# Zusammenfassung Theoretische Informatik und Logik

Henrik Tscherny

12. August 2021

## Inhaltsverzeichnis

# 1 Grundlegende Definitionen

## 1.1 DTM

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

- $Q$: endliche Zustandsmenge
- $\Sigma$: Eingabealphabet
- $\Gamma$: Arbeitsalphabet mit $\Sigma \cup \{\_\}$
- $\delta$: Übergangsfunktion (partiell) mit $Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$
- $q_0$: Startzustand mit $q_0 \in Q$
- $F$: Menge akzeptierender Endzustände mit $F \subseteq Q$

Der Ausdruck $\delta(q, a) = (p, b, D)$ bedeutet:

- Ist die TM im Zustand q
- Ließt die TM das Zeichen a
  - wechsele in den Zustand p
  - überschreibe a mit b
  - verschiebe Head nach $D \in \{L, R, N\}$

## 1.2 NTM

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

- $Q$: endliche Zustandsmenge
- $\Sigma$: Eingabealphabet
- $\Gamma$: Arbeitsalphabet mit $\Sigma \cup \{\_\}$
- $\delta$: Übergangsfunktion (total) mit $Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R,N\}}$
- $q_0$: Startzustand mit $q_0 \in Q$
- $F$: Menge akzeptierender Endzustände mit $F \subseteq Q$

Anders als bei einer DTM ist die Übergangsfunktion einer NTM mehrdeutig.
Eine NTM akzeptiert sobald mindestens ein Endzustand erreicht werden kann.
Dabei werden stets alle möglichen Entscheidungen in $\delta$ betrachtet.
Eine NTM ist genauso berechnungsstark wie eine DTM

## 1.3 Church-Turing-These

- Jede Funktion die intuitiv berechenbar ist, kann auch von einer TM berechnet werden
- intuitiv berechenbar meint das es auf irgendeine Weise berechnet werden kann und ist nicht Formal definiert
- Alle Computer sind gleich (berechnungsstark)

## 1.4 TM und Funktionen

- eine DTM berechnet eine partielle Funktion der Form $f_M : \Sigma^* \rightarrow \Sigma^*$
- das von der TM berechnete Resultat der Funktion (v) steht nach dem Halten alleinig auf dem Band $w \in \Sigma^* : f_M(w) = v$
- Funktion undefiniert (darum partiell) bei:
  - M hält auf w aber Bandformat ungültig → es steht noch was anderes auf dem Band
  - M hält nicht auf w
- ist $f_M$ total so muss M immer halten

## 1.5 Berechenbarkeit

eine Funktion heißt berechenbar, wenn:

- es gibt eine TM M die f berechnet mit $f = f_M$
- es gibt einen Algorithmus auf einer TM um das Problem zu lösen

Berechenbare totale Funktionen nennt man auch **rekursiv**
Berechenbare partielle Funktionen nennt man auch **partiell rekursiv**

## 1.6 Entscheidbarkeit und Sprachen

Sei L eine Sprache und M eine TM
Sei $L(M)$ die Menge aller Wörter die von einer TM akzeptiert werden d.h. die TM hält in einem Endzustand
Sei $w \in L(M)$ das Wortproblem, d.h. ob das Wort w in der Sprache L liegt

- **Entscheidbar**: es gibt eine TM die entscheidet ob $w \in L(M)$, M heißt in diesem Fall auch Entscheider
- **Semi-Entscheidbar**: die TM verhält sich wie folgt

$$\begin{cases} w \in L(M) \rightarrow \text{TM hält,} \\ w \notin L(M) \rightarrow \text{TM kann, muss aber nicht halten} \end{cases}$$

- **Co-Semi-Entscheidbar**: Die TM verhält sich umgekehrt zur Semi-Entscheidbarkeit
- **Unentscheidbar**: Das Verhalten der TM sagt nichts über das Wortproblem aus

Note: andere Objekte können stets als Wörter einer Sprache kodiert werden, daher sollte man die Entscheidbarkeit intuitiv nicht zu eng mit Sprachen in Verbindung bringen

### 1.6.1 Sätze

- ist L entscheidbar so ist es auch das Komplement von L
- L ist entscheidbar gdw. L und nicht L ist semi-entscheidbar
- ist L unentscheidbar und semi-entscheidbar so ist nicht L unentscheidbar und nicht semi-entscheidbar

# 2 unentscheidbare Probleme

## 2.1 Existenzbeweis

Folgende Dinge müssen gezeigt werden um die Existenz von unentscheidbaren Problemen zu Beweisen:

- **1.** Die Menge der TM's ist abzählbar
- **2.** die Menge der Sprachen über jedem Alphabet ist überabzählbar

**1.**

- Jede TM lässt sich durch ein endliches Wort kodieren, denn jedes Programm besteht aus endlich viel Code ($len(pgrm) \in \mathbb{N}$)
- alle Wörter sind abzählbar:
  fange mit $\epsilon$ an, dann alle Wörter mit $len(w) = 1$ dann $len(w) = 2$ usw.

**2.**

- Nehme an die Menge aller Sprachen wäre abzählbar
- Erstelle eine große Tabelle welche alle Wörter auf einer Achse aufzählt **und alle Sprachen auf der anderen***
- Markiere in der Tabelle immer ob $w \in L$
- Konstruiere nun eine neue Sprache $L_d \div$ mit $w_i \in L_d \Leftrightarrow w_i \notin L_i$, d.h. die Sprache $L_d$ entspricht eine Sprache welche sich durch genau einen Eintrag von **allen** Sprachen unterscheidet
- es kann sich aber nicht von allen Sprachen unterscheiden, da wir ja davor alle in der Tabelle aufgeschrieben haben **siehe ***, d.h. **Widerspruch!**
- Die Menge aller Sprachen muss überabzählbar sein

## 2.2 Beispiel $L_{\pi 7}$

Sei folgendes Problem gegeben

- Sei $L_{\pi 7}$ die Menge mit Wörtern der Form $7^n$ (Zahl 7, n-mal hintereinander), die in $\pi$ vorkommen
- $L_{\pi 7}$ ist entscheidbar, denn:

  - Enthält $\pi$ beliebig lange Siebenerketten, so wird die Sprache Entschieden
  - Enthält $\pi$ nur Siebenerketten bestimmter Länge, so akzeptiere nur für diese Längen

- Für jeden Fall ist ein Algorithmus denkbar
- aber man weiß nicht welchen
- das Problem ist entscheidbar aber trotzdem nicht berechenbar

## 2.3 Beispiel Bussy-Beaver

Sei eine TM mit leerem Band, n Zuständen und Arbeitsalphabet dem $\Gamma = \{x, \_\}$
Note: es gibt dabei $(2 \cdot 2 \cdot (n+1))^{2n}$ verschiedene TM's die diese Bedingung erfüllen **Frage**:

- Wie viele Zeichen 'x' kann die TM auf das Band schreiben bevor sie hält
- die TM muss halten
- TM's die nicht halten werden ignoriert
- Sei $BB(n)$ die Anzahl maximal geschriebener Zeichen

Note: Alternativ kann man das Problem auch so formulieren: Wie viele Zeichen kann ein beliebiges Python Script ausgeben bevor es sich beendet, wenn das Script gültig ist und das Programm irgendwann terminiert
**Beweis der nicht Berechenbarkeit** Nehme an BB wäre berechenbar, dann:

- alle Berechnungen sind dabei unär kodiert, d.h. 4 = 'xxxx' usw.
- gibt es eine TM $M_{BB}$ die die Funktion $f(n) = BB(n)$ berechnet
- Sei zudem $M_{+1}$ eine TM mit $f(n) = n + 1$ (Inkrementierer)
- Sei zudem $M_{x2}$ eine TM mit $f(n) = 2n$ (Verdoppler)
- Sei k die Summe der Anzahl der Zustände in $M_{BB}$, $M_{+1}$ und $M_{x2}$
- Sei $I_k$ eine TM mit $k + 1$ Zuständen welche das Wort $x^k$ auf ein leeres Band schreibt
- Führe $I_k, M_{x2}, M_{BB}, M_{+1}$ hintereinander aus

- Wir haben somit eine TM mit $\leq 2k$ Zuständen die $BB(2k) + 1$ mal 'x' schreibt
- **Widerspruch!**: $BB(n)$ sollte ja das Maximum sein wie kann es also noch ein 'x' mehr sein
- die BB-Funktion ist nicht berechenbar

## 2.4 Schwierigkeit von unentscheidbaren Problemen

**Existenzbeweis verschieden schwerer unentscheidbarer Probleme**
oder: Sind alle unentscheidbaren Probleme auf das Halteproblem Turing-reduzierbar (ex. v. schwereren Prob.)
oder: Ist das Halteproblem auf alle unentscheidbaren Probleme Turing-reduzierbar (ex. v. leichteren Prob.)

- TM's sind endlich beschreibbar
- Daher gibt es auch nur abzählbar viele TM's
- Die Menge der Probleme ist aber überabzählbar (siehe Diagonalisierung von Cantor)
- Eine TM könnte Subroutinen benutzen welche nicht berechenbar sind (z.B. das Halteproblem lösen)
- Auch solche TM's sind jedoch endlich beschreibbar
- $\rightarrow$ **es existieren schwerere Probleme als das Halteproblem**

### 2.4.1 Beispiele für ein schwereres Problem

Gegeben sei das Halteproblem höher Ordnung ($P_{HALT}$)

- eine DTM M welche $P_{HALT}$ als Funktion benutzen darf
- Frag: Hält M auf w ?

