktion über y kann man leicht zeigen:
$a(1, y) = y + 2$ und $a(2, y) = 2y + 3$ (Übung).

Also gilt $f_P(n) < a(2, n)$.

Primitiv rekursive und μ-rekursive Funktionen

Fall 2: $P = (P_1; P_2)$.

Nach Induktionsvoraussetzung gibt es Zahlen ℓ_1 und ℓ_2 mit

$$\forall n \in \mathbb{N} : f_{P_1}(n) < a(\ell_1, n) \text{ und } f_{P_2}(n) < a(\ell_2, n). \tag{4}$$

Wir zeigen zunächst $f_P(n) \leq f_{P_2}(f_{P_1}(n))$.

Nach Definition von $f_P(n)$ gibt es ein Tupel $(n_1, \ldots, n_k) \in \mathbb{N}^k$ mit $\sum(n_1, \ldots, n_k) \leq n$ und

$$f_P(n) = \sum [P]_k(n_1, \ldots, n_k) = \sum [P_2]_k([P_1]_k(n_1, \ldots, n_k)).$$

Nun gilt nach Definition von $f_{P_1}(n)$: $\sum [P_1]_k(n_1, \ldots, n_k) \leq f_{P_1}(n)$.

Mit der Definition von $f_{P_2}(n)$ folgt schließlich:

$$f_P(n) = \sum [P_2]_k([P_1]_k(n_1, \ldots, n_k)) \leq f_{P_2}(f_{P_1}(n)).$$

Primitiv rekursive und μ-rekursive Funktionen

Mit $\ell_3 := \max\{\ell_1 - 1, \ell_2\}$ folgt nun:

$$\begin{aligned}
f_P(n) &\leq f_{P_2}(f_{P_1}(n)) \\
&< a(\ell_2, a(\ell_1, n)) && \text{((4) und Monotonie von } a\text{)} \\
&\leq a(\ell_3, a(\ell_3 + 1, n)) && \text{(Monotonie von } a\text{)} \\
&= a(\ell_3 + 1, n + 1) && \text{(Definition von } a\text{)} \\
&\leq a(\ell_3 + 2, n) && \text{(Lemma 15)}
\end{aligned}$$

Wir können somit $\ell = \ell_3 + 2$ in Lemma 17 wählen.

Fall 3: $P = (\textsc{Loop}\ x_i\ \textsc{Do}\ Q\ \textsc{End})$

Nach Induktionsvoraussetzung gibt es eine Zahl ℓ_1 mit

$$\forall n \in \mathbb{N} : f_Q(n) < a(\ell_1, n). \tag{5}$$

Beachte: x_i kommt in Q nicht vor.

Primitiv rekursive und μ-rekursive Funktionen

Wähle $m, n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k \leq n$ so, dass gilt:

$$\begin{aligned} f_P(n) &= \max\{\sum[P]_k(n_1, \ldots, n_k) \mid n_1, \ldots, n_k \in \mathbb{N}, \sum(n_1, \ldots, n_k) \leq n\} \\ &= \sum[P]_k(n_1, \ldots, n_{i-1}, m, n_{i+1}, \ldots, n_k), \end{aligned}$$

wobei $n_1 + \cdots + n_{i-1} + n_{i+1} + \cdots + n_k + m \leq n$.

Falls $m = 0$ gilt, so ergibt sich

$$\begin{aligned} f_P(n) &= \sum[P]_k(n_1, \ldots, n_{i-1}, 0, n_{i+1}, \ldots, n_k) \\ &= \sum(n_1, \ldots, n_{i-1}, 0, n_{i+1}, \ldots, n_k) \\ &\leq n \\ &< n + 1 \\ &= a(0, n). \end{aligned}$$

Primitiv rekursive und μ-rekursive Funktionen

Falls $m \geq 1$, so erhalten wir (da x_i in Q nicht vorkommt):

$$
\begin{aligned}
f_P(n) &= \sum [P]_k(n_1, \ldots, n_{i-1}, m, n_{i+1}, \ldots, n_k) \\
&= \sum [Q]_k^m(n_1, \ldots, n_{i-1}, m, n_{i+1}, \ldots, n_k) \\
&\leq \underbrace{f_Q(f_Q(\cdots f_Q}_{m\text{-mal}}(n-m)\cdots)) + m \\
&< a(\ell_1, \underbrace{f_Q(f_Q(\cdots f_Q}_{m-1\text{-mal}}(n-m)\cdots))) + m \\
&\vdots \\
&< \underbrace{a(\ell_1, a(\ell_1, \cdots a(\ell_1}_{m\text{-mal}}, n-m)\cdots)) + m
\end{aligned}
$$

Primitiv rekursive und μ-rekursive Funktionen

Da in dieser Abschätzung m-mal "$<$" vorkommt, erhalten wir:

$$
\begin{aligned}
f_P(n) &\leq \underbrace{a(\ell_1, a(\ell_1, \cdots a(\ell_1,}_{m\text{-mal}} n-m)\cdots)) \\
&< \underbrace{a(\ell_1, \cdots a(\ell_1,}_{m-1\text{-mal}} a(\ell_1+1, n-m))\cdots) && \text{(Monotonie von } a\text{, } m \geq 1) \\
&= a(\ell_1+1, n-1) && \text{(Definition von } a) \\
&< a(\ell_1+1, n) && \text{(Monotonie von } a)
\end{aligned}
$$

Wir können somit $\ell = \ell_1 + 1$ in Lemma 17 wählen. □

Primitiv rekursive und μ-rekursive Funktionen

Wir können nun den Beweis von Satz 12 endlich beenden.

Angenommen die Ackermannfunktion a wäre LOOP-berechenbar.

Dann ist auch die Funktion $g : \mathbb{N} \to \mathbb{N}$ mit $g(n) = a(n, n)$ LOOP-berechenbar.

Sei P ein LOOP-Programm mit

$$\forall n \in \mathbb{N} : g(n) = \pi_0([P]_k(n, 0, \ldots, 0)).$$

Nach Lemma 17 existiert eine Konstante ℓ mit

$$\forall n \in \mathbb{N} : f_P(n) < a(\ell, n).$$

Für $n = \ell$ folgt dann

$$g(\ell) = \pi_0([P]_k(\ell, 0, \ldots, 0)) \leq f_P(\ell) < a(\ell, \ell) = g(\ell).$$

Dies ist ein Widerspruch. □

Unentscheidbarkeit

Überblick

- Zunächst einmal definieren wir formal den Begriff Entscheidbarkeit.
  Was bedeutet es überhaupt, dass ein Problem entscheidbar ist?
- Dann kommen wir zum Begriff Semi-Entscheidbarkeit.
  Hier wird erlaubt, dass das Entscheidungsverfahren bei einer negativen Antwort nicht terminiert und keine Antwort liefert.
- Anschließend geht es um negative Resultate.
  Wie kann man zeigen, dass ein Problem nicht entscheidbar ist?

Entscheidbarkeit

Entscheidbarkeit (Definition)

Eine Sprache $L \subseteq \Sigma^*$ heißt entscheidbar, falls die charakteristische Funktion von L, d.h. die Funktion $\chi_L : \Sigma^* \to \{0, 1\}$ mit

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

berechenbar ist.

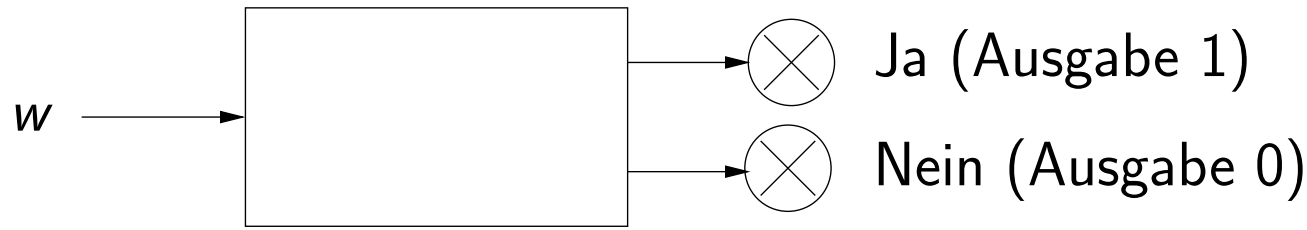
Eine Sprache, die nicht entscheidbar ist, heißt unentscheidbar.

Intuition: Es gibt ein algorithmisches Verfahren, welches bei Eingabe von $w \in \Sigma^*$ folgendes Verhalten zeigt:

- Wenn $w \in L$ gilt, dann terminiert das Verfahren nach endlich vielen Rechenschritten mit der Ausgabe 1 ("Ja, w gehört zu L").
- Wenn $w \notin L$ gilt, dann terminiert das Verfahren nach endlich vielen Rechenschritten mit der Ausgabe 0 ("Nein, w gehört nicht zu L").

Entscheidbarkeit

Darstellung der Entscheidbarkeit an einem Maschinenmodell:



Bei jeder Eingabe rechnet die Maschine endliche Zeit und gibt dann entweder “Ja” oder “Nein” aus.

Entscheidbarkeit

Bei Semi-Entscheidbarkeit erlaubt man, dass die charakteristische Funktion im negativen Fall undefiniert ist, d.h., keine Antwort zurückkommt. Bei konkreten Berechnungsmodellen bedeutet das dann Nicht-Terminierung.

Semi-Entscheidbarkeit (Definition)

Eine Sprache $L \subseteq \Sigma^*$ heißt semi-entscheidbar, falls die "halbe" charakteristische Funktion von L, d.h. die Funktion $\chi'_L : \Sigma^* \to \{1\}$ mit

$$\chi'_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ \textit{undefiniert} & \text{falls } w \notin L \end{cases}$$

berechenbar ist.

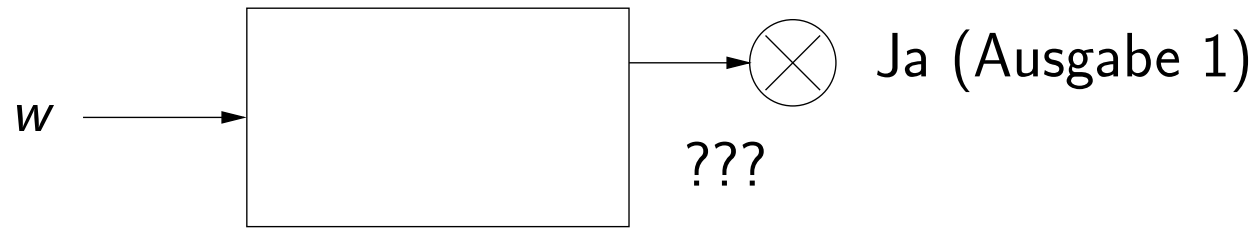
Entscheidbarkeit

Intuition: Es gibt ein algorithmisches Verfahren, welches bei Eingabe von $w \in \Sigma^*$ folgendes Verhalten zeigt:

- Wenn $w \in L$ gilt, dann terminiert das Verfahren nach endlich vielen Rechenschritten mit der Ausgabe 1 ("Ja, w gehört zu L").
- Wenn $w \notin L$ gilt, dann terminiert das Verfahren nicht

Entscheidbarkeit

Darstellung der Semi-Entscheidbarkeit an einem Maschinenmodell:



Bei jeder Eingabe rechnet die Maschine und gibt im Fall $w \in A$ nach endlicher Zeit "Ja" aus. Falls $w \notin A$ gilt, so terminiert die Maschine nie.

Das heißt, man kann sich nie sicher sein, ob nicht doch irgendwann "Ja" ausgegeben wird, da die Antwortzeit der Maschine nicht beschränkt ist.

Entscheidbarkeit

Satz 18 (Semi-Entscheidbarkeit und Chomsky-0-Sprachen)

Eine Sprache A ist semi-entscheidbar genau dann, wenn sie vom Typ 0 ist.

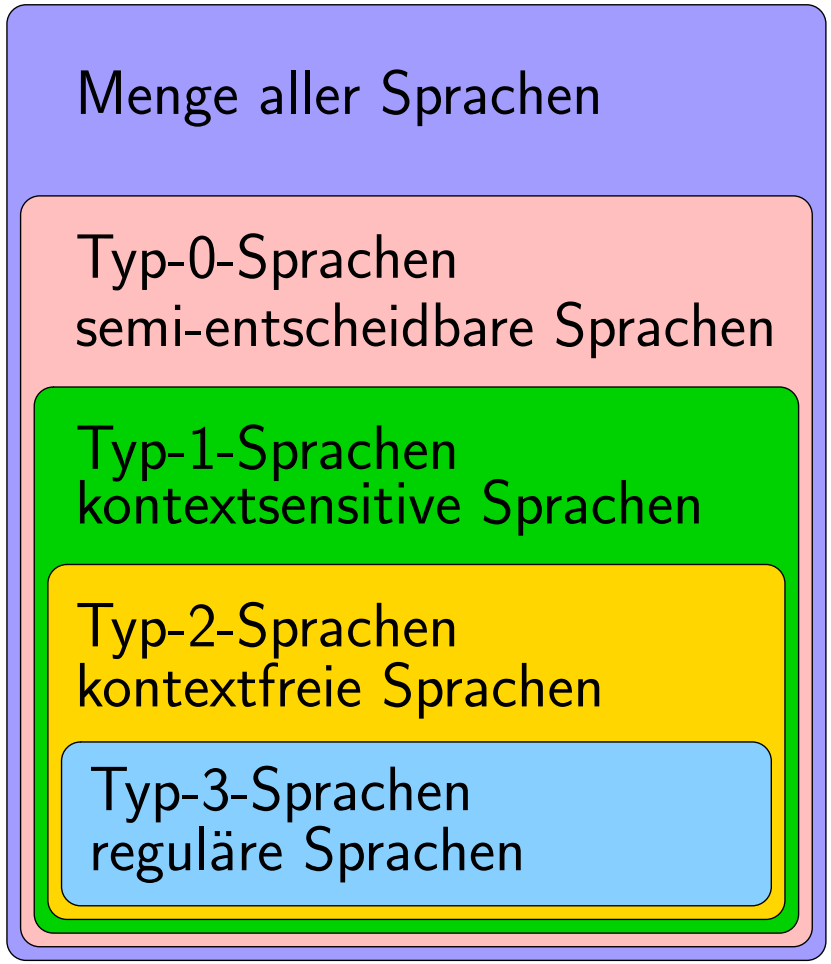
Beweis: Die Chomsky-0-Sprachen sind genau die Sprachen, die von einer Turing-Maschine akzeptiert werden, siehe FSA, Folie 348.

Eine Turing-Maschine, die die halbe charakteristische Funktion χ'_A berechnet, akzeptiert auch die Sprache A, da sie nach Schreiben der 1 in einen Endzustand übergeht.

Eine Turing-Maschine, die eine Sprache A akzeptiert, kann leicht in eine Turing-Maschine umgewandelt werden, die die "halbe" charakteristische Funktion χ'_A berechnet, indem sie nach Erreichen eines Endzustands das Band löscht und danach eine 1 schreibt. □

Entscheidbarkeit

Zur Erinnerung: die Chomsky-Hierarchie



Entscheidbarkeit

Bemerkungen:

- Im Zusammenhang mit Fragestellungen der Entscheidbarkeit werden Sprachen oft auch als Probleme bezeichnet.
- Auch wenn charakteristische Funktionen Wörter als Argumente haben, kann man sie leicht als Funktionen über natürlichen Zahlen auffassen und so mit WHILE- bzw. GOTO-Programmen berechnen. Jedes Wort aus Σ^* kann als Zahl zur Basis b aufgefasst werden, wobei $b \geq |\Sigma|$ (siehe auch die Umwandlung von Turing-Maschinen in GOTO-Programme auf Folie 47).
- Daher werden wir als Probleme im folgenden auch Teilmengen von $\mathbb{N}$ bzw. $\mathbb{N}^k$ betrachten.

Entscheidbarkeit

Typische Beispiele für Probleme:

Beispiel 1: das Wortproblem

Gegeben sei eine feste Chomsky-Grammatik G und das Problem sei $A = L(G)$. Wir wissen bereits, dass A entscheidbar ist, falls G eine Chomsky-1-Grammatik ist. Außerdem gibt es Grammatiken, für die $L(G)$ nicht entscheidbar ist (Beweis kommt bald).

Beispiel 2: das allgemeine Wortproblem

Das allgemeine Wortproblem ist die Menge

$$A = \{(w, G) \mid w \in L(G),\ G \text{ Chomsky-Grammatik über dem Alphabet } \Sigma\},$$

wobei die Paare (w, G) geeignet als Wörter kodiert werden müssen (z.B. durch Auflisten aller Produktionen von G, getrennt durch ein Symbol #).

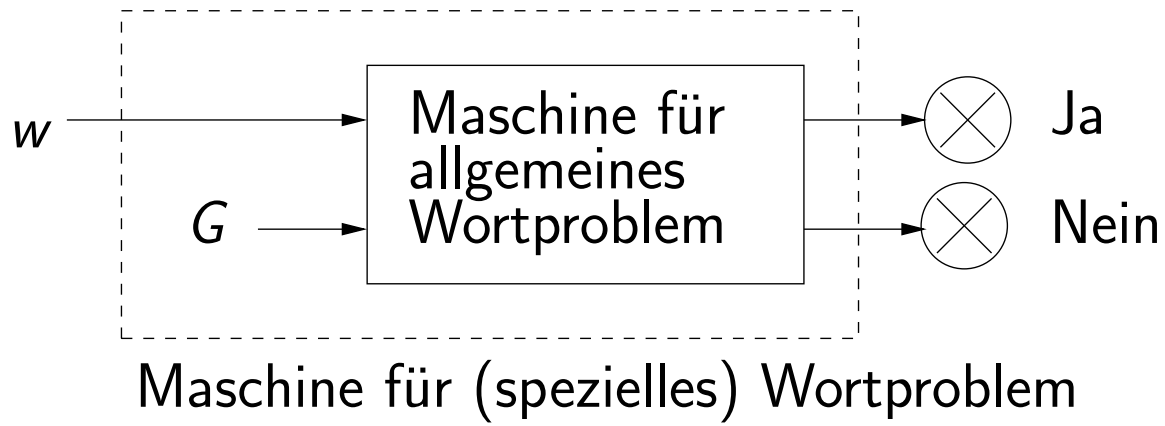
Entscheidbarkeit

Satz 19 (Unentscheidbarkeit des allgemeinen Wortproblems)

Das allgemeine Wortproblem ist unentscheidbar.

Beweis: Sei G eine Grammatik, für die das Wortproblem $L(G)$ unentscheidbar ist (Beweis kommt noch).

Falls das allgemeine Wortproblem A entscheidbar wäre, so wäre auch $L(G)$ entscheidbar. Für ein gegebenes Wort w müßte man dann nur überprüfen, ob $(w, G) \in A$ gilt. $\rightsquigarrow$ Widerspruch!



Entscheidbarkeit

Das heißt, aus einer Maschine zur Lösung des allgemeinen Wortproblems könnte man eine Maschine zur Lösung des (speziellen) Wortproblems bauen. Da es letztere aber nicht für alle Grammatiken G gibt, kann es auch erstere nicht geben.

Argumentationen dieser Art ("wenn es ein Verfahren für A gibt, dann kann man daraus ein Verfahren für B konstruieren") bezeichnet man als Reduktionen. Wir werden sie im folgenden häufiger anwenden.

Entscheidbarkeit

Beispiel 3: das Schnittproblem

Das Schnittproblem für kontextfreie Grammatiken ist die Menge

$$A = \{(G_1, G_2) \mid G_1, G_2 \text{ kontextfreie Grammatiken}, \\ L(G_1) \cap L(G_2) \neq \emptyset\}.$$

Das Schnittproblem ist unentscheidbar (noch ohne Beweis).

Intuitiv gesprochen: Es gibt kein algorithmisches Verfahren (z.B. ein C-Programm), welches als Eingabe zwei kontextfreie Grammatiken G_1 und G_2 als Eingabe bekommt und nach endlich vielen Schritten folgende Ausgabe liefert:

- 1, falls $L(G_1) \cap L(G_2) \neq \emptyset$,
- 0, falls $L(G_1) \cap L(G_2) = \emptyset$.

Entscheidbarkeit

Satz 20 (Entscheidbarkeit und Semi-Entscheidbarkeit)

Eine Sprache A ist entscheidbar, genau dann, wenn sowohl A als auch $\overline{A}$ (das Komplement von A) semi-entscheidbar sind.

Beweis:

Sei zunächst A entscheidbar.

Dann ist also die charakterische Funktion χ_A berechenbar mittels einer Turing-Maschine M.

Die folgende Turing-Maschine M_A berechnet die halbe charakteristische Funktion χ'_A:

- M_A simuliert die Maschine M.
- Falls M mit der Ausgabe 0 terminiert, geht jedoch M_A in eine Endlosschleife über.

Entscheidbarkeit

Die folgende Turing-Maschine $M_{\overline{A}}$ berechnet die halbe charakteristische Funktion $\chi'_{\overline{A}}$:

- $M_{\overline{A}}$ simuliert die Maschine M.
- Falls M mit der Ausgabe 1 terminiert, geht jedoch $M_{\overline{A}}$ in eine Endlosschleife über.
- Falls M mit der Ausgabe 0 terminiert, so terminiert $M_{\overline{A}}$ mit der Ausgabe 1.

Also sind sowohl A als auch $\overline{A}$ semi-entscheidbar.

Sei nun sowohl A als auch $\overline{A}$ semi-entscheidbar.

Sei M_A (bzw. $M_{\overline{A}}$) eine Turing-Maschine, die die halbe charakteristische Funktion χ'_A (bzw. $\chi'_{\overline{A}}$) berechnet.

Der folgende Algorithmus berechnet dann die charakteristische Funktion von A:

Entscheidbarkeit

INPUT w
$t := 1$;
WHILE true DO
  IF M_A terminiert bei Eingabe w nach t Schritten THEN
    OUTPUT(1)
  ELSEIF $M_{\overline{A}}$ terminiert bei Eingabe w nach t Schritten THEN
    OUTPUT(0)
  END
  $t := t + 1$
END

Beachte: Die WHILE-Schleife wird stets nach endlich vielen Schritten terminieren, da entweder $w \in A$ oder $w \notin A$ gilt.

Also gibt es eine Zahl t, so dass entweder M_A oder $M_{\overline{A}}$ bei Eingabe w nach t Schritten terminiert. □

Entscheidbarkeit

Rekursive Aufzählbarkeit (Definition)

Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv aufzählbar, falls $L = \emptyset$ oder es eine totale und berechenbare Funktion $f : \mathbb{N} \to \Sigma^*$ gibt mit

$$L = \{f(n) \mid n \in \mathbb{N}\} = \{f(1), f(2), f(3), \dots\}.$$

Bemerkungen:

- Sprechweise: die Funktion f zählt die Sprache L auf.
- Äquivalente Definition von rekursiver Aufzählbarkeit: es gibt eine totale, berechenbare und surjektive Funktion $f : \mathbb{N} \to L$.
- Der mathematische Begriff der Abzählbarkeit ist ganz ähnlich definiert. Nur fordert man dort nicht, dass f berechenbar ist. Daraus folgt:
  L rekursiv aufzählbar $\Rightarrow$ L abzählbar.

Entscheidbarkeit

Satz 21 (Rekursive Aufzählbarkeit und Semi-Entscheidbarkeit)

Eine Sprache L ist rekursiv aufzählbar, genau dann, wenn sie semi-entscheidbar ist.

Beweis:

Rekursive Aufzählbarkeit $\rightarrow$ Semi-Entscheidbarkeit:

Gegeben: rekursiv aufzählbare Sprache $L \subseteq \Sigma^*$, beschrieben durch eine berechenbare und totale Funktion $f : \mathbb{N} \rightarrow \Sigma^*$.

Gesucht: TM, die bestimmt, ob ein Wort $w \in \Sigma^*$ in L liegt.

Lösung: Berechne $f(1), f(2), f(3), \ldots$, solange bis $w = f(i)$ für ein i gilt. In diesem Fall gibt die TM 1 aus, ansonsten terminiert sie nicht.

Entscheidbarkeit

Semi-Entscheidbarkeit $\rightarrow$ Rekursive Aufzählbarkeit:

Gegeben: semi-entscheidbare Sprache $L \subseteq \Sigma^*$, beschrieben durch eine deterministische TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ mit $L = T(M)$.

Gesucht: TM M', die eine Funktion f mit $\{f(1), f(2), f(3), \dots\} = L$ berechnet.

Der Fall, dass L endlich ist, ist einfach: Falls $L = \{w_1, w_2, \dots, w_n\}$ gilt, ist die Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ mit $f(i) = w_i$ für $1 \leq i \leq n$ und $f(i) = w_n$ für alle $i > n$ berechenbar.

Sei nun L unendlich.

Erinnern Sie sich, wie wir in FSA (Folien 352 und 353) erfolgreiche Berechnungen von M durch Wörter über dem Alphabet $\Omega = Z \cup \Gamma \cup \{\#\}$ beschrieben haben.

Da M deterministisch ist, gibt es zu jedem Wort $w \in T(M)$ genau eine erfolgreiche Berechnung $c(w)$.

Entscheidbarkeit

M' arbeitet nun wie folgt:

INPUT: $i \in \mathbb{N}$
$c := \varepsilon$,
$j := 0$;
WHILE true DO
  $c :=$ das längenlexikographisch nach c kommende Wort aus Ω^*;
  IF c ist eine erfolgreiche Berechnung THEN
    $j := j + 1$
    IF $j = i$ THEN OUTPUT w falls $c = c(w)$
  END
END

Begründung der Korrektheit:

- Sei $c(w_1), c(w_2), \ldots$ die längenlexikographische Auflistung aller erfolgreichen Berechnungen von M.
- Dann berechnet M' die Funktion $i \mapsto w_i$. □

Entscheidbarkeit

Aus den Sätzen 7 aus FSA (Folie 348), 18 (Folie 109) und 21 (Folie 120) folgt, dass die folgenden Aussagen für eine Sprache L äquivalent sind.

Semi-Entscheidbarkeit und äquivalente Begriffe

- L ist semi-entscheidbar, d. h. χ'_L ist (Turing, WHILE-, GOTO-)berechenbar.
- L ist rekursiv aufzählbar
- L ist vom Typ 0.
- $L = T(M)$ für eine Turing-Maschine M.

Halteproblem/Kodierung von Turing-Maschinen

Unser Ziel ist es nun zu zeigen, dass das sogenannte Halteproblem unentscheidbar ist.

Halteproblem (informell)

- **Eingabe:** deterministische Turing-Maschine M mit Eingabe w.
- **Ausgabe:** Hält M auf w?

Dazu werden wir jedoch zunächst genauer definieren, wie eine Turing-Maschine kodiert werden kann, um als Eingabe einer berechenbaren (charakteristischen) Funktion verwendet zu werden.

Halteproblem/Kodierung von Turing-Maschinen

Ziel: Kodierung einer deterministischen Turing-Maschine

$$M = (Z, \{0,1\}, \Gamma, \delta, z_0, \square, E)$$

durch ein Wort über dem Alphabet $\{0,1\}$.

Ohne Beschränkung der Allgemeinheit können wir folgendes annehmen:

$$\begin{aligned} \Gamma &= \{0,1,\ldots,m\} \text{ wobei } \square = m \geq 2, \\ Z &= \{1,\ldots,n\} \text{ wobei } z_0 = 1 \text{ und} \\ E &= \{k+1,\ldots,n\} \end{aligned}$$

Im Folgenden bezeichnet 1^i das Wort $\underbrace{11\cdots1}_{i \text{ viele}}$.

Der wichtigste Teil von M ist die Überführungsfunktion δ. Diese kann folgendermaßen als Wort über dem Alphabet $\{0,1\}$ kodiert werden:

Halteproblem/Kodierung von Turing-Maschinen

Für alle $1 \leq i \leq k$ und $0 \leq j \leq m$ definieren wir das Wort $w_{i,j}$ wie folgt:

Sei $\delta(i,j) = (i',j',y)$. Dann ist

$$w_{i,j} = 1^i 0 1^j 0 1^{i'} 0 1^{j'} 0 1^{\text{code}(y)} 0 \in \{0,1\}^*.$$

$$\text{Dabei ist } \text{code}(y) = \begin{cases} 1 & \text{falls } y = L \\ 2 & \text{falls } y = R \\ 3 & \text{falls } y = N \end{cases}$$

Dann kann die deterministischen Turing-Maschine M durch das folgende Wort $\text{code}(M) \in \{0,1\}^*$ kodiert werden:

$$\text{code}(M) = 1^n 0 1^m 0 1^k 0 \prod_{1 \leq i \leq k} \prod_{0 \leq j \leq m} w_{i,j}$$

Halteproblem/Kodierung von Turing-Maschinen

Bemerkungen: Viele der Festlegungen bei unserer Kodierung von Turing-Maschinen sind hochgradig willkürlich. Es gibt viele andere Möglichkeiten, Turing-Maschinen zu kodieren. Wichtig ist hier nur, dass es eine mögliche Kodierung gibt, auf die wir uns festlegen.

Beachte: Nicht jedes Wort $w \in \{0,1\}^*$ ist der Code einer Turing-Maschine. Dennoch wollen wir im Folgenden jedem Wort $w \in \{0,1\}^*$ eine Turing-Maschine M_w zuordnen.

Zu diesem Zweck fixieren wir eine beliebige aber feste gewählte deterministische Turing-Maschine $\widehat{M}$ (die Default-Maschine) mit Eingabealphabet $\{0,1\}$. Dann sei

$$M_w := \begin{cases} M & \text{falls } \mathrm{code}(M) = w \\ \widehat{M} & \text{falls kein Turing-Maschine } M \text{ mit } \mathrm{code}(M) = w \text{ existiert} \end{cases}$$

Halteproblem/Kodierung von Turing-Maschinen

Damit können wir nun zwei verschiedene Varianten des Halteproblems definieren.

Halteproblem

Das (allgemeine) Halteproblem ist die Sprache

$$\begin{aligned} H &= \{w\#x \mid w, x \in \{0,1\}^*, M_w \text{ angesetzt auf } x \text{ hält}\} \\ &= \{w\#x \mid w, x \in \{0,1\}^*, x \in T(M_w)\} \end{aligned}$$

Spezielles Halteproblem

Das spezielle Halteproblem ist die Sprache

$$\begin{aligned} K &= \{w \in \{0,1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\} \\ &= \{w \in \{0,1\}^* \mid w \in T(M_w)\}. \end{aligned}$$

Universelle Turing-Maschine

Man beachte die Selbstbezüglichkeit in der Definition von K: Eine Turing-Maschine $M = M_w$ erhält als Eingabe ihre eigene Kodierung w.

Dies ist aber problemlos möglich.

Haben Sie z.B. ein C-Programm P geschrieben, welches eine Textdatei als Eingabe bekommt, so können Sie problemlos das Programm P auf seinen eigenen Programmtext anwenden.

Universelle Turing-Maschine

Einige der folgenden Ergebnisse beruhen darauf, dass es eine deterministische Turing-Maschine gibt, die eine andere Turing-Maschine simulieren kann, wenn deren Kodierung gegeben ist. So eine Turing-Maschine heißt auch universelle Turing-Maschine.

Universelle Turing-Maschine

Eine deterministische Turing-Maschine U heißt universelle Turing-Maschine, wenn sie sich bei Eingabe von $w\#x$ wie folgt verhält:

- Wenn M_w bei Eingabe x nicht hält, so hält U bei Eingabe $w\#x$ auch nicht.
- Wenn M_w bei Eingabe x mit der Ausgabe y hält, so hält U bei Eingabe $w\#x$ ebenfalls mit y.

Im folgenden sei U eine universelle Turing-Maschine.

Unentscheidbarkeit des Halteproblems

Satz 22 (Unentscheidbarkeit des Halteproblems)

Das spezielle Halteproblem K ist nicht entscheidbar.

Beweis:

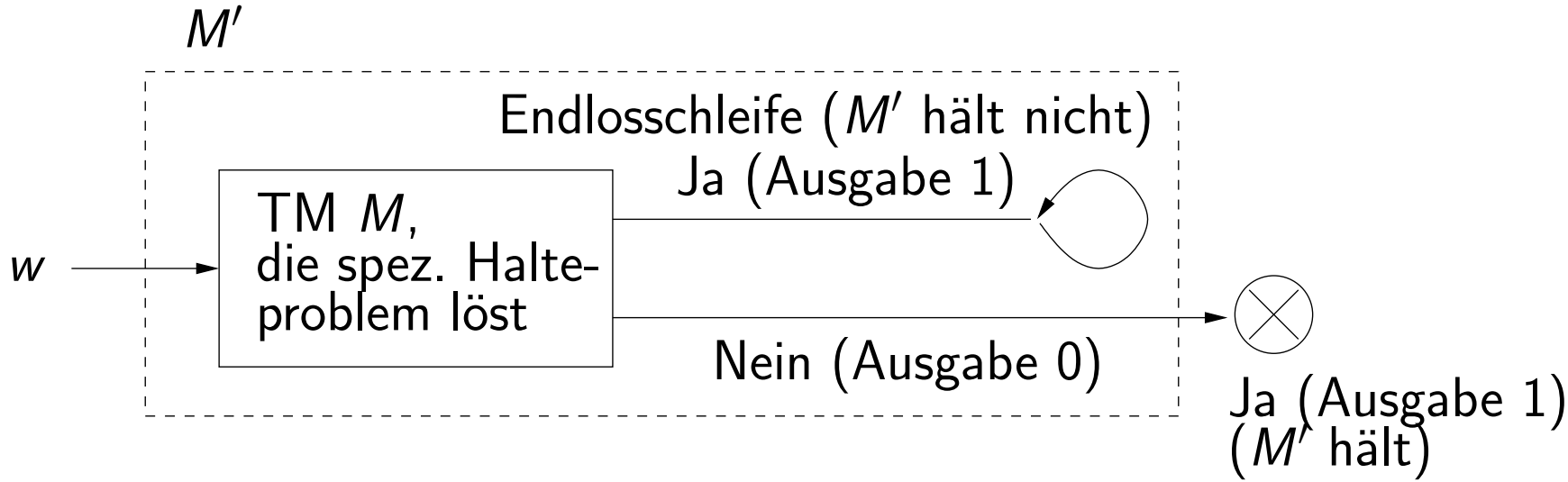
Angenommen das spezielle Halteproblem K ist entscheidbar, d. h. die charakteristische Funktion $\chi_K : \{0,1\}^* \to \{0,1\}$ wird durch eine Turing-Maschine M berechnet.

Dann können wir eine Turing-Maschine M' konstruieren, die sich wie folgt verhält:

INPUT $w \in \{0,1\}^*$
Simuliere M auf Eingabe w
IF $\chi_K(w) = 0$ THEN OUTPUT(1)
ELSE Gehe in Endlosschleife über
END

Unentscheidbarkeit des Halteproblems

Veranschaulichung der Turing-Maschine M':



Sei $w' \in \{0,1\}^*$ so, dass $M' = M_{w'}$.

Dann erhalten wir den folgenden Widerspruch:

$$
\begin{aligned}
M' = M_{w'} \text{ hält bei Eingabe } w' &\iff M \text{ gibt 0 bei Eingabe } w' \text{ aus} \\
&\iff \chi_K(w') = 0 \\
&\iff w' \notin K \\
&\iff M_{w'} \text{ hält bei Eingabe } w' \text{ nicht}
\end{aligned}
$$

Unentscheidbarkeit des Halteproblems

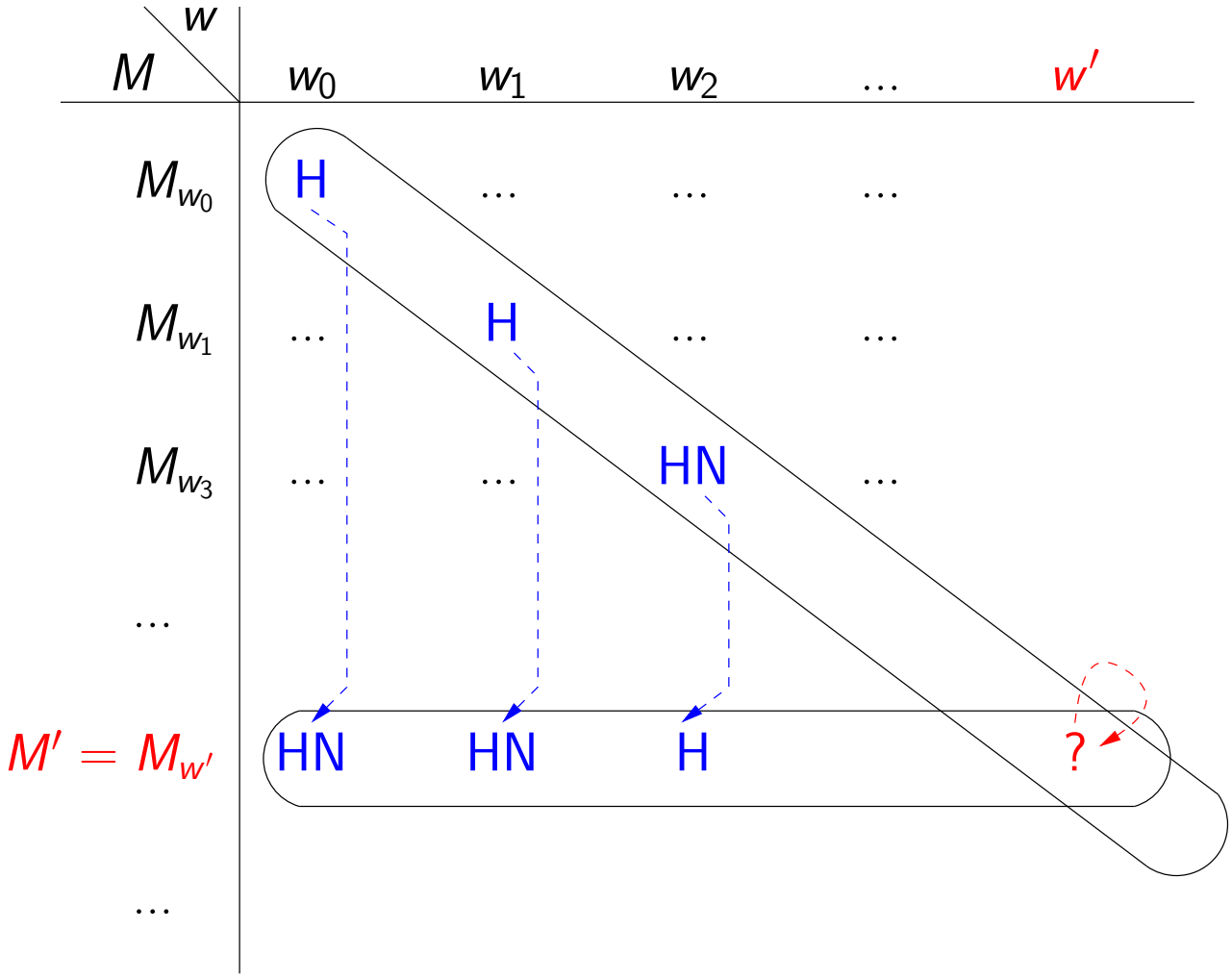
Es handelt sich hierbei um einen Diagonalisierungsbeweis:

Sei $w_0, w_1, w_2, \ldots$ eine Aufzählung aller Wörter aus $\{0,1\}^*$, beispielsweise $w_0 = \varepsilon$, $w_1 = 0$, $w_2 = 1$, $w_3 = 00$, $\ldots$

Wir tragen in eine Tabelle folgendes ein: "Wie verhält sich M_{w_i} auf w_j?"

Es gibt zwei Möglichkeiten: H (M_{w_i} hält) oder HN (M_{w_i} hält nicht).

Unentscheidbarkeit des Halteproblems



Unentscheidbarkeit des Halteproblems

Die konstruierte Turing-Maschine M' und ein w' mit $M' = M_{w'}$ sind ebenfalls in die Tabelle eingetragen.

Aufgrund der Konstruktion von M': die Felder in der Diagonalen bedingen die Felder in der Zeile von M'.

