

DEFINITION

**Alphabet**

DEFINITION

**Menge der endlichen Folgen**

DEFINITION

**Wort**

DEFINITION

**Induktiv  $w^n$  definieren**

DEFINITION

$y, w$  sind Wörter über  $\Sigma$ . Dann heißt  $y$ :

DEFINITION

**Sprachen**

DEFINITION

**Präfix**

DEFINITION

**Infix**

DEFINITION

**Suffix**

DEFINITION

**formale Sprachen**

Für eine Menge  $X$  ist  $X^*$  die Menge der endlichen Folgen über  $X$ .

Beispiel: Elemente von  $\{a, b, c, d\}^*$ :  $(a, b, c), (), (a, c, d) \dots$

$$w^n = \begin{cases} \epsilon & \text{falls } n = 0 \\ w * w^{n-1} & \text{falls } n > 0 \end{cases}$$

f: Menge der möglichen Eingaben  $\rightarrow$  Menge der möglichen Ausgaben  
Spezialfall  $A = 0, 1$  heißt Entscheidungsproblem. Sie ist gegeben durch die Menge der Eingaben.

Seien  $y, w$  Wörter über  $\Sigma$ . Dann heißt Infix/Faktor von  $w$ , wenn es  $x, z \in \Sigma^*$  gibt mit  $xyz = w$ .

Sei  $\Sigma$  ein Alphabet. Teilmengen von  $\Sigma^*$  werden formale Sprachen über  $\Sigma$  genannt.  
Eine Menge  $L$  ist eine formale Sprache wenn es ein Alphabet  $\Sigma$  gibt, so dass  $L$  formale Sprache über  $\Sigma$  ist (d.h.  $L \subseteq \Sigma^*$ ).

Ein Alphabet ist eine endliche nichtleere Menge. Üblicherweise heißen Alphabete hier  $\Sigma, \Gamma, \Delta$ . Ist  $\Sigma$  Alphabet, so nennen wir die Elemente oft Buchstaben und die Elemente von  $\Sigma^*$  auch Wörter über  $\Sigma$  (auch String/Zeichenkette).

Sind  $u = (a_1, a_2, \dots, a_n)$  und  $v = (b_1, b_2, \dots, b_n)$  Wörter, so ist  $u * v$  das Wort  $(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n)$ ; es wird als Verkettung/Konkatenation von  $u$  und  $v$  bezeichnet. An Stelle von  $u * v$  schreibt man auch  $uv$ .

- Präfix/Anfangsstück von  $w$ , wenn es  $z \in \Sigma^*$  gibt mit  $yz = w$
- Infix/Faktor von  $w$ , wenn es  $x, z \in \Sigma^*$  gibt mit  $xyz = w$
- Suffix/Endstück von  $w$ , wenn es  $x \in \Sigma^*$  gibt mit  $xy = w$

Seien  $y, w$  Wörter über  $\Sigma$ . Dann heißt Präfix/Anfangsstück von  $w$ , wenn es  $z \in \Sigma^*$  gibt mit  $yz = w$ .

Seien  $y, w$  Wörter über  $\Sigma$ . Dann heißt Suffix/Endstück von  $w$ , wenn es  $x \in \Sigma^*$  gibt mit  $xy = w$ .

DEFINITION

## **Verkettung von Sprachen**

DEFINITION

## **Kleene Abschluss**

DEFINITION

## **Prioritätsregeln für Operationen auf Sprachen**

DEFINITION

## **Grammatik**

DEFINITION

## **Ableitung einer Grammatik**

DEFINITION

## **Wort ist Satzform**

DEFINITION

## **erzeugte Sprache**

DEFINITION

## **Chomsky-0**

DEFINITION

## **Chomsky-1**

DEFINITION

## **Chomsky-2**

Sei  $L$  eine Sprache. Dann ist  $L^* = \bigcup_{n \geq 0} L^n$  der Kleene-Abschluss oder die Kleene-Iteration von  $L$ .

Weiter ist  $L^+ = \bigcup_{n \geq 0} L^n$   
 $(L^+ = L * L = L^* * L)$

Sind  $L_1$  und  $L_2$  Sprachen, so heißt die Sprache  $L_1 L_2 = \{w \mid \exists w_1 \in L_1, w_2 \in L_2 : w = w_1 w_2\}$  (auch  $L_1 * L_2$ ) die Konkatenation oder Verkettung von  $L_1$  und  $L_2$ .