Note: Auf diese weise können unendliche viele noch schwerere Probleme konstruiert werden
Ein weiteres Problem ist das Universalitätsproblem:

- Sei eine TM M über $\Sigma$ gegeben
- akzeptiert die TM alle Eingaben für $w \in \Sigma^*$
- Es stellt sich heraus, dass dieses Problem auf $P^2_{HALT}$ Turing-reduzierbar ist

# 3 Loop

## 3.1 Regeln

Sei P ein Loop-Programm:

- alle Variablen sind vom Typ int
- Wertzuweisung der Form $x := y + n$ für $n \in \mathbb{Z}$
- For-Schleifen **LOOP** x **DO** ... **END**
- Programmverkettung: $P_1; P_2$
- Loop in Loop: **LOOP** x **DO** P **END**

## 3.2 Eigenschaften

- ein Loop-Programm terminiert immer
- jede Loop-berechenbare Funktion ist total
- es gibt berechenbare Funktionen die nicht Loop-berechenbar sind (z.B.) Ackermann-Funktion
- alle kleiner als $O(n^{n^{n^{\cdots^x}}})$ laufenden Funktionen sind Loop berechenbar

### 3.2.1 BB-Funktion für Loop

Sei $BB_{Loop}$ die eine Funktion die für ein Loop Programm der Länge l, die größte Zahl die ein Loop Programm dieser Länge erzeugen kann, bei leerer Startkonfiguration
Wir zeigen 2 Dinge:

- $BB_{Loop}$ ist berechenbar
- $BB_{Loop}$ ist nicht Loop-berechenbar

**1.**

- Es gibt endlich viele Loop Programme der Länge $\leq l$
- Durch diese kann man in endlicher Zeit durch iterieren
- gebe am Ende das Maximum zurück

- ein Loop Programm muss vorher wissen wie viele Schleifendurchläufe benötigt werden, da es dies selbst nicht berechnen kann (diese Aussage könnte jedoch eine TM für das Loop Programm treffen)
- Loop ist berechnungsschwächer als eine TM

**2.**

- Annahme, es $BB_{Loop}$ ist Loop-berechenbar
- Sei k die Länge eines Loop Programms $P_{BB}$ welches $BB_{Loop}$ berechnet
- Wähle $m \geq k + 17 + log_{10}m$
- Sei $P_m$ ein Programm welches m zu einer Zahl x addiert (Länge 7)
- Sei $P_{++}$ ein Programm welches x inkrementiert (Länge 8)
- Sei $P = P_m; P_{BB}; P_{++}$ mit Länge $k + 17 + log_{10}m$
- P gibt somit $BB_{Loop}(m) + 1$ aus, jedoch ist die Länge von P kürzer als m
- Widerspruch

# 4 While

## 4.1 Regeln

- jedes Loop Programm ist ein While Programm
- **WHILE** condition **DO** P **END**

## 4.2 Eigenschaften

- While Programme sind genauso berechnungsstark wie TM's
- → DTM's können White Programme simulieren
- → While Programme können TM's simulieren
- While Programme terminieren nicht immer
- können alle berechenbaren totalen und partiellen Funktionen berechnen

## 4.3 Universalmachine

- TM's sind stark genug andere TM's zu simulieren
- Die simulierende TM kodiert dabei die andere(n) TM'(s) in ihrem Alphabet (Emulator)
- Die Universalmachine U simuliert für eingaben der Form $enc(M)\#\#enc(w)$ w auf M

$$\begin{cases} \text{M hält auf w} \rightarrow \text{U hält auf enc(w)}, \\ \text{M hält nicht auf w} \rightarrow \text{U hält nicht auf enc(w)} \end{cases}$$

- $\rightarrow$ Universalrechner sind möglich
- $\rightarrow$ es gibt Software

# 5 Halteproblem

## 5.1 Definition

Das Halteproblem ist die Frage:
Wird eine TM M auf einem Wort w jemals anhalten ?
Formal:
$P_{HALT} = \{end(M)\#\#enc(w)|\text{M hält auf w}\}$

## 5.2 Beweis der Unentscheidbarkeit

$P_{HALT}$ ist entscheidbar gdw. es und sein Komplement semi-entscheidbar sind

- Nehme an das Komplement sei semi-entscheidbar
- Konstruiere eine TM F wie folgt

$$\begin{cases} \text{hält} \rightarrow \text{TM hält auf w nicht}, \\ \text{hält-nicht} \rightarrow \text{sonst} \end{cases}$$

  , diese TM zeigt also immer das genau entgegengesetzte Halteverhalten
- gebe F in sich selbst ein (F ist Programm und Maschine zugleich)
- 1. Möglichkeit: F hält auf F nicht, darum müsste F halten, Widerspruch da F halten muss wenn F nicht hält laut Definition

- 2. Möglichkeit: F hält auf F, darum müsste F nicht halten nach Definition von F, Widerspruch
- → das Halteproblem ist semi-entscheidbar

# 6 Reduktionen

Mittels Reduktion können Aussagen bzgl. der relativen Schwierigkeit von Problemen getroffen werden, z.B. A ist mindestens genauso schwierig wie B oder B ist nicht leichter als A

- Many-One Reduktion kann als Turing-Reduktion ausgedrückt werden
- es gibt Probleme die Turing aber nicht Many-One reduzierbar sind
- Bsp. $P_{HALT} \leq_T \bar{P_{HALT}}$ aber nicht $P_{HALT} \leq_m \bar{P_{HALT}}$
- Turing-Reduktion kann nur Entscheidbarkeit oder Unentscheidbarkeit, nicht aber Semi-Entscheidbarkeit zeigen

## 6.1 Turing Reduktion

- Bei der Turing Reduktion benutzt man ein zwei Programme P und Q
- P sei auf ein Problem Q Turing reduzierbar ($P \leq_T Q$) wenn man P mittel Q lösen kann
- sprich, P ist nicht schwerer zu lösen aus Q (kann aber leichter sein)
- Q ist mindestens so schwer wie P (oder schwerer)
- lässt sich beispielsweise ein Problem auf das Halteproblem reduzieren, so ist das Problem nicht berechenbar
- sprich, ist $P \leq_T Q$ und $P = P_{HALT}$ dann 'ist das Halteproblem höchstens so schwierig wie Q' oder 'Q ist mindestens so leicht wie das Halteproblem'

## 6.2 Many-One Reduktion

- Es sind zwei Probleme P und Q gegeben
- Findet man eine Funktion f mit $w \in P \Leftrightarrow f(w) \in Q$ so ist P auf Q Many-One reduzierbar $P \leq_m Q$

Beispiel: Reduktion des Halteproblems auf das $\epsilon$-Halteproblem (Wird M auf $\epsilon$ halten?)

- Sei M eine TM und eine andere TM $M_w$ wie folgt konstruiert:
- Lösche Eingabeband und fülle es mit dem Wort w
- Verarbeite die Eingabe wie M
- $$f(v) = \begin{cases} enc(M_w) \rightarrow v = enc(M)\#\#enc(w) \text{für eine TM M,} \\ \# \rightarrow \text{ungültige Eingabe} \end{cases}$$
- Ist $P \leq_m Q$ und Q ist semi-entscheidbar, dann ist auch P semi-entscheidbar

Note: ist f eine in polynomiell Zeit berechnenbare Funktion so ist es eine **polynomielle Many-One Reduktion** und man schreibt $P \leq_p Q$

# 7 Satz von Rice

## 7.1 Definition

- Jede nicht-triviale Frage über von einer Turingmaschine ausgeführten Berechnung ist unentscheidbar
- Sei E einer Eigenschaft von Sprachen, welche für manche (semi)-entscheidbare Sprachen gilt und für manche nicht

## 7.2 Beweis

- Konstruiere eine Many-One Reduktion vom $\epsilon$-Halteproblem auf die 'E-Haltigkeit'
- Sei $\emptyset \notin E$
- Sei $M_L$ eine TM, die eine Sprache $L \in E$ akzeptiert
- Sei $M^*$ wie folgt: Simuliere das $\epsilon$-Halteproblem und simuliere $M_L$ auf w sofern M hält
- somit, hält M auf so ist die von $M^*$ erkannte Sprach eine Sprache mit der Eigenschaft E
- wenn nicht, dann ist die $L(M^*)$ leer und hat nicht die Eigenschaft E

- $$\begin{cases} enc(M^*) \rightarrow v = enc(M) \text{für eine TM M,} \\ \# \rightarrow \text{ungültige Eingabe} \end{cases}$$

# 8 Postsches-Korrespondenz-Problem (PCP)

## 8.1 Definition und Eigenschaften

Gegeben ist folgender Sachverhalt:

- Dominosteine mit Wortpaaren über $\Sigma^*$
- die Länge der Wörter auf einem Dominostein kann sich unterscheiden
- Frage: kann man eine Kombination aus gegeben Dominosteinen finden, sodass die beiden Gesamtwörter (also oben und unten) gleich sind?
- modifizieren des PCP, sodass ein Startsein gegeben ist mit dem man anfangen muss (MPCP)
- das PCP ist unentscheidbar

## 8.2 Beweis der Unentscheidbarkeit

Ziel: Reduzieren des MPCP auf das Halteproblem, und des MPCP auf das PCP
**Konstruktion**

- Nimm eine Instanz des MPCP, d.h. das PCP mit einem gegebenen Startstein
- Finde dazu eine Many-One Reduktion die das MPCP auf das Halteproblem reduziert (MPCP ist mindestens so schwer wie das Halteproblem)
- Kodiere das entstehende Lösungswort des MPCP als Abfolge von Konfigurationen (Zustand, Bandinhalt, HEAD-Position) wie folgt:
  - $(vqw)$ sei eine Konfiguration (v = Bandinhalt links, q = HEAD, w = Bandinhalt rechts)
  - Folge von Konfigurationen (Lauf) getrennt mit Trennzeichen (#) $\rightarrow$ $(c_1\#c_2\#...)$
- Lösung ist dann wie folgt:
  - Hält die TM, d.h. ist die Abfolge endlich: es gibt eine Lösung