Problem: nichts passt an die Stelle des Fragezeichens!

$\leadsto$ es kann keine Turing-Maschine geben, die das Halteproblem löst. $\square$

Unentscheidbarkeit des Halteproblems

Satz 23 (Semi-Entscheidbarkeit des speziellen Halteproblems)

Das spezielle Halteproblem K ist semi-entscheidbar.

Beweis:

Die "halbe" charakteristische Funktion $\chi'_K : \{0,1\}^* \to \{1\}$ kann wie folgt berechnet werden:

- Bei Eingabe w starten wir die universelle Turing-Maschine U mit der Eingabe $w\#w$.
- Falls U bei Eingabe $w\#w$ terminiert, wird die produzierte Ausgabe durch 1 überschrieben. □

Reduktionen

Wir haben nun die Unentscheidbarkeit eines Problems, des speziellen Halteproblems, nachgewiesen.

Daraus sollen weitere Unentscheidbarkeitsresultate gewonnen werden.

Dies erfolgt mit Argumentationen folgender Art:

1. Wenn man Problem B lösen könnte, dann könnte man auch A lösen. (Reduktionsschritt)
2. Daraus folgt, dass B schwieriger bzw. allgemeiner ist als A ($A \leq B$).
3. Wir wissen jedoch bereits, dass A unentscheidbar ist.
4. Also muss das schwierigere Problem B auch unentscheidbar sein.

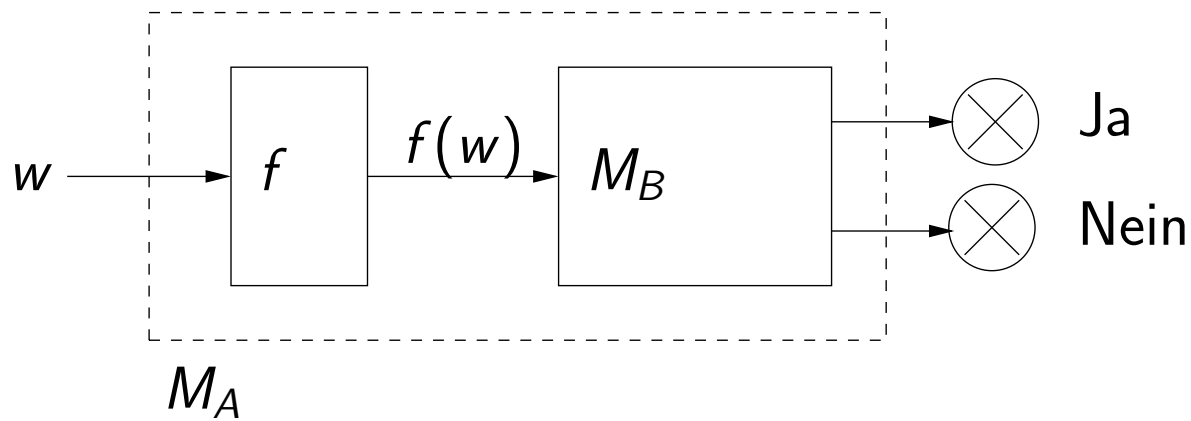
Reduktionen

Reduktion/Reduzierbarkeit (Definition)

Gegeben seien Sprachen $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$. Dann heißt A auf B reduzierbar (in Zeichen $A \leq B$), falls es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$ gilt:

$$w \in A \iff f(w) \in B.$$

Anschaulich: $A \leq B$, falls man aus einer Maschine M_B für B und einer Funktion f eine Maschine M_A für A bauen kann. Das heißt, M_B wird nach einer Vorverarbeitung der Eingabe durch f als Unterprozedur aufgerufen.



Reduktionen

Die folgende Aussage ist dann offensichtlich:

Lemma 24 (Reduktionen und Entscheidbarkeit)

Es sei $A \leq B$.

- Falls B entscheidbar ist, dann ist auch A entscheidbar.
- Falls A unentscheidbar ist, dann ist auch B unentscheidbar.

Kochrezept um die Unentscheidbarkeit eines Problems B zu zeigen

- Finde ein geeignetes Problem A, von dem bekannt ist, dass es unentscheidbar ist.

  Bisher kennen wir nur das spezielle Halteproblem K, wir werden allerdings bald weitere geeignete Probleme kennenlernen.
- Finde eine geeignete Funktion f, mit Hilfe derer A auf B reduziert werden kann und beweise, dass sie korrekt ist.
- Dann folgt daraus unmittelbar, dass B unentscheidbar ist.

Unentscheidbarkeit des Halteproblems

Satz 25 (Unentscheidbarkeit des Halteproblems)

Das (allgemeine) Halteproblem H ist nicht entscheidbar.

Beweis:

Sei die berechenbare Funktion f definiert durch $f(w) = w\#w$ für $w \in \{0,1\}^*$.

Dann gilt für alle $w \in \{0,1\}^*$:

$$w \in K \iff w\#w \in H \iff f(w) \in H.$$

Also gilt $K \leq H$.

Da K nach Satz 23 unentscheidbar ist, muss auch H unentscheidbar sein. □

Unentscheidbarkeit des Halteproblems

Halteproblem auf leerem Band (Definition)

Das Halteproblem auf leerem Band ist die Sprache

$$\begin{aligned} H_0 &= \{w \in \{0,1\}^* \mid M_w \text{ hält gestartet mit dem leeren Band}\} \\ &= \{w \in \{0,1\}^* \mid \varepsilon \in T(M_w)\}. \end{aligned}$$

Satz 26 (Unentscheidbarkeit des Halteproblems auf leerem Band)

Das Halteproblem auf leerem Band H_0 ist nicht entscheidbar.

Beweis: Wir zeigen $H \leq H_0$.

Jedem Wort $w\#x$ mit $w, x \in \{0,1\}^*$ ordnen wir eine Turing-Maschine $M(w\#x)$ zu, die, wenn auf dem leeren Band gestartet, wie folgt arbeitet:

1. Schreibe x auf das Band.
2. Simuliere dann die Maschine M_w.

Unentscheidbarkeit des Halteproblems

Es ist egal, wie die Maschine $M(w\#x)$ auf einem nicht-leeren Band arbeitet.

Wir definieren nun die Funktion $f : \{0,1,\#\}^* \to \{0,1\}^*$ durch die Vorschrift

$$f(w\#x) = \text{code}(M(w\#x)),$$

d.h. $M(w\#x) = M_{f(w\#x)}$ für alle $w, x \in \{0,1\}^*$.

Für Wörter der Form $y \in \{0,1,\#\}^* \setminus \{0,1\}^*\#\{0,1\}^*$ sei $f(y)$ der Code einer Turing-Maschine M_0, die nicht auf dem leeren Band hält.

Die Funktion f ist dann berechenbar: Sei $y \in \{0,1,\#\}^*$ die Eingabe.

- Falls y nicht genau ein $\#$ enthält (kann mit einem DFA überprüft werden), wird das feste Wort code(M_0) ausgegeben.
- Falls $y = w\#x$ mit $w, x \in \{0,1\}^*$ konstruieren wir algorithmisch die Turing-Maschine $M(w\#x)$ und berechnen dann code$(M(w\#x))$.

Unentscheidbarkeit des Halteproblems

Es gilt dann:

$$\begin{aligned} w\#x \in H &\iff M_w \text{ hält bei Eingabe } x \\ &\iff M(w\#x) \text{ hält auf dem leeren Band} \\ &\iff M_{f(w\#x)} \text{ hält auf dem leeren Band} \\ &\iff f(w\#x) \in H_0 \end{aligned}$$

Außerdem gilt für alle $y \in \{0,1,\#\}^* \setminus \{0,1\}^*\#\{0,1\}^*$:

$$y \notin H \text{ und } f(y) = \text{code}(M_0) \notin H_0.$$

Also vermittelt f in der Tat eine Reduktion von H auf H_0. □

Satz von Rice

Das nächste Resultat zeigt, dass es unentscheidbar ist, ob die Funktion, die von einer Turing-Maschine M berechnet wird, eine bestimmte Eigenschaft $\mathcal{S}$ hat.

Das bedeutet, es gibt keine Methode, mit der man für alle Turing-Maschinen verläßliche Aussagen über die von ihnen berechneten Funktionen machen kann.

Satz 27 (Satz von Rice)

Sei $\mathcal{R}$ die Klasse aller Turing-berechenbaren Funktionen und sei $\mathcal{S}$ eine beliebige Teilmenge hiervon mit $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$.
Dann ist die Sprache

$$C(\mathcal{S}) = \{w \in \{0,1\}^* \mid \text{die von } M_w \text{ berechnete Funktion liegt in } \mathcal{S}\}$$

unentscheidbar.

Satz von Rice

Beweis:

Sei Ω die überall undefinierte Funktion.

Es gilt entweder $\Omega \in \mathcal{S}$ oder $\Omega \notin \mathcal{S}$.

Fall 1: $\Omega \in \mathcal{S}$

Da $\mathcal{S} \neq \mathcal{R}$ gilt, gibt es eine Funktion $q \in \mathcal{R} \setminus \mathcal{S}$.

Sei Q eine Turing-Maschine, welche q berechnet.

Wir ordnen nun jedem Wort $w \in \{0,1\}^*$ eine Turing-Maschine $M(w)$ zu, die sich bei einer Eingabe $y \in \{0,1\}^*$ wie folgt verhält.

1. $M(w)$ ignoriert die Eingabe y zunächst und simuliert M_w auf dem leeren Band.
2. Falls diese Simulation schließlich hält, so simuliert $M(w)$ die Maschine Q auf y.

Satz von Rice

Dann gilt für die von $M(w)$ berechnete Funktion g:

$$g = \begin{cases} \Omega & \text{falls } M_w \text{ auf dem leeren Band nicht hält, d. h. } w \notin H_0 \\ q & \text{sonst, d. h. } w \in H_0 \end{cases}$$

Die totale Funktion $f : \{0,1\}^* \to \{0,1\}^*$ mit

$$f(w) = \text{der Code der Maschine } M(w)$$

ist offensichtlich berechenbar.

Beachte: Es gilt $M(w) = M_{f(w)}$.

Wir erhalten:

$$\begin{aligned} w \in H_0 &\implies g = q \\ &\implies \text{die von } M_{f(w)} \text{ berechnete Funktion liegt nicht in } \mathcal{S} \\ &\implies f(w) \notin C(\mathcal{S}) \end{aligned}$$

Satz von Rice

Umgekehrt gilt:

$$\begin{aligned} w \notin H_0 &\implies g = \Omega \\ &\implies \text{die von } M_{f(w)} \text{ berechnete Funktion liegt in } \mathcal{S} \\ &\implies f(w) \in C(\mathcal{S}) \end{aligned}$$

Es gilt also $w \in \overline{H_0} \iff f(w) \in C(\mathcal{S})$, d. h. $\overline{H_0} \leq C(\mathcal{S})$.

Da H_0 nach Satz 26 unentscheidbar ist, ist auch $\overline{H_0}$ und somit $C(\mathcal{S})$ unentscheidbar.

Fall 2: $\Omega \notin \mathcal{S}$

Da $\mathcal{S} \neq \emptyset$ gilt, gibt es eine Funktion $q \in \mathcal{S}$.

Sei Q eine Turing-Maschine, welche q berechnet.

Für $w \in \{0,1\}^*$ seien die Maschine $M(w)$ sowie die berechenbare totale Funktion $f : \{0,1\}^* \to \{0,1\}^*$ exakt wie in Fall 1 definiert.

Satz von Rice

Wir erhalten diesmal:

$$\begin{aligned} w \in H_0 &\implies g = q \\ &\implies \text{die von } M_{f(w)} \text{ berechnete Funktion liegt in } \mathcal{S} \\ &\implies f(w) \in C(\mathcal{S}) \end{aligned}$$

Umgekehrt gilt:

$$\begin{aligned} w \notin H_0 &\implies g = \Omega \\ &\implies \text{die von } M_{f(w)} \text{ berechnete Funktion liegt nicht in } \mathcal{S} \\ &\implies f(w) \notin C(\mathcal{S}) \end{aligned}$$

Hieraus folgt die Unentscheidbarkeit von $C(\mathcal{S})$ wie in Fall 1. □

Satz von Rice

Konsequenzen aus dem Satz von Rice

Folgende Probleme sind unentscheidbar:

- Konstante Funktion: $\{w \mid M_w \text{ berechnet eine konstante Funktion}\}$
- Identität: $\{w \mid M_w \text{ berechnet die Identitätsfunktion}\}$
- Totale Funktion: $\{w \mid M_w \text{ berechnet eine totale Funktion}\}$
- Überall undefinierte Funktion: $\{w \mid M_w \text{ berechnet } \Omega\}$

Der Satz von Rice erlaubt es Unentscheidbarkeitsresultat für die Eigenschaften der von einer Turing-Maschine berechneten Funktion zu zeigen, nicht jedoch für andere Eigenschaften einer Turing-Maschine (wie beispielsweise die Anzahl ihrer Zustände oder das Bandalphabet).

Konsequenz des Satzes von Rice auf die Verifikation von Programmen: Kein Programm kann automatisch die Korrektheit von Software überprüfen.

Satz von Rice

Der Satz von Rice und seine Varianten gelten natürlich auch für andere universelle Berechnungsmodelle.

Satz 28 (Halteproblem für GOTO-/WHILE-Programme)

Für ein gegebenes GOTO-/WHILE-Programm und Anfangswerte für die Variablen ist es nicht entscheidbar, ob das Programm auf dieser Eingabe hält.

Beweis:

Das Halteproblem für Turing-Maschinen ist auf dieses Problem reduzierbar. Dazu muss nur die Turing-Maschine in das entsprechende GOTO-/WHILE-Programm und die Eingabe der Maschine in die entsprechenden Variablenbelegungen übersetzt werden.

Siehe Transformation "Turing-Maschine $\rightarrow$ GOTO-Programm" als Reduktionsfunktion f. □

Satz von Rice

Es ist bereits folgendes Problem unentscheidbar:

Satz 29 (Halteproblem für GOTO-Programme mit zwei Variablen)

Für ein gegebenes GOTO-Programm mit zwei Variablen, die beide mit 0 vorbelegt sind, ist es nicht entscheidbar, ob das Programm hält.

(Ohne Beweis)

Für GOTO-Programme mit nur einer Variablen ist das Halteproblem übrigens entscheidbar, denn eine Variable kann durch einen Kellerautomaten simuliert werden.

Nicht-berechenbare Funktionen

Ähnlich zur Unentscheidbarkeit von Problemen kann auch gezeigt werden, dass bestimmte Funktionen nicht berechenbar sind. Ein Beispiel dafür ist die sogenannte Busy-Beaver-Funktion.

Busy Beaver

Wir betrachten alle Turing-Maschinen mit dem zweielementigen Bandalphabet $\Gamma = \{1, \square\}$ und n Zuständen. Von diesen Maschinen halten einige auf dem leeren Band, andere terminieren nicht.

Der Wert der Busy-Beaver-Funktion BB an der Stelle n ist die maximale Anzahl von Rechenschritten, die von einer Turing-Maschine mit n Zuständen ausgeführt werden kann, die auf dem leeren Band terminiert.

Nicht-berechenbare Funktionen

Von der Busy-Beaver-Funktion $\mathrm{BB}\colon \mathbb{N} \to \mathbb{N}$ ist folgendes bekannt:

- Sie ist nicht berechenbar.
- Über die Funktionswerte weiß man folgendes (siehe z.B. Scott Aaronson, The Busy Beaver Frontier, `https://www.scottaaronson.com/papers/bb.pdf`):

n	$\mathrm{BB}(n)$
1	1
2	6
3	21
4	107
5	47.176.870 (wurde in 2024 gezeigt)
6	$\geq 7,4 \times 10^{36.534}$
7	$\geq 10^{2\times 10^{10^{10^{18.705.352}}}}$

Unentscheidbare Probleme

Wir verwenden nun die Reduktions-Beweistechnik und zeigen die Unentscheidbarkeit folgender Probleme:

- Postsches Korrespondenzproblem PCP

  PCP: ein kombinatorisches Problem auf Wörtern, wichtiges (Hilfs-)Problem, um damit die Unentscheidbarkeit anderer Probleme zu zeigen
- Schnittproblem für kontextfreie Grammatiken

Postsches Korrespondenzproblem

Wir betrachten nun ein wichtiges unentscheidbares Problem, das dazu benutzt wird, die Unentscheidbarkeit vieler anderer Probleme zu zeigen:

Postsches Korrespondenzproblem (PCP)

- **Eingabe:** Eine endliche Liste von Wortpaaren $I = ((x_1, y_1), \ldots, (x_k, y_k))$ mit $x_i, y_i \in \Sigma^+$.

  Dabei ist Σ ein beliebiges Alphabet.
- **Frage:** Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$ mit $n \geq 1$ und $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$?

Eine Folge $(i_1, \ldots, i_n)$ mit $n \geq 1$, $i_1, \ldots, i_n \in \{1, \ldots, k\}$ und $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$ nennen wir eine Lösung für die PCP-Eingabe $I = ((x_1, y_1), \ldots, (x_k, y_k))$ und $x_{i_1} \cdots x_{i_n}$ ist das Lösungswort.

Postsches Korrespondenzproblem

Beispiel 1: Die folgende PCP-Eingabe ist lösbar:

$$\begin{array}{llllll} x_1 = 0 & x_2 = 1 & x_3 = 0101 \\ y_1 = 010 & y_2 = 101 & y_3 = 01 \end{array}$$

Eine mögliche Lösung: $(3, 3, 1, 2)$:

$$\begin{array}{l} 01 \quad 01 \mid 010 \quad 1 \mid 0 \mid 1 \\ 01 \mid 01 \mid 010 \mid 1 \quad 0 \quad 1 \end{array}$$

Eine weitere (kürzere) Lösung ist: $(3, 1)$

Postsches Korrespondenzproblem

Beispiel 2: Die folgende PCP-Eingabe ist lösbar:

$$\begin{array}{llllllllllll} x_1 & = & 001 & x_2 & = & 01 & x_3 & = & 01 & x_4 & = & 10 \\ y_1 & = & 0 & y_2 & = & 011 & y_3 & = & 101 & y_4 & = & 001 \end{array}$$

Eine kürzeste Lösung besteht bereits aus 66 Indizes:
$(2, 4, 3, 4, 4, 2, 1, 2, 4, 3, 4, 3, 4, 4, 3, 4, 4, 2, 1, 4, 4, 2, 1, 3, 4, 1, 1, 3,$
$4, 4, 4, 2, 1, 2, 1, 1, 1, 3, 4, 3, 4, 1, 2, 1, 4, 4, 2, 1, 4, 1, 1, 3, 4, 1, 1, 3,$
$1, 1, 3, 1, 2, 1, 4, 1, 1, 3)$.

An der Komplexität dieser Lösung kann man bereits die Schwierigkeit des Problems ablesen.

Postsches Korrespondenzproblem

Satz 30 (Semi-Entscheidbarkeit des PCP)

Das Postsche Korrespondenzproblem ist semi-entscheidbar.

Beweis:

Probiere erst alle Indexfolgen der Länge 1 aus, dann alle Indexfolgen der Länge 2, ...

Falls irgendwann eine passende Indexfolge gefunden wird, so gib 1 aus. □

Postsches Korrespondenzproblem

Der erste Schritt des Unentscheidbarkeitsbeweises ist es, das folgende modifizierte Problem zu betrachten.

Modifiziertes PCP (MPCP)

- **Eingabe:** I wie beim PCP.
- **Frage:** Gibt es eine Lösung $(i_1, \ldots, i_n)$ für I mit $i_1 = 1$?

Wir beweisen nun zwei Reduktions-Lemmata, aus denen die Unentscheidbarkeit des Postschen Korrespondenzproblems folgt:

Lemma 31 (MPCP auf PCP reduzierbar)

$\text{MPCP} \leq \text{PCP}$

Postsches Korrespondenzproblem

Beweis:

Sei $I = ((x_1, y_1), \ldots, (x_k, y_k))$ mit $x_i, y_i \in \Sigma^+$ eine endliche Folge von Wortpaaren.

Seien $\#$ und \$ zwei neue Symbole.

Für ein Wort $w = a_1 a_2 \cdots a_n$ mit $a_1, \ldots, a_n \in \Sigma$ und $n \geq 1$ definieren wir die Wörter ${}^{\#}w$, $w^{\#}$, ${}^{\#}w^{\#}$ wie folgt:

$$\begin{aligned} {}^{\#}w &= \#a_1\#a_2\#\cdots\#a_n \\ w^{\#} &= a_1\#a_2\#\cdots\#a_n\# \\ {}^{\#}w^{\#} &= \#a_1\#a_2\#\cdots\#a_n\# \end{aligned}$$

Wir ordnen nun der Liste I die Liste

$$f(I) = (({}^{\#}x_1^{\#}, {}^{\#}y_1), ({}x_1^{\#}, {}^{\#}y_1), \ldots, ({}x_k^{\#}, {}^{\#}y_k), (\$, \#\$))$$

zu. Diese besteht aus $k + 2$ Paaren.

Die Funktion f ist offensichtlich berechenbar.

Postsches Korrespondenzproblem

Beispiel: Sei I das folgende PCP:

$$\begin{array}{llllll} x_1 = 0101 & x_2 = 0 & x_3 = 1 \\ y_1 = 01 & y_2 = 010 & y_3 = 101 \end{array}$$

Dann ist $f(I)$ das folgende PCP:

$x'_1 = \#0\#1\#0\#1\#$ $x'_2 = 0\#1\#0\#1\#$ $x'_3 = 0\#$ $x'_4 = 1\#$

$y'_1 = \#0\#1$ $y'_2 = \#0\#1$ $y'_3 = \#0\#1\#0$ $y'_4 = \#1\#0\#1$

$x'_5 = \$$

$y'_5 = \#\$$

Postsches Korrespondenzproblem

Wir behaupten, dass f eine Reduktion von MPCP nach PCP vermittelt.

Sei zunächst $(i_1, i_2, \ldots, i_n)$ eine Lösung für I mit $i_1 = 1$ $(n \geq 1)$.

Dann ist $(1, i_2 + 1, \ldots, i_n + 1, k + 2)$ eine Lösung für $f(I)$.

Sei nun $(i_1, \ldots, i_n)$ eine kürzeste Lösung von $f(I)$.

Dann muss $i_1 = 1$, $i_2, \ldots, i_{n-1} \in \{2, \ldots, k + 1\}$ und $i_n = k + 2$ gelten.

Dann ist $(1, i_2 - 1, \ldots, i_{n-1} - 1)$ eine Lösung für I. □

Postsches Korrespondenzproblem

Lemma 32 (Halteproblem auf MPCP reduzierbar)

$H \leq \text{MPCP}$

Beweis:

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine deterministische Turing-Maschine (gegeben durch ihre Kodierung) und $w \in \Sigma^*$.

Wir konstruieren hieraus eine MPCP-Eingabe

$$I(M, w) = ((x_1, y_1), \ldots, (x_k, y_k)),$$

welche lösbar ist genau dann, wenn M auf Eingabe w hält.

$I(M, w)$ wird über dem Alphabet $Z \cup \Gamma \cup \{\$, \#\}$ definiert (o.B.d.A. sei $Z \cap \Gamma = \emptyset$).

Postsches Korrespondenzproblem

Dann sieht $I(M, w)$ wie folgt aus:

- 1. Wortpaar:
  $(\$, \$\square z_0 w \square \#)$
- Kopierpaare:
  (a, a) für alle $a \in \Gamma \cup \{\#\}$
- Transitionspaare:
  $(za, z'c)$ falls $\delta(z, a) = (z', c, N)$
  (za, cz') falls $\delta(z, a) = (z', c, R)$
  $(bza, z'bc)$ falls $\delta(z, a) = (z', c, L)$ und $b \in \Gamma$
- Paar um Blanks bei Bedarf am Rand hinzuzufügen:
  $(\#, \square\#)$ und $(\#, \#\square)$
- Löschpaare:
  (az_e, z_e) und $(z_e a, z_e)$ für alle $z_e \in E$ und $a \in \Gamma$
- Abschlusspaar:
  $(z_e\#\#, \#)$ für alle $z_e \in E$

Postsches Korrespondenzproblem

Behauptung: M hält auf Eingabe w genau dann, wenn $I(M, w)$ lösbar ist.

Angenommen M hält auf Eingabe w.

Dann gibt es eine Folge von Konfigurationen $k_0, k_1, \ldots, k_t \in \Gamma^+ Z \Gamma^+$ mit:

- $k_0 = \square z_0 w \square$
- $k_i \vdash_M k_{i+1}$ für alle $0 \leq i \leq t-1$
- $k_t \in \Gamma^+ E \Gamma^+$

Dann erhalten wir eine Lösung von $I(M, w)$, wobei das Lösungswort wie folgt aussieht:

$$\$k_0 \# k_1 \# \cdots \# k_t \# k_t' \# k_t'' \# k_t''' \cdots \# z_e \# \#.$$

Hierbei entstehen $k_t', k_t'', k_t''', \ldots$ aus k_t, indem jeweils das Symbol links oder rechts von z_e gelöscht wird.

Postsches Korrespondenzproblem

Angenommen $I(M, w)$ hat nun eine Lösung $(1, i_2, \ldots, i_t)$, welche also mit $(\#, \#\square\square z_0 w \square\#)$ beginnt.

Solange in der "partiellen Lösung" $(x_1 x_{i_2} \cdots x_{i_n}, y_1 y_{i_2} \cdots y_{i_n})$ in $y_1 y_{i_2} \cdots y_{i_n}$ (dem längeren Wort) noch kein Endzustand $z_e \in E$ vorkommt, muss mittels der Kopierpaare, Transitionspaare und den Paaren zum Hinzufügen von Blanks eine Berechnung von M auf Eingabe w korrekt simuliert werden.

Da wir aber eine endliche Lösung $(1, i_2, \ldots, i_t)$ haben, muss für ein $m \leq t$ ein Endzustand $z_e \in E$ in $y_1 y_{i_1} \cdots y_{i_m}$ vorkommen.

Also hält M bei Eingabe w. $\square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \Box$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \Box, \#\}$ und $y \in \{a, b, c, \Box\}$

$(\$, \$\Box z_0 ab\Box\#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \Box\#)$ $(\#, \#\Box)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$$

$\$ \ \Box \ z_0 \ a \ \ b \ \Box \ \#$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$$ $\square$

$\$$ $\square$ z_0 a b $\square$ $\#$ $\square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \Box$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0 ab \vdash b z_1 b \vdash z_2 bc \vdash z_e cc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \Box, \#\}$ und $y \in \{a, b, c, \Box\}$

$(\$, \$\Box z_0 ab \Box \#)$ (x, x) $(z_0 a, b z_1)$ $(y z_1 b, z_2 y c)$ $(z_2 b, z_e c)$

$(\#, \Box\#)$ $(\#, \#\Box)$ $(y z_e, z_e)$ $(z_e y, z_e)$ $(z_e \#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \Box \ z_0 \ a$

$\$ \ \Box \ z_0 \ a \ \ b \ \Box \ \# \ \Box \ b \ \ z_1$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \#$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \#$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0 ab \vdash b z_1 b \vdash z_2 bc \vdash z_e cc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$$(\$, \$\square z_0 ab \square \#) \quad (x, x) \quad (z_0 a, b z_1) \quad (y z_1 b, z_2 y c) \quad (z_2 b, z_e c)$$
$$(\#, \square \#) \quad (\#, \# \square) \quad (y z_e, z_e) \quad (z_e y, z_e) \quad (z_e \# \#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b$
$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$$(\$, \$\square z_0 ab \square \#) \quad (x, x) \quad (z_0a, bz_1) \quad (yz_1b, z_2yc) \quad (z_2b, z_ec)$$
$$(\#, \square\#) \quad (\#, \#\square) \quad (yz_e, z_e) \quad (z_ey, z_e) \quad (z_e\#\#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square$
$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \#$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \#$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \Box$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \Box, \#\}$ und $y \in \{a, b, c, \Box\}$

$$(\$, \$\Box z_0 ab\Box\#) \quad (x, x) \quad (z_0a, bz_1) \quad (yz_1b, z_2yc) \quad (z_2b, z_ec)$$
$$(\#, \Box\#) \quad (\#, \#\Box) \quad (yz_e, z_e) \quad (z_ey, z_e) \quad (z_e\#\#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \Box \ z_0 \ a \ \ b \ \Box \ \# \ \Box \ b \ z_1 \ b \ \Box \ \# \ \Box \ z_2 \ b$
$\$ \ \Box \ z_0 \ a \ \ b \ \Box \ \# \ \Box \ b \ z_1 \ b \ \Box \ \# \ \Box \ z_2 \ b \ \ c \ \Box \ \# \ \Box \ z_e \ c$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \#$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square \ \#$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$$(\$, \$\square z_0 ab \square \#) \quad (x, x) \quad (z_0a, bz_1) \quad (yz_1b, z_2yc) \quad (z_2b, z_ec)$$
$$(\#, \square\#) \quad (\#, \#\square) \quad (yz_e, z_e) \quad (z_ey, z_e) \quad (z_e\#\#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e$
$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square \ \# \ z_e$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square \ \# \ z_e \ c$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$(\$, \$\square z_0 ab \square \#)$ (x, x) (z_0a, bz_1) (yz_1b, z_2yc) (z_2b, z_ec)

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) (z_ey, z_e) $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$\ \square\ z_0\ a\ \ b\ \square\ \#\ \square\ b\ \ z_1\ b\ \square\ \#\ \square\ z_2\ b\ \ c\ \ \square\ \#\ \square\ z_e\ c\ \ c$

$\$\ \square\ z_0\ a\ \ b\ \square\ \#\ \square\ b\ \ z_1\ b\ \square\ \#\ \square\ z_2\ b\ \ c\ \ \square\ \#\ \square\ z_e\ c\ \ c\ \ \square\ \#\ z_e\ c\ \ c$

Postsches Korrespondenzproblem

Beispiel: Betrachten wir eine Turing-Maschine M mit den Zuständen z_0, z_1, z_2, z_e, den Bandsymbolen $a, b, c, \square$ und folgenden Übergängen:

$$\delta(z_0, a) = (z_1, b, R) \quad \delta(z_1, b) = (z_2, c, L) \quad \delta(z_2, b) = (z_e, c, N)$$

mit $z_e \in E$. Dann gibt es die folgende akzeptierende Berechnung bei Eingabe ab:

$$z_0ab \vdash bz_1b \vdash z_2bc \vdash z_ecc$$

$I(M, ab)$ besteht aus folgenden Paaren für alle $x \in \{a, b, c, \square, \#\}$ und $y \in \{a, b, c, \square\}$

$$(\$, \$\square z_0 ab \square \#) \quad (x, x) \quad (z_0a, bz_1) \quad (yz_1b, z_2yc) \quad (z_2b, z_ec)$$
$$(\#, \square\#) \quad (\#, \#\square) \quad (yz_e, z_e) \quad (z_ey, z_e) \quad (z_e\#\#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square$

$\$ \ \square \ z_0 \ a \ \ b \ \square \ \# \ \square \ b \ \ z_1 \ b \ \square \ \# \ \square \ z_2 \ b \ \ c \ \square \ \# \ \square \ z_e \ c \ \ c \ \square \ \# \ z_e \ c \ \ c \ \square$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$$(\$, \$\square z_0 ab\square\#) \quad (x, x) \quad (z_0 a, bz_1) \quad (yz_1 b, z_2 yc) \quad (z_2 b, z_e c)$$
$$(\#, \square\#) \quad (\#, \#\square) \quad (yz_e, z_e) \quad (z_e y, z_e) \quad (z_e\#\#, \#)$$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square$

$\cdots \square \# z_e\, c \;\; c \;\; \square$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \#$
$\cdots \square \# z_e\, c\;\; c\;\; \square \#$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\Box z_0 ab\Box\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \Box\#)$ $(\#, \#\Box)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \Box \# z_e\, c$
$\cdots \Box \# z_e\, c \;\; c \;\; \Box \# z_e$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e c \quad c$
$\cdots \square \# z_e c \quad c \quad \square \# z_e c$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e\, c \;\; c \;\; \square$
$\cdots \square \# z_e\, c \;\; c \;\; \square \# z_e\, c \;\; \square$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e\, c \;\; c \;\; \square \#$
$\cdots \square \# z_e\, c \;\; c \;\; \square \# z_e\, c \;\; \square \#$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\Box z_0 ab\Box\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \Box\#)$ $(\#, \#\Box)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \Box \# z_e\, c \;\; c \;\; \Box \# z_e\, c$
$\cdots \Box \# z_e\, c \;\; c \;\; \Box \# z_e\, c \;\; \Box \# z_e$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab \square \#)$ (x, x) $(z_0 a, b z_1)$ $(y z_1 b, z_2 y c)$ $(z_2 b, z_e c)$
$(\#, \square \#)$ $(\#, \#\square)$ $(y z_e, z_e)$ $(z_e y, z_e)$ $(z_e \# \#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square$
$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \#$
$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square \#$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$
$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots\ \square\ \#\ z_e\ c\ \ c\ \ \square\ \#\ z_e\ c\ \ \square\ \#\ z_e\ \square$
$\cdots\ \square\ \#\ z_e\ c\ \ c\ \ \square\ \#\ z_e\ c\ \ \square\ \#\ z_e\ \square\ \#\ z_e$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab \square \#)$ (x, x) $(z_0 a, b z_1)$ $(y z_1 b, z_2 y c)$ $(z_2 b, z_e c)$
$(\#, \square \#)$ $(\#, \#\square)$ $(y z_e, z_e)$ $(z_e y, z_e)$ $(z_e \# \#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square \#$
$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square \# z_e \#$

Postsches Korrespondenzproblem

Beispiel (Fortsetzung)

$(\$, \$\square z_0 ab\square\#)$ (x, x) $(z_0 a, bz_1)$ $(yz_1 b, z_2 yc)$ $(z_2 b, z_e c)$

$(\#, \square\#)$ $(\#, \#\square)$ (yz_e, z_e) $(z_e y, z_e)$ $(z_e\#\#, \#)$

Eine Lösung dieses PCPs, die zu der obigen akzeptierenden Berechnung gehört, baut sich dann wie folgt auf:

$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square \# z_e \# \#$

$\cdots \square \# z_e c \;\; c \;\; \square \# z_e c \;\; \square \# z_e \square \# z_e \# \#$

Postsches Korrespondenzproblem

Satz 33 (PCP unentscheidbar)

Das Postsche Korrespondenzproblem ist unentscheidbar.

Beweis: Die Behauptung folgt direkt aus den beiden vorherigen Lemmata:

Aus $H \leq \text{MPCP} \leq \text{PCP}$ folgt $H \leq \text{PCP}$ (durch Komposition der Reduktionsabbildungen).

Da außerdem das allgemeine Halteproblem H nicht entscheidbar ist, ist auch PCP nicht entscheidbar. □

Postsches Korrespondenzproblem

Bemerkungen:

Sei $\mathrm{PCP}_{m,n}$ die Einschränkung des PCPs auf Eingaben der Form $((x_1, y_1), \ldots, (x_k, y_k))$ mit $k \leq m$, $x_1, y_1, \ldots, x_k, y_k \in \{a_1, \ldots, a_n\}^+$
(d. h. n-elementiges Alphabet und nur maximal m Wortpaare)

- Bereits $\mathrm{PCP}_{5,2}$ ist unentscheidbar.
  (Turlough Neary 2015, `http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=4948`
- $\mathrm{PCP}_{m,1}$ und $\mathrm{PCP}_{2,n}$ sind entscheidbar (für m und n beliebig).
- Es ist unbekannt, ob $\mathrm{PCP}_{k,2}$ für $k \in \{3, 4\}$ entscheidbar ist.

Schnittproblem für kontextfreie Grammatiken

Wir werden nun das PCP dazu nutzen, um die Unentscheidbarkeit des Schnittproblems für kontextfreie Grammatiken zu zeigen.

Schnittproblem für kontextfreie Grammatiken

- **Eingabe:** zwei kontextfreie Grammatiken G_1, G_2.
- **Frage:** Gilt $L(G_1) \cap L(G_2) \neq \emptyset$, d.h., es gibt ein Wort, das sowohl von G_1 als auch von G_2 erzeugt wird?

Satz 34 (Schnittproblem unentscheidbar)

Das Schnittproblem für kontextfreie Grammatiken ist unentscheidbar.

Schnittproblem für kontextfreie Grammatiken

Beweis:

Aufgrund von Satz 33 (PCP unentscheidbar) genügt es zu zeigen, dass PCP auf das Schnittproblem für kontextfreie Grammatiken reduzierbar ist.

Sei hierzu $I = ((x_1, y_1), \ldots, (x_k, y_k))$ eine beliebige PCP-Eingabe mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

Sei $\Gamma = \Sigma \cup \{\$, 1, \ldots, k\}$.

Wir definieren nun zwei kontextfreie Grammatiken G_1 und G_2 über dem Terminalalphabet Γ.

Produktionen von G_1 (S ist das Startsymbol):

$$\begin{array}{lcl} S & \to & A\,\$\,B \\ A & \to & 1\,A\,x_1 \mid 1\,x_1 \mid \cdots \mid k\,A\,x_k \mid k\,x_k \\ B & \to & y_1^{\text{rev}}\,B\,1 \mid y_1^{\text{rev}}\,1 \mid \cdots \mid y_k^{\text{rev}}\,B\,k \mid y_k^{\text{rev}}\,k \end{array}$$

Schnittproblem für kontextfreie Grammatiken

Hierbei ist w^{rev} das Wort w von recht nach links gelesen.

Dann gilt:

$$L(G_1) = \{i_n \cdots i_1 x_{i_1} \cdots x_{i_n} \$ (y_{j_1} \cdots y_{j_m})^{\text{rev}} j_1 \cdots j_m \mid n, m \geq 1, 1 \leq i_1, \ldots, i_n, j_1, \ldots, j_m \leq k\}.$$

Produktionen von G_2 (S ist das Startsymbol):

$$\begin{array}{lcl} S & \to & 1S1 \mid \cdots kSk \mid T \\ T & \to & aTa \text{ für alle } a \in \Sigma \mid \$ \end{array}$$

Dann gilt: $L(G_2) = \{uv\$v^{\text{rev}}u^{\text{rev}} \mid u \in \{1, \ldots, k\}^*, v \in \Sigma^*\}$.

Also gilt: I lösbar $\iff$ $L(G_1) \cap L(G_2) \neq \emptyset$. □

Schnittproblem für kontextfreie Grammatiken

In dem vorherigen Beweis entsprechen Lösungen der PCP-Eingabe I genau den Wörtern in $L(G_1) \cap L(G_2)$.

Nun gilt für jede PCP-Eingabe J: J lösbar genau dann, wenn J unendlich viele Lösungen hat (wenn $(i_1, \ldots, i_k)$ eine Lösung ist, dann ist auch $(i_1, \ldots, i_k, i_1, \ldots, i_k)$ eine Lösung).

Also gilt: I lösbar genau dann, wenn $L(G_1) \cap L(G_2)$ unendlich ist.

Wir erhalten:

Satz 35

Es ist unentscheidbar, ob für gegebene kontextfreie Grammatiken G_1 und G_2 der Schnitt $L(G_1) \cap L(G_2)$ unendlich ist.

Schnittproblem für kontextfreie Grammatiken

Es ist einfach, eine kontextfreie Grammatik G_2' für die Sprache $\Gamma^* \setminus L(G_2)$ anzugeben (Übung).

Sei nun G_3 eine kontextfreie Grammatik für $L(G_1) \cup L(G_2')$.

Dann gilt:

$$\begin{aligned} L(G_1) \cap L(G_2) = \emptyset \quad &\iff \quad L(G_1) \subseteq L(G_2') \\ &\iff \quad L(G_1) \cup L(G_2') = L(G_2') \\ &\iff \quad L(G_3) = L(G_2') \end{aligned}$$

Wir erhalten:

Satz 36

Es ist unentscheidbar, ob für gegebene kontextfreie Grammatiken G_1 und G_2 gilt:

- $L(G_1) \subseteq L(G_2)$
- $L(G_1) = L(G_2)$

Schnittproblem für kontextfreie Grammatiken

Schließlich kann man von den Grammatiken G_1, G_2 und G_2' zeigen, dass sie deterministisch kontextfreie Sprachen erzeugen, und man kann aus G_1, G_2, G_2' äquivalente deterministische Kellerautomten A_1, A_2, A_2' konstruieren.