Grammatiken sind ein Mittel um alle syntaktisch korrekten Sätze einer Sprache zu erzeugen.

Eine Grammatik  $G$  ist ein 4-Tupel  $G = (V, \Sigma, P, S)$  das folgende Bedingungen erfüllt

- $V$  ist eine endliche Menge von Nicht-Terminalen oder Variablen
- $\Sigma$  ist ein Alphabet (Menge der Terminale) mit  $V \cap \Sigma = \emptyset$ , d.h. kein Zeichen ist gleichzeitig Terminal und Nicht-Terminal
- $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$  ist eine endliche Menge von Regeln oder Produktionen (Produktionsmenge)
- $S \in V$  ist das Startsymbol/ die Startvariable oder das Axiom

Jede Grammatik hat nur endlich viele Regeln!

- Potenz/Iteration binden stärker als Konkatenation
- Konkatenation stärker als Vereinigung/Durchschnitt/Differenz

Ein Wort  $w \in (V \cup \Sigma)^*$  heißt Satzform, wenn es eine Ableitung gibt, deren letztes Wort  $w$  ist.

Sei  $G = (V, \Sigma, P, S)$  eine Grammatik. Eine Ableitung ist eine endliche Folge von Wörtern  $w_0, w_1, w_2, \dots, w_n$  mit  $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$ .

Jede Grammatik ist vom Typ 0 (Semi-Thue-System) und wird auch als rekursiv-aufzählbar bezeichnet.

Die Sprache  $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$  aller Satzformen aus  $\Sigma^*$  heißt von  $G$  erzeugte Sprache.

Eine Regel  $(l \rightarrow r)$  heißt kontext-frei wenn  $l \in V$  und  $r \in (V \cup \Sigma)^*$  gilt. Eine Grammatik ist vom Typ 2, falls sie nur kontext-freie Regeln enthält

Eine Regel heißt kontext-sensitiv, wenn es Wörter  $u, v, w \in (V \cup \Sigma)^*$ ,  $|v| > 0$  und ein Nichtterminal  $A \in V$  gibt mit  $l = uAw$  und  $r = vww$ . Eine Grammatik ist vom Typ 1 (oder kontext-sensitiv) falls

- alle Regeln aus  $P$  kontext-sensitiv sind
- $(S \rightarrow \epsilon) \in P$  die einzige nicht kontext-sensitive Regel in  $P$  ist und  $S$  auf keiner rechten Seite einer Regel aus  $P$  vorkommt

DEFINITION

**Chomsky-3**

BEWEISE

**Es gibt einen Algorithmus, der als Eingabe eine Typ-1-Grammatik  $G$  und ein Wort  $w$  bekommt und nach endlicher Zeit entscheidet ob  $w \in L(G)$  gilt.**

DEFINITION

**Deterministische endliche Automaten**

DEFINITION

**DFA mit Funktion  $\hat{\delta}$**

DEFINITION

**von einem DFA akzeptierte Sprache**

DEFINITION

**Wann ist eine Sprache regulär?**

DEFINITION

**ein nichtdeterministischer endlicher Automat  $M$**

DEFINITION

**Zu einem gegebenen NFA  $M$  definieren wir die Funktion  $\hat{\delta} : P(Z) \times \Sigma^* \rightarrow P(Z)$**

DEFINITION

**die von einem NFA  $M$  akzeptierte Sprache ist**

SATZ

**Sei  $\Sigma$  ein Alphabet und  $L \subseteq \Sigma^*$  eine Sprache. Dann sind äquivalent**

1.  $w = \epsilon$ : Da  $G$  vom Typ 1 ist, gilt  $w \in L(G)$  genau dann wenn  $(S \rightarrow \epsilon) \in P$ . Dies kann ein Algorithmus entscheiden

2.  $|w| \geq 1$ : Definiere einen gerichteten Graphen  $(W, E)$  wie folgt

- Knoten sind die nichtleeren Wörter über  $V \cup \Sigma$  der Länge  $\geq |w|$  (insbes.  $S, w \in W$ )
- $(u, v) \in E$  genau dann wenn  $u \Rightarrow_G v$

da kontextsensitiv ist, gilt  $1 = |u_0| \geq |u_1| \geq |u_2| \geq \dots \geq |u_n| = |w|$ , also  $u_i \in W$  f.a.  $1 \leq i \leq n$ . Also existiert Pfad von  $S$  nach  $w$  im Graphen  $(W, E)$ , womit die Behauptung bewiesen ist.