  - Hält die TM nicht: es gibt keine Lösung

- nehme als Startstein einen welcher oben leer ist und unten bereits eine Konfiguration steht
  $\begin{bmatrix} \# \\ \#c_0\# \end{bmatrix}$
- finde nun Steine die oben das haben was unten gefordert ist
- dazu benutze die folgenden Regeln und das untere Wort nach oben zu kopieren

$$\begin{cases} \begin{bmatrix} qa \\ bp \end{bmatrix} \rightarrow \delta(q,a) = \langle p, b, R \rangle, \\ \begin{bmatrix} cqa \\ pcb \end{bmatrix} \rightarrow \delta(q,a) = \langle p, b, L \rangle \text{und } c \in \Gamma \text{ beliebig}, \\ \begin{bmatrix} qa \\ pb \end{bmatrix} \rightarrow \delta(q,a) = \langle q, b, N \rangle \end{cases}$$

- zusätzliche Randregeln:

$$\begin{cases} \begin{bmatrix} \#qa \\ \#bp \end{bmatrix} \rightarrow \delta(q,a) = \langle p, b, L \rangle \text{anstoßen am linken Rand}, \\ \begin{bmatrix} q\# \\ q\_\# \end{bmatrix} \rightarrow \forall q \in Q \text{unendlich nach rechts} \end{cases}$$

- zusätzlich nehmen wir eine Kopierregel welche es erlaubt von unten nach oben zu kopieren:
  $\begin{bmatrix} x \\ x \end{bmatrix} \rightarrow \forall x \in \Gamma \cup \{\#\}$
- Startregel:
  $\begin{bmatrix} \# \\ \#q_0 w\# \end{bmatrix}$ mit $q_0 = S, w \in \Sigma^*$
- Zudem definieren wir noch ein weites Symbol \$ um den Endstein zu markieren:

$$\begin{cases} \begin{bmatrix} qa \\ \$ \end{bmatrix} \rightarrow \delta(q,a), \text{undefiniert und } a \in \Gamma, \\ \begin{bmatrix} a\$ \\ \$ \end{bmatrix}, \text{und } \begin{bmatrix} \$ \\ a\$ \end{bmatrix}, \rightarrow \forall a \in \Gamma. \\ \begin{bmatrix} \$\#\# \\ \# \end{bmatrix} \rightarrow \text{endgültiger Abschluss} \end{cases}$$

**Beweis**

- man kann nun eine Abfolge von Konfigurationen finden, welche der Konstruktion folgt:

- der Ausgang des Laufs, kann das wie folgt interpretiert werden:
  - hält: MPCP hat Lösung
  - nicht hält: undefiniert
- Um das MPCP auf das PCP zu reduzieren definieren wir noch 3 Dinge:
  - ${}_{\#}w_{\#}$, Wörter mit Trennzeichen vor und hinter jedem Zeichen
  - ${}_{\#}w$, Wörter mit Trennzeichen vor jedem Zeichen
  - $w_{\#}$, Wörter mit Trennzeichen nach jedem Zeichen
- Sei eine Reduktion nun wie folgt:
  - beginne mit einem Domino mit beidseitigem Trennzeichen oben und nur linksseitigen Trennzeichen unten $\begin{bmatrix} {}_{\#}x_{1\#} \\ {}_{\#}y_1 \end{bmatrix}$
  - setze die Kette nun mit Steinen fort, welche oben rechtsseitig und unten linksseitig getrennt sind $\begin{bmatrix} x_{n\#} \\ {}_{\#}y_n \end{bmatrix}$
  - der letzte Stein soll ein spezieller Stein sein $\begin{bmatrix} \$ \\ \#\$ \end{bmatrix}$
- Besitzt nun das MPCP eine Lösung, so hat das entsprechende PCP auch eine Lösung
- jedes Symbol ist dabei von einer # umgeben
- das Wort endet auf $
- Besitzt das PCP eine Lösung, so muss es mit dem ersten Wortpaar beginnen, da nur dies oben und unten eine # besitzt
- lässt man alle # und $ weg, so hat man wieder eine Lösung für das MPCP
- d.h. PCP und MPCP unterscheiden sich nur durch Trennzeichen
- $\Rightarrow$ **PCP und MPCP sind semi-entscheidbar**

### 8.2.1 Beweis Beispiel

- gegebene TM:

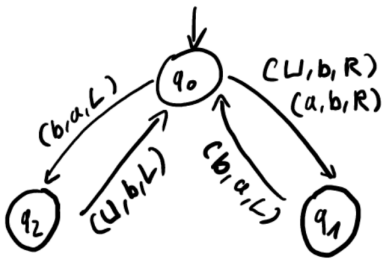



Abbildung 1: gegebene TM

- Wort w: **aba**
- Lässt man die TM nun auf w laufen so erhält man: $\#q_0aba\#bq_1ba\#q_0baa\#q_2\_aaa\#q_0\_baaa\#bq_1baaa$ usw.
- Ziel: das erzeugte Resultat oben und unten auf den Steinen
- dazu nutzte die Regeln von Oben
- $\begin{bmatrix} \# \\ \#q_0aba\# \end{bmatrix} \rightarrow \begin{bmatrix} \# \\ \#q_0aba\# \end{bmatrix} \begin{bmatrix} q_0a \\ bq_1 \end{bmatrix}$
- Regel 1: in $q_0$ lese a $\rightarrow$ gehe in $q_1$ schreibe b
- nutze Regeln bis oben=unten

# 9 Komplexität

## 9.1 asymptotische Laufzeit

Note: alles gesagte ist äquivalent auf Speicherbedarf (benutzte Speicherzellen) übertragbar

**O-Notation**

- sei $f, g : \mathbb{N} \to \mathbb{R}$ und $f \in O(g)$
- nehme eine Zahl $c > 0$ (Konstante) und ein $n_0 \in \mathbb{N}$ (untere Schranke)
- gilt $n > n_0$ und $f(n) \leq c \cdot g(n)$ wächst f nicht wesentlich schneller als g
- d.h. f unterscheidet sich von g nicht mehr als um einen linearen Teil
- Sei: $f : 2x^2$ und $g : x^2 + 10$
  - beide Funktionen steigen etwa quadratisch
  - f unterscheidet sich von g um den Faktor 2x, dies ist aber im Rahmen (Konstante c>0)
  - und $f \geq g$ für ein $n_0$, also der Schnittpunkt der Parabeln
  - die Notation versteckt absolute Anteile (Konstanten) (z.B.) $10n^3 = O(n^3)$
- Trotz kleiner O-Notation kann der tatsächliche Wert trotzdem extrem abweichen da der Grenzwert betrachtet wird
  - $2^n + n^{2000} \in O(2^n)$
  - $2^{1000} \in O(1)$
  - $4^n \in O(2^n)$
  - $x^2 > 2^n$ für $n \in [2, 4]$, (aber $(O(2^n) > O(n^2))$, siehe Abb.2

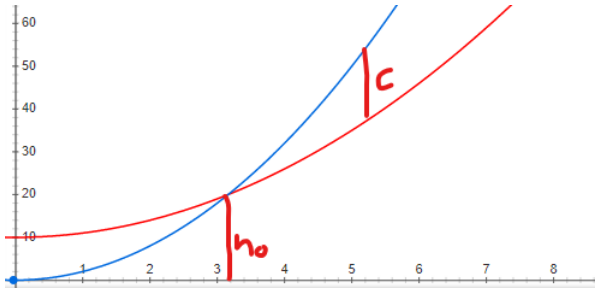



Abbildung 2: Veranschaulichung von f, g, sowie c und $n_0$

**andere Laufzeiten**

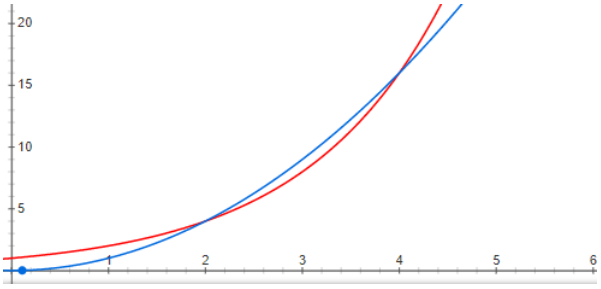



Abbildung 3: Vergleich von $x^2$ und $2^x$

**andere Laufzeiten**

| Notation | $C = \lim_{n\to\infty} \frac{f(n)}{g(n)}$ | Bedeutung |
|---|---|---|
| $f \in O(g)$ | $c < 0$ | $f \leq g$ |
| $f \in \Omega(g)$ | $c > 0$ | $f \geq g$ |
| $f \in \Theta(g)$ | $0 < c < \infty$ | $f = g$ |
| $f \in o(g)$ | $c = 0$ | $f < g$ |
| $f \in \omega(g)$ | $c = \infty$ | $f > g$ |

## 9.2 Linear Speedup Theorem

**Definition**

- Sei M eine multiband-TM mit k Bändern
- M hält bei einer Eingabe der Länge n nach f(n) Schritten
- Es gibt eine k-Band TM' die nach maximal $\frac{f(n)}{c} + n + 2$ Schritten hält für jedes $c > 0, c \in \mathbb{N}$
- (n+2) Kodierungsoverhead → n ... einmal über den Speicher iterieren, 2 ... Feststellen des Endes
- sprich, Jede Turingmaschine kann um einen linearen Faktor beschleunigt werden
- möglich durch Zusammenfassen von Speicherzellen zu einer, durch das benutzen eines unendlichen (sehr großen) Bandalphabets (Bandkompression)
- man benutzt quasi eine riesige Look-up-table über alles was passieren kann

- Problem: reale Rechner können kein unendliches Arbeitsalphabets benutzen, sondern nur $\{0, 1\}$
- Problem 2: in der Praxis kann man nicht beliebig große Daten in einem Schritt lesen
- Problem 3: Linearer Speedup ist immer noch relativ klein $O(n)$