Also ergibt sich:

Satz 37

Es ist unentscheidbar, ob für gegebene deterministische Kellerautomaten A_1 und A_2 gilt:

- $T(A_1) \cap T(A_2) \neq \emptyset$
- $T(A_1) \cap T(A_2)$ ist unendlich.
- $T(A_1) \subseteq T(A_2)$

Schnittproblem für kontextfreie Grammatiken

Bemerkung 1: Die auf Folie 195 konstruierte Sprache $L(G_1) \cup L(G_2')$ ist nicht notwendigerweise deterministisch kontextfrei (die Klasse der deterministisch kontextfreien Sprachen ist nicht unter Vereinigung abgeschlossen).

In der Tat ist es entscheidbar, ob $T(A_1) = T(A_2)$ für zwei gegebene deterministische Kellerautomaten A_1 und A_2 gilt (Senizergues 1997).

Bemerkung 2: Das Schnittproblem für kontextfreie Sprachen ist semi-entscheidbar:

Allgemeiner: Die Menge $\{(u, v) \mid u, v \in \{0, 1\}^*, T(M_u) \cap T(M_v) \neq \emptyset\}$ ist semi-entscheidbar, d.h. das Schnittproblem für Typ-0-Sprachen ist semi-entscheidbar:

Die Sprachen $T(M_u)$ und $T(M_v)$ sind rekursiv aufzählbar.

Zähle "parallel" die Sprachen $T(M_u)$ und $T(M_v)$ auf.

Leerheit von kontextsensitiven Sprachen

Terminiere mit Ausgabe 1 falls irgendwann ein Wort w in beiden Aufzählungen auftaucht.

Konsequenz: Das Komplement des Schnittproblems ist nicht semi-entscheidbar. Ansonsten wäre es nämlich entscheidbar (Folie 116).

Satz 38 (Leerheit von Typ-1-Grammatiken unentscheidbar)

Es ist unentscheidbar, ob für eine gegebene Typ-1-Grammatik G (oder alternativ einen linear beschränkten Automaten) gilt: $L(G) \neq \emptyset$.

Beweis:

Wir reduzieren das Schnittproblem für kontextfreie Grammatiken auf das Leerheitsproblem für Typ-1-Grammatiken.

Mit Satz 34 beweist dies den Satz.

Seien G_1 und G_2 zwei kontextfreie Grammatiken.

Diese sind insbesondere vom Typ-1.

Leerheit von kontextsensitiven Sprachen

Da die Typ-1-Sprachen effektiv unter Schnitt abgeschlossen sind, können wir aus G_1 und G_2 eine Typ-1-Grammatik G mit $L(G) = L(G_1) \cap L(G_2)$ konstruieren.

Etwas detaillierter: Konstruiere aus G_1 und G_2 zwei linear beschränkte Automaten A_1 und A_2 mit $L(G_1) = T(A_1)$ und $L(G_2) = T(A_2)$ (siehe Konstruktion im Beweis des Satzes von Kuroda (FSA, Folie 338).

Aus A_1 und A_2 kann man dann leicht einen linear beschränkten Automaten A mit $T(A) = T(A_1) \cap T(A_2)$ konstruieren.

A kann dann wieder mittels der Konstruktion im Beweis des Satzes von Kuroda in eine äquivalente Typ-1-Grammatik umgewandelt werden. □

Hilbert's 10. Problem

Ein weiteres unentscheidbares Problem (ohne Beweis):

Wir betrachten Polynome $p(x_1, \ldots, x_n)$ (in mehreren Variablen) mit Koeffizienten aus $\mathbb{Z}$.

Beispiel: $p(x_1, x_2, x_3, x_4) = -5x_1^2 x_3^4 x_4 + 3x_2^8 x_3^2 x_4^3 - 8x_1 x_2^6 + 17x_4 - 25$.

Hilbert's 10. Problem ist unentscheidbar (Matiyasevich 1970)

Das folgende Problem ist unentscheidbar:

EINGABE: Ein Polynom $p(x_1, \ldots, x_n)$ (in mehreren Variablen) mit Koeffizienten aus $\mathbb{Z}$.

FRAGE: Existieren $a_1, \ldots, a_n \in \mathbb{Z}$ mit $p(a_1, \ldots, a_n) = 0$?

Komplexitätstheorie

Deterministische Zeitklassen

Sei $f : \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse DTIME(f) besteht aus allen Sprachen L, für die es eine deterministische Turingmaschine M gibt mit:

- M berechnet die charakteristische Funktion von L.
- Für jede Eingabe $w \in \Sigma^*$ erreicht M von der Startkonfiguration $z_0 w \square$ aus nach höchstens $f(|w|)$ Rechenschritten einen Endzustand (und gibt 0 oder 1 aus, je nachdem ob $w \notin L$ oder $w \in L$ gilt).

Poly bezeichet die Menge aller durch ein Polynom mit Koeffizienten aus $\mathbb{N}$ beschriebenen Funktionen auf $\mathbb{N}$ (z. B. $n, 2n, n^2 + 3n, n^{10000}$).

Die Klasse P

$$\mathrm{P} = \bigcup_{f \in \mathrm{Poly}} \mathrm{DTIME}(f)$$

Bemerkungen:

- P wird häufig als die Menge aller effizient lösbaren Probleme betrachtet.
- Die Klasse P ist relativ robust gegen Änderungen des Berechnungsmodells. So ändert sich z. B. die Klasse P nicht, wenn wir anstatt normalen Turingmaschinen Mehrband-Turingmaschinen erlauben würden (was auch etwas realistischer wäre, da man bei Verwendung von Einband-Turingmaschinen sehr viel Information kopieren muss).
- Definiert man P über WHILE- oder GOTO-Programme, so muss man den Zeitaufwand für eine Zuweisung (z. B. $x_i := x_j + 1$) als die aktuelle Anzahl der Bits von x_j (also etwa $\log(x_j)$) bemessen: logarithmisches Kostenmaß.

Komplexitätstheorie

- Würde man das uniforme Kostenmaß (d. h. eine Zuweisung wird als ein Schritt gezählt) verwenden, so wäre z. B. der folgende Algorithmus polynomiell:

  $\mathrm{INPUT}\ n;$
  $x := 2;$
  $\mathrm{LOOP}\ n\ \mathrm{DO}\ x := x * x\ \mathrm{END};$
  $\mathrm{OUTPUT}(x)$

  Dieser Algorithmus berechnet die Zahl 2^{2^n}, und um diese Zahl in Binärdarstellung aufzuschreiben benötigt man schon 2^n viele Bits.

  Bei diesem Beispiel haben wir aber etwas geschummelt, da wir Multiplikation als elementare Operation verwenden. Ersetzt man $x := x * x$ durch ein (Standard-) Loop-Programm, so wird die Rechenzeit des obigen Algorithmus exponentiell.

Nichtdeterministische Zeitklassen

Sei $f : \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse NTIME(f) besteht aus allen Sprachen L, für die es eine nichtdeterministische Turingmaschine M gibt mit:

- Für jede Eingabe $w \in \Sigma^*$ erreicht M von der Startkonfiguration $z_0 w \square$ aus auf jedem Berechnungspfad nach höchstens $f(|w|)$ Rechenschritten einen Endzustand und gibt 0 oder 1 aus.
- Es gilt: $w \in L$ genau dann, wenn M auf mindestens einem Berechnungspfad eine 1 ausgibt.

Die Klasse NP

$$\mathrm{NP} = \bigcup_{f \in \mathrm{Poly}} \mathrm{NTIME}(f)$$

Komplexitätstheorie

Bemerkungen:

- Offensichtlich gilt $\mathrm{P} \subseteq \mathrm{NP}$.
- Ob $\mathrm{P} = \mathrm{NP}$ gilt, gilt als die wichtigste offene Frage in der Theoretischen Informatik. Es wird im Allgemeinen vermutet, dass $\mathrm{P} \neq \mathrm{NP}$ gilt.
- Warum ist die Frage $\mathrm{P} = \mathrm{NP}$ so interessant?

  Von einer Vielzahl von Problemen ist bekannt, dass sie in NP liegen, man weiß jedoch nicht, ob sie in P liegen.

  Man kennt sogar eine große Klasse von Problemen (die NP-vollständigen Probleme, mehr dazu gleich) von denen folgendes bekannt ist: Gehört eines dieser Probleme zu P, so folgt $\mathrm{P} = \mathrm{NP}$.
- Es ist nicht schwer zu sehen, dass alle Sprachen in NP LOOP-entscheidbar sind (d. h. die charakteristischen Funktionen von Sprachen aus NP sind LOOP-berechenbar).

Beispiel: SUBSETSUM

SUBSETSUM

EINGABE: Binär-kodierte Zahlen $t, w_1, \ldots, w_n$
FRAGE: Gibt es eine Teilmenge $U \subseteq \{w_1, \ldots w_n\}$ mit $t = \sum_{w \in U} w$?

Satz 39

SUBSETSUM $\in$ NP

Bemerkung: Dieses Problem gehört zu P, falls man die Zahlen $t, w_1, \ldots, w_n$ unär kodiert.

Bei der unären Kodierung wird die Zahl n durch das Wort a^n (für ein Symbol a) kodiert.

Reduktionen so wie wir sie im Abschnitt über (Un)Entscheidbarkeit (Folie 138) kennengelernt haben, sind für entscheidbare Probleme nicht sehr aussagekräftig:

Lemma 40

Seien $A, B \subseteq \Sigma^*$ entscheidbare Sprachen mit $\emptyset \neq B \neq \Sigma^*$. Dann gilt $A \leq B$.

Wähle hierzu zwei Elemente $x \in B$ und $y \in \Sigma^* \setminus B$.

Definiere die Funktion $f : \Sigma^* \to \Sigma^*$ durch

$$f(w) = \begin{cases} x \text{ falls } w \in A \\ y \text{ falls } w \notin A \end{cases}$$

Da A entscheidbar ist, ist f berechenbar, und es gilt $w \in A \iff f(w) \in B$, d. h. $A \leq B$.

Polynomielle Reduzierbarkeit

Eine Funktion $f : \Sigma^* \to \Gamma^*$ ist polynomiell berechenbar, falls eine deterministische Turingmaschine M und ein Polynom $p(n)$ existiert, so dass für alle $w \in \Sigma^*$ gilt:

Wenn M mit der Eingabe w gestartet wird, hält M nach höchstens $p(|w|)$ vielen Schritten mit der Ausgabe $f(w)$ auf dem Arbeitsband an.

Eine Sprache $A \subseteq \Sigma^*$ ist polynomiell reduzierbar auf eine Sprache $B \subseteq \Gamma^*$ (kurz $A \leq_p B$), falls eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ existiert mit

$$\forall w \in \Sigma^* : w \in A \iff f(w) \in B.$$

Lemma 41

Wenn $A \leq_p B$ und $B \in \mathrm{P}$ (bzw. $B \in \mathrm{NP}$), dann gilt $A \in \mathrm{P}$ (bzw. $A \in \mathrm{NP}$).

Beweis:

Sei zunächst $A \leq_p B$ und $B \in \mathrm{P}$.

Dann existieren Polynome $p(n)$ und $q(n)$ sowie Turingmaschinen M und N mit folgenden Eigenschaften:

- M berechnet aus einer Eingabe $w \in \Sigma^*$ in Zeit $p(|w|)$ ein Wort $f(w)$ mit: $w \in A \iff f(w) \in B$.

  Beachte: Da die Maschine M in $p(|w|)$ Schritten nur eine Ausgabe der Länge höchstens $p(|w|)$ erzeugen kann, gilt $|f(w)| \leq p(|w|)$.

- N akzeptiert die Sprache B in Zeit $q(n)$.

Ein Turingmaschine für die Sprache A arbeitet dann bei einer Eingabe w wie folgt:

1. Berechne $f(w)$ (Zeitbedarf: $p(|w|)$).
2. Simuliere die Maschine N auf $f(w)$ (Zeitbedarf: $q(p(|w|))$).

Der gesamte Zeitbedarf ist also $p(|w|) + q(p(|w|))$, was wieder ein Polynom ist.

Die Aussage für die Klasse NP kann genauso bewiesen werden. □

NP-Vollständigkeit

Eine Sprache A ist NP-hart, falls für alle $B \in \text{NP}$ gilt: $B \leq_p A$
(A ist mindestens so schwer wie jedes Problem in NP).

Eine Sprache A ist NP-vollständig, falls sie zu NP gehört und NP-hart ist.

Intuition: NP-vollständige Sprachen sind die schwierigsten Sprachen in NP.

Noch wissen wir garnicht, ob es überhaupt NP-vollständige Sprachen gibt.
Dies werden wir bald zeigen.

Komplexitätstheorie

Zunächst aber noch ein einfaches Resultat:

Lemma 42

Wenn A NP-vollständig ist, dann gilt: $\mathrm{P} = \mathrm{NP} \iff A \in \mathrm{P}$.

Beweis:

$\Rightarrow$: Sei $\mathrm{P} = \mathrm{NP}$.

Da A NP-vollständig ist, folgt $A \in \mathrm{NP} = \mathrm{P}$.

$\Leftarrow$: Sei $A \in \mathrm{P}$ und sei $B \in \mathrm{NP}$ beliebig.

Da A NP-vollständig ist, folgt $B \leq_p A \in \mathrm{P}$.

Lemma 41 impliziert $B \in \mathrm{P}$.

Also gilt $\mathrm{NP} \subseteq \mathrm{P}$ und damit $\mathrm{NP} = \mathrm{P}$. □

Komplexitätstheorie

Wir werden bald ein konkretes NP-vollständiges Probleme kennenlernen: das Erfüllbarkeitsproblem für aussagenlogische Formeln (SAT).

Für viele weitere Probleme A kann dann die NP-Vollständigkeit durch eine Reduktion SAT $\leq_p A$ gezeigt werden.

Hier sind einige Beispiele für NP-vollständige Probleme
(ohne Beweis; siehe Schöning für Beweise).

SUBSETSUM

EINGABE: Binär-kodierte Zahlen $t, w_1, \ldots, w_n$
FRAGE: Gibt es eine Teilmenge $U \subseteq \{w_1, \ldots w_n\}$ mit $t = \sum_{w \in U} w$?

CLIQUE

EINGABE: Ein ungerichteter Graph $G = (V, E)$ (siehe Vorlesung *Diskrete Mathematik*) und eine Zahl k (unär kodiert)

FRAGE: Hat G eine Clique der Größe k, d. h. existiert eine Menge $U \subseteq V$ mit $|U| \geq k$ und für alle $u, v \in U$ mit $u \neq v$: $\{u, v\} \in E$?

VERTEX-COVER

EINGABE: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl k (unär kodiert)

FRAGE: Hat G eine Knotenüberdeckung der Größe k, d. h. existiert eine Menge $U \subseteq V$ mit $|U| \leq k$ und für alle $\{u, v\} \in E$ gilt $U \cap \{u, v\} \neq \emptyset$?

3-FÄRBBARKEIT

EINGABE: Ein ungerichteter Graph $G = (V, E)$

FRAGE: Ist die Färbungszahl von G höchstens 3.

HAMILTON-CIRCUIT

EINGABE: Ein ungerichteter Graph $G = (V, E)$

FRAGE: Hat G einen Hamiltonkreis (siehe Vorlesung *Diskrete Mathematik*)?

Logik

Wir kommen später nochmals zur Komplexitätstheorie zurück (und zeigen, dass SAT NP-vollständig ist)

Zunächst wollen wir uns aber mit der Logik beschäftigen.

Intuitiv gesprochen ist eine Logik eine Sprache, mittels derer sich formale Sachverhalte (z.B. Aussagen aus der Mathematik, Korrektheitsaussagen für Programme, etc) formulieren lassen.

Zum Teil haben wir solche logischen Aussagen auch bereits verwendet.

Wir werden zwei wichtige (wohl die wichtigsten) Logiken kennenlernen:

- Aussagenlogik
- Prädikatenlogik

Logik

Das Vorgehen bei der Einführung einer neuen Logik ist immer das gleiche:

- Zunächst definieren wir die Syntax der Logik. Dabei definieren wir eine Sprache von syntaktisch korrekten Formeln.
- Danach definieren wir die Semantik der Logik, d.h. wir definieren, wann eine Formel wahr bzw. falsch ist.

Syntax der Aussagenlogik

Eine atomare Formel hat die Form A_i (wobei $i = 1, 2, 3, \ldots$).
Formeln werden durch folgenden induktiven Prozeß definiert:

1. Alle atomaren Formeln sind Formeln
2. Falls F und G Formeln sind, sind auch $(F \wedge G)$ und $(F \vee G)$ Formeln.
3. Falls F eine Formel ist, ist auch $\neg F$ eine Formel.

Sprechweise:

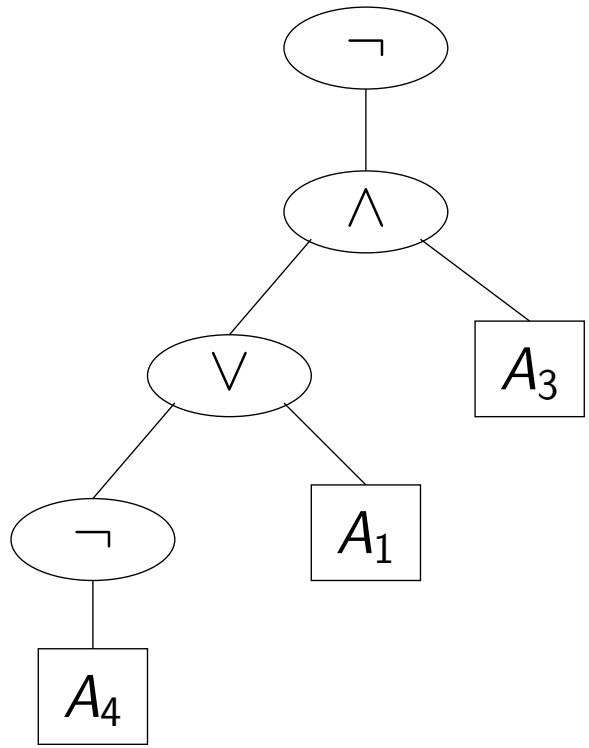
- $(F \wedge G)$: F und G, Konjunktion von F und G
- $(F \vee G)$: F oder G, Disjunktion von F und G
- $\neg F$: nicht F, Negation von F

Beispiel: $\neg((\neg A_4 \vee A_1) \wedge A_3)$ ist eine Formel.

Formel als Syntaxbaum

Jede Formel kann auch durch einen Syntaxbaum dargestellt werden.

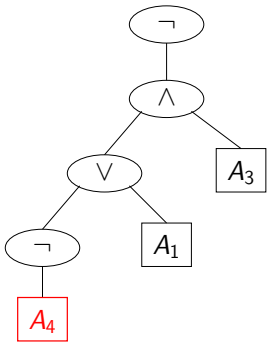
Beispiel: $F = \neg((\neg A_4 \vee A_1) \wedge A_3)$



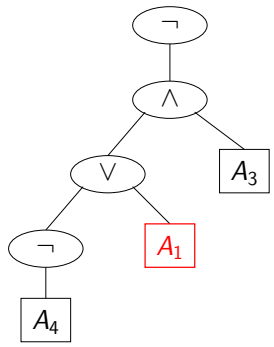
Teilformel

Die Teilformeln einer Formel F entsprechen dann den Teilbäumen.

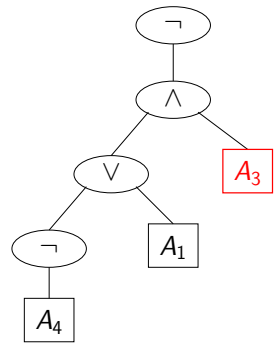
A_4



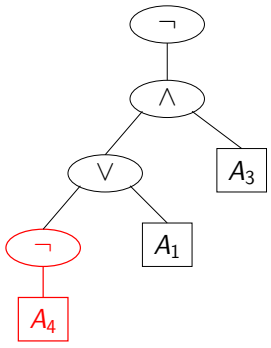
A_1



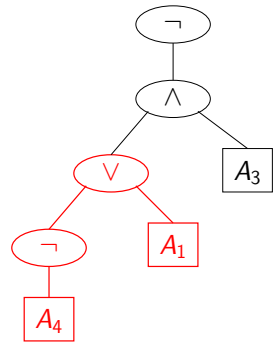
A_3



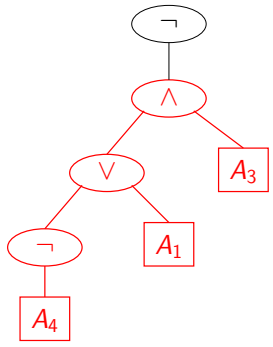
$\neg A_4$



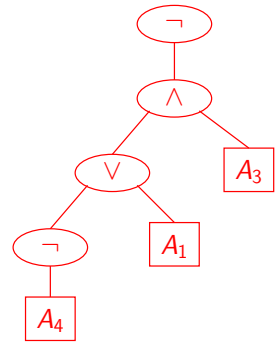
$(\neg A_4 \vee A_1)$



$((\neg A_4 \vee A_1) \wedge A_3)$



$\neg((\neg A_4 \vee A_1) \wedge A_3)$



Semantik der Aussagenlogik: Belegungen

Die Elemente der Menge $\{0, 1\}$ heißen Wahrheitswerte.

Eine Belegung ist eine Funktion $\mathcal{B}\colon D \to \{0, 1\}$, wobei $D \subseteq \{A_1, A_2, A_3, \ldots\}$ eine Teilmenge der atomaren Formeln ist.

Auf der nächsten Folie erweitern wir $\mathcal{B}$ zu einer Funktion $\widehat{\mathcal{B}}\colon E \to \{0, 1\}$, wobei $E \supseteq D$ die Menge aller Formeln ist, die nur aus den atomaren Formeln in D aufgebaut sind.

Beispiel: Sei $D = \{A_1, A_5, A_8\}$.

Dann gilt $F = \neg((\neg A_5 \vee A_1) \wedge A_8) \in E$ aber $\neg((\neg A_4 \vee A_1) \wedge A_3) \notin E$.

Ein mögliche Wahrheitsbelegung könnte definiert werden durch: $\mathcal{B}(A_1) = 1$, $\mathcal{B}(A_5) = 0$, $\mathcal{B}(A_8) = 1$.

Frage: Was ist wohl $\widehat{\mathcal{B}}(F)$?

Semantik der Aussagenlogik: Belegungen

$$
\begin{aligned}
\widehat{\mathcal{B}}(A) &= \mathcal{B}(A) \quad \text{falls } A \in D \text{ eine atomare Formel ist} \\
\widehat{\mathcal{B}}((F \wedge G)) &= \begin{cases} 1 & \text{falls } \widehat{\mathcal{B}}(F) = 1 \text{ und } \widehat{\mathcal{B}}(G) = 1 \\ 0 & \text{sonst} \end{cases} \\
\widehat{\mathcal{B}}((F \vee G)) &= \begin{cases} 1 & \text{falls } \widehat{\mathcal{B}}(F) = 1 \text{ oder } \widehat{\mathcal{B}}(G) = 1 \\ 0 & \text{sonst} \end{cases} \\
\widehat{\mathcal{B}}(\neg F) &= \begin{cases} 1 & \text{falls } \widehat{\mathcal{B}}(F) = 0 \\ 0 & \text{sonst} \end{cases}
\end{aligned}
$$

Wir schreiben im folgenden $\mathcal{B}$ anstatt $\widehat{\mathcal{B}}$.

Verknüpfungstafeln für $\wedge, \vee$, und $\neg$

Berechnung von $\mathcal{B}$ mit Hilfe von Verknüpfungstafeln, auch Wahrheitstafeln genannt.

Beobachtung: Der Wert $\mathcal{B}(F)$ hängt nur davon ab, wie $\mathcal{B}$ auf den den in F vorkommenden atomaren Formeln definiert ist.

Tafeln für die Operatoren $\vee$, $\wedge$, $\neg$:

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	$\neg A$
0	1
1	0

Abkürzungen

A, B, C oder
P, Q, R oder ... statt $A_1, A_2, A_3 \ldots$

$(F_1 \to F_2)$ statt $(\neg F_1 \vee F_2)$

$(F_1 \leftrightarrow F_2)$ statt $((F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2))$

$(\bigvee_{i=1}^{n} F_i)$ statt $(\ldots((F_1 \vee F_2) \vee F_3) \vee \ldots \vee F_n)$

$(\bigwedge_{i=1}^{n} F_i)$ statt $(\ldots((F_1 \wedge F_2) \wedge F_3) \wedge \ldots \wedge F_n)$

Verknüpfungstafeln für $\rightarrow$ und $\leftrightarrow$

Verknüpfungstafeln für die Operatoren $\rightarrow$, $\leftrightarrow$:

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Name: *Implikation*

Interpretation: Wenn A gilt, dann muß auch B gelten.

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

Name: *Äquivalenz*

Interpretation: A gilt genau dann, wenn B gilt.

Achtung!!!

$A \to B$ sagt nicht, dass A eine Ursache für B ist.

"Pinguine schwimmen $\to$ Hunde bellen"
ist wahr (in unserer Welt).

$A \to B$ sagt nichts darüber, ob A wahr oder falsch ist.

"$x = y \to 2x = 2y$"
ist wahr für alle Zahlen x und y

Eine falsche Aussage impliziert alles.

"Pinguine fliegen $\to$ Katzen bellen"
ist wahr (in unserer Welt).

Formalisierung natürlicher Sprache (I)

Ein Gerät besteht aus einem Bauteil A, einem Bauteil B und einem roten Licht. Folgendes ist bekannt:

- Bauteil A oder Bauteil B (oder beide) sind kaputt.
- Wenn Bauteil A kaputt ist, dann ist auch Bauteil B kaputt.
- Wenn Bauteil B kaputt ist und das rote Licht leuchtet, dann ist Bauteil A nicht kaputt.
- Das rote Licht leuchtet.

Formalisieren Sie diese Situation als aussagenlogische Formel und stellen Sie die Wahrheitstafel zu dieser Formel auf. Verwenden Sie dazu folgende atomare Formeln: RL (rotes Licht leuchtet), AK (Bauteil A kaputt), BK (Bauteil B kaputt)

Formalisierung natürlicher Sprache (II)

Gesamte Wahrheitstafel:

RL	AK	BK	$(AK \vee BK) \wedge (AK \rightarrow BK) \wedge ((BK \wedge RL) \rightarrow \neg AK) \wedge RL$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Formalisierung von Sudoku

Formalisieren Sie das Sudoku-Problem:

4				9				2
		1				5		
	9		3	4	5		1	
		8				2	5	
7		5		3		4	6	1
	4	6				9		8
	6		1	5	9		8	
		9				6		
5				7				4

Verwenden Sie dazu eine atomare Formel $A[n, x, y]$ für jedes Tripel $(n, x, y) \in \{1, \ldots, 9\}^3$:

$A[n, x, y] = 1$, falls: Auf der Zeile x, Spalte y liegt die Zahl n.

Formalisierung von Sudoku

Beispiel: In der erste Zeile stehen alle Zahlen von 1 bis 9

$$\bigwedge_{n=1}^{9} \left(\bigvee_{y=1}^{9} A[n, 1, y] \right)$$

Die Wahrheitstabelle hat

$$2^{729} = 282401395870821749694910884220462786335135391185$$
$$157752468340193086269383036119849990587392099522$$
$$999697089786549828399657812329686587839094762655$$
$$308848694610643079609148271612057263207249270352$$
$$772375735947883453036573491 2$$

Zeilen. Warum?

Modelle

Sei F eine Formel und $\mathcal{B}$ eine Belegung.
Falls $\mathcal{B}$ für alle in F vorkommenden atomaren Formeln definiert ist so heißt $\mathcal{B}$ zu F passend.

Sei $\mathcal{B}$ passend zu F:

Falls $\mathcal{B}(F) = 1$	so schreiben wir	$\mathcal{B} \models F$
	und sagen	F gilt unter $\mathcal{B}$
	oder	$\mathcal{B}$ ist ein Modell für F
Falls $\mathcal{B}(F) = 0$	so schreiben wir	$\mathcal{B} \not\models F$
	und sagen	F gilt nicht unter $\mathcal{B}$
	oder	$\mathcal{B}$ ist kein Modell für F

Gültigkeit und Erfüllbarkeit

Erfüllbarkeit: Eine Formel F heißt erfüllbar, falls F mindestens ein Modell besitzt, andernfalls heißt F unerfüllbar.

Eine (endliche oder unendliche!) Menge von Formeln M heißt erfüllbar, falls es eine Belegung gibt, die für jede Formel in M ein Modell ist.

Gültigkeit: Eine Formel F heißt gültig (oder allgemeingültig oder Tautologie) falls jede zu F passende Belegung ein Modell für F ist. Wir schreiben $\models F$, falls F gültig ist, und $\not\models F$ sonst.

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A			
$A \vee B$			
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N		
$A \vee B$			
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	
$A \vee B$			
$A \vee \neg A$			
$A \wedge \neg A$			
$A \to \neg A$			
$A \to B$			
$A \to (B \to A)$			
$A \to (A \to B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$			
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N		
$A \vee \neg A$			
$A \wedge \neg A$			
$A \to \neg A$			
$A \to B$			
$A \to (B \to A)$			
$A \to (A \to B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$			
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J		
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$			
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N		
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$			
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N		
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$			
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N		
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$			
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J		
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$			
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N		
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N	J	
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N	J	N
$A \leftrightarrow \neg A$			

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N	J	N
$A \leftrightarrow \neg A$	N		

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N	J	N
$A \leftrightarrow \neg A$	N	N	

Aufgabe

	Gültig	Erfüllbar	Unerfüllbar
A	N	J	N
$A \vee B$	N	J	N
$A \vee \neg A$	J	J	N
$A \wedge \neg A$	N	N	J
$A \rightarrow \neg A$	N	J	N
$A \rightarrow B$	N	J	N
$A \rightarrow (B \rightarrow A)$	J	J	N
$A \rightarrow (A \rightarrow B)$	N	J	N
$A \leftrightarrow \neg A$	N	N	J

Aufgabe

Gelten die folgenden Aussagen?

				J/N	Gegenb.
Wenn	F gültig,	dann	F erfüllbar		
Wenn	F erfüllbar,	dann	$\neg F$ unerfüllbar		
Wenn	F gültig,	dann	$\neg F$ unerfüllbar		
Wenn	F unerfüllbar,	dann	$\neg F$ gültig		

Aufgabe

Gelten die folgenden Aussagen?

				J/N	Gegenb.
Wenn	F gültig,	dann	F erfüllbar	J	
Wenn	F erfüllbar,	dann	$\neg F$ unerfüllbar		
Wenn	F gültig,	dann	$\neg F$ unerfüllbar		
Wenn	F unerfüllbar,	dann	$\neg F$ gültig		

Aufgabe

Gelten die folgenden Aussagen?

				J/N	Gegenb.
Wenn	F gültig,	dann	F erfüllbar	J	
Wenn	F erfüllbar,	dann	$\neg F$ unerfüllbar	N	$F = A$
Wenn	F gültig,	dann	$\neg F$ unerfüllbar		
Wenn	F unerfüllbar,	dann	$\neg F$ gültig		

Aufgabe

Gelten die folgenden Aussagen?

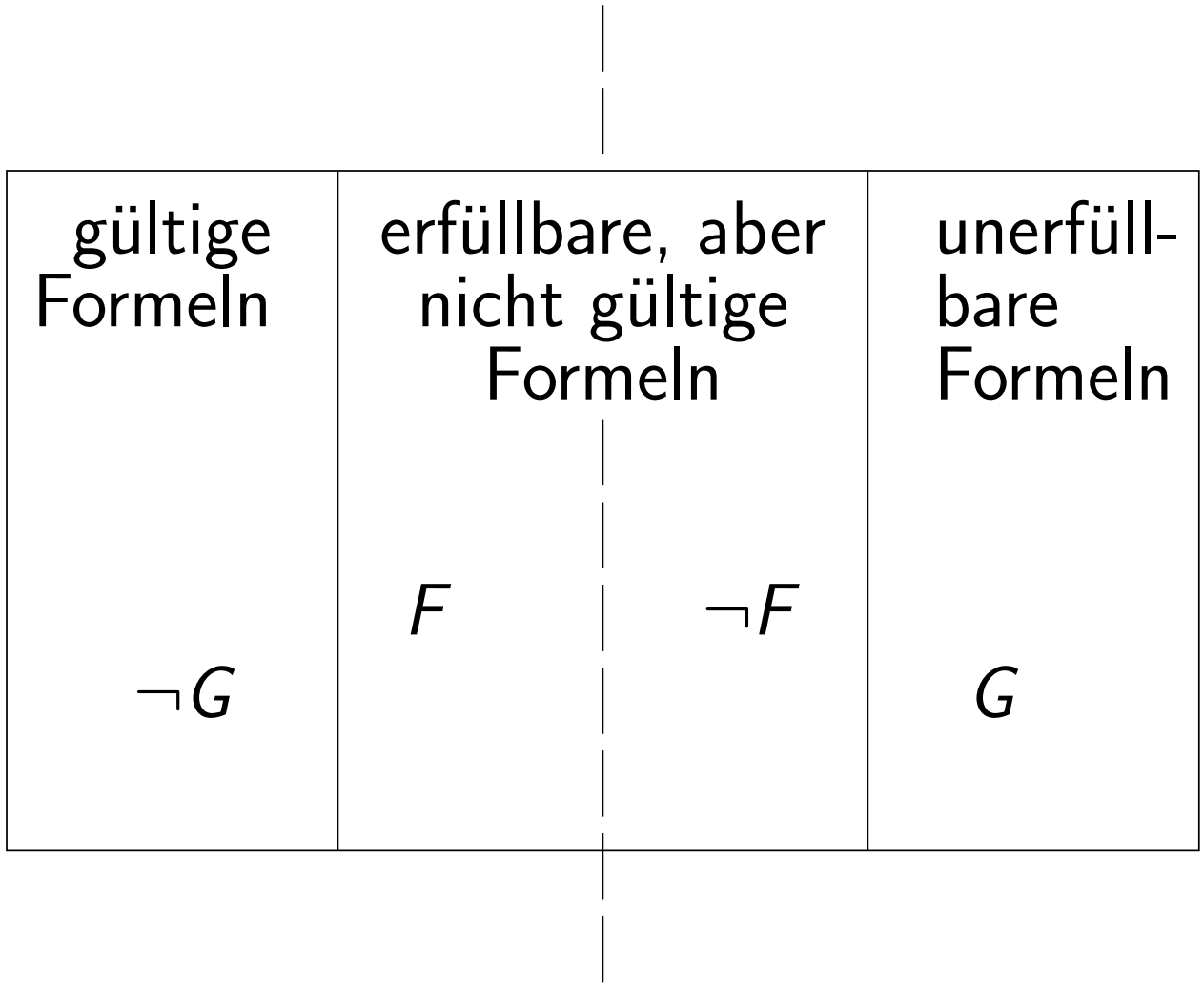
				J/N	Gegenb.
Wenn	F gültig,	dann	F erfüllbar	J	
Wenn	F erfüllbar,	dann	$\neg F$ unerfüllbar	N	$F = A$
Wenn	F gültig,	dann	$\neg F$ unerfüllbar	J	
Wenn	F unerfüllbar,	dann	$\neg F$ gültig		

Aufgabe

Gelten die folgenden Aussagen?

				J/N	Gegenb.
Wenn	F gültig,	dann	F erfüllbar	J	
Wenn	F erfüllbar,	dann	$\neg F$ unerfüllbar	N	$F = A$
Wenn	F gültig,	dann	$\neg F$ unerfüllbar	J	
Wenn	F unerfüllbar,	dann	$\neg F$ gültig	J	

Spiegelungsprinzip



Ein Gültigkeitstest

Wie kann man überprüfen, ob eine Formel F gültig/erfüllbar ist?

Eine Möglichkeit: Wahrheitstafel aufstellen

Angenommen, die Formel F enthält n verschiedene atomare Formeln. Wie groß ist die Wahrheitstafel?

Anzahl Zeilen in der Wahrheitstafel: 2^n

Geht es auch effizienter?

Wahrscheinlich nicht: Erfüllbarkeit von aussagenlogischen Formeln ist NP-vollständig und damit nicht in polynomieller Zeit möglich, es sei denn P = NP.

Komplexitätstheorie: SAT

Das Problem SAT

EINGABE: Eine aussagenlogische Formel F

FRAGE: Ist F erfüllbar?

Aussagenlogische Formeln lassen sich z. B. durch Wörter über dem Alphabet $\{a, \vee, \wedge, \neg,), (\}$ kodieren (atomare Formeln werden durch Wörter der Form a^n kodiert).

Satz 43

SAT $\in$ NP

Komplexitätstheorie: SAT

Beweis: Sei F eine aussagenlogische Formel, in der die atomaren Formeln $A_1, \ldots, A_n$ vorkommen.

Eine nichtdeterministische Turingmaschine "rät" nun in einer ersten Phase eine Belegung $\mathcal{B} : \{A_1, \ldots, A_n\} \to \{0, 1\}$:

- Im ersten Schritt verzweigt sich die Turingmaschine (d. h. es gibt zwei Folgekonfigurationen).
  Im ersten Zweig schreibt die Turingmaschine $A_1 0$ auf das Band, im zweiten Zweig schreibt sie $A_1 1$ auf das Band.
- Im zweiten Schritt verzweigt sich die Turingmaschine wieder.
  Im ersten Zweig schreibt sie (hinter $A_1 b$ mit $b \in \{0, 1\}$) $A_2 0$ auf das Band, im zweiten Zweig schreibt sie $A_2 1$ auf das Band.
  $\vdots$

Komplexitätstheorie: Der Satz von Cook

Nach n Schritten steht in jedem der 2^n Berechnungszweige ein Wort der Form $A_1 b_1 A_2 b_2 \cdots A_n b_n$ mit $b_1, \ldots, b_n \in \{0, 1\}$ auf dem Band.

Dieses Wort kodiert die Belegung $\mathcal{B}$ mit $\mathcal{B}(A_i) = b_i$ für $1 \leq i \leq n$.

In einer zweiten Phase kann die Turingmaschine nun den Wert $\mathcal{B}(F)$ deterministisch ausrechnen, indem die Formel F einmal von links nach rechts durchlaufen wird und jedesmal, wenn eine atomare Formel A_i gesehen wird, der Wert $\mathcal{B}(A_i) = b_i$ aus dem gespeicherten "Belegungswort" ermitelt wird.

Dies benötigt höchstens $|F|^2$ Schritte, die Turingmaschine rechnet also auf jedem Berechnungspfad nur $O(|F|^2)$ Schritte.

Die Maschine gibt am Ende 1 aus, genau dann, wenn $\mathcal{B}(F) = 1$.

Also gibt es einen Berechnungspfad, auf dem die Maschine 1 ausgibt, genau dann, wenn F erfüllbar ist. □

Komplexitätstheorie: Der Satz von Cook

Satz 44 (Satz von Cook)

SAT ist NP-vollständig.

Beweis:

Wir müssen noch zeigen, dass SAT NP-hart ist.

Sei hierfür $L \in \mathrm{NP}$, $L \subseteq \Sigma^*$.

Zu $w \in \Sigma^*$ konstruieren wir eine aussagenlogische Formel $f(w)$ mit:
$w \in L \iff f(w)$ ist erfüllbar.

Die Abbildung f wird polynomiell berechenbar sein.

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine nichtdeterministische Turingmaschine für L, die bei einer Eingabe der Länge n auf jedem Berechnungspfad nach $\leq p(n)$ Schritten terminiert ($p(n)$ ist ein Polynom).

Sei $w = w_1 w_2 \cdots w_n \in \Sigma^*$ eine Eingabe der Länge n.