Zu einem gegebenen DFA definieren wir die Funktion

$$\hat{\delta} : Z \times \Sigma^* \rightarrow Z \text{ induktiv wie folgt, wobei } z \in Z, w \in \Sigma^+ \text{ und } a \in \Sigma:$$

- $\hat{\delta}(z, \epsilon) = z$
- $\hat{\delta}(z, aw) = \hat{\delta}(\delta(z, a), w)$

Der Zustand  $\hat{\delta}(z, w)$  ergibt sich indem man vom Zustand  $z$  aus dem Pfad folgt der mit  $w$  beschriftet ist.

Eine Sprache  $L \subseteq \Sigma^*$  ist regulär, wenn es einen DFA mit  $L(M) = L$  gibt (bzw. wird von einem DFA akzeptiert). Jede reguläre Sprache ist rechtslinear.

induktiv wie folgt, wobei  $Y \subseteq Z$ ,  $w \in \Sigma^*$  und  $a \in \Sigma$ :  $\hat{\delta}(Y, \epsilon) = Y$ ,  $\hat{\delta}(Y, aw) = \bigcup \delta(z, a), w$

1.  $L$  ist regulär (d.h. von einem DFA akzeptiert) 2.  $L$  wird von einem NFA akzeptiert 3.  $L$  ist rechtslinear (d.h. von einer Typ-3 Grammatik erzeugt)

Eine Regel ist rechtslinear, wenn  $l \in V$  und  $r \in \Sigma \cup V \cup \epsilon$  gilt. Eine Grammatik ist vom Typ 3 wenn sie nur rechtslineare Regeln enthält.

ein deterministischer endlicher Automat  $M$  ist ein 5-Tupel  $M = (Z, \Sigma, z_0, \delta, E)$

- $Z$  eine endliche Menge von Zuständen
- $\Sigma$  das Eingabealphabet (mit  $Z \cap \Sigma = \emptyset$ )
- $z_0 \in Z$  der Start/Anfangszustand (max. einer)
- $\delta : Z \times \Sigma \rightarrow Z$  die Übergangsfunktion
- $E \subseteq Z$  die Menge der Endzustände

Abkürzung: DFA (deterministic finite automaton)

die von einem DFA akzeptierte Sprache ist:

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E\}$$

Mit anderen Worten: Ein Wort  $w$  wird genau dann akzeptiert, wenn derjenige Pfad, der im Anfangszustand beginnt und dessen Übergänge mit den Zeichen von  $w$  markiert sind, in einem Endzustand endet.

ist ein 5-Tupel  $M = (Z, \Sigma, S, \delta, E)$  mit -  $Z$  ist eine endliche Menge von Zuständen -  $\Sigma$  ist das Eingabealphabet -  $S \subseteq Z$  die Menge der Startzustände (können mehrere sein) -  $\delta : Z \times \Sigma \rightarrow P(Z)$  ist die (Menge der) Überföhrungs/Übergangsfunktion -  $E \subseteq Z$  die Menge der Endzustände

$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(S, w) \cap E \neq \emptyset\}$  (Das Wort wird akzeptiert wenn es mindestens einen Pfad vom anfangs in den endzustand gibt)

DEFINITION

Gegeben sei eine Klasse  $K$  und ein  $n$ -stelliger Operator  $\otimes : K^n \rightarrow K$ .