**Beweis**

- vergrößere das Bandalphabet von M': $\Gamma' = \Sigma \cup \Gamma^{6c}$ (6c-große Tupel)
- möchte man z.B. einen Speedup von 1000-x so müssen 6000 Zeichen an einer Bandposition kodiert werden
- M' kodiert nun 6c Zeichen des Eingabebandes und schreibt ein Zeichen aus $\Gamma^{6c}$ auf Band 2
- nun simuliere M
  - lese das Band von den jetzigen Positionen der k-Bänder und Rechts und Links davon (benötigt 4 Schritte)
  - Speichere das gelesene wird als Zustand gespeichert es gibt somit $|Q \times \{1, ..., k\}^{18ck}|$ Zustände
  - dieser Zustand enthält nun die gelesenen Information vom TM-Head sowie rechts und links davon, aber als ein Zustand
  - Simuliere nun in 2 Schritten die nächsten 6c Schritte von M
- Simulation von 6c M-Schritten in 6 M'-Schritten

## 9.3 Komplexitätsklassen

**Definitionen**
Sei f(n) Eine obere Schranke von Einheiten einer Resource welche benutzt werden können, bei einer Eingabe der Länge n, dann:

- Sei $DTIME(f(n))$ die Klasse aller Sprachen L, welche durch eine $O(f)$ zeitbeschränkte **DTM** entschieden werden können
- Sei $DSPACE(f(n))$ analog zum Speicher
- Sei $NTIMEf(n))$ die Klasse aller Sprachen L, welche durch eine $O(f)$ zeitbeschränkte **NTM** entschieden werden können

- Sei $NSPACE(f(n))$ analog zum Speicher

**Eigenschaften**

- Verschiedene Modelle von DTM's (z.B. Mehrband-TM, WHILE, etc.) resultieren in verschiedenen Klassen
- Der Verzicht auf mehrere Bänder verursacht **maximal quadratische Kosten** (bei NTM's evl. größer)
- Die Art der Kodierung hat in der Regel maximal polynomiellen Einfluss
- Die Implementierungsdetails wirken sich maximal polynomiell oder eher konstant aus (sonst würde jede Programmiersprache eine andere Laufzeitklasse haben)
- Problem: genaue Bestimmung der Komplexität ist sehr schwer
- Lösung: verwenden allgemeiner Sprachklassen

**Wichtige Sprachklassen**

| | |
|---|---|
| $P = PTime = \cup_{d\geq 1} DTime(n^d)$ | det. poly. Zeit |
| $Exp = ExpTime = \cup_{d\geq 1} DTime(2^{n^d})$ | det. exp. Zeit |
| $E = ETime = \cup_{d\geq 1} DTime(2^{dn})$ | det. exp. Zeit m. lin. exp. |
| $L = LogSpace = DSpace(log\ n)$ | det. log. Speicher |
| $PSpace = \cup_{d\geq 1} DSpace(n^d)$ | det. poly. Speicher |
| $NP = NPTime = \cup_{d\geq 1} NTime(n^d)$ | n. det. poly Zeit |
| $NExp = NExpTime \cup_{d\geq 1} NTime(2^{n^d})$ | n. det exp. Zeit |
| $NL = NLogSpace = NSpace(log\ n)$ | n. det. log. Speicher |
| $NPSpace = \cup_{d\geq 1} NSpace(n^d)$ | n. det. poly. Speicher |

Note: der Speicher bezieht sich auf den vom Programm benutzten Speicher, daher kann dieser weniger als die Eingabe betragen
Des Weiteren gilt:

- $L \subseteq NL$, $P \subseteq NP$, $PSpace \subseteq NPSpace$, $Exp \subseteq NExp$
  also $DTM \subseteq NTM$
- $P \subsetneq Exp$
- $L \subsetneq PSpace$
- $L \subseteq P \subseteq PSpace \subseteq Exp$
- $P \subseteq PSpace, NP \subseteq NPSpace$
  also $Zeit \subseteq Speicher$

- $NL \subseteq P, NPSpace \subseteq Exp$
  also $(N)Speicher \subseteq 2^{(D)Zeit}$
- $PSpace = NPSpace$ **Satz von Savitch**
- $L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$
  es ist unbekannt ob $\subseteq oder \subsetneq oder =$ auch für größere Abstände z.B.
  $L \subseteq NP$ ??

Note: Speicher kann man wiederverwenden, Zeit nicht
**Robustheit**

- Es wird mindestens lineare Zeit benötigt um die gesamte Eingabe zu lesen
- Die Anzahl der Bänder einer TM hat nur einen quadratischen Einfluss auf deren Zeitbeschränkung
- Konstante Faktoren haben keinen Einfluss auf die Speicherbeschränkung einer TM (LST für Speicher)
- Die Anzahl der Bänder hat keinen Einfluss auf die Speicherbeschränkung einer TM (man schreibt einfach den Inhalt von k Bändern auf ein Band mit einer 1/k Speicherreduktion)

**Bezug von Raum und Zeit**

- $DTIME(f) \subseteq DSPACE(f)$ In n Zeiteinheiten können nur max n Speicherzellen beschrieben werden
- $NTIME(f) \subseteq NSPACE(f)$
- $DSPACE(f) \subseteq DTIME(2^{O(f)})$ Im Worstcase wird Exponentiell soviel Speicher benötigt um alles im gegebenen Speicher zu erledigen
- $NSPACE(f) \subseteq DTIME(2^{O(f)})$ Kann eine NTM ein Problem mit f(n) Platz lösen, so benötigt eine DTM $2^{O(f)}$ Zeiteinheiten dafür
  → Im Übergangsgraphen einer NTM kann es von einem Knoten mehrere ausgehende Kanten geben, diese können von einer DTM jedoch in exponentieller Zeit durchsucht werden

## 9.4 NP

**Definition**

- Probleme welche in polynomielle Zeit gelöst werden können, hätte man eine NTM die immer 'richtig rät'
- Ein Problem liegt in NP, sofern es einen polynomiellen Verifikator für das Problem gibt
  - für jedes Wort der Sprache ex. ein Zertifikat, welches angibt ob $w \in L$
  - **Achtung!** kein Zertifikat für $w \notin L$
  - die Prüfung des Zertifikats kann in PTime abgeschlossen werden
  - Beispiel: Primfaktorzerlegung: schwer zu finden, aber leicht mit Zertifikat (Menge der Primfaktoren) zu belegen
- NP ist nicht symmetrisch, coNP ist die Menge der Sprachen für welche ein negativ Zertifikat existiert $w \notin L$
  $\rightarrow$ Es gibt Probleme welche in NP und coNP liegen $\rightarrow$ Zertifikat für $w \in L$ und $w \notin L$
  Beispiel: Zertifikat für ex. Hamiltonpfad ist der Hamiltonpfad, aber kein Zertifikat für kein Hamiltonpfad
- es kann jede NTM-Klasse komplementiert werden (coN...)
- L ist nachweis-polynomiell $\Leftrightarrow L \in NP$
- eine Sprache ist **NP-schwer** wenn jede Sprache aus NP polyn. darauf reduziert werden kann
- eine Sprache ist **NP-vollständig** wenn sie NP-schwer ist und in NP liegt (NP-schwer + Zertifikat)
- $\rightarrow$ um zu Zeigen das ein Problem P NP-vollständig ist muss man zeigen, dass:
  - $P \in NP$
  - $Q \leq_p P$ Q bekanntes Problem in NP
- sollte gelten $P \neq NP$ dann ex. Probleme welche weder NP-vollständig sind noch in P liegen (Satz von Ladner) $\rightarrow$ **NP-intermediate**

Note: P ist anders als NP unter Komplement abgeschlossen (vertausche akzeptierende mit nicht-akzeptierenden Zuständen)

### 9.4.1 SAT

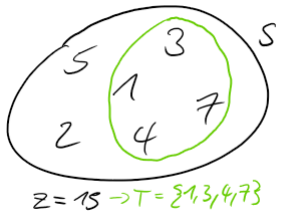
**Problem** Sei F eine Aussagenlogische Formel, gibt es eine Belegung, sodass F erfüllt ist ? **Beweisidee**

- Reduktion von Wortproblemen in NPTime
- verwenden so vieler aussagenlogischer Variablen, dass sie jeden polynomiellen Lauf kodieren könnten
- verwenden Formeln, so dass ede erfüllende Belegung einen korrekten akzeptierenden Lauf kodiert

Note: SAT ist in linearem Speicher lösbar

### 9.4.2 Teilmengen-Summe

**Problem** Sei eine Menge S von Gegenständen $a_i$ gegeben. Jeder Gegenstand hat dabei einen Wert $v(a_i)$.
Sei zudem ein Zielzahl z gegeben.
Gibt es eine Teilmenge $T \subseteq S$ sodass $\sum_{a \in T} v(a) = z$




**Beweis**

- TMS ∈ NP: T dient selbst als Zertifikat
- NP-schwere zeigen durch Reduktion auf SAT
  - Sei F eine KNF
  - seien $p_1, ..., p_n$ die Variablen in F
  - Definiere Gegenstände $t_i$ und $f_i$ für jede Variable $p_i$
  - die Kosten $v(t_i)$ und $v(f_i)$ seien also Binärfolge $a_1...a_n c_1...c_k$ gegeben
    - die $a_i$'s kodieren die Variablen in 2er Potenzen
    - die $c_i$'s geben jeweils an ob die Variable $p_i$ für $t_i$ oder $\neq p_i$ für $f_i$ in der jeweiligen Disjunktion der KNF vorkommt