Komplexitätstheorie: Der Satz von Cook

Wir stellen o.B.d.A. folgende Forderungen an M:

1. Der Kopf von M wandert nie links von der Position, wo er zu Beginn steht (kann durch spezielle Markierung erreicht werden).
2. M terminiert genau dann mit Ausgabe 1, wenn M in den speziellen Zustand $z_1 \in E$ übergeht. D. h. der Zustand z_1 signalisiert Akzeptanz der Eingabe.
3. Alle Tupel der Form (z_1, a, z_1, a, N) (mit $a \in \Gamma$) gehören zu δ.
4. $(z, a, z', a', D), (z, b, z'', b', D') \in \delta \implies a = b, a' = b', D = D'$

   Nur hinsichtlich des Folgezustands z' haben wir also eine nichtdeterministische Wahl (siehe nächste Folie).

Komplexitätstheorie: Der Satz von Cook

Eigenschaft (4) kann wie folgt erzwungen werden.

Zunächst können wir die Sprache L durch $\$L$ für ein neues Symbol \$ ersetzen, da $L \leq_p \$L$ gilt (d. h. aus $\$L \leq_p$ SAT folgt wieder $L \leq_p$ SAT).

Wir wissen also, dass alle positiven Eingaben mit einem \$ beginnen müssen.

Definiere nun die Zustandsmenge und die Transitionsrelation wie folgt um:

$$
\begin{aligned}
Z' =& \{(z[a, a', D] \mid z \in Z,\ a, a' \in \Gamma,\ D \in \{L, R, N\}\} \cup \{z_0[\$, \$, R]\} \\
\delta' =& \{(z[a, a', D], a, z'[b, b', D'], a', D) \mid (z, a, z', a', D) \in \delta, \\
& \qquad b, b' \in \Gamma, D' \in \{L, R, N\}\} \cup \\
& \{(z_0[\$, \$, R], \$, z_0[a, a', D], \$, R) \mid a, a' \in \Gamma,\ D \in \{L, R, N\}\}
\end{aligned}
$$

Der neue Anfangszustand ist $z_0[\$, \$, R]$.

Komplexitätstheorie: Der Satz von Cook

Jede von der Startkonfiguration erreichbare Konfiguration kann durch ein Wort aus

$$\text{Conf} = \{\Box uzv\Box \mid z \in Z;\ u, v \in \Gamma^*;\ |uv| = p(n)\}$$

beschrieben werden.

Die Startkonfiguration ist $\Box z_0 w \Box^{p(n)+1-n}$.

Wegen Punkt (2) und (3) akzeptiert M auf einem bestimmten Berechnungspfad die Eingabe w genau dann, wenn sich M nach $p(n)$ vielen Schritten im Zustand z_1 befindet.

Notation: Für ein $\alpha \in \text{Conf}$ schreiben wir

$$\alpha = \alpha[-1]\alpha[0]\cdots\alpha[p(n)]\alpha[p(n)+1]$$

wobei $\alpha[-1] = \Box$, $\alpha[0], \ldots, \alpha[p(n)] \in Z \cup \Gamma$, $\alpha[p(n)+1] = \Box$.

Komplexitätstheorie: Der Satz von Cook

Definiere die Menge der 4-Tupel

$$\begin{aligned}\Delta = &\{(a,b,c,b) \mid a,b,c \in \Gamma\}\\ &\cup \{(c,b,z,z'),(b,z,a,b),(z,a,d,a') \mid (z,a,z',a',L) \in \delta, c,b,d \in \Gamma\}\\ &\cup \{(c,b,z,b),(b,z,a,z'),(z,a,d,a') \mid (z,a,z',a',N) \in \delta, c,b,d \in \Gamma\}\\ &\cup \{(c,b,z,b),(b,z,a,a'),(z,a,d,z') \mid (z,a,z',a',R) \in \delta, c,b,d \in \Gamma\}\end{aligned}$$

Idee der Δ-Tupel: Wenn drei aufeinanderfolgende Positionen $i-1$, i, $i+1$ einer Konfiguration α die Symbole $x, y, z \in Z \cup \Gamma$ enthalten, dann muss für jede Folgekonfiguration α' von α ein Tupel (x, y, z, y') existieren, in der Position i das Symbol y' enthält.

Wegen Punkt (4) gilt dann für alle $\alpha, \alpha' \in \square(Z \cup \Gamma)^*\square$ mit $|\alpha| = |\alpha'|$:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \text{Conf und } \forall i \in \{0,\ldots,p(n)\} : (\alpha[i-1], \alpha[i], \alpha[i+1], \alpha'[i]) \in \Delta.$$

Komplexitätstheorie: Der Satz von Cook

Beispiel:

Falls $(z, a, z', a', L) \in \delta$ ist folgende lokale Bandänderung für alle $b \in \Gamma$ möglich:

Position			i–1	i	i+1		
α	$=$	$\cdots$	$\cdots$	b	z	a	$\cdots$	$\cdots$
α'	$=$	$\cdots$	$\cdots$	z'	b	a'	$\cdots$	$\cdots$

Falls $(z, a, z', a', R) \in \delta$ ist folgende lokale Bandänderung für alle $b \in \Gamma$ möglich:

Position			i–1	i	i+1		
α	$=$	$\cdots$	$\cdots$	b	z	a	$\cdots$	$\cdots$
α'	$=$	$\cdots$	$\cdots$	b	a'	z'	$\cdots$	$\cdots$

Komplexitätstheorie: Der Satz von Cook

Eine Rechnung von M können wir nun als Matrix beschreiben:

$$\begin{array}{lllllll}
\alpha_0 & = & \square & \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,p(n)} \quad \square \\
\alpha_1 & = & \square & \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,p(n)} \quad \square \\
 & & & & \vdots & & \\
\alpha_{p(n)} & = & \square & \alpha_{p(n),0} & \alpha_{p(n),1} & \cdots & \alpha_{p(n),p(n)} \quad \square
\end{array}$$

Für jedes Tripel (a, i, t) ($a \in Z \cup \Gamma$, $-1 \leq i \leq p(n) + 1$, $0 \leq t \leq p(n)$) sei $x(a, i, t)$ eine aussagenlogische Variable (atomare Formel).

Interpretation: $x(a, i, t) =$ **true** genau dann, wenn zum Zeitpunkt t das i-te Zeichen der aktuellen Konfiguration ein a ist.

Komplexitätstheorie: Der Satz von Cook

An den Positionen -1 und $p(n)+1$ steht immer $\Box$:

$$G(n) \quad = \quad \bigwedge_{0 \leq t \leq p(n)} \Big(x(\Box, -1, t) \wedge x(\Box, p(n)+1, t) \Big)$$

Für jedes Paar (i, t) ist genau eine Variable $x(a, i, t)$ wahr (zu jedem Zeitpunkt kann auf einem Bandfeld nur ein Zeichen stehen):

$$X(n) \quad = \quad \bigwedge_{\substack{0 \leq t \leq p(n) \\ -1 \leq i \leq p(n)+1}} \left(\bigvee_{a \in Z \cup \Gamma} \left(x(a, i, t) \wedge \bigwedge_{b \neq a} \neg x(b, i, t) \right) \right)$$

Komplexitätstheorie: Der Satz von Cook

Zum Zeitpunkt $t = 0$ ist die Konfiguration gleich $\Box z_0 w \Box^{p(n)+1-n}$

$$S(w) \quad = \quad \left(x(z_0, 0, 0) \wedge \bigwedge_{i=1}^{n} x(w_i, i, 0) \wedge \bigwedge_{i=n+1}^{p(n)} x(\Box, i, 0) \right)$$

Die Berechnung respektiert die lokale Relation Δ:

$$D(n) \quad = \quad \bigwedge_{\substack{0 \leq i \leq p(n) \\ 0 \leq t < p(n)}} \quad \bigvee_{(a,b,c,d) \in \Delta} \left(\begin{array}{l} x(a, i-1, t) \ \wedge \ x(b, i, t) \wedge \\ x(c, i+1, t) \ \wedge \ x(d, i, t+1) \end{array} \right)$$

Komplexitätstheorie: Der Satz von Cook

Sei schließlich

$$R(w) = G(n) \wedge X(n) \wedge S(w) \wedge D(n).$$

Es ergibt sich eine natürliche Bijektion zwischen der Menge der $R(w)$ erfüllenden Belegungen und der Menge derjenigen Rechnungen von M auf die Eingabe w, die aus $p(n)$ Rechenschritten bestehen.

Für $f(w) = R(w) \wedge \bigvee_{i=0}^{p(n)} x(z_1, i, p(n))$ gilt somit:

$$f(w) \text{ erfüllbar} \quad \Longleftrightarrow \quad w \in L.$$

Zahl der Variablen von $f(w) \in \mathcal{O}(p(n)^2)$

Länge von $f(w) \in \mathcal{O}(p(n)^2 \log p(n))$

Der Faktor $\mathcal{O}(\log p(n))$ ist notwendig, da zum Aufschreiben der Indizes $\log p(n)$ viele Bits benötigt werden. □

Folgerung

Eine Formel G heißt eine Folgerung der Formeln $F_1, \ldots, F_k$ falls für jede Belegung $\mathcal{B}$, die sowohl zu $F_1, \ldots, F_k$ als auch zu G passend ist, gilt:

> *Wenn $\mathcal{B}$ Modell von $\{F_1, \ldots, F_k\}$ ist (d.h. Modell von F_1 und Modell von F_2 und ... und Modell von F_k), dann ist $\mathcal{B}$ auch Modell von G.*

Wir schreiben $F_1, \ldots, F_k \models G$, falls G eine Folgerung von $F_1, \ldots, F_k$ ist.

Folgerung: Beispiel

$$(AK \vee BK), (AK \rightarrow BK),$$
$$((BK \wedge RL) \rightarrow \neg AK), RL \quad \models \quad (RL \wedge \neg AK) \wedge BK$$

Wenn Bauteil A oder Bauteil B kaputt ist *und* daraus, dass Bauteil A kaputt ist, immer folgt, dass Bauteil B kaputt ist *und* ...

... dann kann man die Folgerung ziehen: das rote Licht leuchtet, Bauteil A ist nicht kaputt und Bauteil B ist kaputt.

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	
A	$A \wedge B$	
A, B	$A \vee B$	
A, B	$A \wedge B$	
$A \wedge B$	A	
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	
A, B	$A \vee B$	
A, B	$A \wedge B$	
$A \wedge B$	A	
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	
A, B	$A \wedge B$	
$A \wedge B$	A	
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	J
A, B	$A \wedge B$	
$A \wedge B$	A	
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	J
A, B	$A \wedge B$	J
$A \wedge B$	A	
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	J
A, B	$A \wedge B$	J
$A \wedge B$	A	J
$A \vee B$	A	
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	J
A, B	$A \wedge B$	J
$A \wedge B$	A	J
$A \vee B$	A	N
$A, A \rightarrow B$	B	

Aufgabe

M	F	Gilt $M \models F$?
A	$A \vee B$	J
A	$A \wedge B$	N
A, B	$A \vee B$	J
A, B	$A \wedge B$	J
$A \wedge B$	A	J
$A \vee B$	A	N
$A, A \rightarrow B$	B	J

Folgerung, Gültigkeit und Unerfüllbarkeit

Satz 45

Folgende Aussagen sind äquivalent:

1. $F_1, \ldots, F_k \models G$, *d.h.,* G *ist eine Folgerung von* $F_1, \ldots, F_k$.
2. $((\bigwedge_{i=1}^{k} F_i) \rightarrow G)$ *ist gültig.*
3. $((\bigwedge_{i=1}^{k} F_i) \wedge \neg G)$ *ist unerfüllbar.*

Folgerung, Gültigkeit und Unerfüllbarkeit

Beweis:

1 $\Rightarrow$ **2**: Gelte $F_1, \ldots, F_k \models G$.

Behauptung: $((\bigwedge_{i=1}^{k} F_i) \rightarrow G)$ ist gültig.

Sei $\mathcal{B}$ eine beliebige zu $((\bigwedge_{i=1}^{k} F_i) \rightarrow G)$ passende Belegung.

1.Fall: Es gibt ein $i \in \{1, \ldots, k\}$ mit $\mathcal{B}(F_i) = 0$:

Dann gilt auch $\mathcal{B}(\bigwedge_{i=1}^{k} F_i) = 0$ und somit $\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \rightarrow G) = 1$.

2.Fall: Für alle $i \in \{1, \ldots, k\}$ gilt $\mathcal{B}(F_i) = 1$:

Aus $F_1, \ldots, F_k \models G$ folgt $\mathcal{B}(G) = 1$ und somit auch
$\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \rightarrow G) = 1$.

Folgerung, Gültigkeit und Unerfüllbarkeit

2 $\Rightarrow$ **3**: Sei $((\bigwedge_{i=1}^{k} F_i) \rightarrow G)$ gültig.

Behauptung: $((\bigwedge_{i=1}^{k} F_i) \wedge \neg G)$ ist unerfüllbar.

Sei $\mathcal{B}$ eine beliebige Belegung.

1.Fall: $\mathcal{B}(G) = 1$:

Dann gilt $\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \wedge \neg G) = 0$.

2.Fall: $\mathcal{B}(\bigwedge_{i=1}^{k} F_i) = 0$:

Dann gilt wieder $\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \wedge \neg G) = 0$.

3.Fall: $\mathcal{B}(\bigwedge_{i=1}^{k} F_i) = 1$ und $\mathcal{B}(G) = 0$:

Dann gilt $\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \rightarrow G) = 0$, dies widerspricht jedoch der Tatsache, dass $((\bigwedge_{i=1}^{k} F_i) \rightarrow G)$ gültig ist.

Also kann Fall 3 nicht eintreten.

Folgerung, Gültigkeit und Unerfüllbarkeit

3 $\Rightarrow$ **1**: Sei $((\bigwedge_{i=1}^{k} F_i) \wedge \neg G)$ unerfüllbar.

Behauptung: $F_1, \ldots, F_k \models G$

Sei $\mathcal{B}$ eine beliebige Belegung mit $\mathcal{B}(F_i) = 1$ für alle $i \in \{1, \ldots, k\}$.

Da $((\bigwedge_{i=1}^{k} F_i) \wedge \neg G)$ unerfüllbar ist, muss $\mathcal{B}(G) = 1$ gelten (sonst wäre $\mathcal{B}((\bigwedge_{i=1}^{k} F_i) \wedge \neg G) = 1$).

□

Äquivalenz

Zwei Formeln F und G heißen (semantisch) äquivalent, falls für alle Belegungen $\mathcal{B}$, die sowohl für F als auch für G passend sind, gilt $\mathcal{B}(F) = \mathcal{B}(G)$. Hierfür schreiben wir $F \equiv G$.

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{aligned}
(A \wedge (A \vee B)) &\equiv A \\
\neg(A \vee B) &\equiv (\neg A \wedge \neg B) \\
(A \wedge (B \vee C)) &\equiv ((A \wedge B) \vee C) \\
(A \wedge (B \vee C)) &\equiv ((A \wedge B) \vee (A \wedge C)) \\
(A \to B) \to C &\equiv A \to (B \to C) \\
(A \to B) \to C &\equiv (A \wedge B) \to C \\
(A \leftrightarrow B) \leftrightarrow C &\equiv A \leftrightarrow (B \leftrightarrow C)
\end{aligned}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & \\
(A \to B) \to C & \equiv & A \to (B \to C) & \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & \\
(A \to B) \to C & \equiv & A \to (B \to C) & \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & N : \mathcal{B}(A) = 0, \mathcal{B}(C) = 1 \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & \\
(A \to B) \to C & \equiv & A \to (B \to C) & \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & N : \mathcal{B}(A) = 0, \mathcal{B}(C) = 1 \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & J \\
(A \to B) \to C & \equiv & A \to (B \to C) & \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & N : \mathcal{B}(A) = 0, \mathcal{B}(C) = 1 \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & J \\
(A \to B) \to C & \equiv & A \to (B \to C) & N : \mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = 0 \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & N : \mathcal{B}(A) = 0, \mathcal{B}(C) = 1 \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & J \\
(A \to B) \to C & \equiv & A \to (B \to C) & N : \mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = 0 \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & N : \mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = 0 \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) &
\end{array}
$$

Aufgabe

Gelten die folgenden Äquivalenzen?

$$
\begin{array}{rcll}
(A \wedge (A \vee B)) & \equiv & A & J \\
\neg(A \vee B) & \equiv & (\neg A \wedge \neg B) & J \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee C) & N : \mathcal{B}(A) = 0, \mathcal{B}(C) = 1 \\
(A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) & J \\
(A \to B) \to C & \equiv & A \to (B \to C) & N : \mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = 0 \\
(A \to B) \to C & \equiv & (A \wedge B) \to C & N : \mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = 0 \\
(A \leftrightarrow B) \leftrightarrow C & \equiv & A \leftrightarrow (B \leftrightarrow C) & J
\end{array}
$$

Aufgabe

Wahrheitstafeln für $(A \leftrightarrow B) \leftrightarrow C$ und $A \leftrightarrow (B \leftrightarrow C)$

A	B	C	$(A \leftrightarrow B) \leftrightarrow C$	$A \leftrightarrow (B \leftrightarrow C)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Die Hauptprobleme der Aussagenlogik

In der "informatischen" Aussagenlogik sucht man nach Verfahren, die folgende Aufgaben (Probleme) lösen:

- Modellprüfung
  Sei F eine Formel und sei $\mathcal{B}$ eine passende Belegung. Gilt $\mathcal{B}(F) = 1$?
- Erfüllbarkeit (SAT)
  Sei F eine Formel. Ist F erfüllbar ?
- Gültigkeit
  Sei F eine Formel. Ist F gültig ?
- Folgerung
  Seien F und G Formeln. Gilt $F \models G$?
- Äquivalenz
  Seien F und G Formeln. Gilt $F \equiv G$?

Aufgabe

Zeigen Sie, dass die folgenden Aussagen gelten:

Wenn	$(F \to G)$ gültig	dann	$F \models G$.
Wenn	$F \models G$	dann	$(F \to G)$ gültig.
Wenn	$(F \leftrightarrow G)$ gültig	dann	$F \equiv G$.
Wenn	$F \equiv G$	dann	$(F \leftrightarrow G)$ gültig.

Reduktion von Problemen

Welche Probleme lassen sich auf welche (polynomiell) reduzieren ($A \equiv_p B$ steht für $A \leq_p B \leq_p A$)?

- Gültigkeit $\equiv_p$ (Nichterfüllbarkeit (Komplement von SAT):
  F gültig genau dann, wenn $\neg F$ nicht erfüllbar
  F erfüllbar genau dann, wenn $\neg F$ nicht gültig.
- Gültigkeit $\leq_p$ Folgerung:
  F gültig genau dann, wenn $T \models F$ (T beliebige gültige Formel).
- Folgerung $\leq_p$ Gültigkeit:
  $F \models G$ genau dann, wenn $F \rightarrow G$ gültig.
- Gültigkeit $\leq_p$ Äquivalenz:
  F gültig genau dann, wenn $F \equiv T$ (T beliebige gültige Formel).
- Äquivalenz $\leq_p$ Gültigkeit:
  $F \equiv G$ genau dann, wenn $F \leftrightarrow G$ gültig.

Einschub: Äquivalenzrelationen

Sei R eine binäre Relation auf der Menge A, d. h. $R \subseteq A \times A$.

- R ist reflexiv, falls für alle $a \in A$ gilt: $(a, a) \in R$.
- R ist symmetrisch, falls für alle $a, b \in A$ gilt:
  Wenn $(a, b) \in R$, dann auch $(b, a) \in R$.
- R ist transitiv, falls für alle $a, b, c \in A$ gilt:
  Wenn $(a, b) \in R$ und $(b, c) \in R$, dann auch $(a, c) \in R$.

Eine reflexive, symmetrische und transitive Relation wird auch als Äquivalenzrelation bezeichnet.

Für eine binäre Relation R schreiben wir im folgenden auch $a\,R\,b$ anstatt $(a, b) \in R$ (Infixschreibweise).

Beispiel: Für eine natürliche Zahl $k \geq 1$ definieren wir die binäre Relation $\equiv_k$ auf $\mathbb{Z}$: $n \equiv_k m$ genau dann, wenn $n - m$ durch k teilbar ist.

Übung: Zeigen Sie, dass $\equiv_k$ für jedes $k \geq 1$ eine Äquivalenzrelation ist.

Einschub: Kongruenzrelationen

Sei f ein n-stelliger Operator auf A, d. h. $f : A^n \to A$, wobei $A^n = \{(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in A\}$.

Die binäre Relation $R \subseteq A \times A$ ist abgeschlossen unter dem Operator f, falls gilt:

Für alle $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in A^n$ gilt:
Wenn $a_1 \, R \, b_1$ und $\ldots$ $a_n \, R \, b_n$, dann auch $f(a_1, \ldots, a_n) \, R \, f(b_1, \ldots, b_n)$.

Man sagt auch, dass R und f verträglich sind.

Seien $f_1, \ldots, f_n$ Operatoren auf A (beliebiger Stelligkeit).
R ist eine Kongurenzrelation auf A (bezüglich $f_1, \ldots, f_n$), falls gilt:

- R ist eine Äquivalenzrelationen.
- R ist abgeschlossen unter $f_1, \ldots, f_n$.

Beispiel: $\equiv_k$ ist eine Kongruenzrelation auf $\mathbb{Z}$ bezüglich der 2-stelligen Operatoren $+$ und $\cdot$ (mal).

Äquivalenz ist eine Kongurenzrelation

Die Äquivalenz $\equiv$ von Formeln ist eine binäre Relation auf der Menge aller Formeln: Sei $\mathcal{F}$ die Menge aller Formeln. Dann gilt $\equiv \subseteq \mathcal{F} \times \mathcal{F}$.

$\wedge$ und $\vee$ sind 2-stellige Operatoren auf $\mathcal{F}$.
$\neg$ ist ein 1-stelliger Operator auf $\mathcal{F}$.

Die Äquivalenz $\equiv$ ist eine Kongruenzrelation auf der Menge aller Formeln (bezüglich der Operatoren $\wedge, \vee$ und $\neg$):

reflexiv: Es gilt $F \equiv F$ für jede Formel F (jede Formel ist zu sich selbst äquivalent)

symmetrisch: Falls $F \equiv G$ gilt, so gilt auch $G \equiv F$

transitiv: Falls $F \equiv G$ und $G \equiv H$ gilt, so gilt auch $F \equiv H$

abgeschlossen unter Operatoren: Falls $F_1 \equiv F_2$ und $G_1 \equiv G_2$ gilt, so gilt auch $(F_1 \wedge G_1) \equiv (F_2 \wedge G_2)$, $(F_1 \vee G_1) \equiv (F_2 \vee G_2)$ und $\neg F_1 \equiv \neg F_2$.

Ersetzbarkeitstheorem

Die Abgeschlossenheit läßt sich auch folgendermaßen formulieren:

Ersetzbarkeitstheorem

Seien F und G äquivalente Formeln. Sei H eine Formel mit (mindestens) einem Vorkommen der Teilformel F. Dann ist H äquivalent zu H', wobei H' aus H hervorgeht, indem (irgend-) ein Vorkommen von F in H durch G ersetzt wird.

Beweis des Ersetzbarkeitstheorems

Beweis (durch Induktion über den Formelaufbau von H):

Induktionsanfang: Falls H eine atomare Formel ist, dann kann nur $H = F$ sein. Und damit ist klar, dass H äquivalent zu H' ist, denn $H' = G$.

Induktionsschritt: Falls F gerade H selbst ist, so trifft dieselbe Argumentation wie im Induktionsanfang zu.

Nehmen wir also an, dass F eine Teilformel von H mit $F \neq H$ ist. Dann müssen wir drei Fälle unterscheiden.

Beweis des Ersetzbarkeitstheorems

Fall 1: H hat die Bauart $H = \neg H_1$.
Nach Induktionsvoraussetzung ist H_1 äquivalent zu H_1', wobei H_1' aus H_1 durch Ersetzung von F durch G hervorgeht.
Nun ist aber $H' = \neg H_1'$.
Aus der (semantischen) Definition von „$\neg$“ folgt dann, dass H und H' äquivalent sind.

Fall 2: H hat die Bauart $H = (H_1 \vee H_2)$.

Dann kommt F entweder in H_1 oder H_2 vor. Nehmen wir den ersteren Fall an (der zweite ist völlig analog).
Dann ist nach Induktionssannahme H_1 wieder äquivalent zu H_1', wobei H_1' aus H_1 durch Ersetzung von F durch G hervorgeht.
Mit der Definition von „$\vee$“ ist dann klar, dass $H \equiv (H_1' \vee H_2) = H'$.

Fall 3: H hat die Bauart $H = (H_1 \wedge H_2)$.
Diesen Fall beweist man völlig analog zu *Fall 2*.

Äquivalenzen (I)

Satz

Es gelten die folgenden Äquivalenzen:

$$
\begin{array}{rcll}
(F \wedge F) & \equiv & F & \\
(F \vee F) & \equiv & F & \text{(Idempotenz)} \\
 & & & \\
(F \wedge G) & \equiv & (G \wedge F) & \\
(F \vee G) & \equiv & (G \vee F) & \text{(Kommutativität)} \\
 & & & \\
((F \wedge G) \wedge H) & \equiv & (F \wedge (G \wedge H)) & \\
((F \vee G) \vee H) & \equiv & (F \vee (G \vee H)) & \text{(Assoziativität)} \\
 & & & \\
(F \wedge (F \vee G)) & \equiv & F & \\
(F \vee (F \wedge G)) & \equiv & F & \text{(Absorption)} \\
 & & & \\
(F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) & \\
(F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) & \text{(Distributivität)}
\end{array}
$$

Äquivalenzen (II)

Satz

Es gelten die folgenden Äquivalenzen:

$$\begin{array}{rcll}
\neg\neg F & \equiv & F & \text{(Doppelnegation)} \\
\\
\neg(F \wedge G) & \equiv & (\neg F \vee \neg G) & \\
\neg(F \vee G) & \equiv & (\neg F \wedge \neg G) & \text{(deMorgansche Regeln)} \\
\\
(F \vee G) & \equiv & F, \text{ falls } F \text{ Tautologie} & \\
(F \wedge G) & \equiv & G, \text{ falls } F \text{ Tautologie} & \text{(Tautologieregeln)} \\
\\
(F \vee G) & \equiv & G, \text{ falls } F \text{ unerfüllbar} & \\
(F \wedge G) & \equiv & F, \text{ falls } F \text{ unerfüllbar} & \text{(Unerfüllbarkeitsregeln)}
\end{array}$$

Beweis: Übung

Normalformen: **KNF**

Definition (Normalformen)
Ein Literal ist eine atomare Formel oder die Negation einer atomaren Formel. Im ersten Fall sprechen wir von einem positiven, im zweiten Fall von einem negativen Literal.

Eine Formel F ist in konjunktiver Normalform (**KNF**), falls sie eine Konjunktion von Disjunktionen von Literalen ist:

$$F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j})),$$

wobei $L_{i,j} \in \{A_1, A_2, \ldots\} \cup \{\neg A_1, \neg A_2, \ldots\}$

Beispiel: $A_1 \wedge \neg A_2 \wedge (\neg A_1 \vee A_2 \vee \neg A_3) \wedge (A_2 \vee A_4)$

Normalformen: **DNF**

Eine Formel F ist in disjunktiver Normalform (**DNF**), falls sie eine Disjunktion von Konjunktionen von Literalen ist:

$$F = (\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} L_{i,j})),$$

wobei $L_{i,j} \in \{A_1, A_2, \ldots\} \cup \{\neg A_1, \neg A_2, \ldots\}$

Beispiel: $A_1 \vee \neg A_2 \vee (\neg A_1 \wedge A_2 \wedge \neg A_3) \vee (A_2 \wedge A_4)$

Satz

Zu jeder Formel F existiert eine äquivalente Formel in KNF, sowie eine äquivalente Formel in DNF.

Methode 1: Ablesen aus Wahrheitstafel

Für eine atomare Formel A_i definiere

$$A_i^0 := \neg A_i \quad \text{und} \quad A_i^1 := A_i.$$

Für eine Formel F, in der genau die atomaren Formeln $A_1, \ldots, A_n$ vorkommen, definiere

$$\mathrm{DNF}(F) := \bigvee_{\substack{\mathcal{B}:\{A_1,\ldots,A_n\}\to\{0,1\},\\ \mathcal{B}(F)=1}} \bigwedge_{i=1}^{n} A_i^{\mathcal{B}(A_i)}$$

$$\mathrm{KNF}(F) := \bigwedge_{\substack{\mathcal{B}:\{A_1,\ldots,A_n\}\to\{0,1\},\\ \mathcal{B}(F)=0}} \bigvee_{i=1}^{n} A_i^{1-\mathcal{B}(A_i)}$$

Lemma 46

Für jede Formel F gilt: $F \equiv \mathrm{DNF}(F) \equiv \mathrm{KNF}(F)$.

Methode 1: Ablesen aus Wahrheitstafel

Beweis: Wir zeigen $F \equiv \mathrm{KNF}(F)$, $F \equiv \mathrm{DNF}(F)$ folgt analog.

Sei $\mathcal{B}' : \{A_1, \ldots, A_n\} \to \{0, 1\}$ eine beliebige Belegung.

Wir zeigen: $\mathcal{B}'(F) = 0$ genau dann, wenn $\mathcal{B}'(\mathrm{KNF}(F)) = 0$.

1. Sei $\mathcal{B}'(F) = 0$.

Behauptung: Für alle $i \in \{1, \ldots, n\}$ gilt $\mathcal{B}'(A_i^{1-\mathcal{B}'(A_i)}) = 0$
(dies gilt für jede passende Belegung):

Fall A: $\mathcal{B}'(A_i) = 0$.

Dann gilt $A_i^{1-\mathcal{B}'(A_i)} = A_i^1 = A_i$, und somit $\mathcal{B}'(A_i^{1-\mathcal{B}'(A_i)}) = \mathcal{B}'(A_i) = 0$.

Fall B: $\mathcal{B}'(A_i) = 1$.

Dann gilt $A_i^{1-\mathcal{B}'(A_i)} = A_i^0 = \neg A_i$, und somit $\mathcal{B}'(A_i^{1-\mathcal{B}'(A_i)}) = \mathcal{B}'(\neg A_i) = 0$.

Methode 1: Ablesen aus Wahrheitstafel

Aus der Behauptung folgt $\mathcal{B}'(\bigvee_{i=1}^{n} A_i^{1-\mathcal{B}'(A_i)}) = 0$.

Da $\mathcal{B}'(F) = 0$ gilt, ist $\mathcal{B}'$ eine der Belegungen, über die in $\mathrm{KNF}(F)$ außen das große $\bigwedge$ gebildet wird.

Also gibt es eine Formel G, so dass

$$\mathrm{KNF}(F) \equiv G \wedge \bigvee_{i=1}^{n} A_i^{1-\mathcal{B}'(A_i)}$$

Wegen $\mathcal{B}'(\bigvee_{i=1}^{n} A_i^{1-\mathcal{B}'(A_i)}) = 0$ gilt $\mathcal{B}'(\mathrm{KNF}(F)) = 0$.

Methode 1: Ablesen aus Wahrheitstafel

2. Sei $\mathcal{B}'(\mathrm{KNF}(F)) = 0$.

Aus

$$\mathrm{KNF}(F) = \bigwedge_{\substack{\mathcal{B}:\{A_1,\ldots,A_n\}\to\{0,1\},\\ \mathcal{B}(F)=0}} \bigvee_{i=1}^{n} A_i^{1-\mathcal{B}(A_i)}$$

folgt, dass eine der Disjunktionen in $\mathrm{KNF}(F)$ unter $\mathcal{B}'$ gleich 0 ist.

Es gibt somit eine Belegung $\mathcal{B}$ mit: $\mathcal{B}(F) = 0$ und $\mathcal{B}'(\bigvee_{i=1}^{n} A_i^{1-\mathcal{B}(A_i)}) = 0$.

Also: $\mathcal{B}'(A_i^{1-\mathcal{B}(A_i)}) = 0$ für alle $i \in \{1, \ldots, n\}$.

Dies impliziert $\mathcal{B}'(A_i) = \mathcal{B}(A_i)$ für alle $i \in \{1, \ldots, n\}$ und somit $\mathcal{B}'(F) = 0$ (wegen $\mathcal{B}(F) = 0$). □

Methode 1: Ablesen aus Wahrheitstafel

Beachte:

- Ist F unerfüllbar, d. h. $\mathcal{B}(F) = 0$ für alle passenden Belegungen $\mathcal{B}$, so ist $\mathrm{DNF}(F)$ die leere Disjunktion. Diese soll eine unerfüllbare Formel sein.
- Ist F gültig, d. h. $\mathcal{B}(F) = 1$ für alle passenden Belegungen $\mathcal{B}$, so ist $\mathrm{KNF}(F)$ die leere Konjunktion. Diese soll eine Tautologie sein.

Methode 1: Beispiel

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

DNF: Aus jeder Zeile mit Wahrheitswert 1 wird eine Konjunktion, aus einer 0 in der Spalte A wird $\neg A$, aus einer 1 wird A.

$$(\neg A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge C)$$
$$\vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge C)$$

KNF: Aus jeder Zeile mit Wahrheitswert 0 wird eine Disjunktion, aus einer 0 in der Spalte A wird A, aus einer 1 wird $\neg A$.

$$(A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C)$$
$$\wedge (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee C)$$

Methode 2: syntaktisches Umformen

Gegeben: eine Formel F.
Wir bilden die KNF von F wie folgt:

1. Ersetze in F jedes Vorkommen einer Teilformel der Bauart

$$\begin{array}{rcl} \neg\neg G & \text{durch} & G \\ \neg(G \wedge H) & \text{durch} & (\neg G \vee \neg H) \\ \neg(G \vee H) & \text{durch} & (\neg G \wedge \neg H) \end{array}$$

   bis keine derartige Teilformel mehr vorkommt.

2. Ersetze jedes Vorkommen einer Teilformel der Bauart

$$\begin{array}{rcl} (F \vee (G \wedge H)) & \text{durch} & ((F \vee G) \wedge (F \vee H)) \\ ((F \wedge G) \vee H) & \text{durch} & ((F \vee H) \wedge (G \vee H)) \end{array}$$

   bis keine derartige Teilformel mehr vorkommt.

Mengendarstellung

Eine Klausel ist eine Disjunktion von Literalen.

Die Klausel $L_1 \vee L_2 \vee \cdots \vee L_n$, wobei $L_1, \ldots, L_n$ Literale sind, wird auch mit der Menge $\{L_1, \ldots, L_n\}$ identifiziert.

Eine Formel in **KNF** (= Konjunktion von Klauseln) wird mit einer Menge von Klauseln (d.h. Menge von Mengen von Literalen) identifiziert:
$(\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j}))$ wird mit $\{\{L_{i,j} \mid 1 \leq j \leq m_i\} \mid 1 \leq i \leq n\}$ identifiziert.

Die leere Klausel (= leere Disjunktion) ist äquivalent zu einer unerfüllbaren Formel.

Die leere **KNF**-Formel (= leere Konjunktion) ist äquivalent zu einer gültigen Formel.

Mengendarstellung

Beispiel: Die Mengenschreibweise der KNF

$$A_1 \wedge \neg A_2 \wedge (\neg A_1 \vee A_2 \vee \neg A_3) \wedge (A_2 \vee A_4)$$

ist $\{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2, \neg A_3\}, \{A_2, A_4\}\}$.

Präzedenzen

Präzedenz der Operatoren:

$\leftrightarrow$	bindet am schwächsten
$\rightarrow$	...
$\vee$	...
$\wedge$	...
$\neg$	bindet am stärksten

Es gilt also:

$$A \leftrightarrow B \vee \neg C \rightarrow D \wedge \neg E \equiv (A \leftrightarrow ((B \vee \neg C) \rightarrow (D \wedge \neg E)))$$

Dennoch: Zu viele Klammern schaden i.A. nicht.

Erfüllbarkeit und Gültigkeit in DNF und KNF

Erfüllbarkeit ist leicht (lösbar in linearer Zeit) für Formeln in DNF:

> *Eine Formel in DNF ist erfüllbar genau dann, wenn es eine Konjunktion gibt, die nicht gleichzeitig A und $\neg A$ für eine atomare Formel A enthält.*

Erfüllbar: $(\neg B \wedge A \wedge B) \vee (\neg A \wedge C)$
Nicht erfüllbar: $(A \wedge \neg A \wedge B) \vee (C \wedge \neg C)$

Gültigkeit ist leicht (lösbar in linearer Zeit) für Formeln in KNF:

> *Eine Formel in KNF ist gültig genau dann, wenn jede Disjunktion gleichzeitig A und $\neg A$ für eine atomare Formel A enthält. (Oder es handelt sich um die leere Konjunktion.)*

Gültig: $(A \vee \neg A \vee B) \wedge (C \vee \neg C)$
Nicht gültig: $(A \vee \neg A) \wedge (\neg A \vee C)$

Erfüllbarkeit von KNF-Formeln

Der Satz von Cook kann wie folgt verschärft werden:

3-KNF-SAT

EINGABE: Eine aussagenlogische Formel F in KNF, bei der jede Klausel aus höchstens 3 Literalen (atomare Formeln oder negierte atomare Formeln) besteht.
FRAGE: Ist F erfüllbar?

Satz 47

3-KNF-SAT ist NP-vollständig

Bemerkung: 2-KNF-SAT $\in$ P

Effiziente Erfüllbarkeitstests

Im folgenden:

- Ein sehr effizienter Erfüllbarkeitstest für eine spezielle Klasse von Formeln, sogenannte Hornformeln
- Ein im allgemeinen effizienter Unerfüllbarkeitstest für Formeln in KNF (Resolution)

Hornformel

Eine Formel F ist eine Hornformel (benannt nach Alfred Horn, 1918–2001), falls F in **KNF** ist, und jede Klausel in F höchstens ein positives Literal enthält.

Notation:

$$\begin{array}{rll} (\neg A \vee \neg B \vee C) & \text{wird zu} & (A \wedge B \rightarrow C) \\ (\neg A \vee \neg B) & \text{wird zu} & (A \wedge B \rightarrow 0) \\ A & \text{wird zu} & (1 \rightarrow A) \end{array}$$

0: steht für eine beliebige unerfüllbare Formel
1: steht für eine beliebige gültige Formel

Allgemein:

$$\begin{array}{rcl} \neg A_1 \vee \cdots \vee \neg A_k \vee B & \equiv & \neg(A_1 \wedge \cdots \wedge A_k) \vee B \equiv (A_1 \wedge \cdots \wedge A_k) \rightarrow B \\ \neg A_1 \vee \cdots \vee \neg A_k & \equiv & \neg(A_1 \wedge \cdots \wedge A_k) \vee 0 \equiv (A_1 \wedge \cdots \wedge A_k) \rightarrow 0 \end{array}$$

$(A_1 \wedge \cdots \wedge A_k) \rightarrow B$ (bzw. $(A_1 \wedge \cdots \wedge A_k) \rightarrow 0$) ist die implikative Form der Klausel $\neg A_1 \vee \cdots \vee \neg A_k \vee B$ (bzw. $\neg A_1 \vee \cdots \vee \neg A_k$).

Erfüllbarkeitstest für Hornformeln

Markierungsalgorithmus

Eingabe: eine Hornformel F.

(1) Versehe jedes Vorkommen einer atomaren Formel A in F mit einer Markierung, falls es in F eine Teilformel der Form $(1 \to A)$ gibt;

(2) **while** es gibt in F eine Teilformel G der Form $(A_1 \wedge \ldots \wedge A_k \to B)$ oder $(A_1 \wedge \ldots \wedge A_k \to 0)$, $k \geq 1$, wobei $A_1, \ldots, A_k$ bereits markiert sind und B noch nicht markiert ist **do**:

    if G hat die erste Form **then**

        markiere jedes Vorkommen von B

    else gib "unerfüllbar" aus und stoppe;

    endwhile

(3) Gib "erfüllbar" aus und stoppe.

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Beispiele für den Markierungsalgorithmus

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_4) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist erfüllbar.