SATZ

Wenn  $L \subseteq \Sigma^*$  eine reguläre Sprache ist,

SATZ

Wenn  $L_1$  und  $L_2$  reguläre Sprachen sind,

SATZ

Wenn  $L_1$  und  $L_2$  reguläre Sprachen sind,

SATZ

Wenn  $L_1$  und  $L_2$  reguläre Sprachen sind,

SATZ

Wenn  $L$  eine reguläre Sprache ist,

SATZ

Wenn  $L$  eine reguläre Sprache ist,

DEFINITION

Die Menge  $Reg(\Sigma)$  der regulären Ausdrücke über dem Alphabet  $\Sigma$

DEFINITION

Für einen regulären Ausdruck  $\alpha \in Reg(\Sigma)$  ist die Sprache  $L(\alpha) \subseteq \Sigma^*$

SATZ

Für jedes Alphabet  $\Sigma$  ist die Menge  $P(\Sigma^*) = L|L$  Sprache über  $\Sigma$  überabzählbar

dann ist auch  $\Sigma^* \setminus L$  regulär

Man sagt, eine Klasse  $K' \subseteq K$  ist unter  $\otimes$  abgeschlossen, wenn für beliebige Elemente  $k_1, k_2, \dots, k_n \in K'$  gilt  $\otimes(k_1, k_2, \dots, k_n) \in K'$

dann ist auch  $L_1 \cap L_2$  regulär.

dann ist auch  $L_1 \cup L_2$  regulär.

dann ist auch  $L^+$  regulär

dann ist auch  $L_1 L_2$  regulär

ist die kleinste Menge mit folgenden Eigenschaften:

- $\emptyset \in \text{Reg}(\Sigma), \lambda \in \text{Reg}(\Sigma), \Sigma \subseteq \text{Reg}(\Sigma)$
- Wenn  $\alpha, \beta \in \text{Reg}(\Sigma)$ , dann auch  $(\alpha * \beta), (\alpha + \beta), (\alpha^*) \in \text{Reg}(\Sigma)$

dann ist auch  $L^*$  regulär.

, d.h. es gibt keine bijektive Funktion  $F: \mathbb{N} \rightarrow P(\Sigma^*)$ .

induktiv definiert

$$L(\alpha) = \begin{cases} \emptyset & \text{falls } \alpha = \emptyset \\ \epsilon & \text{falls } \alpha = \lambda \\ a & \text{falls } \alpha = a \in \Sigma \\ L(\beta) \cup L(\gamma) & \text{falls } \alpha = (\beta + \gamma) \\ L(\beta)L(\gamma) & \text{falls } \alpha = (\beta * \gamma) \\ (L(\beta))^* & \text{falls } \alpha = (\beta^*) \end{cases}$$

DEFINITION

## Pumping Lemma

## Myhill-Nerode-Äquivalenz

DEFINITION

SATZ

**Für eine Sprache  $L$  und ein Wort  $x \in \Sigma^*$  ist  $[x]_L = \{y \in \Sigma^* \mid xR_L y\}$**

**Satz von Myhill-Nerode**

DEFINITION

DEFINITION

**Ein DFA  $M$  heißt reduziert,**

**Sei  $M$  ein DFA. Zwei Zustände  $z, z' \in Z$  heißen erkenntungsäquivalent**

DEFINITION

DEFINITION

**Sei  $M$  ein DFA. Dann ist  $M' = (Z_{\equiv}, \Sigma, [z_0], \sigma', E')$  mit**

**Homomorphismus**

SATZ

SATZ

**surjektiver Homomorphismus**

**Seien  $M_1$  und  $M_2$  reduzierte DFAs mit  $L(M_1) = L(M_2)$ . Sei  $M'_1$  der Quotient von  $M$  bzgl  $\equiv$**

Für eine Sprache  $L \subseteq \Sigma^*$  definieren wir eine binäre Relation  $R_L \subseteq \Sigma^* \times \Sigma^*$  wie folgt: Für alle  $x, y \in \Sigma^*$  setze  $(x, y) \in R_L$  genau dann, wenn  $\forall z \in \Sigma^* : (xz \in L \leftrightarrow yz \in L)$  gilt. Wir schreiben hierfür auch  $xR_Ly$ .