- definiere ein $r = |C_i| - 1$
- definiere die Menge S als
  $S = \{t_i f_i | 1 \leq i \leq n\} \cup \{m_{i,j} | 1 \leq i \leq k, 1 \leq j \leq |C_i| - 1\}$
- z kann nun wie folgt bestimmt werden: $z = a_1 ... a_n c_1 .... c_k$ mit $a_i = 1$ und $c_i = |C_i|$
- Die Teilmengen-Summe ist erfüllbar gdw. F erfüllbar ist
  * Sei w eine erfüllende Belegung für F mit $w(p_i) = 1$ falls $t_i \in T$ und $w(p_i) = 0$ falls $f_i \in T$
  * definiere
    $T_1 = \{t_i | w(p_i) = 1, 1 \leq i \leq m\} \cup \{f_i | w(p_i) = 0, 1 \leq i \leq m\}$
  * definiere $T_2 = \{m_{i,j} | 1 \leq i \leq k, 1 \leq j \leq |C_i| - r_j\}$
  * die gesuchte Menge an Gegenständen ist $T = T_1 \cup T_2$
- so folgt: $\sum_{s \in T} v(s) = z$
- w ist wohldefiniert das $\forall i : t_i \in T$ xor $f_i \in T$

$$(p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_4) \wedge (p_4 \vee p_5 \vee \neg p_2 \vee \neg p_3)$$

| | | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $C_1$ | $C_2$ | $C_3$ |
|---|---|---|---|---|---|---|---|---|---|
| $v(t_1)$ | = | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $v(f_1)$ | = | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $v(t_2)$ | = | | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| $v(f_2)$ | = | | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $v(t_3)$ | = | | | 1 | 0 | 0 | 1 | 0 | 0 |
| $v(f_3)$ | = | | | 1 | 0 | 0 | 0 | 0 | 1 |
| $v(t_4)$ | = | | | | 1 | 0 | 0 | 0 | 1 |
| $v(f_4)$ | = | | | | 1 | 0 | 0 | 1 | 0 |
| $v(t_5)$ | = | | | | | 1 | 0 | 0 | 1 |
| $v(f_5)$ | = | | | | | 1 | 0 | 0 | 0 |
| $v(m_{1,1})$ | = | | | | | | 1 | 0 | 0 |
| $v(m_{1,2})$ | = | | | | | | 1 | 0 | 0 |
| $v(m_{2,1})$ | = | | | | | | | 1 | 0 |
| $v(m_{3,1})$ | = | | | | | | | | 1 |
| $v(m_{3,2})$ | = | | | | | | | | 1 |
| $v(m_{3,3})$ | = | | | | | | | | 1 |
| $z$ | = | 1 | 1 | 1 | 1 | 1 | 3 | 2 | 4 |

### 9.4.3 Nicht-Primzahl

**Problem** Sei eine natürliche Zahl $z > 1$ gegeben. Gibt es eine $p, q > 1$ mit $p \cdot q = z$ ? (Primfaktorzerlegung)

### 9.4.4 Clique

**Problem**
Sei ein Graph G und eine Zahl k gegeben. Enthält G eine vollvermaschte Knotenteilmenge mit k Knoten (Clique) ? **Beweis**

- Clique ∈ NP: Die Clique selbst ist ein Zertifikat
- Clique ist NP-schwer. z.z. durch Reduktion auf SAT
  - Sei F eine KNF
  - Sei $G_F$ ein Graph für den gilt: $G_F$ hat eine Clique der Größe l gdw. F erfüllbar ist
  - Knoten: Stehen in unterschiedlichen Disjunktionen der KNF, jeweils ein Paar aus Atom und Index der Disjunktion innerhalb der KNF
  - Kannten: zwischen Paaren, für welche der Index der Paares unterschiedlich ist und die Konjunktion der Atome erfüllbar ist
- $G_F$ kann in PTime berechnet werden

### 9.4.5 Rucksackproblem

**Problem** Sei G eine Menge von Gegenständen $a_1, ..., a_n$ mit je einem Wert $v(a_i)$ und einem Gewicht $g(a_i)$.
Sei zudem w ein Mindestwert und l ein Gewichtslimit
Gibt es eine Teilmenge, der Gegenstände, welche das Gewichtslimit einhalten und trotzdem den Mindestwert erreichen ?
**Beweis**

- Rucksackproblem ∈ NP: Zertifikat ist die gewählte Teilmenge
- Rucksackprobem ist NP-schwer
  - Reduktion des Rucksackproblems auf Teilmengen-Summe
  - Nehme eine Instanz des Teilmengen-Summe Problems mit $S = \{a_1, ..., a_n$ und Wert $v(a_i)$, Wunschwert z
  - Setzte für alle Gegenstände $i \in G$ den Wert gleich dem Gewicht
  - Setze den Mindestwert w und das Maximalgewicht l gleich dem Wunschwert z
  - diese Übersetzung ist in PTime
  - somit gilt: $\sum_{vi \in T} v(a_i) = z$ gdw. $\sum_{i \in T} v(i) \geq w = z$ und $\sum_{i \in T} g(i) \geq l = z$
- → Das Rucksackproblem ist NP-vollständig

## 9.5 Pseudopolynomielle Probleme

- Ein Problem ist in pseudopolynomieller Zeit lösbar, wenn es von einer DTM gelöst wird welche polynomiell Zeitbeschränkt ist bzgl. der Eingabe **und** des Betrages aller zahlen in der Eingabe
- oder: pseudopolynomieller wenn es in PTime liegt und alle Zahlen unär kodiert sind
- durch pseudopolynomielle Zeit kann es so scheinen, als ob das Problem im PTime lösbar ist, ist es aber nicht
- durch ineffiziente Kodierung der Eingabe wird so viel Platz geschaffen, dass ein Problem scheinbar in PTime lösbar wird
  da Resourcen nur im Bezug zur Inputgröße gemessen werden.
- Beispiel: Input 1 Mrd = 1.000.000.000 → nur 10 Stellen aber eig. exponentiell Größer $10^9$
  ⇒ Kodierung lässt Wert klein aussehen
- durch den übergroßen Input steht mehr Zeit zur Verfügung um das Problem scheinbar in PTime zu lösen

Note: Ist selbst nach einer unärkodierung ein Problem immer noch NP-vollständig, so heißt es **stark NP-vollständig**

## 9.6 PSpace

Es wird angenommen, dass Probleme in PSpace schwerer sind als NP

### 9.6.1 Quantifizierte Boolsche Formel (QBF)

- Eine QBF ist eine logische Formel, in welcher die Atome zusätzlich quantifiziert sind
- z.B. $\forall p_1 \forall p_2 \exists p_2 . (p_1 \wedge p_2) \vee (p_3 \rightarrow p_1)$
- aufeinanderfolgende Quantoren können zusammengefasst werden: $\forall p_1 \forall p_2 = \forall p_1, p_2$
- $\phi[p/q]$ bedeutet $\phi$ mit p ersetzt durch q (im jeweiligen Scope)
- **TrueQBF** ist das Problem ob $W(Q) = 1$

- TrueQBF lässt sich auf SAT reduzieren, indem alle Quantoren Existenzquantoren werden
- TrueQBF ist PSpace-vollständig
  - TrueQBF $\in$ PSpace
    - Sei F eine aussagenlogische Formel
    - enthält F keine Quantoren: löse F mit SAT
    - enthält F Existenzquantoren:
      - Teile die Formel auf
      - Teil 1: ersetze alle vorkommen der Quantifizierten Variable mit True
      - Teil 2: ersetzte alle vorkommen der Quantifizierten Variable mit False
      - Wende den Algorithmus rekursiv erneut auf Teil 1 und Teil 2 an
      - bilde die **Disjunktion** der beiden Ergebnisse
    - enthält F Allquantoren
      - Teile die Formel auf
      - Teil 1: ersetze alle vorkommen der Quantifizierten Variable mit True
      - Teil 2: ersetzte alle vorkommen der Quantifizierten Variable mit False
      - Wende den Algorithmus rekursiv erneut auf Teil 1 und Teil 2 an
      - bilde die **Konjunktion** der beiden Ergebnisse
- TrueQBF ist PSpace-schwer
  - Darstellen des Laufes einer TM durch aussagenlogische Atome
  - Speicher ist wie bei NP polynomiell
  - Zeit kann in PSpace exponentiell sein, daher kann nicht einfach für jeden Zeitschritt eine neue Version der Konfiguration angelegt werden
  - durchsuchen des exponentiell großen Graphen der TM durch **Middle-First-Search**
    - Ziel: gelangt man von s nach t ?
    - Prüfe ob s = t oder s direkter Vorgänger von t

        * Rate einen Punkt m in der Mitte eines Pfades von s nach t (Raten erfordert NTM)
        * prüfe Rekursiv ob man von s nach m und von m nach t gelangen kann (wieder durch Raten)
        * Anzahl zu ratender Mittelpunkte ist logarithmisch zur Länge des Pfades
          → somit braucht es durch logarithmisch viele Schritte in einem Exponentiellen Graph nur linear viele Schritte
        * Speicher kann in jedem Schritt wiederverwendet werden
    - der obrige Algorithmus lässt sich polynomiell in QBF kodieren

```
TrueQBF(F) :
 if F „hat keine Quantoren“ :
   return „Aussagenlogische Auswertung von F“
 else if F = ∃p.G :
   return (TrueQBF(G[p/⊤]) OR TrueQBF(G[p/⊥]))
 else if F = ∀p.G :
   return (TrueQBF(G[p/⊤]) AND TrueQBF(G[p/⊥]))
```

### 9.7 Weitere Klassen

- es lassen sich beliebig schwere Probleme der **k-Exp-Klasseen** konstruieren (k Höhe des Potenzturms)
- es gibt Probleme jenseits der exponentiellen Klassen. Diese nennt man nicht-elementare Klassen
- z.B. liegt Regex + Negationsoperator in dieser nicht-elementaren Klasse