Wir wenden den Markierungsalgorithmus auf folgende Hornformel an:

$$(1 \to A_1) \wedge (1 \to A_2) \wedge (A_1 \wedge A_2 \to A_3) \wedge (A_2 \wedge A_3 \to A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \to 0)$$

Die Formel ist nicht erfüllbar.

Induktionsprinzip

Um die Aussage

Für jedes $n \in \{0, 1, 2, 3, \ldots\}$ *gilt* $P(n)$.

zu zeigen, gehen wir im allgemeinen folgendermaßen vor:

- Wir zeigen, dass $P(0)$ gilt. (Induktionsanfang)
- Wir zeigen, dass für jedes n gilt:
  Wenn $P(n)$ gilt, dann gilt auch $P(n+1)$. (Induktionsschritt)

Dann kann man schließen, dass $P(n)$ für jedes beliebige n gilt.

Alternatives Induktionsprinzip: Wir zeigen, dass für jedes n gilt:
Wenn $P(k)$ für alle $k < n$ gilt, dann gilt auch $P(n)$.

Anwendung: Beweis, dass eine Bedingung während des Ablaufs eines Algorithmus immer erfüllt ist (Invariante). Hierzu zeigt man durch Induktion, dass die Bedingung nach n Schritten des Algorithmus erfüllt ist.

Korrektheit des Markierungsalgorithmus

Satz 48

Der Markierungsalgorithmus ist korrekt und terminiert immer nach spätestens n Markierungschritten.
Dabei ist n die Anzahl der atomaren Formeln in der Eingabeformel F.

Beweis:

(A) Algorithmus terminiert:

Nach spätestens n Schritten sind alle atomare Formeln markiert.

(B) Wenn der Algorithmus eine atomare Formel A markiert, dann gilt $\mathcal{B}(A) = 1$ für jede erfüllende Belegung $\mathcal{B}$ von F.

Beweis von (B) mittels Induktion:

1.Fall: atomare Formel A wird in Schritt (1) markiert: klar

Korrektheit des Markierungsalgorithmus

2.Fall: atomare Formel A wird in Schritt (2) markiert:

Dann gibt es eine Teilformel $(A_1 \wedge \ldots \wedge A_k \rightarrow A)$, so dass $A_1, \ldots, A_k$ zu frühreren Zeitpunkten markiert wurden.

Also gilt $\mathcal{B}(A_1) = \cdots = \mathcal{B}(A_k) = 1$ für jede erfüllende Belegung $\mathcal{B}$ von F.

Dann muss aber auch $\mathcal{B}(A) = 1$ für jede erfüllende Belegung $\mathcal{B}$ von F gelten.

Dies beweist (B).

Korrektheit des Markierungsalgorithmus

(C) Wenn der Algorithmus "unerfüllbar" ausgibt, dann ist F unerfüllbar.

Sei $(A_1 \wedge \ldots \wedge A_k \rightarrow 0)$ die Teilformel von F, die die Ausgabe "unerfüllbar" verursacht.

Nach (B) gilt $\mathcal{B}(A_1) = \cdots = \mathcal{B}(A_k) = 1$ für jede erfüllende Belegung $\mathcal{B}$ von F.

Aber für solche Belegungen gilt: $\mathcal{B}(A_1 \wedge \cdots \wedge A_k \rightarrow 0) = 0$ und damit $\mathcal{B}(F) = 0$.

Also kann es keine erfüllende Belegung von F geben.

(D) Wenn der Algorithmus "erfüllbar" ausgibt, dann ist F erfüllbar.

Angenommen, der Algorithmus gibt "erfüllbar" aus.

Definiere eine Belegung $\mathcal{B}$ wie folgt:

$$\mathcal{B}(A_i) = \begin{cases} 1 & \text{der Algoritmus markiert } A_i \\ 0 & \text{sonst} \end{cases}$$

Wir behaupten, dass die Belegung $\mathcal{B}$ die Konjunktion F erfüllt:

- In $(A_1 \wedge \ldots \wedge A_k \rightarrow B)$ ist B markiert oder mindestens ein A_i nicht markiert.
- In $(A_1 \wedge \ldots \wedge A_k \rightarrow 0)$ ist mindestens ein A_i nicht markiert (sonst hätte der Algorithmus mit "unerfüllbar" terminiert).

□

Bemerkung: Mit einer geeigneten Implementierung läuft der Algorithmus in linearer Zeit.

Beispiel: MYCIN

MYCIN: Expertensystem zur Untersuchung von Blutinfektionen (entwickelt in den 70er Jahren)

Beispiel:
IF the infection is pimary-bacteremia AND the site of the culture is one of the sterile sites AND the suspected portal of entry is the gastrointestinal tract THEN there is suggestive evidence (0.7) that infection is bacteroid.

Resolution (Idee)

Resolution ist ein Verfahren, mit dem man feststellen kann, ob eine Formel F in **KNF** unerfüllbar ist.

Idee: $(F \vee A) \wedge (F' \vee \neg A) \equiv (F \vee A) \wedge (F' \vee \neg A) \wedge (F \vee F')$

Aus der Herleitung der leeren Disjunktion (= leere Klausel) folgt Unerfüllbarkeit.

Zwei Fragen:

- Kann man aus einer unerfüllbaren Formel immer die leere Klausel herleiten? (Vollständigkeit)
- Gibt es eine Möglichkeit, die Herleitung kompakter aufzuschreiben?

Mengendarstellung

Zur Erinnerung:

- Klausel: Menge von Literalen (Disjunktion).

  $\{A, B\}$ stellt $(A \vee B)$ dar.

- Formel in **KNF**: Menge von Klauseln (Konjunktion von Klauseln).

  $\{\{A, B\}, \{\neg A, B\}\}$ stellt $((A \vee B) \wedge (\neg A \vee B))$ dar.

Die leere Klausel (= leere Disjunktion) ist äquivalent zu einer unerfüllbaren Formel.
Diese wird auch mit $\square$ bezeichnet.

Die leere KNF-Formel (= leere Konjunktion) ist äquivalent zu einer gültigen Formel.

Beachte: Die KNF-Formel $\{\}$ (= leere Konjunktion = Tautologie) ist zu unterscheiden von der Formel $\{\square\}$ (= unerfüllbare Formel).

Vorteile der Mengendarstellung

Man erhält automatisch:

- Kommutativität:
  $(A \vee B) \equiv (B \vee A)$,
  beide dargestellt durch $\{A, B\}$
- Assoziativität:
  $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$,
  beide dargestellt durch $\{A, B, C\}$
- Idempotenz:
  $(A \vee A) \equiv A$,
  beide dargestellt durch $\{A\}$

Resolvent

Definition: Seien K_1, K_2 und R Klauseln. Dann heißt R Resolvent von K_1 und K_2, falls es ein Literal L gibt mit $L \in K_1$ und $\overline{L} \in K_2$ und R die folgende Form hat:

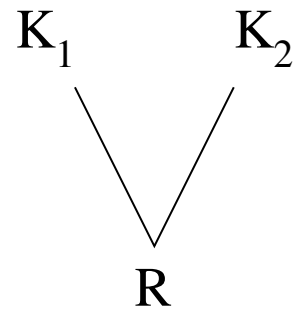
$$R = (K_1 \setminus \{L\}) \cup (K_2 \setminus \{\overline{L}\}).$$

Hierbei ist $\overline{L}$ definiert als

$$\overline{L} = \begin{cases} \neg A_i & \text{falls } L = A_i \text{ für ein } i \geq 1, \\ A_i & \text{falls } L = \neg A_i \text{ für ein } i \geq 1 \end{cases}$$

Resolvent

Wir stellen diesen Sachverhalt durch folgendes Diagramm dar



Sprechweise: R wird aus K_1, K_2 nach L resolviert.

Ferner: falls $K_1 = \{L\}$ und $K_2 = \{\overline{L}\}$, so entsteht die leere Menge als Resolvent. Diese wird mit dem speziellen Symbol $\square$ bezeichnet, das eine unerfüllbare Formel darstellt.

Beispiel: Alle Resolventen von $\{A, \neg B, \neg C\}$ und $\{\neg A, B, D\}$:

$$\{\neg B, B, \neg C, D\} \text{ sowie } \{A, \neg A, \neg C, D\}$$

Resolutionslemma

Resolutionslemma

Sei F eine Formel in **KNF**, dargestellt als Klauselmenge. Ferner sei R ein Resolvent zweier Klauseln K_1 und K_2 in F. Dann sind F und $F \cup \{R\}$ äquivalent.

Beweis: Folgt direkt aus

$$\underbrace{(F_1 \vee A)}_{K_1} \wedge \underbrace{(F_2 \vee \neg A)}_{K_2} \equiv \underbrace{(F_1 \vee A)}_{K_1} \wedge \underbrace{(F_2 \vee \neg A)}_{K_2} \wedge \underbrace{(F_1 \vee F_2)}_{R}$$

Resolutionshülle

Definition: Sei F eine Menge von Klauseln. Dann ist $Res(F)$ definiert als

$$Res(F) = F \cup \{R \mid R \text{ ist Resolvent zweier Klauseln in } F\}.$$

Außerdem setzen wir:

$$\begin{aligned} Res^0(F) &= F \\ Res^{n+1}(F) &= Res(Res^n(F)) \qquad \text{für } n \geq 0 \end{aligned}$$

und schließlich sei

$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F).$$

$Res^*(F)$ wird auch als die Resolutionshülle von F bezeichnet.

Aus dem Resolutionslemma folgt sofort

$$F \equiv Res^*(F).$$

Resolutionshülle

Angenommen, die Formel F (in **KNF**) enthält n atomare Formeln.
Wie groß kann dann $Res^*(F)$ höchstens werden?

(A) $|Res^*(F)| \leq 2^n$

(B) $|Res^*(F)| \leq 4^n$

(C) $|Res^*(F)|$ kann unendlich werden

Dabei bezeichnet $|Res^*(F)|$ die Anzahl der Elemente in $Res^*(F)$.

Beispiel: Die Resolutionshülle von $\{\{A, \neg B, \neg C\}, \{\neg A, B, D\}\}$ besteht aus folgenden Klauseln:

$$\{A, \neg B, \neg C\}, \{\neg A, B, D\},$$
$$\{\neg B, B, \neg C, D\}, \{A, \neg A, \neg C, D\}$$
$$\{A, \neg B, \neg C, D\}, \{\neg A, B, \neg C, D\}$$

Resolutionssatz

Wir zeigen nun die Korrektheit und Vollständigkeit der Resolution:

Resolutionssatz der Aussagenlogik

Eine endliche Menge F von Klauseln ist unerfüllbar genau dann, wenn $\square \in Res^*(F)$.

Beweis:

Korrektheit: Wenn $\square \in Res^*(F)$, dann ist F unerfüllbar.

Sei $\square \in Res^*(F)$.

Aus dem Resolutionslemma folgt $F \equiv Res^*(F)$.

Da $\square$ unerfüllbar ist, ist auch $Res^*(F)$ und somit F unerfüllbar.

Beweis des Resolutionssatz (Vollständigkeit)

Vollständigkeit: Wenn F unerfüllbar ist, dann gilt $\square \in Res^*(F)$.

Sei F im folgenden unerfüllbar.

Wir beweisen die Vollständigkeit durch eine Induktion über die Anzahl $n(F)$ der atomaren Formeln, die in F vorkommen.

Induktionsanfang: $n(F) = 0$. Dann muss $F = \{\square\}$ gelten. Also gilt: $\square \in F \subseteq Res^*(F)$.

Induktionsschritt: Sei $n(F) > 0$.

Wähle eine beliebige in F vorkommende atomare Formel A aus.
Wir definieren aus F die Formel F_0 wie folgt:

$$F_0 = \{K \setminus \{A\} \mid K \in F, \neg A \notin K\}.$$

Intuition: F_0 entsteht aus F indem A durch 0 ersetzt wird, und die "offensichtlichen" Vereinfachungen durchgeführt werden.

Beweis des Resolutionssatz (Vollständigkeit)

Analog definieren wir aus F Formel F_1:

$$F_1 = \{K \setminus \{\neg A\} \mid K \in F, A \notin K\}.$$

Intuition: F_1 entsteht aus F indem A durch 1 ersetzt wird, und die "offensichtlichen" Vereinfachungen durchgeführt werden.

Da F unerfüllbar ist, sind auch F_0 und F_1 unerfüllbar:

Würde z. B. F_0 durch die Belegung $\mathcal{B}$ erfüllt werden, so würde F durch die folgende Belegung $\mathcal{B}'$ erfüllt werden:

$$\mathcal{B}'(A_i) = \begin{cases} 0 & \text{falls } A_i = A \\ \mathcal{B}(A_i) & \text{falls } A_i \neq A \end{cases}$$

Da $n(F_0) = n(F_1) = n(F) - 1$ gilt, können wir aus der Induktionsannahme schließen, dass $\square \in Res^*(F_0)$ und $\square \in Res^*(F_1)$ gilt.

Beweis des Resolutionssatz (Vollständigkeit)

Somit gibt es eine Folge von Klauseln $K_1, K_2, \ldots, K_m$ mit

- $K_m = \square$
- $K_i \in F_0$ oder K_i ist Resolvent von K_j und K_ℓ mit $j, \ell < i$.

Wir definieren nun eine Folge von Klauseln $K'_1, K'_2, \ldots, K'_m$, wobei $K'_i = K_i$ oder $K'_i = K_i \cup \{A\}$, wie folgt:

1. Fall $K_i \in F_0$ und $K_i \in F$: $K'_i := K_i$
2. Fall $K_i \in F_0$ und $K_i \notin F$: $K'_i := K_i \cup \{A\} \in F$
3. Fall $K_i \notin F_0$ und K_i wird aus K_j und K_ℓ $(j, \ell < i)$ nach dem Literal L resolviert:
   K'_i wird aus K'_j und K'_ℓ bebildet, indem nach L resolviert wird.

Dann gilt entweder $K'_m = \square$ oder $K'_m = \{A\}$ und somit

$$\square \in Res^*(F) \quad \text{oder} \quad \{A\} \in Res^*(F)$$

Analog folgt durch Betrachtung von F_1:

$$\square \in Res^*(F) \quad \text{oder} \quad \{\neg A\} \in Res^*(F)$$

Hieraus folgt $\square \in Res^*(F)$. $\square$

Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \{\neg A_2,\ A_4\}, \{\neg A_1, A_2,\ A_4\}, \{A_3, \neg A_4\},\ \{\neg A_1, \neg A_3, \neg A_4\}\ \}$

Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \{\neg A_2, \cancel{A_4}\}, \{\neg A_1, A_2, \cancel{A_4}\}, \cancel{\{A_3, \neg A_4\}}, \cancel{\{\neg A_1, \neg A_3, \neg A_4\}}\}$

$F_0 = \{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2\}\}$

Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \cancel{\{\neg A_2, A_4\}}, \cancel{\{\neg A_1, A_2, A_4\}}, \{A_3, \cancel{\neg A_4}\}, \{\neg A_1, \neg A_3, \cancel{\neg A_4}\}\}$

$F_0 = \{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2\}\}$

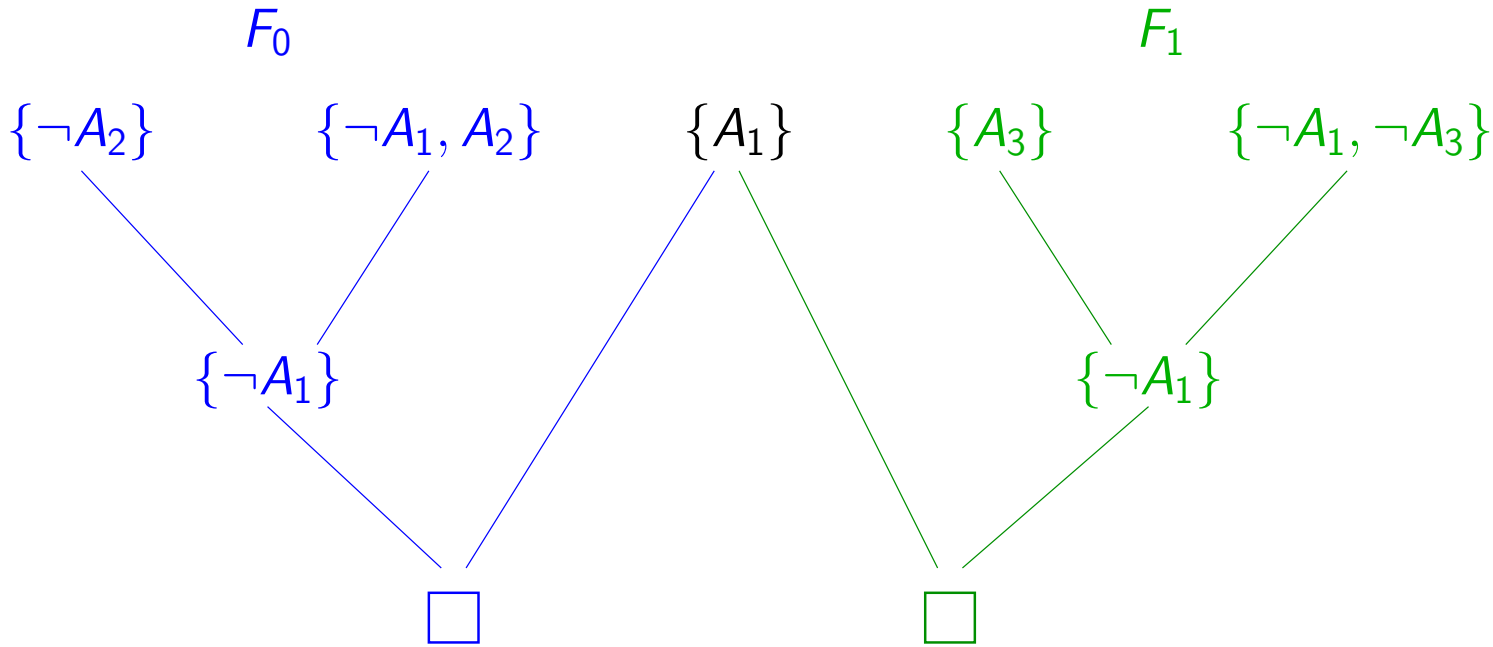
$F_1 = \{\{A_1\}, \{A_3\}, \{\neg A_1, \neg A_3\}\}$

Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \{\neg A_2, A_4\}, \{\neg A_1, A_2, A_4\}, \{A_3, \neg A_4\}, \{\neg A_1, \neg A_3, \neg A_4\}\}$

$F_0 = \{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2\}\}$

$F_1 = \{\{A_1\}, \{A_3\}, \{\neg A_1, \neg A_3\}\}$

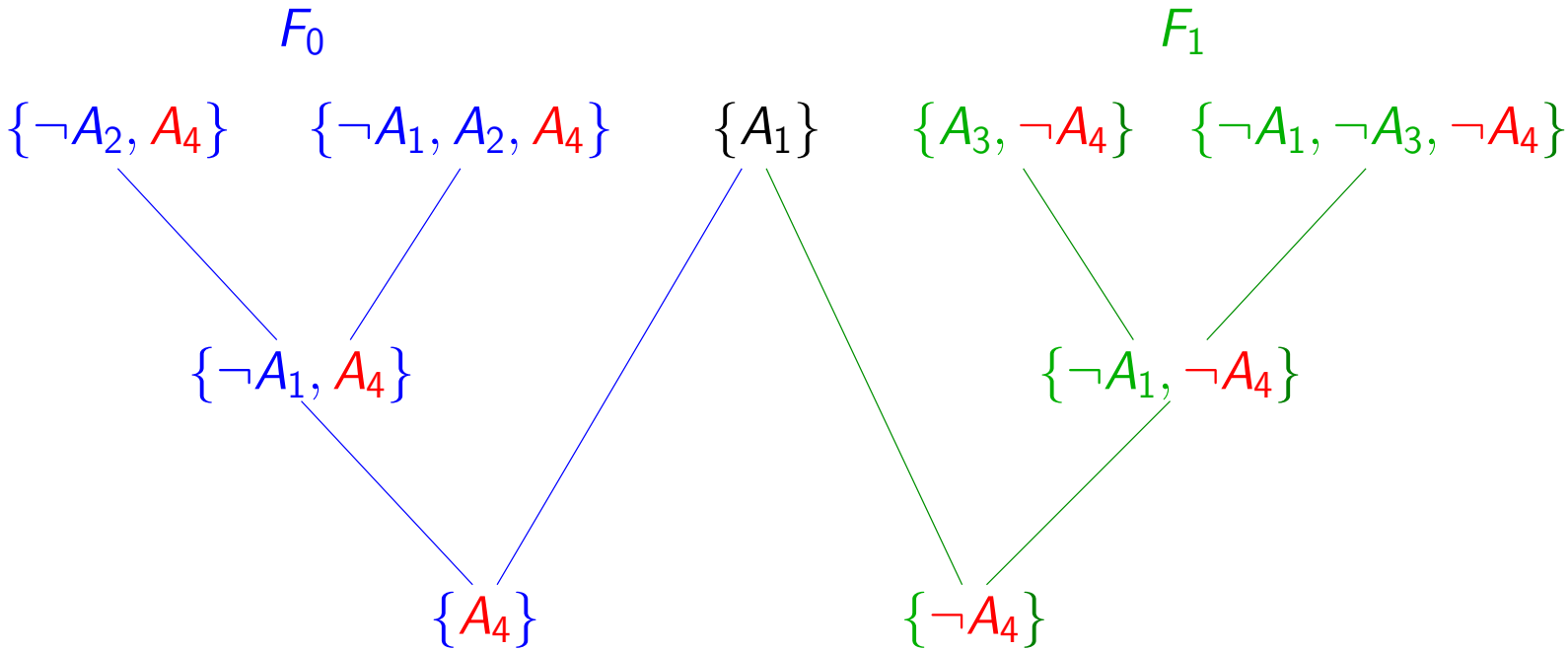


Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \{\neg A_2, A_4\}, \{\neg A_1, A_2, A_4\}, \{A_3, \neg A_4\}, \{\neg A_1, \neg A_3, \neg A_4\}\}$

$F_0 = \{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2\}\}$

$F_1 = \{\{A_1\}, \{A_3\}, \{\neg A_1, \neg A_3\}\}$

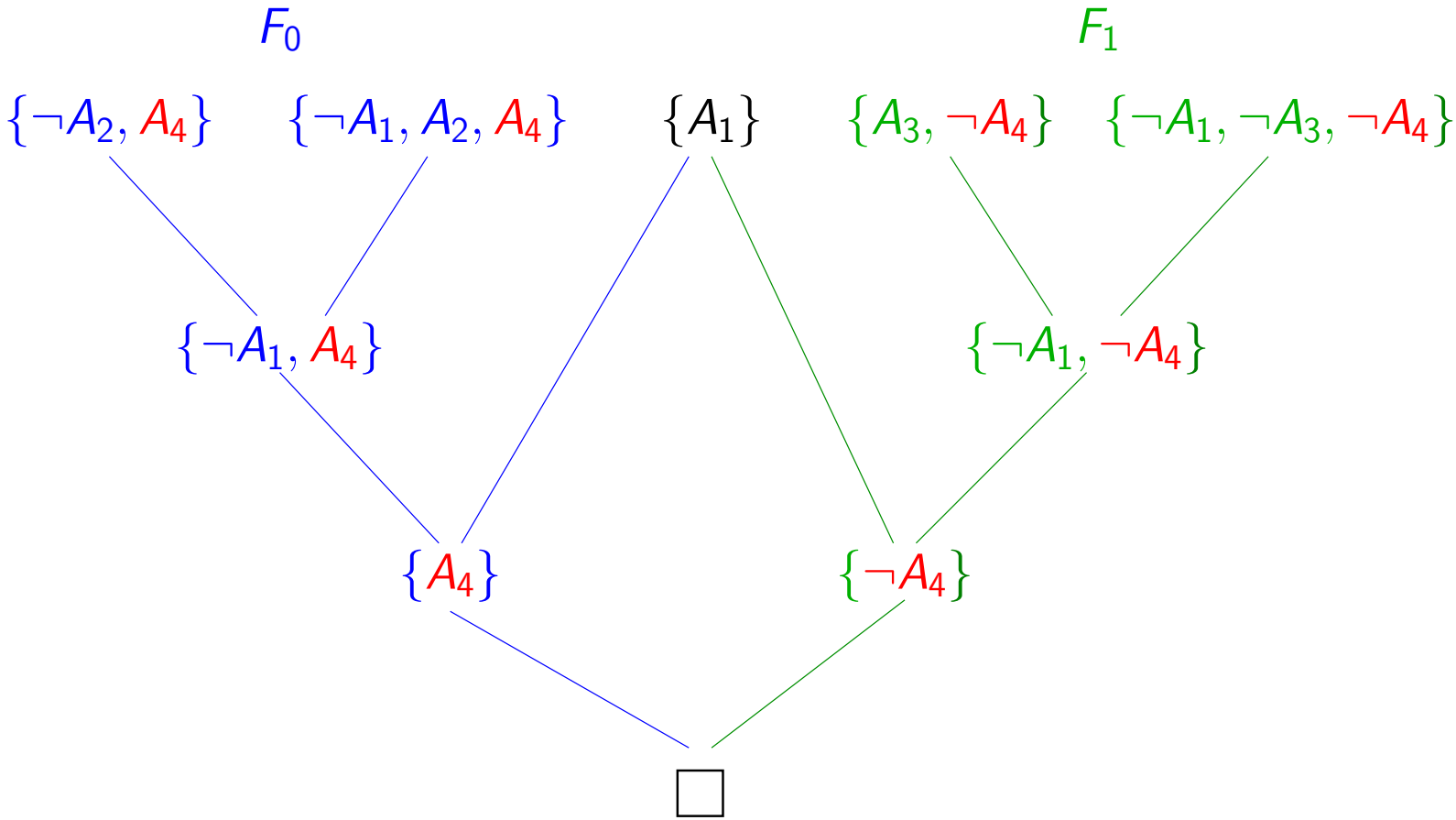


Veranschaulichung des Induktionsschritts

$F = \{\{A_1\}, \{\neg A_2, A_4\}, \{\neg A_1, A_2, A_4\}, \{A_3, \neg A_4\}, \{\neg A_1, \neg A_3, \neg A_4\}\}$

$F_0 = \{\{A_1\}, \{\neg A_2\}, \{\neg A_1, A_2\}\}$

$F_1 = \{\{A_1\}, \{A_3\}, \{\neg A_1, \neg A_3\}\}$



Deduktionen

Eine Deduktion (oder Herleitung oder Beweis) der leeren Klausel aus einer Klauselmenge F ist eine Folge von $K_1, K_2, \ldots, K_m$ von Klauseln mit folgenden Eigenschaften:

> *K_m ist die leere Klausel und für jedes $i \in \{1, \ldots, m\}$ gilt, dass K_i Element von F ist oder aus gewissen Klauseln K_j,K_ℓ mit $j, \ell < i$ resolviert werden kann.*

Aus dem Resolutionssatz folgt:
Eine endliche Klauselmenge ist unerfüllbar genau dann, wenn eine Deduktion der leeren Klausel existiert.

Bemerkung: Es kann sein, dass $Res^*(F)$ sehr groß ist, aber trotzdem eine kurze Deduktion der leeren Klausel existiert.

Resolutionskalkül

Mit dem Begriff Kalkül bezeichnet man eine Menge von syntaktischen Umformungsregeln, mit denen man semantische Eigenschaften der Eingabeformel herleiten kann.

Für den Resolutionskalkül:

- Syntaktische Umformungsregeln: Resolution, Stopp bei Erreichen der leeren Klausel
- Semantische Eigenschaft der Eingabeformel: Unerfüllbarkeit

Wünschenswerte Eigenschaften eines Kalküls:

- Korrektheit: Wenn die leere Klausel aus F abgeleitet werden kann, dann ist F unerfüllbar.
- Vollständigkeit: Wenn F unerfüllbar ist, dann ist die leere Klausel aus F ableitbar.

Beispiel zur Resolution

Wir wollen zeigen, dass

$$((AK \vee BK) \wedge (AK \rightarrow BK) \wedge (BK \wedge RL \rightarrow \neg AK) \wedge RL) \rightarrow (\neg AK \wedge BK)$$

gültig ist.

Das ist genau dann der Fall, wenn

$$(AK \vee BK) \wedge (\neg AK \vee BK) \wedge (\neg BK \vee \neg RL \vee \neg AK) \wedge RL \wedge (AK \vee \neg BK)$$

unerfüllbar ist. (Wegen: $F \rightarrow G$ gültig gdw. $F \wedge \neg G$ unerfüllbar.)

In Mengendarstellung:

$$\{\{AK, BK\},\ \{\neg AK, BK\},\ \{\neg BK, \neg RL, \neg AK\},\ \{RL\},\ \{AK, \neg BK\}\}$$

Beispiel zur Resolution

Eine mögliche Deduktion der leeren Klausel aus

$$\{\{AK, BK\},\ \{\neg AK, BK\},\ \{\neg BK, \neg RL, \neg AK\},\ \{RL\},\ \{AK, \neg BK\}\} \quad (6)$$

sieht wie folgt aus:

Beispiel zur Resolution

Eine mögliche Deduktion der leeren Klausel aus

$$\{\{AK, BK\},\ \{\neg AK, BK\},\ \{\neg BK, \neg RL, \neg AK\},\ \{RL\},\ \{AK, \neg BK\}\} \quad (6)$$

sieht wie folgt aus:

$\{AK, BK\}$	gehört zu (6)	(7)
$\{\neg AK, BK\}$	gehört zu (6)	(8)
$\{BK\}$	aus (7) und (8)	(9)
$\{\neg BK, \neg RL, \neg AK\}$	gehört zu (6)	(10)
$\{AK, \neg BK\}$	gehört zu (6)	(11)
$\{\neg BK, \neg RL\}$	aus (10) und (11)	(12)
$\{RL\}$	gehört zu (6)	(13)
$\{\neg BK\}$	aus (12) und (13)	(14)
$\square$	aus (9) und (14)	(15)

Bemerkungen zur Resolution

Armin Haken hat 1985 für jedes n eine Menge von Klauseln F_n angegeben mit:

- F_n ist unerfüllbar.
- F_n enthält $n \cdot (n+1)$ viele atomare Teilformeln.
- F_n besteht aus $\frac{n^3+n^2}{2} + n + 1$ vielen Klauseln.
- Jede Deduktion der leeren Klausel aus F_n hat Länge mindestens c^n für eine feste Konstante $c > 1$.

Fazit: Deduktionen können sehr lang werden.

Der Endlichkeitssatz der Aussagenlogik

Satz 49 (Endlichkeitssatz (compactness theorem))

Sei M eine (eventuell unendliche) Menge von Formeln. Dann ist M genau dann erfüllbar, wenn jede endliche Teilmenge von M erfüllbar ist.

Beweis:

1. Wenn M erfüllbar ist, dann ist jede endliche Teilmenge von M erfüllbar.

Diese Aussage ist trivial, denn jedes Modell von M ist auch ein Modell für jede Teilmenge von M.

Der Endlichkeitssatz der Aussagenlogik: Beweis

2. Sei jede endliche Teilmenge von M erfüllbar. Dann ist auch M erfüllbar.

Zur Erinnerung: Die Menge der atomaren Formeln ist $\{A_1, A_2, A_3, \ldots\}$.

Sei $M_n \subseteq M$ die Menge der Formeln in M, die nur atomare Teilformeln aus $\{A_1, \ldots, A_n\}$ enthalten.

Beachte: M_n kann unendlich sein; z. B. könnte M_1 die Formeln $A_1,\ A_1 \wedge A_1,\ A_1 \wedge A_1 \wedge A_1, \ldots$ enthalten.

Aber: Es gibt nur 2^{2^n} viele verschiedene Wahrheitstafeln mit den atomaren Formeln $A_1, \ldots, A_n$.

Deshalb gibt es in M_n eine endliche Teilmenge $M'_n \subseteq M_n$ mit:

1. $|M'_n| \leq 2^{2^n}$
2. Für jede Formel $F \in M_n$ existiert eine Formel $F' \in M'_n$ mit $F \equiv F'$

Der Endlichkeitssatz der Aussagenlogik: Beweis

Nach Annahme hat M'_n ein Modell $\mathcal{B}_n$.

Also ist $\mathcal{B}_n$ auch ein Modell von M_n.

Wir konstruieren nun ein Modell $\mathcal{B}$ von M wie folgt in Stufen.

In Stufe n wird dabei der Wert $\mathcal{B}(A_n) \in \{0, 1\}$ festgelegt.

$I_0 := \{1, 2, 3, \ldots\}$
for all $n \geq 1$ **do**
    if es gibt unendlich viele Indizes $i \in I_{n-1}$ mit $\mathcal{B}_i(A_n) = 1$ **then**
        $\mathcal{B}(A_n) := 1$
        $I_n := \{i \in I_{n-1} \mid \mathcal{B}_i(A_n) = 1\}$
    else (es gibt unendlich viele $i \in I_{n-1}$ mit $\mathcal{B}_i(A_n) = 0$)
        $\mathcal{B}(A_n) := 0$
        $I_n := \{i \in I_{n-1} \mid \mathcal{B}_i(A_n) = 0\}$
endfor

Der Endlichkeitssatz der Aussagenlogik: Beweis

Es gilt zwar $I_0 \supseteq I_1 \supseteq I_2 \supseteq I_3 \cdots$ aber:

Behauptung 1: Für jedes $n \geq 0$ ist I_n ist unendlich.

Dies zeigt man durch Induktion über n:

$I_0 = \{1, 2, 3, \ldots\}$ offensichtlich unendlich.

Ist I_{n-1} unendlich, so ist nach Konstruktion auch I_n unendlich, denn es gilt $I_{n-1} = \{i \in I_{n-1} \mid \mathcal{B}_i(A_n) = 1\} \cup \{i \in I_{n-1} \mid \mathcal{B}_i(A_n) = 0\}$.

Behauptung 2: Für alle $n \geq 1$ und alle $i \in I_n$ gilt:

$$\mathcal{B}_i(A_1) = \mathcal{B}(A_1), \ldots, \mathcal{B}_i(A_{n-1}) = \mathcal{B}(A_{n-1}),\ \mathcal{B}_i(A_n) = \mathcal{B}(A_n).$$

Sei $i \in I_n$. Dann gilt $\mathcal{B}_i(A_n) = \mathcal{B}(A_n)$ nach Konstruktion von $\mathcal{B}$.

Sei nun $j \in \{1, \ldots, n-1\}$.

Wegen $I_n \subseteq I_j$ gilt $i \in I_j$ und damit wieder $\mathcal{B}_i(A_j) = \mathcal{B}(A_j)$.

Der Endlichkeitssatz der Aussagenlogik: Beweis

Behauptung 3: Die konstruierte Belegung $\mathcal{B}$ ist ein Modell von M.

Beweis von Behauptung 3: Sei $F \in M$.

Dann existiert ein $n \geq 1$ mit $F \in M_n$ (denn in F kommen nur endlich viele atomare Formeln vor).

Also gilt $F \in M_i$ für alle $i \geq n$.

Also ist jede der Belegungen $\mathcal{B}_i$ mit $i \geq n$ ein Modell von F.

Da I_n nach Behauptung 1 unendlich ist, existiert ein $i \geq n$ mit $i \in I_n$.

Für dieses i gilt nach Behauptung 2:

$$\mathcal{B}_i(A_1) = \mathcal{B}(A_1), \quad \mathcal{B}_i(A_2) = \mathcal{B}(A_2), \ldots, \mathcal{B}_i(A_n) = \mathcal{B}(A_n)$$

Da $\mathcal{B}_i$ wegen $i \geq n$ ein Modell von F ist, ist auch $\mathcal{B}$ ein Modell von F. $\square$

Folgerungen aus dem Endlichkeitssatz

Die folgende Aussage haben wir bisher nur für eine endliche Klauselmenge F bewiesen.

Resolutionssatz der Aussagenlogik für beliebige Formelmengen

Eine Menge F von Klauseln ist unerfüllbar genau dann, wenn $\square \in Res^*(F)$.

Beweis:

Korrektheit: Wenn $\square \in Res^*(F)$, dann ist F unerfüllbar:
Beweis wie im endlichen Fall

Vollständigkeit: Wenn F unerfüllbar ist, dann gilt $\square \in Res^*(F)$.

Wenn F unerfüllbar ist, dann muss nach dem Endlichkeitssatz eine endliche Teilmenge $F' \subseteq F$ existieren, die ebenfalls unerfüllbar ist.

Aus dem bereits bewiesenen Resolutionssatz für endliche Formelmengen folgt $\square \in Res^*(F')$.

Also gilt auch $\square \in Res^*(F)$. $\square$

Prädikatenlogik

Aussagenlogik ist zur Formulierung mathematischer Sachverhalte im Allgemeinen zu ausdrucksschwach.

Beispiel: In der Analysis will man Aussagen der Form

> *Für alle* $x \in \mathbb{R}$ *und alle* $\varepsilon > 0$ *existiert ein* $\delta > 0$, *so dass für alle* $y \in \mathbb{R}$ *gilt: Wenn* $\mathit{abs}(x - y) < \delta$, *dann* $\mathit{abs}(f(x) - f(y)) < \varepsilon$.

Hierbei verwenden wir:

- Relationen wie z. B. $<$ oder $>$
- Funktionen wie z. B. $\mathit{abs} : \mathbb{R} \to \mathbb{R}_+$ (Absolutbetrag) oder $- : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ (Minus)
- Quantifikationen wie z. B. "für alle $x \in \mathbb{R}$" oder "existiert ein $\delta > 0$".

Die führt zur Prädikatenlogik

Syntax der Prädikatenlogik: Variablen, Terme

Die Menge der Variablen ist $\{x_0, x_1, x_2, \ldots\}$.

Ein Prädikatensymbol hat die Form P_i^k mit $i, k \in \{0, 1, 2, \ldots\}$.

Ein Funktionssymbol hat die Form f_i^k mit $i, k \in \{0, 1, 2, \ldots\}$.

In beiden Fällen heißt i der Unterscheidungsindex und k die Stelligkeit.

Variablen werden auch mit u, x, y, z bezeichnet,
Prädikatensymbole (Funktionssymbole) auch mit P, Q, R (f, g, h).

Wir definieren nun die Menge der Terme induktiv:

1. Jede Variable ist ein Term.
2. Falls f ein Funktionssymbol mit der Stelligkeit k ist, und falls $t_1, \ldots, t_k$ Terme sind, so ist auch $f(t_1, \ldots, t_k)$ ein Term.

Hierbei sollen auch Funktionssymbole der Stelligkeit 0 eingeschlossen sein, und in diesem Fall sollen die Klammern wegfallen.

Nullstellige Funktionssymbole heißen auch Konstanten
(werden meist mit a, b, c bezeichnet).

Formeln

Nun können wir (wiederum induktiv) definieren, was Formeln (der Prädikatenlogik) sind.

1. Falls P ein Prädikatsymbol der Stelligkeit k ist, und falls $t_1, \ldots, t_k$ Terme sind, dann ist $P(t_1, \ldots, t_k)$ eine Formel.
2. Für jede Formel F ist auch $\neg F$ eine Formel.
3. Für alle Formeln F und G sind auch $(F \wedge G)$ und $(F \vee G)$ Formeln.
4. Falls x eine Variable ist und F eine Formel, so sind auch $\exists x F$ und $\forall x F$ Formeln.
   Das Symbol $\exists$ wird Existenzquantor und $\forall$ Allquantor genannt.

Atomare Formeln nennen wir genau die, die gemäß 1. aufgebaut sind.

Falls F eine Formel ist und F als Teil einer Formel G auftritt, so heißt F Teilformel von G.

Freie und gebundene Variablen, Aussagen

Alle Vorkommen von Variablen in einer Formel, welche nicht direkt hinter einem Quantor stehen, werden in freie und gebundene Vorkommen unterteilt:

Ein Vorkommen der Variablen x in der Formel F, welches nicht direkt hinter einem Quantor steht, ist gebunden, falls dieses Vorkommen in einer Teilformel von F der Form $\exists x G$ oder $\forall x G$ vorkommt.
Andernfalls heißt dieses Vorkommen von x frei.