Sei  $L$  eine Sprache.  $L$  ist regulär  $\leftrightarrow \text{index}(R_L) < \infty$  (d.h. nur wenn die Myhill-Nerode-Äquivalenz endliche Klassen hat)

(in Zeichen  $z \equiv z'$ ) wenn für jedes Wort  $w \in \Sigma^*$  gilt:  
 $\hat{\sigma}(z, w) \in E \leftrightarrow \hat{\sigma}(z', w) \in E$

Seien  $M_i$  DFAs (für  $i \in \{1, 2\}$ ) und  $f : Z_1 \rightarrow Z_2$  eine Funktion. Dann ist  $f$  ein Homomorphismus von  $M_1$  auf  $M_2$ , falls gilt:

- $f(l_1) = l_2$
- $f(\sigma_1(z, a)) = \sigma_2(f(z), a)$  für alle  $z \in Z_1$  und  $a \in \Sigma$
- $z \in E_1 \leftrightarrow f(z) \in E_2$  für alle  $z \in Z_1$  (bildet Endzustände aufeinander ab)

- $M_2$  hat wenigstens so viele Zustände wie  $M_1'$
- Hat  $M_2$  genauso viele Zustände wie  $M_1'$ , so sind  $M_2$  und  $M_1'$  bis auf Umbenennung der Zustände identisch (sie sind Isomorph)

Folgerung: Seien  $M_1$  und  $M_2$  reduzierte DFAs mit  $L(M_1) = L(M_2)$ . Seien  $M_1'$  und  $M_2'$  die Quotienten bzgl  $\equiv$ . Dann sind  $M_1'$  und  $M_2'$  isomorph, d.h. für jede reguläre Sprache gibt es (bis auf Umbenennung der Zustände) genau einen minimalen DFA

Um den minimalen DFA zu erhalten bildet man den Quotienten eines beliebigen zur Sprache passenden DFA.

Wenn  $L$  eine reguläre Sprache ist, dann gibt es  $n \leq 1$  derart, dass für alle  $x \in L$  mit  $|x| \geq n$  gilt: es gibt Wörter  $u, v, w \in \Sigma^*$  mit: 1.  $x = uvw$  2.  $|uv| \leq n$  3.  $|v| \geq 1$  4.  $uv^i w \in L$  für alle  $i \geq 0$  Dieses Lemma spricht nicht über Automaten, sondern nur über die Eigenschaften der Sprache. Es ist geeignet, Aussagen über Nicht-Regularität zu machen. Dabei ist es aber nur eine notwendige Bedingung. Es kann nicht genutzt werden, um die Regularität einer Sprache  $L$  zu zeigen.

die Äquivalenzklasse von  $x$ . Ist  $L$  klar, so schreiben wir einfacher  $[x]$ .

wenn es für jeden Zustand  $z \in Z$  ein Wort  $x_z \in \Sigma^*$  gibt mit  $\hat{\sigma}(l, x_z) = z$

- $\sigma'([z], a) = [\sigma(z, a)]$  für  $z \in Z$  und  $a \in \Sigma$  und
- $E' = \{[z] | z \in E\}$   
der Quotient von  $M$  bzgl  $\equiv$

Seien  $M_i$  reduzierte DFAs mit  $L(M_1) = L(M_2)$ . Sei weiter  $M_2'$  der Quotient von  $M_2$  bzgl  $\equiv$ . Dann existiert ein surjektiver Homomorphismus von  $M_1$  auf  $M_2'$  - die Abbildung  $f$  ist surjektiv (auf  $M_2$ ). Und damit ist  $M_2 < M_1$  - die Abbildung  $f$  ist ein Homomorphismus

SATZ

Markierungsalgorithmus

Algorithmus Minimalautomat

SATZ

Minimierungsalgorithmus

Wortproblem

Leerheitsproblem

Endlichkeitsproblem

Schnittproblem

Inklusionsproblem

Äquivalenzproblem

Kontextfreie Sprachen

Eingabe: reduzierter DFA M

Ausgabe: Menge der Paare erkenntnisäquivalenter Zustände  
1. Stelle eine Tabelle aller ungeordneten Zustandspaare  $\{z, z'\}$  mit  $z \neq z'$  auf  
2. Markiere alle Paare  $\{z, z'\}$  mit  $z \in E$  und  $z' \notin E$   
3. Markiere ein beliebiges unmarkiertes Paar  $\{z, z'\}$ , für das es ein  $a \in \Sigma$  gibt, sodass  $\{\sigma(z, a), \sigma(z', a)\}$  bereits markiert ist (falls möglich)  
4. Wiederhole den vorherigen Schritt, bis sich keine Änderung in der Tabelle mehr ergibt

Für einen reduzierten DFA M wird ein Paar  $z, z' \subseteq Z$  mit  $z \neq z'$  genau dann durch den Markierungsalgorithmus markiert werden, wenn  $z \not\equiv z'$

Gilt  $w \in L$  für eine gegebene reguläre Sprache L und  $w \in \Sigma^*$ ?