## 10 Logik

### 10.1 Aussagenlogik

- jedes Atom $p \in P$ ist eine aussagenlogische Formel
- Seien F und G aussagenlogische Formeln, so sind es auch:
    - Negation $\neg F$

- Konjunktion $F \wedge G$
- Disjunktion $F \vee G$
- Implikation $F \rightarrow G$
- Äquivalenz $F \leftrightarrow G$

## 10.2 Prädikatenlogik

- In der Prädikatenlogik benutzt man eine Art Methode (Prädikate) mit dem Returntype Bool um Aussagen zu tätigen
- Die Prädikatenlogik ist **monoton**
  - Je mehr Sätze es gibt desto weniger Modelle gibt es
  - Je mehr Sätze es gibt desto mehr Schlussfolgerungen kann man ziehen
  - Mehr Annahmen führen zu mehr Schlüssen
  - Menge an Variablen V
  - Menge an Konstanten C
  - Menge and Prädikatensymbole P mit Stelligkeit $\geq 0$
- **Atom**: Ausdruck $p(t_1, ..., t_n)$ für ein n-stelliges Prädikatensymbol $p \in P$ und $t_i \in V \cup C$
- die prädikatenlogischen Formeln sind wie die aussagenlogischen Formeln definiert mit dem Zusatz:
- Existenzquantor $\exists x.F$
- Allquantor $\forall x.F$

### 10.2.1 Variablen

- Alle Variablen in einem Atom sind frei
- durch Verknüpfung ändert sich nichts an der 'Freiheit' einer Variable
- eine Variable wird gebunden, wenn die im Bereich (Scope) eines Quantors auftaucht
- Note: eine Variable kann gleichzeitig gebunden und frei auftauchen: z.B. x in $p(x) \wedge \exists x.q(x)$

- Formeln ohne freie Variablen heißen geschlossene Formeln oder **Sätze**
- Formeln mit freien Variablen heißen offene Formeln
- eine Menge von Sätzen nennt man auch **Theorie**

### 10.2.2 Interpretation und Zuweisung

Ersetzt die Wertzuweisung der Aussagenlogik **Interpretation**

- $(\Delta^I, \times^I)$ Paar aus einer Domäne $\Delta^I$ und einer Interpretationsfunktion $\times^I$
- die Interpretationsfunktion weißt jeder Konstante $a \in C$ ein Element der Domäne $a^I \in \Delta^I$ zu
  $f(a) = a^I$
- die Interpretationsfunktion weißt jedem n-stelligen Prädikatensymbol $p \in P$ eine Relation $p^I \subseteq (\Delta^I)^n$ zu
  $f(p) = p^I$

**Zuweisung**

- Funktion welche einer Variable $x \in V$ eine Domänenelement $\delta \in \Delta^I$ zuweist
- $Z[x \mapsto \delta]$ ist die Zuweisung, welche x auf $\delta$ und alle anderen Variablen auf $Z(y)$ abbildet

**Beispiel**

- Null ist eine natürliche Zahl, jede natürliche Zahl hat einen Nachfolger, dieser ist ebenfalls eine natürliche Zahl
- $NatNum(null) \wedge \forall x.(NatNum(x) \rightarrow \exists y.(succ(x, y) \wedge NatNum(y)))$
- Interpretation:
  - $\Delta^I = \mathbb{R}$ (Menge der Reellen Zahlen)
  - $null^I = 0$ (definiert erste Zahl)
  - $NatNum^I = \mathbb{N} \subseteq \mathbb{R}$ (Menge der natürlichen Zahlen)
  - $succ^I\{(d, e)|d, e \in \mathbb{R}, d < e$ ($\rightarrow$ succ(2, 7) = 1, da $2 < 7$)
- es kann nun geschlussfolgert werden, dass NatNum(null) wahr ist, da $null^I \in NatNum^I$ gilt
  $\rightarrow NatNum(num)^I = 1$

- Zuweisung:
  - Sei $Z(x) = 42$
  - Sei $Z(y) = 5$
- Unter dieser Interpretation und Zuweisung ist das Atom $succ(x, y)$ falsch, da $(Z(x), Z(y) \notin succ^I$ gilt (da $42 < 5 \rightarrow$ Falsch $\rightarrow succ(x, y)^{I,Z} = 0$

**Atome Interpretieren**

- Sei I ein Interpretation
- Sei Z eine Zuweisung für I
  - Für eine Konstante c definieren wir $c^{I,Z} = c^I$
  - Für eine Variable x definieren wir $x^{I,Z} = Z(x)$
- für ein Atom $p(t_1, ..., t_n)$ setzten wir nun:
  - $p(t_1, ..., t_n)^{I,Z} = 1$ wenn $(t_1^{I,Z}, ..., t_n^{I,Z}) \in p^I$
  - $p(t_1, ..., t_n)^{I,Z} = 0$ wenn $(t_1^{I,Z}, ..., t_n^{I,Z}) \notin p^I$
- Note: Interpretation wird hier auf 2 verschiedenen Ebenen benutzt!
  - $t^{I,Z} \in \Delta 1^I$ Interpretiert den Term
  - $A^{I,Z} \in \{0, 1\}$ Interpretation von A und der Zuweisung Z ist Wahl/Falsch

**Formeln Interpretieren** Eine Interpretation I und eine Zuweisung Z erfüllen eine Formel F ($I, Z \vDash F$) wenn gilt:

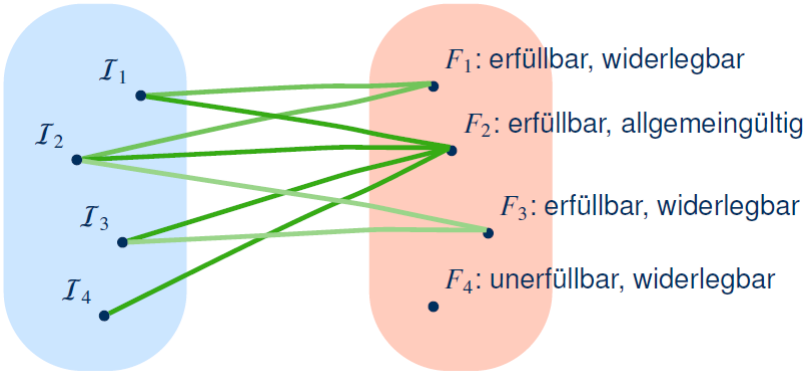
| Formel F | $I, Z \vDash F$ | $I, Z \nvDash F$ |
|---|---|---|
| F Atom | $F^{I,Z} = 1$ | $F^{I,Z} = 0$ |
| $\neg G$ | $I, Z \nvDash G$ | $I, Z \vDash G$ |
| $G_1 \wedge G_2$ | $I, Z \vDash G_1$ und $I, Zt \vDash G_2$ | $I, Z \nvDash G$ oder $I, Z \nvDash G$ |
| $G_1 \vee G_2$ | $I, Z \vDash G_1$ oder $I, Z \vDash G_2$ | $I, Z \nvDash G$ und $I, Z \nvDash G$ |
| $G_1 \rightarrow G_2$ | $I, Z \nvDash G_1$ oder $I, Z \vDash G_2$ | $I, Z \vDash G$ und $I, Z \nvDash G$ |
| $G_1 \leftrightarrow G_2$ | ($I, Z \vDash G_1$ und $I, Zt \vDash G_2$) oder ($I, Z \nvDash G$ und $I, Z \nvDash G$) | $\leftarrow$ -//- |
| $\forall x.G$ | $I, Z[x \mapsto \delta] \vDash G, \forall \delta \in \Delta^I$ | $I, Z[x \mapsto \delta] \nvDash G, \exists \delta \in \Delta^I$ |
| $\exists x.G$ | $I, Z[x \mapsto \delta] \vDash G, \exists \delta \in \Delta^I$ | $I, Z[x \mapsto \delta] \nvDash G, \forall \delta \in \Delta^I$ |

Eine Formel kann nach ihrem Modell

- allgemeingültig (tautologisch)
- widersprüchlich (inkonsistent, unerfüllbar)

- erfüllbar (konsistent)
- widerlegbar

sein




### 10.2.3 Logik auf Sätzen (Formeln ohne freie Variablen)

Note: Eine Menge von Sätzen nennt man auch **Theorie**
ein Beispiel für eine Theorie ist z.B. die Theorie der partiellen Ordnung

- Sei I eine Interpretation und F eine Formel
- Wenn $I \vDash F$ dann nennt man I ein Modell für F (I erfüllt F)
- I ist ein Modell für eine Formelmenge (Theorie) T ($I \vDash T$) wenn
  $I \vDash F, \forall F \in T$
- F ist eine logische Konsequenz aus einer Formel oder Formelmenge G
  ($G \vDash F$), wenn jedes Modell in G auch ein Modell von F ist
  ($I \vDash G \rightarrow G \vDash F$)
  z.B. ist $I_1, I_2 \vDash F$ und $I_1 \vDash G$ dann $G \vDash F$
- ist F eine Tautologie dann $\vDash F$ (ohne Voraussetzung gilt F)
- Note: Alle Tautologien sind semantisch äquivalent, und in jedem Modell wahr, somit auch logische Konsequenz jeder Theorie
- Zwei Formeln oder Formelmengen sind semantisch äquivalent $F \equiv G$,
  wenn sie die gleichen Modell haben
  ($I \vDash F \leftrightarrow I \vDash G, \forall I$) $\rightarrow$ (wechselseitige logische Konsequenz)