Eine Formel ohne Vorkommen einer freien Variablen heißt geschlossen oder eine Aussage.

Die Matrix einer Formel F ist diejenige Formel, die man aus F erhält, indem jedes Vorkommen von $\exists$ bzw. $\forall$, samt der dahinterstehenden Variablen gestrichen wird.

Wir bezeichnen die Matrix der Formel F mit F^*.

Beispiel für Formel der Prädikatenlogik

Sei F die Formel

$$\exists x P(x, f(y)) \ \vee \ \neg \forall y Q(y, g(a, h(z)))$$

Das rot markierte Vorkommen von y in F ist frei, während das grün markierte Vorkommen von y in F gebunden ist:

$$\exists x P(x, f(y)) \ \vee \ \neg \forall y Q(y, g(a, h(z)))$$

Die Matrix von F ist

$$P(x, f(y)) \ \vee \ \neg Q(y, g(a, h(z))).$$

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$			
$\forall x \exists y (Q(x,y) \vee R(x,y))$			
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$			
$\forall x P(x) \vee \forall x Q(x,x)$			
$\forall x (P(y) \wedge \forall y P(x))$			
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$			
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$			
$\forall x P(x) \vee \forall x Q(x,x)$			
$\forall x (P(y) \wedge \forall y P(x))$			
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$			
$\forall x P(x) \vee \forall x Q(x,x)$			
$\forall x (P(y) \wedge \forall y P(x))$			
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$	N	J	N
$\forall x P(x) \vee \forall x Q(x,x)$			
$\forall x (P(y) \wedge \forall y P(x))$			
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$	N	J	N
$\forall x P(x) \vee \forall x Q(x,x)$	N	N	J
$\forall x (P(y) \wedge \forall y P(x))$			
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$	N	J	N
$\forall x P(x) \vee \forall x Q(x,x)$	N	N	J
$\forall x (P(y) \wedge \forall y P(x))$	N	J	N
$P(x) \rightarrow \exists x Q(x, P(x))$			
$\forall f \; \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$	N	J	N
$\forall x P(x) \vee \forall x Q(x,x)$	N	N	J
$\forall x (P(y) \wedge \forall y P(x))$	N	J	N
$P(x) \rightarrow \exists x Q(x, P(x))$	J	N	N
$\forall f\ \exists x P(f(x))$			

Aufgabe

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

(x, y sind Variablen, a ist Konstante, P, Q, R sind Prädikatensymbole, f ist Funktionssymbol)

	NF	F	A
$\forall x P(a)$	N	N	J
$\forall x \exists y (Q(x,y) \vee R(x,y))$	N	N	J
$\forall x Q(x,x) \rightarrow \exists x Q(x,y)$	N	J	N
$\forall x P(x) \vee \forall x Q(x,x)$	N	N	J
$\forall x (P(y) \wedge \forall y P(x))$	N	J	N
$P(x) \rightarrow \exists x Q(x, P(x))$	J	N	N
$\forall f \; \exists x P(f(x))$	J	N	N

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$			
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$			
$\exists x(\neg P(x) \vee P(f(a)))$			
$P(x) \rightarrow \exists x P(x)$			
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$			
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$			
$\exists x(\neg P(x) \vee P(f(a)))$			
$P(x) \rightarrow \exists x P(x)$			
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$			
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$	N	J	N
$\exists x(\neg P(x) \vee P(f(a)))$			
$P(x) \rightarrow \exists x P(x)$			
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$			
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$	N	J	N
$\exists x(\neg P(x) \vee P(f(a)))$	N	N	J
$P(x) \rightarrow \exists x P(x)$			
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$			
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$	N	J	N
$\exists x(\neg P(x) \vee P(f(a)))$	N	N	J
$P(x) \rightarrow \exists x P(x)$	N	J	N
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$			
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$	N	J	N
$\exists x(\neg P(x) \vee P(f(a)))$	N	N	J
$P(x) \rightarrow \exists x P(x)$	N	J	N
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$	N	N	J
$\exists x\forall x Q(x,x)$			

NF: Nicht-Formel F: Formel, aber nicht Aussage A: Aussage

	NF	F	A
$\forall x(\neg\forall y Q(x,y) \wedge R(x,y))$	N	J	N
$\exists z(Q(z,x) \vee R(y,z)) \rightarrow \exists y(R(x,y) \wedge Q(x,z))$	N	J	N
$\exists x(\neg P(x) \vee P(f(a)))$	N	N	J
$P(x) \rightarrow \exists x P(x)$	N	J	N
$\exists x\forall y((P(y) \rightarrow Q(x,y)) \vee \neg P(x))$	N	N	J
$\exists x\forall x Q(x,x)$	N	N	J

Einschub: Relationen und Funktionen beliebiger Stelligkeit

Sei A eine beliebige Menge.

Für $n \geq 0$ sei $A^n = \{(a_1, a_2, \ldots, a_n) \mid a_1, a_2, \ldots, a_n \in A\}$ die Menge aller n-Tupel über der Menge A.

Beachte: $A^0 = \{()\}$ ist eine 1-elementige Menge und $A^1 = \{(a) \mid a \in A\}$ kann mit A identifiziert werden.

Eine n-stellige Relation R über A ist eine Teilmenge von A^n: $R \subseteq A^n$.

Insbesondere gibt es nur zwei 0-stellige Relationen: $\emptyset$ und $\{()\}$.

Ist R eine 2-stellige Relation (man spricht auch von einer binären Relation), so schreibt man häufig $a\,R\,b$ anstatt $(a, b) \in R$ (für $a, b \in A$).

Eine n-stellige Funktion f auf A ist eine Funktion $f : A^n \to A$.

Beachte: Eine 0-stellige Funktion f kann mit einem Element von A identifiziert werden, nämlich mit dem Element $f(())$. Wir schreiben hierfür auch $f()$ oder kurz f.

Semantik der Prädikatenlogik: Strukturen

Eine Struktur ist ein Paar $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ mit:

$U_\mathcal{A}$ ist eine beliebige aber nicht leere Menge, die die Grundmenge von $\mathcal{A}$ (oder der Grundbereich, der Individuenbereich, das Universum) genannt wird.

Ferner ist $I_\mathcal{A}$ eine partiell definierte Abbildung, die

- jedem k-stelligen Prädikatensymbol P aus dem Definitionsbereich von $I_\mathcal{A}$ eine k–stellige Relation $I_\mathcal{A}(P) \subseteq U_\mathcal{A}^k$ zuordnet,
- jedem k-stelligen Funktionssymbol f aus dem Definitionsbereich von $I_\mathcal{A}$ eine k-stellige Funktion $I_\mathcal{A}(f) : U_\mathcal{A}^k \to U_\mathcal{A}$ zuordnet, und
- jeder Variablen x aus dem Definitionsbereich von $I_\mathcal{A}$ ein Element $I_\mathcal{A}(x) \in U_\mathcal{A}$ zuordnet.

Sei F eine Formel und $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ eine Struktur.

$\mathcal{A}$ heißt zu F passend, falls $I_\mathcal{A}$ für alle in F vorkommenden Prädikatsymbole, Funktionssymbole und freien Variablen definiert ist.

Mit anderen Worten, der Definitionsbereich $\mathrm{Def}(I_\mathcal{A})$ von $I_\mathcal{A}$ ist eine Teilmenge von

$$\{P_i^k \mid i, k \geq 0\} \cup \{f_i^k \mid i, k \geq 0\} \cup \{x_i \mid i \geq 0\},$$

und der Wertebereich von $I_\mathcal{A}$ ist die Menge aller Relationen, Funktionen und Elemente von $U_\mathcal{A}$.

Wir schreiben abkürzend statt $I_\mathcal{A}(P)$ einfach $P^\mathcal{A}$, statt $I_\mathcal{A}(f)$ einfach $f^\mathcal{A}$ und statt $I_\mathcal{A}(x)$ einfach $x^\mathcal{A}$.

Beispiel für Struktur

Sei F die Formel $\forall x P(x, f(x)) \ \wedge \ Q(g(a, z))$.

Eine zu F passende Struktur ist z. B. $\mathcal{A} = (\mathbb{N}, I_{\mathcal{A}})$, wobei

$$\begin{aligned} P^{\mathcal{A}} &= \{(n, m) \mid n, m \in \mathbb{N}, n < m\} \\ Q^{\mathcal{A}} &= \{n \mid n \text{ ist Primzahl}\} \\ f^{\mathcal{A}}(n) &= n + 1 \text{ für alle } n \in \mathbb{N} \\ g^{\mathcal{A}}(n, m) &= n + m \text{ für alle } n, m \in \mathbb{N} \\ a^{\mathcal{A}} &= 2 \\ z^{\mathcal{A}} &= 3 \end{aligned}$$

In einem intuitiven Sinn gilt die Formel F in der Struktur $\mathcal{A}$:
$\forall x \in \mathbb{N}\,(x < x + 1) \ \wedge \ 2 + 3$ ist Primzahl) ist eine wahre Aussage.

Auswertung in einer Struktur

Sei F eine Formel und $\mathcal{A}$ eine zu F passende Struktur.

Für jeden Term t, den man aus den Bestandteilen von F bilden kann (also aus den freien Variablen und Funktionssymbolen), definieren wir induktiv den Wert $\mathcal{A}(t) \in U_{\mathcal{A}}$ von t in der Struktur $\mathcal{A}$:

1. Falls t eine Variable ist (also $t = x$), so ist $\mathcal{A}(t) = x^{\mathcal{A}}$.
2. Falls t die Form hat $t = f(t_1, \ldots, t_k)$ wobei $t_1, \ldots, t_k$ Terme und f ein k–stelliges Funktionssymbol ist, so ist $\mathcal{A}(t) = f^{\mathcal{A}}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k))$.

Der Fall 2 schließt auch die Möglichkeit ein, dass f nullstellig ist, also t die Form $t = a$ hat.

In diesem Fall ist also $\mathcal{A}(t) = a^{\mathcal{A}}$.

Auswertung in einer Struktur

Auf analoge Weise definieren wir induktiv den (Wahrheits–) Wert $\mathcal{A}(F)$ der Formeln F unter der Struktur $\mathcal{A}$:

- Falls F die Form hat $F = P(t_1, \ldots, t_k)$ mit den Termen $t_1, \ldots, t_k$ und k–stelligem Prädikatsymbol P, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \textit{falls } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P^{\mathcal{A}} \\ 0, & \textit{sonst} \end{cases}$$

- Falls F die Form $F = \neg G$ hat, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \textit{falls } \mathcal{A}(G) = 0 \\ 0, & \textit{sonst} \end{cases}$$

Auswertung in einer Struktur

- Falls F die Form $F = (G \wedge H)$ hat, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \textit{falls } \mathcal{A}(G) = 1 \textit{ und } \mathcal{A}(H) = 1 \\ 0, & \textit{sonst} \end{cases}$$

- Falls F die Form $F = (G \vee H)$ hat, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \textit{falls } \mathcal{A}(G) = 1 \textit{ oder } \mathcal{A}(H) = 1 \\ 0, & \textit{sonst} \end{cases}$$

Auswertung in einer Struktur

- Falls F die Form $F = \forall x G$ hat, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \text{falls für alle } d \in U_{\mathcal{A}} \text{ gilt: } \mathcal{A}_{[x/d]}(G) = 1 \\ 0, & \text{sonst} \end{cases}$$

- Falls F die Form $F = \exists x G$ hat, so ist

$$\mathcal{A}(F) = \begin{cases} 1, & \text{falls es ein } d \in U_{\mathcal{A}} \text{ gibt mit: } \mathcal{A}_{[x/d]}(G) = 1 \\ 0, & \text{sonst} \end{cases}$$

Hierbei ist $\mathcal{A}_{[x/d]}$ für eine Variable x und $d \in U_{\mathcal{A}}$ diejenige Struktur, die mit $\mathcal{A}$ identisch ist, außer das x durch d interpretiert wird:

$$x^{\mathcal{A}_{[x/d]}} = d$$

Auswertung in einer Struktur

Genauer: Die Struktur $\mathcal{A}_{[x/d]}$ ist wie folgt definiert.

- $U_{\mathcal{A}} = U_{\mathcal{A}_{[x/d]}}$
- $\mathrm{Def}(I_{\mathcal{A}_{[x/d]}}) = \mathrm{Def}(I_{\mathcal{A}}) \cup \{x\}$
- $x^{\mathcal{A}_{[x/d]}} = d$ (unabhängig davon, ob $x^{\mathcal{A}}$ definiert ist)
- $y^{\mathcal{A}_{[x/d]}} = y^{\mathcal{A}}$ für alle Variablen $y \in \mathrm{Def}(I_{\mathcal{A}}) \setminus \{x\}$
- $P^{\mathcal{A}_{[x/d]}} = P^{\mathcal{A}}$ für alle Prädikatensymbole $P \in \mathrm{Def}(I_{\mathcal{A}})$
- $f^{\mathcal{A}_{[x/d]}} = f^{\mathcal{A}}$ für alle Funktionssymbole $f \in \mathrm{Def}(I_{\mathcal{A}})$

Beachte: $x^{\mathcal{A}}$ kann definiert sein, dies hat jedoch auf den Wahrheitswerte $\mathcal{A}(\forall x G)$ und $\mathcal{A}(\exists x G)$ keinen Einfluß. Beim Auswerten von $\forall x G$ und $\exists x G$ in der Struktur $\mathcal{A}$ wird $x^{\mathcal{A}}$ "überschrieben".

Konvention zu $=$

Wir machen für das Weitere folgende Konvention:

Wann immer wir das Prädikatensymbol $=$ in Formeln verwenden, soll $=$ 2-stellig sein, und für jede Struktur $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ soll gelten:

> Für alle $a, b \in U_\mathcal{A}$: $a =^\mathcal{A} b$ genau dann, wenn a und b gleich sind.

Modell, Gültigkeit, Erfüllbarkeit

Falls für eine Formel F und eine zu F passende Struktur $\mathcal{A}$ gilt $\mathcal{A}(F) = 1$, so schreiben wir wieder $\mathcal{A} \models F$.

Sprechweise: F gilt in $\mathcal{A}$ oder $\mathcal{A}$ ist Modell für F.

$\mathcal{A}$ ist Modell für eine Menge M von Formeln, falls $\mathcal{A}$ ein Modell für jede Formel $F \in M$ ist.

Falls jede zu F passende Struktur ein Modell für F ist, so schreiben wir $\models F$, andernfalls $\not\models F$.

Sprechweise: F ist (allgemein–)gültig.

Falls es mindestens ein Modell für die Formel F gibt, so heißt F erfüllbar, andernfalls unerfüllbar.

Beispiele

Modelle für die Formel $\forall x \exists y P(x, y)$ wären $\mathcal{A} = (\mathbb{N}, I_{\mathcal{A}})$ sowie $\mathcal{B} = (\mathbb{N}, I_{\mathcal{B}})$ mit

$$\begin{aligned} P^{\mathcal{A}} &= \{(n, m) \in \mathbb{N} \times \mathbb{N} \mid n < m\} \text{ und} \\ P^{\mathcal{B}} &= \{(n, m) \in \mathbb{N} \times \mathbb{N} \mid n \leq m\}. \end{aligned}$$

Ein anderes (endliches) Modell wäre $\mathcal{C} = (\{0\}, I_{\mathcal{C}})$ mit $P^{\mathcal{C}} = \{(0, 0)\}$.

Betrachte nun die Formel $\exists x \forall y P(x, y)$.

$\mathcal{B}$ und $\mathcal{C}$ sind auch Modelle von $\exists x \forall y P(x, y)$.

$\mathcal{A}$ ist kein Modelle von $\exists x \forall y P(x, y)$.

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$			
$\exists x(\neg P(x) \vee P(a))$			
$P(a) \rightarrow \exists x P(x)$			
$P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$			
$P(a) \rightarrow \exists x P(x)$			
$P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$	J	J	N
$P(a) \rightarrow \exists x P(x)$			
$P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$	J	J	N
$P(a) \rightarrow \exists x P(x)$	J	J	N
$P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$	J	J	N
$P(a) \rightarrow \exists x P(x)$	J	J	N
$P(x) \rightarrow \exists x P(x)$	J	J	N
$\forall x P(x) \rightarrow \exists x P(x)$			
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$	J	J	N
$P(a) \rightarrow \exists x P(x)$	J	J	N
$P(x) \rightarrow \exists x P(x)$	J	J	N
$\forall x P(x) \rightarrow \exists x P(x)$	J	J	N
$\forall x P(x) \wedge \neg \forall y P(y)$			

Aufgabe

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x P(a)$	N	J	N
$\exists x(\neg P(x) \vee P(a))$	J	J	N
$P(a) \rightarrow \exists x P(x)$	J	J	N
$P(x) \rightarrow \exists x P(x)$	J	J	N
$\forall x P(x) \rightarrow \exists x P(x)$	J	J	N
$\forall x P(x) \wedge \neg \forall y P(y)$	N	N	J

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x(P(x,x) \rightarrow \exists x \forall y P(x,y))$			
$\forall x \forall y(x = y \rightarrow f(x) = f(y))$			
$\forall x \forall y(f(x) = f(y) \rightarrow x = y)$			
$\exists x \exists y \exists z(f(x) = y \wedge f(x) = z \wedge y \neq z)$			

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x(P(x,x) \to \exists x \forall y P(x,y))$	N	J	N
$\forall x \forall y(x = y \to f(x) = f(y))$			
$\forall x \forall y(f(x) = f(y) \to x = y)$			
$\exists x \exists y \exists z(f(x) = y \wedge f(x) = z \wedge y \neq z)$			

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x(P(x,x) \rightarrow \exists x \forall y P(x,y))$	N	J	N
$\forall x \forall y(x = y \rightarrow f(x) = f(y))$	J	J	N
$\forall x \forall y(f(x) = f(y) \rightarrow x = y)$			
$\exists x \exists y \exists z(f(x) = y \wedge f(x) = z \wedge y \neq z)$			

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x(P(x,x) \rightarrow \exists x \forall y P(x,y))$	N	J	N
$\forall x \forall y(x = y \rightarrow f(x) = f(y))$	J	J	N
$\forall x \forall y(f(x) = f(y) \rightarrow x = y)$	N	J	N
$\exists x \exists y \exists z(f(x) = y \wedge f(x) = z \wedge y \neq z)$			

G: Gültig E: Erfüllbar U: Unerfüllbar

	G	E	U
$\forall x(P(x,x) \rightarrow \exists x \forall y P(x,y))$	N	J	N
$\forall x \forall y(x = y \rightarrow f(x) = f(y))$	J	J	N
$\forall x \forall y(f(x) = f(y) \rightarrow x = y)$	N	J	N
$\exists x \exists y \exists z(f(x) = y \wedge f(x) = z \wedge y \neq z)$	N	N	J

Folgerung und Äquivalenz

Eine Formel G heißt eine Folgerung der Formeln $F_1, \ldots, F_k$, falls für jede Struktur $\mathcal{A}$, die sowohl zu $F_1, \ldots, F_k$ als auch zu G passend ist, gilt:

> *Wenn $\mathcal{A}$ Modell von $\{F_1, \ldots, F_k\}$ ist, dann ist $\mathcal{A}$ auch Modell von G.*

Wir schreiben $F_1, \ldots, F_k \models G$, falls G eine Folgerung von $F_1, \ldots, F_k$ ist.

Zwei Formeln F und G heißen (semantisch) äquivalent, falls für alle Strukturen $\mathcal{A}$, die sowohl für F als auch für G passend sind, gilt $\mathcal{A}(F) = \mathcal{A}(G)$.

Hierfür schreiben wir $F \equiv G$.

Aufgabe

1. $\forall x P(x) \vee \forall x Q(x, x)$
2. $\forall x (P(x) \vee Q(x, x))$
3. $\forall x (\forall z P(z) \vee \forall y Q(x, y))$

	J	N
1. $\models$ 2.		
2. $\models$ 3.		
3. $\models$ 1.		

Aufgabe

1. $\forall x P(x) \vee \forall x Q(x, x)$
2. $\forall x (P(x) \vee Q(x, x))$
3. $\forall x (\forall z P(z) \vee \forall y Q(x, y))$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 3.		
3. $\models$ 1.		

Aufgabe

1. $\forall x P(x) \vee \forall x Q(x,x)$
2. $\forall x (P(x) \vee Q(x,x))$
3. $\forall x (\forall z P(z) \vee \forall y Q(x,y))$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 3.		✓
3. $\models$ 1.		

Beachte: $\forall x (\forall z P(z) \vee \forall y Q(x,y)) \equiv \forall z P(z) \vee \forall x \forall y Q(x,y)$

Aufgabe

1. $\forall x P(x) \vee \forall x Q(x,x)$
2. $\forall x (P(x) \vee Q(x,x))$
3. $\forall x (\forall z P(z) \vee \forall y Q(x,y))$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 3.		✓
3. $\models$ 1.	✓	

Beachte: $\forall x (\forall z P(z) \vee \forall y Q(x,y)) \equiv \forall z P(z) \vee \forall x \forall y Q(x,y)$

Aufgabe

1. $\exists y \forall x\ P(x, y)$
2. $\forall x \exists y\ P(x, y)$

	J	N
1. $\models$ 2.		
2. $\models$ 1.		

Aufgabe

1. $\exists y \forall x\ P(x, y)$
2. $\forall x \exists y\ P(x, y)$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 1.		

Aufgabe

1. $\exists y \forall x\ P(x, y)$
2. $\forall x \exists y\ P(x, y)$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 1.		✓

Aufgabe

1. $\exists y \forall x\ P(x, y)$
2. $\forall x \exists y\ P(x, y)$

	J	N
1. $\models$ 2.	✓	
2. $\models$ 1.		✓

Beispiel für 2. $\not\models$ 1.

Sei $\mathcal{A} = (\{0, 1\}, \mathcal{I})$ mit $P^{\mathcal{I}} = \{(0, 0), (1, 1)\}$.

Dann gilt $\mathcal{A} \models \forall x \exists y\ P(x, y)$ und $\mathcal{A} \not\models \exists y \forall x\ P(x, y)$.

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$		
$\forall x \exists y F \equiv \exists x \forall y F$		
$\exists x \exists y F \equiv \exists y \exists x F$		
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$		
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		
$\exists x \exists y F \equiv \exists y \exists x F$		
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$		
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$		
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$		
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$	✓	
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$		
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$	✓	
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		✓
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$		
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$	✓	
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		✓
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$	✓	
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$		
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$	✓	
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		✓
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$	✓	
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$	✓	
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		

Aufgabe

	J	N
$\forall x \forall y F \equiv \forall y \forall x F$	✓	
$\forall x \exists y F \equiv \exists x \forall y F$		✓
$\exists x \exists y F \equiv \exists y \exists x F$	✓	
$\forall x F \vee \forall x G \equiv \forall x (F \vee G)$		✓
$\forall x F \wedge \forall x G \equiv \forall x (F \wedge G)$	✓	
$\exists x F \vee \exists x G \equiv \exists x (F \vee G)$	✓	
$\exists x F \wedge \exists x G \equiv \exists x (F \wedge G)$		✓

Äquivalenzen

Satz (Äquivalenzen der Prädikatenlogik)

Seien F und G beliebige Formeln:

1. $\neg\forall xF \equiv \exists x\neg F$
   $\neg\exists xF \equiv \forall x\neg F$
2. Falls x in G nicht frei vorkommt, gilt:
   $(\forall xF \wedge G) \equiv \forall x(F \wedge G)$
   $(\forall xF \vee G) \equiv \forall x(F \vee G)$
   $(\exists xF \wedge G) \equiv \exists x(F \wedge G)$
   $(\exists xF \vee G) \equiv \exists x(F \vee G)$
3. $(\forall xF \wedge \forall xG) \equiv \forall x(F \wedge G)$
   $(\exists xF \vee \exists xG) \equiv \exists x(F \vee G)$
4. $\forall x\forall yF \equiv \forall y\forall xF$
   $\exists x\exists yF \equiv \exists y\exists xF$

Äquivalenzen

Wir beweisen exemplarisch die Äquivalenz

$$(\forall x F \wedge G) \equiv \forall x (F \wedge G)$$

wobei x in G nicht frei vorkommt.

Sei $\mathcal{A}$ eine beliebige zu $(\forall x F \wedge G)$ und $\forall x (F \wedge G)$ passende Struktur. Dann gilt:

$\mathcal{A}(\forall x F \wedge G) = 1$

gdw. $\mathcal{A}(\forall x F) = 1$ und $\mathcal{A}(G) = 1$

gdw. für alle $d \in U_{\mathcal{A}}$ gilt $\mathcal{A}_{[x/d]}(F) = 1$ und $\mathcal{A}(G) = 1$

gdw. für alle $d \in U_{\mathcal{A}}$ gilt $\mathcal{A}_{[x/d]}(F) = 1$ und $\mathcal{A}_{[x/d]}(G) = 1$

(beachte: da x nicht frei in G vorkommt gilt $\mathcal{A}(G) = \mathcal{A}_{[x/d]}(G)$

gdw. für alle $d \in U_{\mathcal{A}}$ gilt $\mathcal{A}_{[x/d]}(F \wedge G) = 1$

gdw. $\mathcal{A}(\forall x (F \wedge G)) = 1$

Umbenennung von gebundenen Variablen

Für eine Formel G, eine Variable x und einen Term t sei $G[x/t]$ die Formel, die aus G entsteht, indem jedes freie Vorkommen von x in G durch den Term t ersetzt wird.

Beispiel:

$$\Big(\exists x P\Big(x, f(y)\Big) \ \vee \ \neg\forall y Q\Big(y, g(a, h(z))\Big)\Big)[y/f(u)] =$$
$$\Big(\exists x P\Big(x, f(f(u))\Big) \ \vee \ \neg\forall y Q\Big(y, g(a, h(z))\Big)\Big)$$

Übung: Definiere $G[x/t]$ formal durch Induktion über den Aufbau von G.

Umbenennung von gebundenen Variablen

Lemma (Umbenennung von Variablen)

Sei $F = QxG$ eine Formel mit $Q \in \{\forall, \exists\}$. Sei y eine Variable, die in G nicht vorkommt. Dann gilt $F \equiv QyG[x/y]$.

Die Bedingung, dass y in G nicht vorkommt ist hier wichtig.

Sei z. B. $G = P(x, y)$.

Dann gilt $G[x/y] = P(y, y)$ und damit

$$\exists xG = \exists xP(x, y) \not\equiv \exists yP(y, y) = \exists yG[x/y].$$

Bereinigte Formeln

Eine Formel heißt bereinigt, sofern es

- keine Variable gibt, die in der Formel sowohl gebunden als auch frei vorkommt, und sofern
- hinter allen vorkommenden Quantoren verschiedene Variablen stehen.

Beispiele:

- $P(x) \wedge \forall x Q(x)$ ist nicht bereinigt.
- $\exists x P(x) \wedge \forall x Q(x)$ ist nicht bereinigt.
- $P(x) \wedge \forall y Q(y)$ ist bereinigt.
- $\exists x P(x) \wedge \forall y Q(y)$ ist bereinigt.

Wiederholte Anwendung des Lemmas "Umbennung von Variaben" liefert:

Lemma 50

Zu jeder Formel F gibt es eine äquivalente bereinigte Formel.

Pränexform

Eine Formel heißt pränex oder in Pränexform, falls sie die Bauart

$$Q_1 y_1 \, Q_2 y_2 \cdots Q_n y_n F$$

hat, wobei

- $n \geq 0$, $Q_1, \ldots, Q_n \in \{\exists, \forall\}$, $y_1, \ldots, y_n$ Variablen sind, und
- in F kein Quantor vorkommt.

Eine bereinigte Formel in Pränexform ist in **BPF**.

Satz

Für jede Formel gibt es eine äquivalente Formel in **BPF**.

Beweis: Wiederholte Anwendung der Äquivalenzen (1) und (2) aus dem Satz "Äquivalenzen der Prädikatenlogik" (Folie 324).

Bildung der Pränexform

Sei F eine beliebige Formel.

Wir definieren eine zu F äquivalente **BPF**-Formel F' durch Induktion über den Aufbau von F:

- F ist atomar:
  Dann ist F bereits in **BPF** und wir können $F' = F$ setzen.
- $F = \neg G$:
  Nach Induktion existiert eine zu G äquivalente **BPF**-Formel

$$G' = Q_1 y_1 Q_2 y_2 \cdots Q_n y_n H,$$

  wobei $Q_1, \ldots, Q_n \in \{\exists, \forall\}$ und H keine Quantoren enthält.
  Aus Punkt (1) im Satz "Äquivalenzen der Prädikatenlogik" folgt:

$$\begin{aligned} F = \neg G \equiv \neg G' &= \neg Q_1 y_1 Q_2 y_2 \cdots Q_n y_n H \\ &\equiv \overline{Q}_1 y_1 \overline{Q}_2 y_2 \cdots \overline{Q}_n y_n \neg H \end{aligned}$$

  wobei $\overline{\exists} = \forall$ und $\overline{\forall} = \exists$. Letztere Formel ist in **BPF**.

Bildung der Pränexform

- $F = F_1 \wedge F_2$:
  Nach Induktion existieren zu F_1 und F_2 äquivalente **BPF**-Formeln

  $$F_1' = Q_1 y_1 Q_2 y_2 \cdots Q_m y_m G_1 \text{ und } F_2' = P_1 z_1 P_2 z_2 \cdots P_n z_n G_2,$$

  wobei $Q_1, \ldots, Q_m, P_1, \ldots, P_n \in \{\exists, \forall\}$ und G_1, G_2 keine Quantoren enthalten.
  Auf Grund des Umbennungslemmas können wir annehmen, dass $y_1, \ldots, y_m$ nicht in F_2' vorkommen und $z_1, \ldots, z_n$ nicht in F_1' vorkommen.
  Aus Punkt (2) im Satz "Äquivalenzen der Prädikatenlogik" folgt dann

  $$\begin{aligned} F &= F_1 \wedge F_2 \equiv F_1' \wedge F_2' \\ &= (Q_1 y_1 Q_2 y_2 \cdots Q_m y_m G_1) \wedge F_2' \\ &\equiv Q_1 y_1 Q_2 y_2 \cdots Q_m y_m (G_1 \wedge F_2') \\ &= Q_1 y_1 Q_2 y_2 \cdots Q_m y_m (G_1 \wedge P_1 z_1 P_2 z_2 \cdots P_n z_n G_2) \\ &\equiv Q_1 y_1 Q_2 y_2 \cdots Q_m y_m P_1 z_1 P_2 z_2 \cdots P_n z_n (G_1 \wedge G_2) \end{aligned}$$

Bildung der Pränexform

- $F = F_1 \vee F_2$: gleiche Argumentation wie bei $\wedge$
- $F = QxG$ mit $Q \in \{\exists, \forall\}$:

  Nach Induktion existiert eine zu G äquivalente **BPF**-Formel

  $$G' = Q_1 y_1 Q_2 y_2 \cdots Q_n y_n H,$$

  wobei H keine Quantoren enthält.

  Auf Grund des Umbennungslemmas können wir annehmen, dass $x \notin \{y_1, \ldots, y_n\}$ gilt.

  Damit gilt

  $$F = QxG \equiv QxQ_1 y_1 Q_2 y_2 \cdots Q_n y_n H$$

  und letztere Formel ist in **BPF**. □

Bildung der Pränexform: Beispiel

$$\Big(\forall x \exists y\ P(x, g(y, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \neg \forall x\ R(x, y)$$

$$\equiv \Big(\forall x \exists y\ P(x, g(y, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists x\ \neg R(x, y)$$

$$\equiv \forall x \exists y \Big(P(x, g(y, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists x\ \neg R(x, y)$$

$$\equiv \forall x \exists y \Big(P(x, g(y, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists w\ \neg R(w, y)$$

$$\equiv \forall x \Big(\exists y \Big(P(x, g(y, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists w\ \neg R(w, y)\Big)$$

$$\equiv \forall x \Big(\exists v \Big(P(x, g(v, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists w\ \neg R(w, y)\Big)$$

$$\equiv \forall x \exists v \Big(\Big(P(x, g(v, f(x)))\ \vee\ \neg Q(z)\Big)\ \vee\ \exists w\ \neg R(w, y)\Big)$$

$$\equiv \forall x \exists v \exists w \Big(P(x, g(v, f(x)))\ \vee\ \neg Q(z)\ \vee\ \neg R(w, y)\Big)$$

Skolemform

Für jede Formel F in **BPF** definieren wir ihre Skolemform(–el) als das Resultat der Anwendung von folgenden Algorithmus auf F:

while F enthält einen Existenzquantor **do**
begin

F habe die Form $F = \forall y_1 \forall y_2 \cdots \forall y_n \exists z G$ für eine Formel G in **BPF** und $n \geq 0$ (der Allquantorblock kann auch leer sein);

Sei f ein neues bisher in F nicht vorkommendes n–stelliges Funktionssymbol;

$F := \forall y_1 \forall y_2 \cdots \forall y_n G[z/f(y_1, y_2, \ldots, y_n)]$;
(d. h. der Existenzquantor in F wird gestrichen und jedes Vorkommen der Variablen z in G durch $f(y_1, y_2, \ldots, y_n)$ ersetzt)

end

Skolemform: Beispiel

Beispiel 1: Wir wollen die Skolemform von

$$\forall x \exists v \exists w \Big(P(x, g(v, f(x))) \ \vee \ \neg Q(z) \ \vee \ \neg R(w, y) \Big)$$

bilden.

Nach 1. Durchlauf durch **while**-Schleife:

$$\forall x \exists w \Big(P(x, g(f_1(x), f(x))) \ \vee \ \neg Q(z) \ \vee \ \neg R(w, y) \Big)$$

Nach 2. Durchlauf durch **while**-Schleife:

$$\forall x \Big(P(x, g(f_1(x), f(x))) \ \vee \ \neg Q(z) \ \vee \ \neg R(f_2(x), y) \Big)$$

Beispiel 2: Die Skolemform von $\exists x P(x)$ ist $P(a)$, wobei a eine Konstante ist.

Skolemform(F) erfüllbar gdw. F erfüllbar.

Satz

Für jede Formel F in **BPF** gilt: F ist erfüllbar genau dann, wenn die Skolemform von F erfüllbar ist.

Für den Beweis benötigen wir das folgende einfache Überführungslemma:

Überführungslemma

Sei F eine Formel, x eine Variable, und t ein Term, der keine in F gebundene Variable enthält. Dann gilt für jede zu F und $F[x/t]$ passende Struktur $\mathcal{A}$:

$$\mathcal{A}(F[x/t]) = \mathcal{A}_{[x/\mathcal{A}(t)]}(F)$$

Beweis des Überführungslemmas

Wir zeigen zunächst durch Induktion über den Aufbau von Termen:

Für jeden Term t' gilt: $\mathcal{A}(t'[x/t]) = \mathcal{A}_{[x/\mathcal{A}(t)]}(t')$

- $t' = y$ für eine Variable y.
  - $y = x$, d. h. $t' = x$:

    Dann gilt $t'[x/t] = t$.

    Also: $\mathcal{A}_{[x/\mathcal{A}(t)]}(t') = \mathcal{A}_{[x/\mathcal{A}(t)]}(x) = \mathcal{A}(t) = \mathcal{A}(t'[x/t])$
  - $y \neq x$.

    Dann gilt $t'[x/t] = t' = y$.

    Also: $\mathcal{A}_{[x/\mathcal{A}(t)]}(t') = \mathcal{A}(t') = \mathcal{A}(t'[x/t])$.

Beweis des Überführungslemmas

- $t' = f(t_1, \ldots, t_n)$.

  Dann gilt:

$$\begin{aligned}
\mathcal{A}(t'[x/t]) &= \mathcal{A}(f(t_1[x/t], \ldots, t_n[x/t])) \\
&= f^{\mathcal{A}}(\mathcal{A}(t_1[x/t]), \ldots, \mathcal{A}(t_n[x/t])) \\
&= f^{\mathcal{A}_{[x/\mathcal{A}(t)]}}(\mathcal{A}_{[x/\mathcal{A}(t)]}(t_1), \ldots, \mathcal{A}_{[x/\mathcal{A}(t)]}(t_n)) \\
&= \mathcal{A}_{[x/\mathcal{A}(t)]}(f(t_1, \ldots, t_n)) \\
&= \mathcal{A}_{[x/\mathcal{A}(t)]}(t')
\end{aligned}$$

Beweis des Überführungslemmas

Nun können wir $\mathcal{A}(F[x/t]) = \mathcal{A}_{[x/\mathcal{A}(t)]}(F)$ für eine Formel F durch Induktion über den Aufbau von F beweisen:

- F atomar, d. h. $F = P(t_1, \ldots, t_n)$ für ein Prädikatensymbol P und Terme $t_1, \ldots, t_n$.

  Dann gilt:

$$
\begin{array}{lll}
\mathcal{A}(F[x/t]) = 1 & \text{gdw.} & \mathcal{A}(P(t_1[x/t], \ldots, t_n[x/t])) = 1 \\
& \text{gdw.} & (\mathcal{A}(t_1[x/t]), \ldots, \mathcal{A}(t_n[x/t])) \in P^{\mathcal{A}} \\
& \text{gdw.} & (\mathcal{A}_{[x/\mathcal{A}(t)]}(t_1), \ldots, \mathcal{A}_{[x/\mathcal{A}(t)]}(t_n)) \in P^{\mathcal{A}_{[x/\mathcal{A}(t)]}} \\
& \text{gdw.} & \mathcal{A}_{[x/\mathcal{A}(t)]}(P(t_1, \ldots, t_n)) = 1 \\
& \text{gdw.} & \mathcal{A}_{[x/\mathcal{A}(t)]}(F) = 1
\end{array}
$$

Beweis des Überführungslemmas

- $F = \neg G$.

  Dann gilt:

$$\begin{aligned} \mathcal{A}(F[x/t]) = 1 \quad & \text{gdw.} \quad \mathcal{A}(\neg G[x/t]) = 1 \\ & \text{gdw.} \quad \mathcal{A}(G[x/t]) = 0 \\ & \text{gdw.} \quad \mathcal{A}_{[x/\mathcal{A}(t)]}(G) = 0 \\ & \text{gdw.} \quad \mathcal{A}_{[x/\mathcal{A}(t)]}(\neg G) = 1 \\ & \text{gdw.} \quad \mathcal{A}_{[x/\mathcal{A}(t)]}(F) = 1 \end{aligned}$$

- $F = F_1 \wedge F_2$ oder $F = F_1 \vee F_2$:

  Analoges Argument wie im vorherigen Fall.

Beweis des Überführungslemmas

- $F = \exists y G$, wobei y nicht in t vorkommt (denn t soll keine in F gebundene Variable enthalten).

  Wenn $y = x$, dann $\mathcal{A}(F[x/t]) = \mathcal{A}(F) = \mathcal{A}_{[x/\mathcal{A}(t)]}(F)$.

  Die letzte Gleichung gilt, weil x nicht frei in F vorkommt.

  Sei nun $y \neq x$. Dann gilt:

$$
\begin{aligned}
\mathcal{A}(F[x/t]) = 1 \quad & \text{gdw.} \quad \mathcal{A}(\exists y G[x/t]) = 1 \\
& \text{gdw.} \quad \text{es gibt } d \in U_{\mathcal{A}} \text{ mit } \mathcal{A}_{[y/d]}(G[x/t]) = 1 \\
& \text{gdw.} \quad \text{es gibt } d \in U_{\mathcal{A}} \text{ mit } \mathcal{A}_{[y/d][x/\mathcal{A}_{[y/d]}(t)]}(G) = 1 \\
& \text{gdw.} \quad \text{es gibt } d \in U_{\mathcal{A}_{[x/\mathcal{A}(t)]}} \text{ mit } \mathcal{A}_{[x/\mathcal{A}(t)][y/d]}(G) = 1 \\
& \qquad (\mathcal{A}_{[y/d]}(t) = \mathcal{A}(t), \text{ da } y \text{ nicht in } t \text{ vorkommt}) \\
& \text{gdw.} \quad \mathcal{A}_{[x/\mathcal{A}(t)]}(\exists y G) = 1 \\
& \text{gdw.} \quad \mathcal{A}_{[x/\mathcal{A}(t)]}(F) = 1
\end{aligned}
$$

Beweis von Skolemform(F) erfüllbar gdw. F erfüllbar

Beweis des Satzes von Folie 336:

Wir zeigen: Nach jedem Durchlauf durch die **while**-Schleife ist die erhaltene Formel erfüllbar, gdw. die Formel vor dem Durchlauf erfüllbar ist.