Eingabe: DFA M und  $w \in \Sigma^*$

Verfahren: Verfolge die Zustandsübergänge von M, die durch die Symbole  $a_1, \dots, a_n$  vorgegeben sind.

Für einen gegebenen reduzierten DFA M markiert der Minimierungsalgorithmus ein  $\{z, z'\}$  ( $z, z' \in Z, z \neq z'$ ) genau dann, wenn  $z \not\equiv z'$

Ist eine gegebene reguläre Sprache L endlich?

Eingabe: NFA M

Verfahren: Sei  $G = (Z, \rightarrow)$  wieder der gerichtete Graph mit  $z \rightarrow z' \leftrightarrow \exists a \in \Sigma : z' \in \sigma(z, a)$ . Dann gilt  $L(M)$  ist genau dann unendlich, wenn es  $z \in Z, z_0 \in S$  und  $z_1 \in E$  gibt mit  $z_0 \rightarrow^* z \rightarrow^+ z \rightarrow^* z_1$ . D.h. z liegt auf einem Zyklus, ist von einem Startzustand aus erreichbar und von z kann ein Endzustand erreicht werden. Dies kann wieder mit dem Algorithmus von Dijkstra entschieden werden.

Gilt  $L = \emptyset$  für eine gegebene reguläre Sprache L?

Eingabe: NFA M

Verfahren: Sei  $G = (Z, \rightarrow)$  der gerichtete Graph mit  $z \rightarrow z' \leftrightarrow \exists a \in \Sigma : z' \in \sigma(z, a)$ . Dann gilt  $L(M) \neq \emptyset$  genau dann, wenn es in dem Graphen G einen Pfad von einem Knoten aus S zu einem Knoten aus E gibt. Dies kann zB mit dem Algorithmus von Dijkstra entschieden werden.

Gilt  $L_1 \subseteq L_2$  für gegebene reguläre  $L_1, L_2$ ?

Eingabe: NFAs  $M_1$  und  $M_2$

Verfahren: Aus  $M_1$  und  $M_2$  kann ein NFA M mit  $L(M) = L(\bar{M}_2) \cap L(M_1)$  konstruieren. Es gilt  $L(M_1) \subseteq L(M_2)$  genau dann, wenn  $L(M) = \emptyset$ .

Gilt  $L_1 \cap L_2 = \emptyset$  für gegebene reguläre  $L_1, L_2$ ?

Eingabe: NFAs  $M_1$  und  $M_2$

Verfahren: Konstruiere aus  $M_1$  und  $M_2$  einen NFA M mit  $L(M) = L(M_1) \cap L(M_2)$ . Teste ob  $L(M) = \emptyset$

bei Kontext-freien Grammatiken haben alle Produktionen die Form  $A \rightarrow w$  mit  $A \in V$  und  $w \in (V \cup \Sigma)^*$ .

Gilt  $L_1 = L_2$  für gegebene reguläre  $L_1, L_2$ ?

Eingabe: NFAs  $M_1$  und  $M_2$

Verfahren 1: es gilt  $L(M_1) = L(M_2)$  genau dann, wenn  $L(M_1) \subseteq L(M_2)$  und  $L(M_2) \subseteq L(M_1)$ .

Verfahren 2: bestimme zu  $M_i (i \in \{1, 2\})$  den äquivalenten minimalen DFA  $N_i$ . Dann gilt  $L(M_1) = L(M_2)$  genau dann, wenn  $N_1$  und  $N_2$  isomorph sind (d.h. sie können durch Umbenennung der Zustände ineinander überführt werden).