### 10.2.4 Gleichheit und Ungleichheit

**Definition**

- Gleichheit: $\approx^I = \{(\delta, \delta) | \delta \in \Delta^I\}$
- meist infix geschrieben
- Beispiel: $\forall x, y.(erde(x) \wedge erde(y) \rightarrow x \approx y)$ = es gibt nur eine Erde
- Eigenschaften von $\approx$ logisch beschrieben:
  - Reflexivität: $\forall x.eq(x, x)$
  - Symmetrie: $\forall x, y.eq(x, y) \rightarrow eq(y, x)$
  - Transitivität: $\forall x, y, z.eq(x, y) \wedge eq(y, z) \rightarrow eq(x, z)$
  - Kongruenz:
    $\forall x_1, ..., x_n, y.((p(x_1, ..., x_i, ..., x_n) \wedge eq(x_i, y)) \rightarrow p(x_1, ..., y, ..., x_n))$
    da $x_i = y$ können diese ausgetauscht werden (ersetze gleiches mit Gleichem)
- Ungleichheit: $\forall x, y.(x \not\approx y \leftrightarrow \neg x \approx y)$

**Liste von Gleichheiten**

- $\neg\exists.F \equiv \forall x.\neg F$
- $\neg\forall.F \equiv \exists x.\neg F$
- $\exists x.\exists y.F \equiv \exists y.\exists x.F$
- $\forall x.\forall y.F \equiv \forall y.\forall x.F$
- $\exists x.(F \vee G) \equiv (\exists x.F \vee \exists x.G)$
- $\forall x.(F \wedge G) \equiv (\forall x.F \wedge \forall x.G)$

**Liste von Ungleichheiten**

- $\exists x.\forall y.F \not\equiv \forall y.\exists x.F$
  $\rightarrow$ (Jeder hat einen Vater $\neq$ Es gibt einen Vater aller)
- $\forall x.(F \vee G) \equiv (\forall x.F \vee \forall x.G)$
  $\rightarrow$ (Jeder ist entweder glücklich oder nicht $\neg$ es sind alle glücklich oder nicht)

### 10.2.5 Unentscheidbarkeit

Logisches schließen in der Prädikatenlogik ist unentscheidbar
Dies gilt auch wenn nur binäre Prädikatensymbole zugelassen sind
**Beweis**

- Reduktion des CFG-Schnittproblems
  - Gegeben 2 kontextfreie Grammatiken (CFG) $G_1$ und $G_2$
  - Gibt es ein Wort $w \in L(G_1) \cap L(G_2)$ ?
- kodieren von Wörtern als Kette von binären Relationen aus der Prädikatenlogik (für Terminale und nicht Terminale)
  - Beispiel 'baum'
    * $(\delta_0, \delta_1) \in p_b^I$
    * $(\delta_1, \delta_2) \in p_a^I$
    * $(\delta_2, \delta_3) \in p_u^I$
    * $(\delta_3, \delta_4) \in p_m^I$
    * $= \delta_0 \rightarrow^{p_b} \delta_1 \rightarrow^{p_a} \delta_2 \rightarrow^{p_u} \delta_3 \rightarrow^{p_m} \delta_4$
- nehme an, dass $G_1$ und $G_2$ keine gemeinsamen nicht Terminale habe (Terminale schon)
- kodiere jedes nicht Terminal $A \rightarrow \sigma_1 \cdots \sigma_n$ als Formel
- konstruiere eine Formel, welche erkennt ob eine Folge von Terminalen und nicht Terminalen aus einem anderen nicht Terminal entstehen kann
  $(\beta = \alpha_1 \cdots \alpha_n, \alpha_i \in T \cup N, \beta \in N)$
  erkennt ob eine Zeichenkette aus einem anderen nicht Terminal abgeleitet werden kann
- Formel: $\forall x_0, ..., x_n.((p_{\sigma_1}(x_0, x_1) \wedge \cdots \wedge p_{\sigma_n}(x_{n-1}, x_n)) \rightarrow p_A(x_0, x_n))$ gibt es eine Kette in Modell, dann verbinde deren erstes und letztes Symbol mit einem $p_A$ (A ist nicht Terminal)
- Seien $S_1$ und $S_2$ die Startsymbole von $G_1$ und $G_2$
- kodiere das Schnittproblem wie folgt:
  - $\exists x, y.(p_{S_1}(x, y) \wedge p_{S_2}(x, y))$
  - gibt es ein Modell ohne Zyklen oder parallele Relationen welches die obrige Formel erfüllt, so muss es ein Wort geben, welches im Schnitt der Sprachen liegt

- es müssen noch alle möglichen Wörter kodiert werden, dafür
  - $\forall x.\exists y.p_a(x,y), \forall a \in T$
- konstruiere eine Theorie TH aus $G_1$ und $G_2$
- dann muss gelten: $TH \models \exists x, y.(p_{S_1}(x,y) \wedge p_{S_2}(x,y))$ gdw. $w \in L(G_1) \cap L(G_2)$ gilt

## 10.3 Negations Normal Form (NNF)

eine Formel F ist in NNF wenn:

- enthält nur Quantoren und die Junktoren **und** ($\wedge$), **or** ($\vee$), **not** ($\neg$)
- der Junktor $\neg$ kommt nur direkt vor einem Atom vor

**Umwandlung**

- ersetze $F \rightarrow G$ durch $\neg F \vee G$
- ersetze $G \leftrightarrow G$ durch $(\neg F \vee G) \wedge (\neg G \vee F)$

## 10.4 Bereinigte Formel

eine Formel F ist bereinigt wenn:

- keine Variable in F gleichzeitig gebunden und frei vorkommt
- keine Variable in F von mehr als einem Quantor gebunden wird

**Umwandlung**

- entfernen von Quantoren die nichts binden (x kommt in F nicht als freie Variable vor)
  - $\exists x.F \equiv F \equiv \forall x.F$
  - $\exists x.(F \wedge G) \equiv (F \wedge \exists x.G) \equiv (\exists x.F \wedge \exists x.G)$
  - $\forall x.(F \vee G) \equiv (F \vee \forall x.G) \equiv (\forall x.F \vee \forall x.G)$
  - $\exists x.\forall y.F \equiv \forall y.\exists x.F$
  - $\forall x.\exists y.F \equiv \exists y.\forall x.F$
- Variablenumbenennung
  - Eine Variable kann umbenannt werden, sofern dies die Semantik der Formel nicht verändert
  - $Qx.F \equiv Qy.F\{x \mapsto y\}$ mit $Q \in \{\forall, \exists\}$, wenn y nicht in F vorkommt

## 10.5 Pränexform

Eine Formel F ist in Pränexform wenn alle Quantoren am **Anfang** stehen
$Q_1 x_1. \cdots Q_n x_n. F, Q \in \{\forall, \exists\}$
**Umwandlung**

- Sei F eine bereinigte Formel in NNF
- $Qx.F \circ G) \mapsto Qx.(F \circ G)$
- $(G \circ Qx.F) \mapsto Qx.(G \circ F)$
- $Q \in \{\forall, \exists\}, \circ \in \{\wedge, \vee\}$

## 10.6 Skolemisierung

Einführen von Funktionssymbolen in die Logik
**Umwandlung**

- Sei $\forall x_1. \cdots \forall x_n. \exists y. F$ eine Formel in Pränexform
- Die Skolemisierung von y ist die Formel $\forall x_1. \cdots \forall x_n. F'$
  - $F' = F\{y \mapsto f(x_1, ..., x_n)\}$ ersetze jedes freie Vorkommen von y in F durch $f(x_1, ..., x_n)$
  - die Funktion f muss dabei 'neu' sein und darf nirgends sonnst bisher vorkommen
- führe die obrige Umformung für jede existenziell quantifizierte Variable (mit $\exists$), von vorn nach hinten durch

**Beispiel**

- $\forall x. \exists y. \forall z. \exists v. p(x, y, z, v)$
- Skolemisiere y: $= \forall x. \forall z. \exists v. p(x, f(x), z, v)$ (y wird zu f(x), da y von x allquantifiziert wird)
- Skolemisiere v: $= \forall x. \forall z. p(x, f(x), z, g(x, z))$ (v wird zu g(x,z), da v von x und z allquantifiziert wird)

## 10.7 Konjunktive Normalform (KNF)

Eine Formel F ist in KNF wenn:

- sie in Pränexform ist
- die Form $(L_{1,1} \vee \cdots \vee L_{1,m}) \wedge (L_{n,1} \vee \cdots \vee L_{n,m})$ hat
- L sind Literale
- Klauseln sind Disjunktionen ($\vee$) von Literalen

**Umwandlung**

- $F \vee (G \wedge H) \mapsto (F \vee G) \wedge (F \vee H)$

## 10.8 Unifikation

### 10.8.1 Substitution

- Menge von Ersetzungsregeln
- $\{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$
- $x_i \in V$ paarweise verschieden, $t_i \in T$ beliebige Terme
- Komposition von Substitutionen (Hintereinanderausführung)
  - Komposition $\sigma \circ \theta$
  - $\sigma = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$, $\theta = \{y_1 \mapsto s_1, ..., y_m \mapsto s_m\}$
  - $\sigma \circ \theta = \{x_1 \mapsto t_1\theta, ..., x_n \mapsto t_n\theta\} \cup$
    $\{y_i \mapsto s_i | y_i \in \{y_1, ..., y_m\} \setminus \{x_1, ..., x_n\}\}$

**Unifikator**

- Sei G eine Menge von Gleichungen der Form $G = \{s_1 = t_1, ..., s_n = t_n\}$
- Ein Unifikator ist eine Substitution $\sigma$ für welche $s_i\sigma = t_i\sigma, \forall i \in \{1, ..., n\}$ gilt
- ein Unifikator ist also eine Menge von Überführungsregeln um eine Menge von Gleichungen syntaktisch anzugleichen
- Beispiele
  - $\{f(x) = f(y)\}$ hat den Unifikator $\{x \mapsto y\}$