Formel vor dem Durchlauf durch die **while**-Schleife:

$$F = \forall y_1 \forall y_2 \cdots \forall y_n \exists z G$$

Formel nach dem Durchlauf durch die **while**-Schleife:

$$F' = \forall y_1 \forall y_2 \cdots \forall y_n G[z/f(y_1, y_2, \ldots, y_n)]$$

Hierbei ist f ein Funktionssymbol, dass in F nicht vorkommt.

Zu zeigen: F ist erfüllbar genau dann, wenn F' ist erfüllbar.

Beweis von Skolemform(F) erfüllbar gdw. F erfüllbar

(1) Sei $F' = \forall y_1 \forall y_2 \cdots \forall y_n G[z/f(y_1, y_2, \ldots, y_n)]$ erfüllbar.

Dann gibt es eine Struktur $\mathcal{A}$ (passend zu F') mit $\mathcal{A}(F') = 1$.

Dann ist $\mathcal{A}$ auch zu F passend und es gilt:

Für alle $d_1, \ldots, d_n \in U_\mathcal{A}$ gilt:

$$\mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n]}(G[z/f(y_1, y_2, \ldots, y_n)]) = 1$$

Mit dem Überführungslemma (ersetze dort $\mathcal{A} \to \mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n]}$, $t \to f(y_1, y_2, \ldots, y_n)$, $F \to G$, $x \to z$) folgt:

Für alle $d_1, \ldots, d_n \in U_\mathcal{A}$ gilt:

$$\mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n][z/d]}(G) = 1$$

wobei $d = \mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n]}(f(y_1, y_2, \ldots, y_n)) = f^\mathcal{A}(d_1, \ldots, d_n)$.

Beweis von Skolemform(F) erfüllbar gdw. F erfüllbar

Dies impliziert

$$\text{Für alle } d_1, \ldots, d_n \in U_{\mathcal{A}} \text{ gibt es ein } d \in U_{\mathcal{A}} \text{ mit:}$$
$$\mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n][z/d]}(G) = 1$$

d.h. $\mathcal{A}(\forall y_1 \forall y_2 \cdots \forall y_n \exists z G) = 1$.

(2) Sei $F = \forall y_1 \forall y_2 \cdots \forall y_n \exists z G$ erfüllbar.

Dann existiert eine Struktur $\mathcal{A}$ mit

$$\text{Für alle } d_1, \ldots, d_n \in U_{\mathcal{A}} \text{ gibt es ein } d \in U_{\mathcal{A}} \text{ mit:}$$
$$\mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n][z/d]}(G) = 1$$

Wir können also für alle $d_1, \ldots, d_n \in U_{\mathcal{A}}$ ein $u(d_1, \ldots, d_n) \in U_{\mathcal{A}}$ mit

$$\mathcal{A}_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n][z/u(d_1,\ldots,d_n)]}(G) = 1$$

auswählen.

Beweis von Skolemform(F) erfüllbar gdw. F erfüllbar

Wir definieren nun eine Struktur $\mathcal{A}'$ wie folgt:

- $\mathcal{A}'$ ist identisch zu $\mathcal{A}$ außer
- $f^{\mathcal{A}'}(d_1, \ldots, d_n) = u(d_1, \ldots, d_n)$ für alle $d_1, \ldots, d_n \in U_{\mathcal{A}}$.

Dann gilt:

Für alle $d_1, \ldots, d_n \in U_{\mathcal{A}} = U_{\mathcal{A}'}$ gilt:

$$\mathcal{A}'_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n][z/f^{\mathcal{A}'}(d_1,\ldots,d_n)]}(G) = 1$$

Mit dem Überführungslemma (ersetze dort $\mathcal{A} \to \mathcal{A}'_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n]}$, $t \to f(y_1, y_2, \ldots, y_n)$, $F \to G$, $x \to z$) folgt:

Für alle $d_1, \ldots, d_n \in U_{\mathcal{A}'}$ gilt:

$$\mathcal{A}'_{[y_1/d_1][y_2/d_2]\cdots[y_n/d_n]}(G[z/f(y_1, y_2, \ldots, y_n)]) = 1$$

und somit $\mathcal{A}'(\forall y_1 \forall y_2 \cdots \forall y_n G[z/f(y_1, y_2, \ldots, y_n)]) = 1$. □

Bemerkungen zur Skolemform

- Die Skolemform einer Formel F ist i.A. nicht äquivalent zu F.

  Beispiel: Die Skolemform von $\exists x P(x)$ ist $P(a)$ für eine Konstante a.

  Aber: $\exists x P(x) \not\equiv P(a)$
- Der obige Beweis (Punkt (1)) zeigt sogar, dass jedes Modell der Skolemform von F auch ein Modell von F ist.
- Ausserdem erhalten wir ein Modell der Skolemform von F indem wir ein Modell von F erweitern um eine Interpretation der neuen Funktionssymbole (die bei der Bildung der Skolemform eingeführt werden).

Skolemform einer Menge von Formeln

Sei nun M eine (i.A. unendliche) Menge von prädikatenlogischen Formeln, die o.B.d.A. alle in **BPF** sind.

Wir definieren die Skolemform von M als die Menge M' der Skolemformeln, die wir wie folgt erhalten:

Wir ersetzen in M jede Formel F durch seine Skolemform. Dabei achten wir darauf, dass wir für verschiedene Formeln $F, G \in M$ disjunkte Mengen von neuen Funktionssymbolen einführen.

Dann erhalten wir:

Satz 51

Sei M eine (i.A. unendliche) Menge von prädikatenlogischen Formeln in **BPF**. *Dann ist M erfüllbar, genau dann, wenn die Skolemform von M erfüllbar ist.*

Skolemform einer Menge von Formeln

Beweis: Sei $M' = \{F' \mid F \in M\}$ die Skolemform von M, wobei F' eine Skolemform von F ist.

(1) Sei $\mathcal{A}$ ein Modell von M' und sei $F \in M$ beliebig.

Dann ist $\mathcal{A}$ also ein Modell von F' und damit ein Modell von F.

(2) Sei $\mathcal{A}$ ein Modell von M

Für eine Formel $F \in M$ sei $\text{fun}(F)$ die Menge der neuen Funktionssymbole, die wir bei der Bildung der Skolemform F' von F eingeführt haben.

Also gilt $\text{fun}(F) \cap \text{fun}(G) = \emptyset$ für $F \neq G$ aus M.

Wir erhalten ein Modell von $F' \in M'$ indem wir $\mathcal{A}$ um eine Interpretation der Symbole aus $\text{fun}(F)$ erweitern.

Diese Erweiterungen liefern dann ein Modell von M'. □

Klauselform

Eine Aussage (= geschlossene Formel) heißt in Klauselform, falls sie die Bauart

$$\forall y_1 \forall y_2 \cdots \forall y_n \, F$$

hat, wobei F keine Quantoren enthält und in **KNF** ist, und $y_i \neq y_j$ für $i \neq j$.

Eine Ausage in Klauselform kann als Menge von Klauseln dargestellt werden.

Beispiel: Folgende Aussage ist in Klauselform:

$$\forall y_1 \forall y_2 \forall y_3 \forall y_4 ((P(y_1) \vee \neg Q(y_2, y_4)) \wedge (Q(y_1, y_3) \vee \neg P(y_3)) \wedge (Q(y_2, y_2) \vee P(y_4)))$$

Umformung einer beliebigen Formel in eine Aussage in Klauselform

Gegeben: eine prädikatenlogische Formel F (mit eventuellen Vorkommen von freien Variablen).

1. Bereinige F durch systematisches Umbenennen der gebundenen Variablen. Es entsteht eine zu F äquivalente Formel F_1.
2. Seien $y_1, y_2, \ldots, y_n$ die in F bzw. F_1 vorkommenden freien Variablen. Ersetze F_1 durch $F_2 = F_1[y_1/a_1, y_2/a_2, \ldots, y_n/a_n]$, wobei $a_1, \ldots, a_n$ verschiedene Konstanten, die noch nicht in F_1 vorkommen sind. Dann ist F_2 eine Aussage und erfüllbar genau dann, wenn F_1 erfüllbar ist.
3. Stelle eine zu F_2 äquivalente (und damit zu F erfüllbarkeits-äquivalente) Aussage F_3 in Pränexform her.
4. Eliminiere die vorkommenden Existenzquantoren durch Übergang zur Skolemform von F_3. Diese sei F_4 und ist dann erfüllbarkeitsäquivalent zu F_3 und damit auch zu F.
5. Forme die Matrix von F_4 um in **KNF** (und schreibe diese Formel F_5 dann als Klauselmenge auf).

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$				
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$				
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$				
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$				
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$				
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$				
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$				
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$				
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$				
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$				
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$				
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$				
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$	J	J	J	J
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$				
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$				
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$	J	J	J	J
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$	N	N	N	N
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$				
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$	J	J	J	J
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$	N	N	N	N
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$	J	N	N	N
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$				
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$	J	J	J	J
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$	N	N	N	N
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$	J	N	N	N
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$	J	N	N	N
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$				

Aufgabe zu Normalformen

Welche dieser Formel sind bereinigt, in Pränexform, in Skolemform, in Klauselform?

	B	P	S	K
$\forall x(Tet(x) \vee Cube(x) \vee Dodec(x))$	J	J	J	J
$\exists x \exists y(Cube(y) \vee BackOf(x, y))$	J	J	N	N
$\forall x(\neg FrontOf(x, x) \wedge \neg BackOf(x, x))$	J	J	J	J
$\neg \exists x Cube(x) \leftrightarrow \forall x \neg Cube(x)$	N	N	N	N
$\forall x(Cube(x) \rightarrow Small(x)) \rightarrow \forall y(\neg Cube(y) \rightarrow \neg Small(y))$	J	N	N	N
$(Cube(a) \wedge \forall x Small(x)) \rightarrow Small(a)$	J	N	N	N
$\exists x(Larger(a, x) \wedge Larger(x, b)) \rightarrow Larger(a, b)$	J	N	N	N

Herbrand-Universum

Sei $\mathcal{F} \subseteq \{f_i^k \mid i, k \geq 0\}$ eine Menge von Funktionssymbolen, die mindestens eine Konstante enthält.

Das Herbrand-Universum $D(\mathcal{F})$ ist die Menge aller variablenfreien Terme, die aus den Symbolen in $\mathcal{F}$ gebildet werden können.

Etwas formaler wird $D(\mathcal{F})$ wird wie folgt induktiv definiert:

Für jedes n-stellige Funktionssymbol $f \in \mathcal{F}$ ($n = 0$ ist hier möglich) und Terme $t_1, t_2, \ldots, t_n \in D(\mathcal{F})$ gehört auch der Term $f(t_1, t_2, \ldots, t_n)$ zu $D(\mathcal{F})$.

Beispiel: $D(\{f, a\}) = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$

Beachte: $D(\mathcal{F})$ ist endlich genau dann, wenn $\mathcal{F}$ nur Konstanten enthält.

Herbrand-Struktur

Eine Herbrand-Struktur ist eine Struktur $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, so dass eine Menge $\mathcal{F} \subseteq \{f_i^k \mid i, k \geq 0\}$ von Funktionssymbolen existiert mit:

- $\mathcal{F}$ enthält eine Konstante.
- $U_\mathcal{A} = D(\mathcal{F})$
- Ein Funktionssymbol f gehört zu $\text{Def}(I_\mathcal{A})$ genau dann, wenn $f \in \mathcal{F}$.
- Für alle $f \in \mathcal{F}$ (n-stellig) und $t_1, t_2, \ldots, t_n \in D(\mathcal{F})$ gilt $f^\mathcal{A}(t_1, t_2, \ldots, t_n) = f(t_1, t_2, \ldots, t_n)$

Für jede Herbrand-Struktur $\mathcal{A}$ wie oben und alle $t \in D(\mathcal{F})$ gilt $\mathcal{A}(t) = t$.

Sei M eine Menge von Formeln. Ein Herbrand-Modell von M ist ein Modell von M, welches gleichzeitig eine Herbrand-Struktur ist.

Der fundamentale Satz der Prädikatenlogik:

Satz 52

Sei M eine Menge von Aussagen in Skolemform. Dann gilt:

M ist erfüllbar $\iff$ M hat ein Herbrand-Modell.

M erfüllbar gdw. M hat ein Herbrand-Modell

Beweis:

Falls M ein Herbrand-Modell hat, ist M natürlich erfüllbar.

Sei nun M erfüllbar und sei $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ ein Modell von M.

Indem wir, falls nötig, M durch $M \cup \{P(a) \vee \neg P(a)\}$ ersetzen (a ist eine Konstante, P ist ein 1-stelliges Prädikatensymbol), können wir annehmen, dass in M eine Konstante vorkommt.

Sei $\mathcal{F}$ (bzw. $\mathcal{P}$) die Menge aller Funktionssymbole (bzw. Prädikatensymbole) die in M vorkommen.

Wir definieren nun ein Herbrand-Struktur $\mathcal{B} = (D(\mathcal{F}), I_\mathcal{B})$:

Wir müssen $I_\mathcal{B}$ noch auf den Prädikatensymbolen in $\mathcal{P}$ definieren.

Für alle n-stelligen $P \in \mathcal{P}$ und alle $t_1, \ldots, t_n \in D(\mathcal{F})$ sei:

$$(t_1, \ldots, t_n) \in P^\mathcal{B} \text{ gdw. } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)) \in P^\mathcal{A}$$

M erfüllbar gdw. M hat ein Herbrand-Modell

Behauptung: Für jede Aussage F in Skolemform, die aus den Symbolen in $\mathcal{F} \cup \mathcal{P}$ aufgebaut ist, gilt: Wenn $\mathcal{A}(F) = 1$, dann auch $\mathcal{B}(F) = 1$.

Falls F keine Quantoren enthält, zeigen wir sogar $\mathcal{A}(F) = \mathcal{B}(F)$ durch Induktion über den Aufbau von F:

- F ist atomar, d. h. $F = P(t_1, \ldots, t_n)$ für ein Prädikatensymbol $P \in \mathcal{P}$ und variablenfreie Terme $t_1, \ldots, t_n \in D(\mathcal{F})$.
  (beachte: F ist eine Aussage, d. h. F enthält keine freien Variablen).

$$\begin{aligned} \mathcal{A}(F) = 1 \quad & \textbf{gdw.} \quad (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)) \in P^{\mathcal{A}} \\ & \textbf{gdw.} \quad (t_1, \ldots, t_n) \in P^{\mathcal{B}} \\ & \textbf{gdw.} \quad (\mathcal{B}(t_1), \ldots, \mathcal{B}(t_n)) \in P^{\mathcal{B}} \\ & \textbf{gdw.} \quad \mathcal{B}(F) = 1 \end{aligned}$$

- $F = \neg G$: $\mathcal{A}(F) = 1$ **gdw.** $\mathcal{A}(G) = 0$ **gdw.** $\mathcal{B}(G) = 0$ **gdw.** $\mathcal{B}(F) = 1$
- $F = F_1 \wedge F_2$ oder $F = F_1 \vee F_2$: Analoges Argument wie für $F = \neg G$.

M erfüllbar gdw. M hat ein Herbrand-Modell

Damit ist der Fall, dass F keine Quantoren enthält, abgehandelt.

Den allgemeinen Fall behandeln wir durch Induktion über die Anzahl n der Quantoren in F.

Beachte: Da F in Skolemform ist, ist F von der Form $\forall y_1 \cdots \forall y_n H$, wobei H keine Quantoren enthält.

Induktionsanfang: $n = 0$:

Aus $\mathcal{A}(F) = 1$ folgt $\mathcal{B}(F) = 1$, siehe vorherige Folie.

Induktionsschritt: Sei $F = \forall x G$.

Aus $\mathcal{A}(\forall x G) = 1$ folgt:

$$\text{Für alle } d \in U_{\mathcal{A}} \text{ gilt } \mathcal{A}_{[x/d]}(G) = 1$$

Wegen $\{\mathcal{A}(t) \mid t \in D(\mathcal{F})\} \subseteq U_{\mathcal{A}}$ folgt:

$$\text{Für alle } t \in D(\mathcal{F}) \text{ gilt } \mathcal{A}_{[x/\mathcal{A}(t)]}(G) = 1$$

M erfüllbar gdw. M hat ein Herbrand-Modell

Mit dem Überführungslemma folgt:

$$\text{Für alle } t \in D(\mathcal{F}) \text{ gilt } \mathcal{A}(G[x/t]) = 1.$$

Die Aussage $G[x/t]$ ist wieder in Skolemform und hat nur $n-1$ Quantoren.

Nach Induktionsannahme gilt daher:

$$\text{Für alle } t \in D(\mathcal{F}) \text{ gilt } \mathcal{B}(G[x/t]) = 1.$$

Mit dem Überführungslemma folgt wieder:

$$\text{Für alle } t \in D(\mathcal{F}) \text{ gilt } \mathcal{B}_{[x/\mathcal{B}(t)]}(G) = \mathcal{B}_{[x/t]}(G) = 1$$

und damit $\mathcal{B}(F) = \mathcal{B}(\forall x G) = 1$. □

Bemerkung: Im soeben durchgeführten Beweis ist wichtig, dass alle $F \in M$ in Skolemform sind, und damit keine Existenzquantoren enthalten.

Der Satz von Löwenheim und Skolem

Eine Menge A ist abzählbar, falls A endlich ist, oder eine Bijektion $f : \mathbb{N} \to A$ existiert.

Beachte: Das Universum $D(\mathcal{F})$ einer Herbrand-Struktur ist abzählbar.

Satz von Löwenheim und Skolem

Jede erfüllbare Menge von Aussagen der Prädikatenlogik besitzt ein Modell mit einer abzählbaren Grundmenge (ein abzählbares Modell).

Beweis: Sei M eine erfüllbare Menge von Aussagen der Prädikatenlogik.

Sei M' die Skolemform von M.

Satz 51 (Folie 347) $\implies$ M' ist erfüllbar.

Satz 52 (Folie 353) $\implies$ M' hat ein Herbrand-Modell $\mathcal{B}$.

Bemerkung auf Folie 346 $\implies$ $\mathcal{B}$ ist auch ein Modell von M.

Außerdem ist $\mathcal{B}$ abzählbar. □

Herbrand-Expansion

Sei M eine Menge von Aussagen in Skolemform.

Sei $\mathcal{F}$ die Menge der Funktionssymbole, die in Formeln aus M vorkommen, und sei a eine fest gewählte Konstante.

Wir definieren:

$$D(M) = \begin{cases} D(\mathcal{F}) & \text{falls } \mathcal{F} \text{ eine Konstante enthält} \\ D(\mathcal{F} \cup \{a\}) & \text{sonst} \end{cases}$$

Im folgenden bezeichnet F^* stets eine quantorenfreie Formel.

Die Menge von Aussagen

$$E(M) = \{F^*[y_1/t_1][y_2/t_2]\ldots[y_n/t_n] \mid \forall y_1 \forall y_2 \cdots \forall y_n F^* \in M, \\ t_1, t_2, \ldots, t_n \in D(M)\}$$

ist die Herbrand-Expansion von M.

Herbrand-Expansion

Die Formeln in $E(M)$ entstehen also, indem die Terme aus $D(M)$ in jeder möglichen Weise für die Variablen in F ($F \in M$) substituiert werden.

Beachte: Wenn M erfüllbar ist, gibt es ein Herbrand-Modell von M mit Universum $D(M)$.

Beispiel: Für $M = \{\forall x \forall y (P(a, x) \wedge \neg R(f(y)))\}$ gilt

$$D(M) = \{a, f(a), f(f(a)), \ldots\}.$$

Die Herbrand-Expansion von M ist damit

$$\begin{aligned} E(M) = \{ & P(a, a) \wedge \neg R(f(a)), \\ & P(a, f(a)) \wedge \neg R(f(a)), \\ & P(a, a) \wedge \neg R(f(f(a))), \\ & P(a, f(a)) \wedge \neg R(f(f(a))), \ldots\} \end{aligned}$$

Herbrand-Expansion

Wir betrachten die Herbrand-Expansion von M im folgenden als eine Menge von aussagenlogischen Formeln.

Die atomaren Formeln sind hierbei von der Gestalt $P(t_1, \ldots, t_n)$, wobei P ein in M vorkommendes Prädikatensymbol ist und $t_1, \ldots, t_n \in D(M)$.

Im Beispiel auf der vorherigen Folie kommen in der Herbrand-Expansion

$$E(M) = \{P(a, f^n(a)) \wedge \neg R(f^m(a)) \mid n \geq 0, m \geq 1\}$$

(hierbei ist $f^n(a)$ eine Abkürzung für den Term $f(f(\cdots f(a)\cdots))$, wobei f genau n-mal vorkommt) genau die atomoren Formeln aus der Menge

$$\{P(a, f^n(a)) \mid n \geq 0\} \cup \{R(f^m(a)) \mid m \geq 1\}$$

vor.

Herbrand-Expansion

Die Belegung $\mathcal{B}$ mit

$$\mathcal{B}(P(a, f^n(a))) = 1 \text{ für } n \geq 0 \quad \text{und} \quad \mathcal{B}(R(f^m(a))) = 0 \text{ für } m \geq 1$$

erfüllt offenbar auch $E(M)$ im aussagenlogischen Sinn: $\mathcal{B}(G) = 1$ für alle $G \in E(M)$.

Auch die (einzige) Formel $\forall x \forall y(P(a, x) \wedge \neg R(f(y))) \in M$ ist erfüllbar. Wie der folgende Satz zeigt, ist dies kein Zufall.

Satz von Gödel-Herbrand-Skolem

Satz von Gödel-Herbrand-Skolem

Sei M eine Menge von Aussagen in Skolemform. Dann ist M erfüllbar genau dann, wenn die Formelmenge $E(M)$ (im aussagenlogischen Sinn) erfüllbar ist.

Beweis: Es genügt zu zeigen, dass M ein Herbrand-Modell mit Universum $D(M)$ besitzt genau dann, wenn $E(M)$ erfüllbar ist:

$\mathcal{A}$ *ist ein Herbrand-Modell für* M *mit Universum* $D(M)$

gdw. *für alle* $\forall y_1 \forall y_2 \cdots \forall y_n F^* \in M,\ t_1, t_2, \ldots, t_n \in D(M)$ *gilt*

$$\mathcal{A}_{[y_1/t_1][y_2/t_2]\ldots[y_n/t_n]}(F^*) = 1$$

gdw. *für alle* $\forall y_1 \forall y_2 \cdots \forall y_n F^* \in M,\ t_1, t_2, \ldots, t_n \in D(M)$ *gilt*

$$\mathcal{A}(F^*[y_1/t_1][y_2/t_2]\ldots[y_n/t_n]) = 1$$

gdw. *für alle* $G \in E(M)$ *gilt* $\mathcal{A}(G) = 1$

gdw. $\mathcal{A}$ *ist ein Modell für* $E(M)$

Satz von Gödel-Herbrand-Skolem

In dieser Kette von Äquivalenzen ist $\mathcal{A}$ ein Modell von $E(M)$ im Sinne der Prädikatenlogik.

Daraus erhalten wir einfach eine Modell für $E(M)$ im Sinne der Ausagenlogik:
Für alle atomaren Formeln $P(t_1, \ldots, t_n)$ mit $t_1, \ldots, t_n \in D(M)$ gelte

$$\mathcal{B}(P(t_1, \ldots, t_n)) = \mathcal{A}(P(t_1, \ldots, t_n)). \tag{16}$$

Damit gilt dann:

$$\mathcal{A} \text{ ist ein Modell für } E(M) \textbf{ gdw. } \mathcal{B} \text{ ist ein Modell für } E(M)$$

Umgekehrt liefert ein Modell $\mathcal{B}$ für $E(M)$ im Sinne der Ausagenlogik mit der Vorschrift (16) auch wieder ein (Herbrand-)Modell $\mathcal{A}$ für $E(M)$ im Sinne der Prädikatenlogik. □

Endlichkeitssatz der Prädikatenlogik

Endlichkeitssatz der Prädikatenlogik (Gödel 1930)

Eine Menge M von prädikatenlogischen Formeln ist erfüllbar genau dann, wenn jede endliche Teilmenge von M erfüllbar ist.

Beweis: Es genügt die "wenn"-Richtung zu zeigen.

Zunächst ersetzten wir jede freie Variable einer Formel $F \in M$ durch ein neue Konstante a, die in M noch nicht vorkommt (wie auf Folie 350).

Wichtig: Wenn x sowohl frei in $F \in M$ als auch frei in $G \in M$ vorkommt, muss x in beiden Formeln durch die gleiche Konstante a ersetzt werden.

Sei M' die neue Menge von Formeln.

Dann ist M erfüllbar genau dann, wenn M' erfüllbar ist.

Endlichkeitssatz der Prädikatenlogik

Bilde nun die Skolemform M'' von M' (siehe Folie 347).

Nach Satz 51 (Folie 347) ist M'' erfüllbar genau dann, wenn M' (bzw. M erfüllbar) ist.

Wir können also o.B.d.A. annehmen, dass M eine Menge von Aussagen in Skolemform ist.

Sei jede endliche Teilmenge N von M erfüllbar.

Satz von Gödel-Herbrand-Skolem $\Rightarrow$ Für jede endliche Teilmenge $N \subseteq M$ ist die Herbrand-Expansion $E(N)$ im aussagenlogischen Sinn erfüllbar.

Insbesondere ist jede endliche Teilmenge von $E(M)$ im aussagenlogischen Sinn erfüllbar (für jede endlicher Menge $A \subseteq E(M)$ existiert eine endliche Menge $B \subseteq M$ mit $A \subseteq E(B)$).

Endlichkeitssatz der Aussagenlogik (Folie 293) $\Rightarrow$ $E(M)$ erfüllbar.

Satz von Gödel-Herbrand-Skolem $\Rightarrow$ M erfüllbar. $\square$

Satz von Herbrand

Satz von Herbrand

Eine Aussage F in Skolemform ist unerfüllbar genau dann, wenn es eine endliche Teilmenge von $E(F)$ gibt, die (im aussagenlogischen Sinn) unerfüllbar ist.

Beweis: Ummittelbare Folge des Satzes von Gödel-Herbrand-Skolem und des Endlichkeitssatzes der Aussagenlogik (Folie 293). □

Algorithmus von Gilmore

Sei F eine prädikatenlogische Aussage in Skolemform und sei $\{F_1, F_2, F_3, \ldots,\}$ eine Aufzählung der Herbrand-Expansion $E(F)$.

Algorithmus von Gilmore

Eingabe: F
$n := 0$;
repeat $n := n + 1$;
until $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$ ist unerfüllbar;
Gib "unerfüllbar" aus und stoppe.

Wenn F unerfüllbar ist, gibt es nach dem Satz von Herbrand eine endliche Menge $M \subseteq E(F)$, die (im Sinne der Aussagenlogik) unerfüllbar ist.

Es gibt dann ein n, so dass $M \subseteq \{F_1, \ldots, F_n\}$.

Also ist $\{F_1, \ldots, F_n\}$ unerfüllbar und damit $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ unerfüllbar.

Die gültigen Formeln sind semi-entscheidbar

Umgekehrt: Ist F erfüllbar, so ist $E(F)$ erfüllbar und damit auch jede Formel $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ erfüllbar.

Wir erhalten somit:

Satz

Sei F eine prädikatenlogische Aussage in Skolemform. Dann gilt:

- Wenn die Eingabeformel F unerfüllbar ist, dann terminiert der Algorithmus von Gilmore nach endlicher Zeit mit dem Output "unerfüllbar".
- Wenn die Eingabeformel F erfüllbar ist, dann terminiert der Algorithmus von Gilmore nicht, d. h. er läuft unendlich lange.

Die Menge der unerfüllbaren prädikatenlogischen Aussagen ist also semi-entscheidbar (rekursive aufzählbar).

Das Erfüllbarkeitsproblem ist nicht entscheidbar

Korollar

Die Menge der gültigen prädikatenlogischen Aussagen ist semi-entscheidbar.

Beweis: F ist gültig, genau dann, wenn $\neg F$ unerfüllbar ist.

In der Vorlesung Advanced Logic (jedes Sommersemester) wird gezeigt: Die Menge der (un)erfüllbaren prädikatenlogischen Aussagen ist unentscheidbar.

Resolution in der Prädikatenlogik

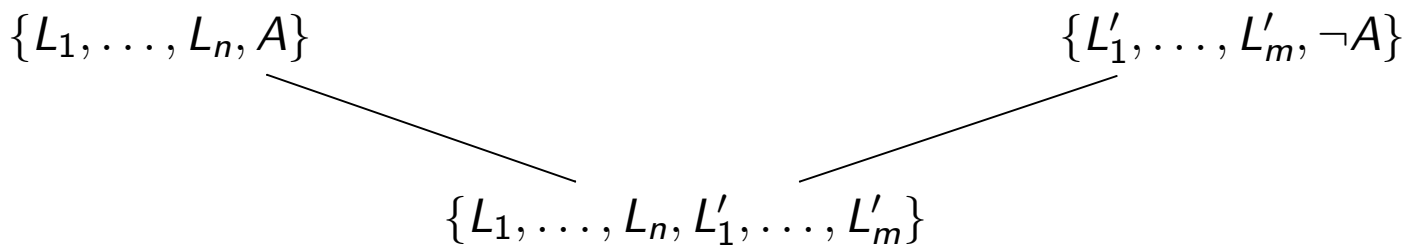
Der Algorithmus von Gilmore funktioniert zwar, ist in der Praxis aber unbrauchbar.

Daher ist unser Programm der nächsten Stunden:

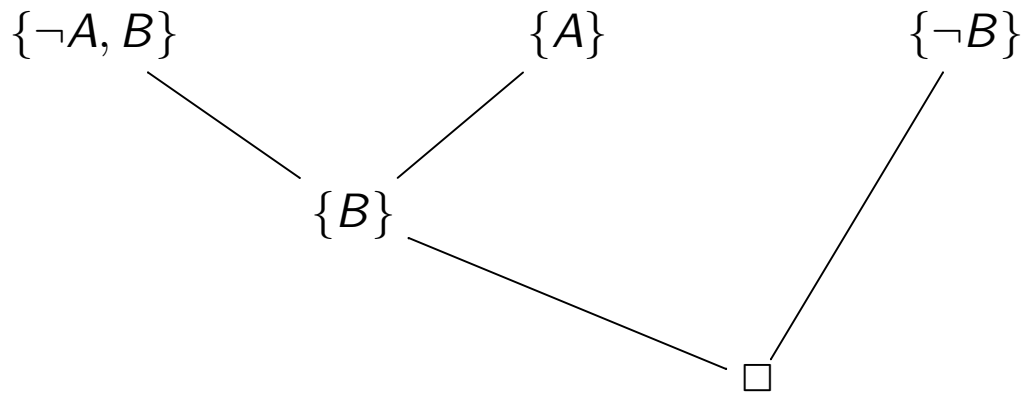
Wie sieht Resolution in der Prädikatenlogik aus?

Wiederholung: Resolution in der Aussagenlogik

Resolutionsschritt:



Mini-Beispiel:



Eine Klauselmenge ist unerfüllbar genau dann, wenn die leere Klausel abgeleitet werden kann.

Anpassung des Algorithmus von Gilmore

Algorithmus von Gilmore:
Sei F eine prädikatenlogische Aussage in Skolemform und sei $\{F_1, F_2, F_3, \ldots\}$ eine Aufzählung von $E(F)$.

Eingabe: F
$n := 0$;
repeat $n := n + 1$;
until $(F_1 \wedge F_2 \wedge \ldots \wedge F_n)$ ist unerfüllbar;
(dies kann mit Mitteln der Aussagenlogik, z.B. Wahrheitstafeln, getestet werden)
Gib "unerfüllbar" aus und stoppe.

"Mittel der Aussagenlogik" $\rightsquigarrow$ wir verwenden Resolution für den Unerfüllbarkeitstest

Definition von $Res(F)$ (Wiederholung)

Definition: Sei F eine Klauselmenge. Dann ist $Res(F)$ definiert als

$$Res(F) = F \cup \{R \mid R \text{ ist Resolvent zweier Klauseln in } F\}.$$

Außerdem setzen wir:

$$\begin{aligned} Res^0(F) &= F \\ Res^{n+1}(F) &= Res(Res^n(F)) \qquad \text{für } n \geq 0 \end{aligned}$$

Schließlich sei

$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F).$$

Grundresolutionsalgorithmus

Sei $F_1, F_2, F_3, \ldots$ wieder eine Aufzählung der Herbrand-Expansion von F.

F sei in Klauselform, d.h. $F = \forall y_1 \forall y_2 \cdots \forall y_n F^*$, wobei F^* in **KNF** ist.

Wir betrachten F^* als eine Menge von Klauseln, dann ist auch jedes F_i eine Menge von Klauseln.

Wir wissen bereits: F ist unerfüllbar genau dann, wenn es ein n gibt, so dass $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ unerfüllbar ist.

Beachte: $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ ist wieder in **KNF** (im Sinne der Aussagenlogik) und kann mit der Menge von Klauseln $\bigcup_{i=1}^{n} F_i$ identifiziert werden.

Aus der Aussagenlogik wissen wir:

$$F_1 \wedge F_2 \wedge \cdots \wedge F_n \text{ ist unerfüllbar} \iff Res^*\left(\bigcup_{i=1}^{n} F_i\right) \text{ ist unerfüllbar}$$

$$\iff \square \in Res^*\left(\bigcup_{i=1}^{n} F_i\right)$$

Grundresolutionsalgorithmus

Dies führt zum Grundresolutionsalgorithmus:

Eingabe: eine Aussage F in Skolemform mit der Matrix F^* in **KNF**
$i := 0$;
$M := \emptyset$;
repeat
$\quad i := i + 1$; $M := M \cup F_i$; $M := Res^*(M)$
until $\square \in M$
Gib "unerfüllbar" aus und stoppe.

Warum der Name Grundresolution?

Im Gegensatz zu späteren Verfahren werden Terme ohne Variablen (auch bekannt als Grundterme) substituiert, um die Formeln der Herbrand-Expansion zu erhalten.

Grundresolutionsalgorithmus

Beispiel: Betrachte die folgende Aussage in Klauselform:

$$F = \forall x \forall y((P(x) \vee \neg Q(y, a)) \wedge (Q(f(x), y) \vee \neg P(y)))$$

Die ersten drei Formeln der Herbrand-Expansion von F sind:

$$\begin{aligned} F_1 &= (P(a) \vee \neg Q(a, a)) \wedge (Q(f(a), a) \vee \neg P(a)), \\ F_2 &= (P(f(a)) \vee \neg Q(a, a)) \wedge (Q(f(f(a)), a) \vee \neg P(a)), \\ F_3 &= (P(a) \vee \neg Q(f(a), a)) \wedge (Q(f(a), f(a)) \vee \neg P(f(a))) \end{aligned}$$

In Mengenschreibweise:

$$\begin{aligned} F_1 &= \{\{P(a), \neg Q(a, a)\}, \{Q(f(a), a), \neg P(a)\}\}, \\ F_2 &= \{\{P(f(a)), \neg Q(a, a)\}, \{Q(f(f(a)), a), \neg P(a)\}\}, \\ F_3 &= \{\{P(a), \neg Q(f(a), a)\}, \{Q(f(a), f(a)), \neg P(f(a))\}\} \end{aligned}$$

Grundresolutionsalgorithmus

Dann erhalten wir nach den ersten drei Durchläufen durch die **repeat**-Schleife für die Mengenvariable M folgende Werte:

Nach 1. Durchlauf:

$$\{P(a), \neg Q(a,a)\},\ \{Q(f(a),a), \neg P(a)\},$$
$$\{\neg Q(a,a), Q(f(a),a)\}$$

Nach 2. Durchlauf:

$$\{P(a), \neg Q(a,a)\},\ \{Q(f(a),a), \neg P(a)\},\ \{\neg Q(a,a), Q(f(a),a)\},$$
$$\{P(f(a)), \neg Q(a,a)\},\ \{Q(f(f(a)),a), \neg P(a)\}$$
$$\{\neg Q(a,a), Q(f(f(a)),a)\}$$

Grundresolutionsalgorithmus

Nach 3. Durchlauf:

$$\{P(a), \neg Q(a,a)\},\ \{Q(f(a),a), \neg P(a)\},\ \{\neg Q(a,a), Q(f(a),a)\},$$
$$\{P(f(a)), \neg Q(a,a)\},\ \{Q(f(f(a)),a), \neg P(a)\},$$
$$\{\neg Q(a,a), Q(f(f(a)),a)\},$$
$$\{P(a), \neg Q(f(a),a)\},\ \{Q(f(a),f(a)), \neg P(f(a))\},$$
$$\{Q(f(a),a), \neg Q(f(a),a)\},\ \{Q(f(f(a)),a), \neg Q(f(a),a)\},$$
$$\{P(a), \neg P(a)\},\ \{\neg Q(a,a), P(a)\}, \{Q(f(a),f(a)), \neg Q(a,a)\}$$

Grundresolutionssatz

Den Grundresolutionsalgorithmus kann man auch in folgenden Grundresolutionssatz umformulieren:

Grundresolutionssatz

Eine Aussage in Skolemform $F = \forall y_1 \ldots \forall y_k F^*$ mit der Matrix F^* in **KNF** ist unerfüllbar genau dann, wenn es eine Folge von Klauseln $K_1, \ldots, K_n$ gibt mit der Eigenschaft:

- K_n ist die leere Klausel
- Für alle $i \in \{1, \ldots, n\}$ gilt:
  - entweder ist K_i eine Grundinstanz einer Klausel $K \in F^*$,d.h. $K_i = K[y_1/t_1] \ldots [y_k/t_k]$ mit $t_i \in D(F)$
  - oder K_i ist (aussagenlogischer) Resolvent zweier Klauseln K_a, K_b mit $a < i$ und $b < i$

Weglassen von Klauseln und Resolutionsschritten, die nicht zur Herleitung der leeren Klausel beitragen.

Substitutionen

Eine Substitution sub ist eine Abbildung von einer endlichen Menge von Variablen in die Menge aller Terme.

Sei $\mathrm{Def}(\mathrm{sub})$ der Definitionsbereich der Substitution sub.

Für einen Term t definieren wir den Term $t\,\mathrm{sub}$ (Anwendung der Substitution sub auf den Term t) wie folgt induktiv.

- $x\,\mathrm{sub} = \mathrm{sub}(x)$, falls $x \in \mathrm{Def}(\mathrm{sub})$.
- $y\,\mathrm{sub} = y$, falls $y \notin \mathrm{Def}(\mathrm{sub})$.
- $f(t_1, \ldots, t_n)\,\mathrm{sub} = f(t_1\,\mathrm{sub}, \ldots, t_n\,\mathrm{sub})$ für Terme $t_1, \ldots, t_n$ und ein n-stelliges Funktionssymbol f
  (dies impliziert $a\,\mathrm{sub} = a$, falls a eine Konstante ist)

Für ein Literal F (= evtl. negierte atomare Formel) definieren wir $F\,\mathrm{sub}$ wie folgt, wobei P ein n-stelliges Prädikatensymbol ist, und $t_1, \ldots, t_n$ Terme sind:

$$\begin{aligned} P(t_1, \ldots, t_n)\,\mathrm{sub} &= P(t_1\,\mathrm{sub}, \ldots, t_n\,\mathrm{sub}) \\ \neg P(t_1, \ldots, t_n)\,\mathrm{sub} &= \neg P(t_1\,\mathrm{sub}, \ldots, t_n\,\mathrm{sub}) \end{aligned}$$

Substitutionen und Ersetzungen

Eine Ersetzung $[x/t]$ (x ist eine Variable, t ein Term) kann mit der Substitution sub mit $\mathrm{Def}(\mathrm{sub}) = \{x\}$ und $\mathrm{sub}(x) = t$ identifiziert werden.