- $\{f(x) = g(x)\}$ hat keinen Unifikator

- Es gibt einen allgemeinsten Unifikator (most general unifier (mgu))
  - Eine Substitution $\sigma$ ist allgemeiner als eine Substitution $\theta$ ($\sigma \preceq \theta$) wenn:
    * es eine Substitution $\lambda$ gibt, sodass $\sigma \circ \lambda = \theta$
    * ein allgemeiner Unifikator kann also erhalten werden, indem eine Teilmenge ($\lambda$) von ihm abgespalten wird
    * daraus folgt auch, dass der MGU nicht weiter zerlegt werden kann

- **Unifikationsalgorithmus**
  - Sei G ein Unifikationsproblem
  - **Löschen**: $\{t = t\} \cup G' \rightarrow G$
    syntaktisch gleiche Gleichungen können aus G entfernt werden
  - **Zerlegung/Dekomposition**:
    $\{f(s_1, ..., s_n) = f(u_1, ..., u_n)\} \cup G' \rightarrow \{s_1 = u_1, ..., s_n = u_n\} \cup G'$
    wird die gleiche Funktion auf eine Anzahl von Variablen angewendet, so kann man die Funktion weglassen
  - **Orientierung**: $\{t = x\} \cup G' \rightarrow \{x = t\} \cup G'$ mit $x \in V$ und $t \notin V$
    Kommutativität
  - **Eleminierung**: $\{x = t\} \cup G' \rightarrow \{x = t\} \cup G'\{x \mapsto t\}$, für $x \in V$ und $x \notin t$
    Einsetzten in die anderen Gleichung, das Eingesetzte verschwindet dabei

$$\begin{aligned}
\{\underline{x \doteq f(a)}, g(x,x) \doteq g(x,y)\} &\quad \text{Eliminierung} \\
\{x \doteq f(a), \underline{g(f(a),f(a)) \doteq g(f(a),y)}\} &\quad \text{Zerlegung} \\
\{x \doteq f(a), \underline{f(a) \doteq f(a)}, f(a) \doteq y\} &\quad \text{Löschen} \\
\{x \doteq f(a), \underline{f(a) \doteq y}\} &\quad \text{Orientierung} \\
\{x \doteq f(a), y \doteq f(a)\} &\quad \text{gelöste Form}
\end{aligned}$$

**Beweis der Korrektheit** Der Unifikationsalgorithmus liefert immer den MGU oder nicht-unifizierbar

- Hilfsaussage ★: Wenn ein Unifikationsproblem $G_1$ in eimem Eretzungschritt in ein Problem $G_2$ umgewandelt werden kann, dann haben $G_1$ und $G_2$ die gleichen Unifikatoren
  - für Löschen, Orientierung, Zerlegung trivial
  - für Eleminierung
    * Sei $G_1 = \{x = t\} \cup G'$ und $\sigma = \{x \mapsto t\}$
    * dann ist $G_2 = \{x = t\} \cup G'\sigma$
    * Sei $\theta$ Unifikator für $G_1$,
    * gdw. $x\theta = t\theta$ und $\theta$ Unifikator für G'
    * gwd. $x\theta = t\theta$ und $\sigma \circ \theta$ Unifikator für G', da $\sigma = \sigma_{\{x=t\}}$ und so $\theta = \sigma \circ \theta$ für alle Unifikatoren von $\theta$ von $\{x = t\}$
    * gdw. $x\theta = t\theta$ und $\theta$ Unifikator für $G'\sigma$
    * gdw. $\theta$ Unifikator für $\{x = t\} \cup G'\sigma = G_2$
- Wenn der Algorithmus eine Substitution $\sigma$ liefert, dann ist $\sigma$ MGU
  - nach ★ erhält jeder Umformungsschritt die Unifikatoren, somit auch die allgemeinsten Unifikatoren
  - Per Induktion gilt, dass jedes Unifikationsproblem, welches der Algorithmus erzeugt, hat den gleichen MGU wie G hat
  - Wenn der Algorithmus also einen Unifikator zurückgibt, ist dieser der MGU
- Wenn der Algorithmus nicht-unifizierbar liefert, dann hat G keinen Unifikator
  - sollte es keinen Unifikator geben, so muss das Problem G' eine der folgenden Gleichungen enthalten
    * x = t, mit $x \in V$ und $x \in t$
    * $f(\cdots) = g(\cdots)$ mit $f \neq g$
  - Fall 1: $s = u$ hat auf einer Seite eine Variable. Dann hat sie Form 1, da sonst Orientierung oder Löschen möglich wäre
  - Fall 2: $s = u$ hat auf keiner Seite eine Variable. Dann hat sie Form 2, da sonst Zerlegung möglich wäre
- Der Algorithmus terminiert nach endlich vielen Schritten
  - anordnen des Unifikationsproblems so, dass das Problem in jedem Schritt kleiner wird

- Sei v die Anzahl nicht gelöster Variablen in G'
- Sei g die Gesamtzahl (Summe) der Vorkommen von Funktionssymbolen, Konstanten und Variablen in G
- Sei r die Anzahl der Gleichung in $s = x \in G'$ mit $x \in V$ auf der **rechten** Seite
- ordne nun das Problem G' lexikographisch bzgl. der Tripel $\kappa(G')$
- Für $G_1$ und $G_2$ mit $\kappa(G') = (v_i, g_i, r_i)$ gilt $G_1 \prec G_2$ falls min eines davon gilt:
  * $v_1 > v_2$
  * $v_1 = v_2$ und $g_1 > g_2$
  * $v_1 = v_2$ und $g_1 = g_2$ und $r_1 > r_2$
  * man vergleicht immer die Stellen im Tripel und hört auf zu vergleichen wenn etwas nicht gleich ist
  * Beispiel: $(4, 2, 1) \prec (3, 42, 23) \prec (3, 42, 22) \prec (3, 41, 1000)$
- Die Behauptung folgt nun, da jede Regelanwendung zu einem kleineren ($\prec$) Unifikationsproblem führt
- Da es keine unendlichen Ketten immer kleiner werdender Probleme gibt muss der Algorithmus terminieren

| | | $v$ | $g$ | $r$ |
|---|---|---|---|---|
| $\{x \doteq f(a), g(x,x) \doteq g(x,y)\}$ | Eliminierung | 2 | 9 | 0 |
| $\{x \doteq f(a), g(f(a),f(a)) \doteq g(f(a),y)\}$ | Zerlegung | 1 | 12 | 0 |
| $\{x \doteq f(a), f(a) \doteq f(a), f(a) \doteq y\}$ | Löschen | 1 | 10 | 1 |
| $\{x \doteq f(a), f(a) \doteq y\}$ | Orientierung | 1 | 6 | 1 |
| $\{x \doteq f(a), y \doteq f(a)\}$ | gelöste Form | 0 | 6 | 0 |

Es gilt: $\langle 2,9,0\rangle \succ \langle 1,12,0\rangle \succ \langle 1,10,1\rangle \succ \langle 1,6,1\rangle \succ \langle 0,6,0\rangle$

## 10.9 Resolution

### 10.9.1 Resolvente

- Sei $K_1 = \{A_1, ..., A_n, L_1, ..., L_k\}$ und $K_2 = \{\neg A'_1, ..., \neg A'_m, L'_1, ..., L'_l\}$ Klauseln

- Sei $\sigma$ der MGU der Menge $\{A_1, ..., A_n, A'_1, ..., A'_m\}$
- Seien $L_i$ und $L'_j$ beliebige Literale
- die Resolvente ist nun die Klausel der Form $\{L_1\sigma, ..., L_k\sigma, L'_1\sigma, ..., L'_l\sigma\}$
- in der Resolvente löschen sich also immer zwei entgegengesetzte Teilterme aus
- führt man die Resolution immer wieder aus und erhält die leere Menge so ist die Formel erfüllbar
- erhält man die leere Menge nicht, so ist die Formel unerfüllbar

## 10.10 Herbrantmodelle

Ziel: **Semantik aus Syntax**:
Konstruktion von Modellen direkt aus den Formeln, welche sie erfüllen sollen
**Herbrantuniversum**

- Sei $\Delta_F$ das Herbrantuniversum für eine Formel F
- die Menge aller Terme ohne Variablen die man mit Konstanten und Funktionssymbolen in F und einer zusätzlichen Konstante a bilden kann
  - $a \in \Delta_F$ (per Definition)
  - $c \in \Delta_F$ für jede Konstante aus F
  - $f(t_1, ..., t_n \in \Delta_F$ für jedes n-stellige Funktionssymbol in F und alle Terme $t_1, ...t_n \in \Delta_F$
- Note: Das Herbranduniversum ist immer abzählbar, nicht aber zwingend endlich
- Beispiel:
  - $F = p(f(x), y, g(z))$
  - $\Delta_F = \{a, f(a), g(a), f(f(a)), f(g(a)), g(f(a)), g(g(a)), ...\}$

**Herbrantinterpretation**

- Ist die Interpretation I für F für welche gilt: $\Delta^I = \Delta_F$
- für jeden Term $t \in \Delta_F$ gilt $t^I = t$
- Ist I ein Modell für F ($I \vDash F$), dann ist I ein **Herbrandmodell** für F

- Es ist egal ob eine Ersetzung auf eine Zuweisung oder eine Formel angewendet wird:
- $I, Z\{x \mapsto t\} \models F$ gdw. $I, Z \models F\{x \mapsto t\}$
- Ein Satz F in Skolenform ist erfüllbar gdw. F ein Herbrandmodell ist

**Beispiele**

- $I_1$ mit $hatVater^{I_1} = \emptyset$ kein Herbrandmodell, Formel nicht erfüllt (niemand hat einen Vater)
- $I_2$ mit $hatVater^{I_2} = \{(t, (f(t)|t \in \Delta_F\}$ Herbrandmodell, da Formel erfüllt
- $I_3$ mit $hatVater^{I_3} = \Delta_F \times \Delta_F$ Herbrandmodell, da Formel erfüllt