Eine Substitution sub mit $\mathrm{Def}(\mathrm{sub}) = \{y_1, \ldots, y_n\}$ (jedes y_i ist eine Variable) kann auch als Folge von Ersetzungen $[y_1/t_1][y_2/t_2]\cdots[y_n/t_n]$ geschrieben werden.

Beachte: Ersetzungen werden von links nach rechts durchgeführt!

Beispiel: Die Substitution sub mit $\mathrm{Def}(\mathrm{sub}) = \{x, y, z\}$ und

$$\mathrm{sub}(x) = f(h(w)), \quad \mathrm{sub}(y) = g(a, h(w)), \quad \mathrm{sub}(z) = h(w)$$

ist gleich der Substitution

$$[x/f(z)]\,[y/g(a,z)]\,[z/h(w)].$$

Verknüpfung von Substitutionen: Bei $\mathrm{sub}_1\mathrm{sub}_2$ wird zuerst sub_1 angewandt, anschließend sub_2.

Vertauschen von Substitutionen

Lemma (Regel für das Vertauschen von Substitutionen)

Falls (i) $x \notin \mathrm{Def}(\mathrm{sub})$ und (ii) x in keinem der Terme $y\,\mathrm{sub}$ mit $y \in \mathrm{Def}(\mathrm{sub})$ vorkommt, so gilt

$$[x/t]\mathrm{sub} = \mathrm{sub}[x/t\,\mathrm{sub}].$$

Beispiele:

- $[x/f(y)]\underbrace{[y/g(z)]}_{\mathrm{sub}} = \underbrace{[y/g(z)]}_{\mathrm{sub}}[x/f(g(z))]$
- aber $[x/f(y)]\underbrace{[x/g(z)]}_{\mathrm{sub}} \neq \underbrace{[x/g(z)]}_{\mathrm{sub}}[x/f(y)]$

Vertauschen von Substitutionen

Beweis des Lemmas:

Wir zeigen $t'[x/t]\mathrm{sub} = t'\mathrm{sub}[x/t\,\mathrm{sub}]$ für alle Terme t' durch Induktion über den Aufbau von t'.

- $t' = x$:
  Dann gilt $x[x/t]\mathrm{sub} = t\,\mathrm{sub}$ und ebenso
  $x\,\mathrm{sub}[x/t\,\mathrm{sub}] = x[x/t\,\mathrm{sub}] = t\,\mathrm{sub}$, da $x\,\mathrm{sub} = x$ wegen
  $x \notin \mathrm{Def}(\mathrm{sub})$.
- $t' = y$ für eine Variable $y \neq x$:
  Dann gilt $y[x/t]\mathrm{sub} = y\,\mathrm{sub}$ und ebenso $y\,\mathrm{sub}[x/t\,\mathrm{sub}] = y\,\mathrm{sub}$, da x in $y\,\mathrm{sub}$ nicht vorkommt.
- $t' = f(t_1, \ldots, t_n)$:
  Diesen Fall kann man sofort mit der Induktionsannahme für $t_1, \ldots, t_n$ erledigen. □

Unifikator/Allgemeinster Unifikator

Gegeben sei eine Menge $\mathbf{L} = \{L_1, \ldots, L_k\}$ ($k \geq 1$) von Literalen (= evtl. negierte atomare prädikatenlogische Formeln).

Eine Substitution sub heißt Unifikator von $\mathbf{L}$, falls

$$L_1\mathrm{sub} = L_2\mathrm{sub} = \cdots = L_k\mathrm{sub}$$

Das ist gleichbedeutend mit $|\mathbf{L}\,\mathrm{sub}| = 1$, wobei
$\mathbf{L}\,\mathrm{sub} = \{L_1\mathrm{sub}, \ldots, L_k\mathrm{sub}\}$.

Ein Unifikator sub von $\mathbf{L}$ heißt allgemeinster Unifikator von $\mathbf{L}$, falls für jeden Unifikator sub' von $\mathbf{L}$ eine Substitution s mit $\mathrm{sub}' = \mathrm{sub}\,s$ existiert.

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		
	$P(x)$	$P(f(y))$		
	$P(x, f(y))$	$P(f(u), z)$		
	$P(x, f(y))$	$P(f(u), f(z))$		
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$		
	$P(x, f(y))$	$P(f(u), z)$		
	$P(x, f(y))$	$P(f(u), f(z))$		
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$		
	$P(x, f(y))$	$P(f(u), f(z))$		
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$	✓	
	$P(x, f(y))$	$P(f(u), f(z))$		
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$	✓	
	$P(x, f(y))$	$P(f(u), f(z))$	✓	
	$P(x, f(x))$	$P(f(y), y)$		
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$	✓	
	$P(x, f(y))$	$P(f(u), f(z))$	✓	
	$P(x, f(x))$	$P(f(y), y)$		✓
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$		
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$	✓	
	$P(x, f(y))$	$P(f(u), f(z))$	✓	
	$P(x, f(x))$	$P(f(y), y)$		✓
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$	✓	
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$		

Aufgabe

Unifizierbar?			Ja	Nein
	$P(f(x))$	$P(g(y))$		✓
	$P(x)$	$P(f(y))$	✓	
	$P(x, f(y))$	$P(f(u), z)$	✓	
	$P(x, f(y))$	$P(f(u), f(z))$	✓	
	$P(x, f(x))$	$P(f(y), y)$		✓
	$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$	✓	
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$	✓	

Aufgabe

- Ein Unifikator für $\{P(x, g(x), g^2(x)), P(f(z), w, g(w))\}$:

$$x \mapsto f(z),\ \ w \mapsto g(f(z))$$

- Ein Unifikator für $\{P(x, f(y)), P(g(y), f(a)), P(g(a), z)\}$:

$$x \mapsto g(a),\ \ y \mapsto a,\ \ z \mapsto f(a)$$

Unifikationsalgorithmus

Eingabe: eine endliche Literalmenge $\mathbf{L} \neq \emptyset$

$\mathrm{sub} := []$; (leere Substitution, d. h. $\mathrm{Def}([]) = \emptyset$)
while $|\mathbf{L}\,\mathrm{sub}| > 1$ **do**
    Nimm zwei verschiedene Literale $L_1, L_2 \in \mathbf{L}\,\mathrm{sub}$.
    Suche die erste Position p, an der sich L_1 und L_2 unterscheiden.
    if keines der beiden Symbole an Position p ist eine Variable **then**
        stoppe mit "nicht unifizierbar"
    else sei x die Variable und t, der Term im anderen Literal der an Position p beginnt (möglicherweise auch eine Variable)
        if x kommt in t vor **then**
            stoppe mit "nicht unifizierbar"
        else $\mathrm{sub} := \mathrm{sub}\,[x/t]$
endwhile

Ausgabe: sub

Unifikationsalgorithmus

Beispiele: (die Position p ist rot markiert)

$$\begin{aligned} L_1 &= P(f(a,x), f(a,y), g(z)) \\ L_2 &= P(f(a,x), g(x), g(z)) \end{aligned}$$

Hier stoppt der der Algorithmus mit "nicht unifizierbar".

$$\begin{aligned} L_1 &= P(f(a,x), y, g(z)) \\ L_2 &= P(f(a,x), g(x), g(z)) \end{aligned}$$

Hier wäre der Term $t = g(x)$ und wir setzen $\mathrm{sub} := \mathrm{sub}\,[y/g(x)]$.

$$\begin{aligned} L_1 &= P(f(a,x), x, g(z)) \\ L_2 &= P(f(a,x), g(x), g(z)) \end{aligned}$$

Hier stoppt der der Algorithmus mit "nicht unifizierbar".

Korrektheit des Unifikationsalgorithmus

Satz

Es gilt:

(A) Der Unifikationsalgorithmus terminiert für jede Eingabe **L**.

(B) Wenn die Eingabe **L** nicht unifizierbar ist, so terminiert der Unifikationsalgorithmus mit der Ausgabe "nicht unifizierbar".

(C) Wenn die Eingabe **L** unifizierbar ist, dann findet der Unifikationsalgorithmus immer einen allgemeinsten Unifikator von **L**.

(C) impliziert insbesondere, dass jede unifizierbare Menge von Literalen einen allgemeinsten Unifikator hat.

Beweis der Korrektheit des Unifikationsalgorithmus

Beweis:

(A) Der Unifikationsalgorithmus terminiert für jede Eingabe **L**.

Dies gilt, denn die Anzahl der in $\mathbf{L}\,\mathrm{sub}$ vorkommenden Variablen wird in jedem Schritt kleiner.

Betrachte hierzu einen Durchlauf durch die **while**-Schleife.

Falls der Algorithmus in diesem Durchlauf nicht terminiert, so wird sub auf $\mathrm{sub}\,[x/t]$ gesetzt.

Hierbei kommt x in $\mathbf{L}\,\mathrm{sub}$ vor und der Term t enthält x nicht.

Also kommt x in $\mathbf{L}\,\mathrm{sub}\,[x/t]$ nicht mehr vor.

Beweis der Korrektheit des Unifikationsalgorithmus

(B) Wenn die Eingabe **L** nicht unifizierbar ist, so terminiert der Unifikationsalgorithmus mit der Ausgabe “nicht unifizierbar”.

Sei die Eingabe **L** nicht unifizierbar.

Falls die Bedingung $|\mathbf{L}\,\mathrm{sub}| > 1$ der **while**-Schleife irgendwann verletzt wäre, so wäre **L** doch unifizierbar.

Da nach (A) der Algorithmus bei Eingabe **L** terminiert, muss schließlich “nicht unifizierbar” ausgegeben werden.

Beweis der Korrektheit des Unifikationsalgorithmus

(C) Wenn die Eingabe **L** unifizierbar ist, dann findet der Unifikationsalgorithmus immer einen allgemeinsten Unifikator von **L**.

Sei **L** unifizierbar und sei sub_i ($i \geq 0$) die nach dem i-ten Durchlauf der **while**-Schleife berechnete Substituiton.

Angenommen der Algorithmus macht N Durchläufe durch die **while**-Schleife.

Beachte: $\mathrm{sub}_0 := []$ und sub_N ist die Ausgabe des Algorithmus (falls diese existiert).

Behauptung:

(1) Für jeden Unifikator sub' von **L** und für alle $0 \leq i \leq N$ existiert eine Substitution s_i mit $\mathrm{sub}' = \mathrm{sub}_i \; s_i$.

(2) Im i-ten Durchlauf durch die **while**-Schleife ($1 \leq i \leq N$) terminiert der Algorithmus entweder erfolgreich (und gibt die Substitution sub_N aus) oder der Algorithmus betritt die beiden **else**-Zweige.

Beweis der Korrektheit des Unifikationsalgorithmus

Beweis der Behauptung:

Sei sub' ein Unifikator von **L**.

Zunächst betrachten wir durch Induktion über i den Fall, dass **L** und $\{y\,\mathrm{sub}' \mid y \in \mathrm{Def}(\mathrm{sub}')\}$ keine gemeinsamen Variablen enthalten.

Wir werden s_i als eine Einschränkung von sub' wählen, d. h. $\mathrm{Def}(s_i) \subseteq \mathrm{Def}(\mathrm{sub}')$ und $s_i(x) = \mathrm{sub}'(x)$ für alle $x \in \mathrm{Def}(s_i)$.

Dann enthalten auch **L** und $\{y\,s_i \mid y \in \mathrm{Def}(s_i)\}$ keine gemeinsamen Variablen.

Induktionsanfang: $i = 0$.

Es gilt: $\mathrm{sub}' = []\,\mathrm{sub}' = \mathrm{sub}_i\,\mathrm{sub}'$. Wir können also $s_0 = \mathrm{sub}'$ setzen.

Beweis der Korrektheit des Unifikationsalgorithmus

Induktionsschritt: Sei $i > 0$ und sei (1) und (2) bereits für $i - 1$ bewiesen.

Nach Induktionshypothese existiert eine Einschränkung s_{i-1} von sub' mit $\mathrm{sub}' = \mathrm{sub}_{i-1} s_{i-1}$.

Falls $|\mathbf{L}\,\mathrm{sub}_{i-1}| = 1$, so terminiert der Algorithmus im i-ten Durchlauf.

Sei nun $|\mathbf{L}\,\mathrm{sub}_{i-1}| > 1$.

Betrachte die erste Position p, an der sich zwei Literale L_1 und L_2 aus $\mathbf{L}\mathrm{sub}_{i-1}$ unterscheiden.

Wegen $|\mathbf{L}\,\mathrm{sub}_{i-1} s_{i-1}| = |\mathbf{L}\,\mathrm{sub}'| = 1$ gilt $L_1 s_{i-1} = L_2 s_{i-1}$.

Also können an Position p in L_1 und L_2 nicht zwei verschiedene Funktionssymbole stehen.

Stehe in L_1 an Position p etwa eine Variable x und in L_2 beginnt an Position p ein Term $t \neq x$.

Beweis der Korrektheit des Unifikationsalgorithmus

Dann gilt $x\, s_{i-1} = t\, s_{i-1}$.

In t kann die Variable x nicht vorkommen:

Dies ist klar, wenn t eine Variable (da $t \neq x$) oder Konstante ist.

Ist t von der Form $f(t_1, \ldots, t_n)$ mit $n \geq 1$, so muss
$x\, s_{i-1} = t\, s_{i-1} = f(t_1 s_{i-1}, \ldots, t_n s_{i-1})$ gelten.
Würde x in einem der Terme t_i vorkommen, so würde $f(t_1 s_{i-1}, \ldots, t_n s_{i-1})$ mehr Symbole als $x\, s_{i-1}$ enthalten.

Also werden die beiden **else**-Zweige im Rumpf der **while**-Schleife betreten (dies zeigt (2)).

Es gilt $\mathrm{sub}_i = \mathrm{sub}_{i-1}[x/t]$.

Sei s_i die Einschränkung von s_{i-1} auf alle von x verschiedenen Variablen.

Beweis der Korrektheit des Unifikationsalgorithmus

Dann gilt:

$$
\begin{aligned}
\mathrm{sub}_i\, s_i &= \mathrm{sub}_{i-1}\,[x/t]\, s_i \\
&= \mathrm{sub}_{i-1}\, s_i\,[x/t s_i] && \text{(denn } x \notin \mathrm{Def}(s_i) \text{ und } x \text{ kommt in keinem der} \\
& && \text{Terme } y\, s_i \text{ für } y \in \mathrm{Def}(s_i) \text{ vor)} \\
&= \mathrm{sub}_{i-1}\, s_i\,[x/t\, s_{i-1}] && \text{(denn } x \text{ kommt in } t \text{ nicht vor)} \\
&= \mathrm{sub}_{i-1}\, s_i\,[x/x\, s_{i-1}] && \text{(wegen } x\, s_{i-1} = t\, s_{i-1}\text{)} \\
&= \mathrm{sub}_{i-1}\, s_{i-1} && \text{(Def. von } s_i \text{ und } x \text{ kommt in keinem der} \\
& && \text{Terme } y\, s_i \text{ für } y \in \mathrm{Def}(s_i) \text{ vor)} \\
&= \mathrm{sub}' && \text{(Induktionshypothese)}
\end{aligned}
$$

Dies zeigt (1).

Der Fall, dass **L** und $\{y\,\mathrm{sub}' \mid y \in \mathrm{Def}(\mathrm{sub}')\}$ keine gemeinsamen Variablen enthalten, ist damit abgeschlossen.

Beweis der Korrektheit des Unifikationsalgorithmus

Im allgemeinen Fall (sub' ist ein beliebiger Unifikator von **L**) sei X die Menge aller Variablen, die in $\{y\,\mathrm{sub}' \mid y \in \mathrm{Def}(\mathrm{sub}')\}$ vorkommen.

Sei Y eine Menge von Variablen mit $|X| = |Y|$, so dass Y und **L** keine gemeinsamen Variablen enthalten.

Sei $u : X \to Y$ eine beliebige Bijektion zwischen X und Y.
Wir nennen u eine Variablenumbenennung.

Dann ist auch $\mathrm{sub}'u$ ein Unifikator von **L**, so dass **L** und $\{y\,\mathrm{sub}'u \mid y \in \mathrm{Def}(\mathrm{sub}')\}$ keine gemeinsame Variablen enthalten.

Also gibt es für alle $0 \leq i \leq N$ eine Substitution s_i mit $\mathrm{sub}'u = \mathrm{sub}_i\ s_i$.

Also gilt $\mathrm{sub}' = \mathrm{sub}_i(s_i u^{-1})$.

Beweis der Korrektheit des Unifikationsalgorithmus

Aus (1) und (2) folgt nun:

Der Algorithmus terminiert nach N Durchläufen durch die **while**-Schleife mit einem Unifikator $\mathrm{sub} = \mathrm{sub}_N$.

Ist sub' ein beliebiger Unifikator von **L**, so existiert wegen (1) eine Substitution s mit $\mathrm{sub}' = \mathrm{sub}\, s$.

Also ist sub ein allgemeinster Unifikator von **L**. □

Beispiel zum Unifikationsalgorithmus

Betrachte $\mathbf{L} = \{P(f(z, g(a, y)), h(z)),\ P(f(f(u, v), w), h(f(a, b)))\}$

Beispiel zum Unifikationsalgorithmus

Betrachte $\mathbf{L} = \{P(f(z, g(a, y)), h(z)),\ P(f(f(u, v), w), h(f(a, b)))\}$

$P(f(z, g(a, y)), h(z))$
$P(f(f(u, v), w), h(f(a, b)))$ $\quad\quad \text{sub} = []$

$P(f(f(u, v), g(a, y)), h(f(u, v)))$
$P(f(f(u, v), w), h(f(a, b)))$ $\quad\quad \text{sub} = [z/f(u, v)]$

$P(f(f(u, v), g(a, y)), h(f(u, v)))$
$P(f(f(u, v), w), h(f(a, b)))$ $\quad\quad \text{sub} = [z/f(u, v)]$

$P(f(f(u, v), g(a, y)), h(f(u, v)))$
$P(f(f(u, v), g(a, y)), h(f(a, b)))$ $\quad\quad \text{sub} = [z/f(u, v)][w/g(a, y)]$

Beispiel zum Unifikationsalgorithmus

$P(f(f(u,v),g(a,y)),h(f(u,v)))$

$P(f(f(u,v),g(a,y)),h(f(a,b)))$ $\quad \text{sub} = [z/f(u,v)][w/g(a,y)]$

$P(f(f(a,v),g(a,y)),h(f(a,v)))$

$P(f(f(a,v),g(a,y)),h(f(a,b)))$ $\quad \text{sub} = [z/f(u,v)][w/g(a,y)][u/a]$

$P(f(f(a,v),g(a,y)),h(f(a,v)))$

$P(f(f(a,v),g(a,y)),h(f(a,b)))$ $\quad \text{sub} = [z/f(u,v)][w/g(a,y)][u/a]$

$P(f(f(a,b),g(a,y)),h(f(a,b)))$

$P(f(f(a,b),g(a,y)),h(f(a,b)))$ $\quad \text{sub} = [z/f(u,v)][w/g(a,y)][u/a][v/b]$

Komplexität des Unifikationsalgorithmus

Zwar ist die Anzahl der Durchläufe durch die while-Schleife in dem Unifikationsalgorithmus durch die Eingabelänge beschränkt, aber:

Durch das wiederholte Einsetzen von Termen können sehr große Terme entstehen.

In der Tat ist die Laufzeit unseres Unifikationsalgorithmus i.A. exponentiell in der Eingabelänge.

Andererseits gilt folgendes Resultat:

Paterson, Wegman 1976

Es gibt einen Unifikationsalgorithmus, dessen Laufzeit linear in der Eingabelänge beschränkt ist.

Prädikatenlogische Resolution

Eine Klausel R heißt prädikatenlogischer Resolvent zweier Klauseln K_1 und K_2, wenn folgendes gilt:

- Es gibt Variablenumbenennungen s_1 und s_2, so dass $K_1 s_1$ und $K_2 s_2$ keine gemeinsamen Variablen enthalten.
- Es gibt $m, n \geq 1$ und Literale $L_1, \ldots, L_m$ aus $K_1 s_1$ und Literale $L'_1, \ldots, L'_n$ aus $K_2 s_2$, so dass

$$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

  unifizierbar ist. Sei sub ein allgemeinster Unifikator von $\mathbf{L}$.
  ($\overline{L}$ bezeichnet das zu L negierte Literal)
- Es gilt

$$R = ((K_1 s_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 s_2 \setminus \{L'_1, \ldots, L'_n\}))\mathrm{sub}.$$

Beispiel für prädikatenlogischen Resolventen

Sei

$$\begin{aligned} K_1 &= \{P(f(x)), \neg Q(z), P(z)\} \\ K_2 &= \{\neg P(x), R(g(x), a)\} \end{aligned}$$

Für die Variablenumbenennungen $s_1 = []$ und $s_2 = [x/u]$ gilt:

$$\begin{aligned} K_1 s_1 &= \{P(f(x)), \neg Q(z), P(z)\} \\ K_2 s_2 &= \{\neg P(u), R(g(u), a)\} \end{aligned}$$

Diese Klauseln haben keine gemeinsamen Variablen.

Sei $L_1 = P(f(x)) \in K_1 s_1$, $L_2 = P(z) \in K_1 s_1$ und $L_1' = \neg P(u) \in K_2 s_2$.

Die Menge $\mathbf{L} = \{\overline{L_1}, \overline{L_2}, L_1'\} = \{\neg P(f(x)), \neg P(z), \neg P(u)\}$ ist unifizierbar.

Ein allgemeinster Unifikator ist $\mathrm{sub} = [z/f(x)][u/f(x)]$.

Somit ist

$$((K_1 s_1 \setminus \{L_1, L_2\}) \cup (K_2 s_2 \setminus \{L_1'\}))\mathrm{sub} = \{\neg Q(f(x)), R(g(f(x)), a)\}$$

Resolvent von K_1 und K_2.

Korrektheit und Vollständigkeit

Zwei Fragen:

- Wenn man mit prädikatenlogischer Resolution aus einer Formel F die leere Klausel $\square$ ableiten kann, ist F dann unerfüllbar? (Korrektheit)
- Kann man für eine unerfüllbare Formel F immer durch prädikatenlogische Resolution die leere Klausel herleiten? (Vollständigkeit)

Aufgabe

Sind diese Klauseln resolvierbar?
Wieviele mögliche Resolventen gibt es?

K_1	K_2	Möglichkeiten
$\{P(x), Q(x,y)\}$	$\{\neg P(f(x))\}$	
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y,z)\}$	

Aufgabe

Sind diese Klauseln resolvierbar?
Wieviele mögliche Resolventen gibt es?

K_1	K_2	Möglichkeiten
$\{P(x), Q(x,y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y,z)\}$	

Aufgabe

Sind diese Klauseln resolvierbar?
Wieviele mögliche Resolventen gibt es?

K_1	K_2	Möglichkeiten
$\{P(x), Q(x,y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	0
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y,z)\}$	

Aufgabe

Sind diese Klauseln resolvierbar?
Wieviele mögliche Resolventen gibt es?

K_1	K_2	Möglichkeiten
$\{P(x), Q(x,y)\}$	$\{\neg P(f(x))\}$	1
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	0
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y,z)\}$	2

Lifting-Lemma

Eine Grundinstanz eines Literals L ist ein Literal $L\mathrm{sub}$, welches keine Variablen enthält.

Eine Grundinstanz einer Klausel $K = \{L_1, \ldots, L_n\}$ ist ein Klausel $K\mathrm{sub} = \{L_1\mathrm{sub}, \ldots, L_n\mathrm{sub}\}$, welche keine Variablen enthält.

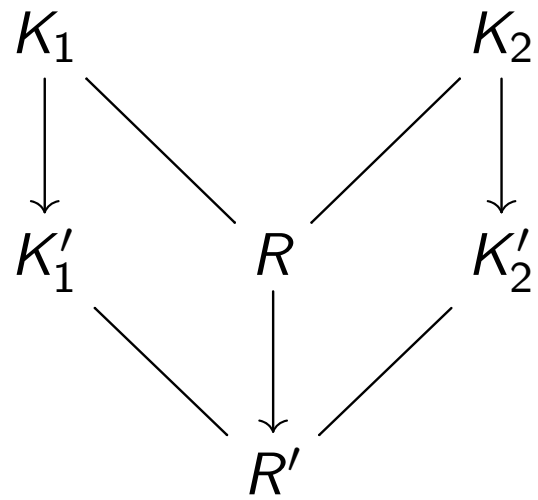
Beispiel: $P(f(a), f(f(a)), g(a, b))$ ist eine Grundinstanz des Literals $P(x, f(x), g(a, y))$.

Lifting-Lemma

Seien K_1, K_2 zwei prädikatenlogische Klauseln und seien K_1', K_2' zwei Grundinstanzen hiervon, die aussagenlogisch resolvierbar sind und den Resolventen R' ergeben.
Dann gibt es einen prädikatenlogischen Resolventen R von K_1, K_2, so dass R' eine Grundinstanz von R ist.

Lifting-Lemma

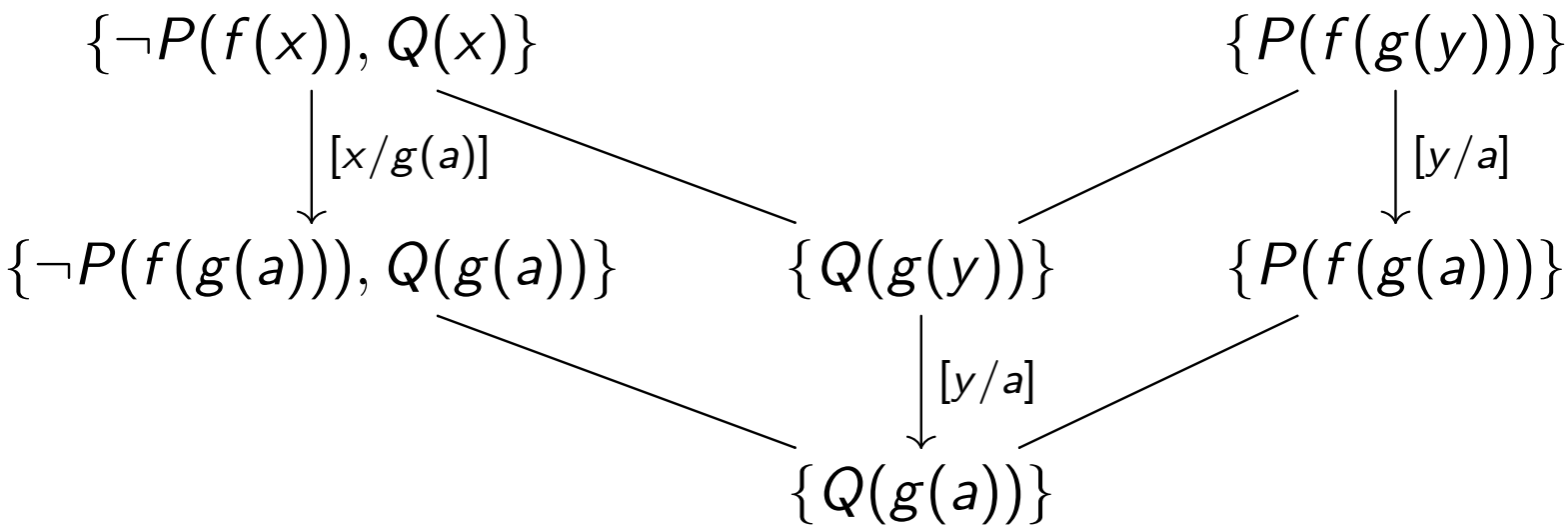
Veranschaulichung des Liftig-Lemma:



—: Resolution

→: Substitution

Beispiel:



Beweis des Lifting-Lemmas

Beweis:

Seien s_1 und s_2 Variablenumbennungen, so dass $K_1 s_1$ und $K_2 s_2$ keine gemeinsamen Variablen haben.

K_i' Grundinstanz von K_i. $\Longrightarrow$ K_i' Grundinstanz von $K_i s_i$.

Sei sub_i eine Substitution mit $K_i' = K_i s_i \, \mathrm{sub}_i$.

Dabei gilt o.B.d.A. für $i \in \{1, 2\}$:

1. $\mathrm{Def}(\mathrm{sub}_i)$ ist die Menge der in $K_i s_i$ vorkommenden Variablen.
2. Für alle $x \in \mathrm{Def}(\mathrm{sub}_i)$ enthält der Term $\mathrm{sub}_i(x)$ keine Variablen (sub_i ist eine Grundsubstitution).

Aus (1) folgt insbesondere $\mathrm{Def}(\mathrm{sub}_1) \cap \mathrm{Def}(\mathrm{sub}_2) = \emptyset$.

Sei $\mathrm{sub} = \mathrm{sub}_1 \mathrm{sub}_2 = \mathrm{sub}_2 \mathrm{sub}_1$.

Es gilt $K_i' = K_i s_i \, \mathrm{sub}_i = K_i s_i \, \mathrm{sub}$.

Nach Voraussetzung ist R' aussagenlogischer Resolvent von K_1' und K_2'.

Beweis des Lifting-Lemmas

Also gibt es $L \in K_1' = K_1 s_1 \,\mathrm{sub}$ und $\overline{L} \in K_2' = K_2 s_2 \,\mathrm{sub}$ mit

$$R' = (K_1' \setminus \{L\}) \cup (K_2' \setminus \{\overline{L}\}).$$

Seien $L_1, \ldots, L_m \in K_1 s_1$ $(m \geq 1)$ alle Literale aus $K_1 s_1$ mit

$$L = L_1 \mathrm{sub} = \cdots = L_m \mathrm{sub}.$$

Seien $L_1', \ldots, L_n' \in K_2 s_2$ $(n \geq 1)$ alle Literale aus $K_2 s_2$ mit

$$\overline{L} = L_1' \mathrm{sub} = \cdots = L_n' \mathrm{sub}.$$

Somit ist sub ein Unifikator der Literalmenge

$$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L_1', \ldots, L_n'\}$$

und die Klauseln K_1 und K_2 sind prädikatenlogisch resolvierbar.

Sei sub_0 ein allgemeinster Unifikator von $\mathbf{L}$.

Beweis des Lifting-Lemmas

Dann ist

$$R = ((K_1 s_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 s_2 \setminus \{L'_1, \ldots, L'_n\}))\mathrm{sub}_0.$$

ein prädikatenlogischer Resolvent von K_1 und K_2.

Da sub_0 allgemeinster Unifikator von $\mathbf{L}$ ist und sub ein Unifikator von $\mathbf{L}$ ist, existiert eine Substitution s mit $\mathrm{sub}_0\, s = \mathrm{sub}$. Es folgt

$$\begin{aligned}
R' &= (K'_1 \setminus \{L\}) \cup (K'_2 \setminus \{\overline{L}\}) \\
&= (K_1 s_1\, \mathrm{sub} \setminus \{L\}) \cup (K_2 s_2\, \mathrm{sub} \setminus \{\overline{L}\}) \\
&= (K_1 s_1\, \mathrm{sub} \setminus \{L_1\mathrm{sub}, \ldots, L_m\mathrm{sub}\}) \cup (K_2 s_2\, \mathrm{sub} \setminus \{L'_1\mathrm{sub}, \ldots, L'_n\mathrm{sub}\}) \\
&= \Big((K_1 s_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 s_2 \setminus \{L'_1, \ldots, L'_n\})\Big)\,\mathrm{sub} \\
&= \Big((K_1 s_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 s_2 \setminus \{L'_1, \ldots, L'_n\})\Big)\,\mathrm{sub}_0 s \\
&= R\, s
\end{aligned}$$

Damit ist gezeigt, dass R' eine Grundinstanz von R ist. □

Resolutionssatz

Resolutionssatz der Prädikatenlogik

Sei F eine Aussage in Skolemform mit einer Matrix F^* in **KNF**. Dann gilt: F ist unerfüllbar genau dann, wenn $\square \in Res^*(F^*)$.

Für den Beweis des Resolutionssatzes benötigen wir folgenden Begriff:

Für eine Formel H mit freien Variablen $x_1, \ldots, x_n$ bezeichnen wir mit

$$\forall H = \forall x_1 \forall x_2 \cdots \forall x_n H$$

ihren Allabschluss.

Lemmata zum Allabschluss

Lemma

Sei F eine Aussage in Skolemform, deren Matrix F^* in **KNF** ist. Dann gilt:

$$F \equiv \forall F^* \equiv \bigwedge_{K \in F^*} \forall K$$

Beweis: Da F eine Aussage ist (also keine freien Variablen hat), gilt $F = \forall F^*$.

Da F^* in **KNF** ist, gilt

$$F^* \equiv \bigwedge_{K \in F^*} K.$$

Das Lemma folgt somit aus der Äquivalenz $\forall y(G \wedge H) \equiv \forall y G \wedge \forall y H$. □

Beispiel:

$$\begin{aligned} F^* &= P(x,y) \wedge \neg Q(y,x) \\ F &\equiv \forall x \forall y (P(x,y) \wedge \neg Q(y,x)) \equiv \forall x \forall y P(x,y) \wedge \forall x \forall y (\neg Q(y,x)) \end{aligned}$$

Lemmata zum Allabschluss

Lemma

Sei R Resolvent zweier Klauseln K_1 und K_2. Dann ist $\forall R$ eine Folgerung von $\forall K_1 \ \wedge \ \forall K_2$.

Beweis:

Sei $\mathcal{A}$ ein Modell von $\forall K_1$ und von $\forall K_2$: $\mathcal{A}(\forall K_1) = \mathcal{A}(\forall K_2) = 1$

Sei

$$R = ((K_1 s_1 \setminus \{L_1, \ldots, L_m\}) \cup (K_2 s_2 \setminus \{L'_1, \ldots, L'_n\}))\mathrm{sub}$$

wobei $L_1, \ldots, L_m \in K_1 s_1$, $L'_1, \ldots, L'_n \in K_2 s_2$ und sub allgemeinster Unifikator von

$$\mathbf{L} = \{\overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n\}$$

ist.

Sei $L = \overline{L_1}\,\mathrm{sub} = \cdots = \overline{L_m}\,\mathrm{sub} = L'_1\,\mathrm{sub} = \cdots = L'_n\,\mathrm{sub}$. Dann gilt

$$(K_1 s_1\,\mathrm{sub} \setminus \{\overline{L}\}) \cup (K_2 s_2\,\mathrm{sub} \setminus \{L\}) \subseteq R.$$

Lemmata zum Allabschluss

Angenommen, es gilt $\mathcal{A}(\forall R) = 0$.

Dann gibt es eine Struktur $\mathcal{A}'$ mit:

- $\mathcal{A}'$ ist identisch zu $\mathcal{A}$ bis auf die Werte $I_{\mathcal{A}'}(x)$ für die in R vorkommenden Variablen x.
- $\mathcal{A}'(R) = 0$.

Also gilt

$$\mathcal{A}'(K_1 s_1 \,\mathrm{sub} \setminus \{\overline{L}\}) = \mathcal{A}'(K_2 s_2 \,\mathrm{sub} \setminus \{L\}) = 0. \tag{17}$$

Aus $\mathcal{A}(\forall K_1) = \mathcal{A}(\forall K_2) = 1$ folgt

$$\mathcal{A}'(K_1 s_1 \,\mathrm{sub}) = \mathcal{A}'(K_2 s_2 \,\mathrm{sub}) = 1. \tag{18}$$

(17) und (18) ergibt zusammen: $\mathcal{A}'(L) = \mathcal{A}'(\overline{L}) = 1$. Widerspruch! □

Beweis des Resolutionssatzes

Beweis des Resolutionssatzes:

(A) Korrektheit: Wenn $\square \in Res^*(F^*)$, dann ist F unerfüllbar.

Gelte $\square \in Res^*(F^*)$.

Aus den soeben bewiesenen Lemmata folgt: $\square = \forall\square$ ist eine Folgerung von $\bigwedge_{K \in F^*} \forall K \equiv F$.

Da $\square$ kein Modell hat, kann auch F kein Modell haben.

(B) Vollständigkeit: Wenn F unerfüllbar ist, dann gilt $\square \in Res^*(F^*)$.

Sei F unerfüllbar.

Beweis des Resolutionssatzes

Aus dem Grundresolutionssatz folgt, dass es eine Folge von Klauseln $K'_1, \ldots, K'_n$ mit folgender Eigenschaft gibt:

- K'_n ist die leere Klausel
- Für $i = 1, \ldots, n$ gilt:
  - K'_i ist eine Grundinstanz einer Klausel $K \in F^*$,d.h. $K'_i = K[y_1/t_1] \ldots [y_k/t_k]$ mit $t_i \in D(F)$
  - **oder** K'_i ist (aussagenlogischer) Resolvent zweier Klauseln K'_a, K'_b mit $a < i$ und $b < i$

Für alle $i \in \{1, \ldots, n\}$ geben wir eine Klausel K_i an, so dass K'_i eine Grundinstanz von K_i ist und $(K_1, \ldots, K_n)$ eine prädikatenlogische Resolutionsherleitung der leeren Klausel $K_n = \square$ aus den Klauseln in F^* ist.

Betrachte ein $i \in \{1, \ldots, n\}$ und seien $K_1, \ldots, K_{i-1}$ bereits konstruiert.

Beweis des Resolutionssatzes

1.Fall: K'_i eine Grundinstanz einer Klausel $K \in F^*$.

Definiere $K_i = K$.

2.Fall: K'_i ist aussagenlogischer Resolvent zweier Klauseln K'_a, K'_b mit $a < i$ und $b < i$.

Aus dem Lifting-Lemma ergibt sich ein prädikatenlogischer Resolvent R von K_a und K_b, so dass K'_i eine Grundinstanz von R ist.

Definiere $K_i = R$. □

Beispiel I

Ist die Klauselmenge

$$\{\{P(f(x))\},\ \{\neg P(x), Q(x, f(x))\},\ \{\neg Q(f(a), f(f(a)))\}\}$$

d.h. die Aussage

$$\forall x\big(P(f(x)) \wedge (\neg P(x) \vee Q(x, f(x))) \wedge \neg Q(f(a), f(f(a)))\big)$$

unerfüllbar?

Beispiel I

Ist die Klauselmenge

$$\{\{P(f(x))\},\ \{\neg P(x), Q(x, f(x))\},\ \{\neg Q(f(a), f(f(a)))\}\}$$

d.h. die Aussage

$$\forall x\big(P(f(x)) \wedge (\neg P(x) \vee Q(x, f(x))) \wedge \neg Q(f(a), f(f(a)))\big)$$

unerfüllbar?

Ja, hier ist eine Resolutionsableitung der leeren Klausel:

$\{P(f(x))\}$ und $\{\neg P(x), Q(x, f(x))\}$ ergeben den Resolventen $\{Q(f(x), f(f(x)))\}$

$\{Q(f(x), f(f(x)))\}$ und $\{\neg Q(f(a), f(f(a)))\}$ ergeben den Resolventen $\{\} = \square$.

Beispiel II

Wir betrachten folgende Klauselmenge
(Beispiel aus dem Buch von Schöning):

$$\begin{aligned} F \quad = \quad & \{\{\neg P(x), Q(x), R(x, f(x))\}, \{\neg P(x), Q(x), S(f(x))\}, \{T(a)\}, \\ & \{P(a)\}, \{\neg R(a, x), T(x)\}, \{\neg T(x), \neg Q(x)\}, \{\neg T(x), \neg S(x)\}\} \end{aligned}$$

Verfeinerung der Resolution (Ausblick)

Probleme bei der prädikatenlogischen Resolution:

- Zu viele Wahlmöglichkeiten
- Immer noch zu viele Sackgassen
- Kombinatorische Explosion des Suchraums

Lösungsansätze:

Strategien und Heuristiken: Verbieten bestimmter Resolutionsschritte, Suchraum wird dadurch eingeschränkt

Vorsicht: Die Vollständigkeit darf dadurch nicht verloren gehen!