DEFINITION

**Ableitungsbaum**

DEFINITION

**Linksableitung**

DEFINITION

**kontextfreie Grammatik**

DEFINITION

**Chomsky Normalform**

SATZ

**Zu jeder kontextfreien Grammatik gibt es eine Grammatik  $G'$  in Chomsky Normalform mit**

**Der Cocke-Younger-Kasami- oder CYK-Algorithmus**

DEFINITION

**Ein Kellerautomat**

DEFINITION

**Ein Konfiguration eines PDA**

DEFINITION

**Seien  $\gamma \in \Gamma^*$ ,  $A_1 B_1, \dots, B_k \in \Gamma$ ,  $w, w' \in \Sigma^*$   
und  $z, z' \in Z$ . Dann gilt  
 $(z, w, A\gamma) \rightarrow (z', w', B_1 \dots B_k \gamma)$  genau dann,  
wenn**

DEFINITION

**Sei  $M$  ein PDA. Dann ist die von  $M$  akzeptierte Sprache:**

Eine Ableitung heißt Linksableitung wenn in jedem Schritt das am weitesten links stehende Nichtterminal ersetzt wird.

Eine kontextfreie Grammatik  $g$  ist in Chomsky Normalform, falls

- alle Produktionen von  $G$  die Form  $A \rightarrow AB$  oder  $A \rightarrow a$  haben
- oder alle Produktionen von  $G$  die Form  $A \rightarrow BC$  oder  $A \rightarrow a$  oder  $S \rightarrow \epsilon$  haben und  $S$  nie auf der rechten Seite einer Produktion vorkommt.

Sei  $G$  kontextfreie Grammatik. Gesucht ist ein Algorithmus mit dessen Hilfe wir entscheiden können, ob ein gegebenes Wort zu  $L(G)$  gehört.

ist ein Tripel  $k \in Z \times \sum^* \times \Gamma^*$  -  $z \in Z$  ist der aktuelle Zustand -  $w \in \sum$  ist der noch zu lesende Teil der Eingabe -  $\gamma \in \Gamma^*$  ist der aktuelle Kellerinhalt. Dabei steht das oberste Kellerzeichen ganz links  
Übergänge zwischen Konfigurationen ergeben sich aus der Überföhrungsfunktion  $\delta$

$$L(M) = \{x \in \sum^* \mid \text{es gibt } z \in Z \text{ mit } (z_0, x, \#)[\dots]^*(z, \epsilon, \epsilon)\}$$

Sei  $G$  eine kontext-freie Grammatik und  $X \in V \cup \sum$ . Ein  $X$ -Ableitungsbaum ist ein gerichteter, geordneter Baum  $T$  mit Wurzel, dessen Knoten mit Elementen von  $V \cup \sum \cup \{\epsilon\}$  beschriftet sind, wobei:

- die Wurzel mit  $X$  beschriftet ist
- Knoten  $v$  mit  $a \in \sum \cup \{\epsilon\}$  beschriftet  $\Rightarrow v$  ist ein Blatt
- Knoten  $v$  mit  $A \in V$  beschriftet und kein Blatt  $\Rightarrow$

- es gibt eine Produktion  $A \rightarrow X_1 \dots X_r$  mit  $X_1 \dots X_r \in \sum \cup V$  ( $r \geq 1$ ) sodass die Nachfolgerknoten von  $v$  mit  $X_1, X_2, \dots, X_r$  beschriftet sind

Eine Kontextfreie Grammatik  $G$  heißt mehrdeutig, wenn es zwei verschiedene vollständige Ableitungsbäume  $T$  und  $T'$  gibt mit  $\alpha(T) = \alpha(T')$ .  
Sonst heißt  $G$  eindeutig, d.h.  $G$  ist eindeutig wenn jedes Wort  $w \in L(G)$  genau eine Ableitung besitzt.

Eine Kontextfreie Sprache heißt inhärent mehrdeutig, wenn jede kontextfreie Grammatik mit  $L = L(G)$   $T$  erhält man, indem man die Beschriftungen der Blätter von links nach rechts betrachtet. Ein Ableitungsbaum ist ein  $S$ -Ableitungsbaum.

- Das Blattwort  $\alpha(T)$  des  $X$ -Ableitungsbaumes  $T$  ist mehrdeutig
- ein  $X$ -Ableitungsbaum ist vollständig, wenn seine Blätter mit Elementen von  $\sum \cup \{\epsilon\}$  beschriftet sind.

$$L(G) = L(G')$$

$M$  ist ein 6-Tupel  $M = (Z, \sum, \Gamma, z_0, \delta, \#)$ , wobei

- $Z$  die endliche Menge der Zustände
- $\sum$  das Eingabealphabet
- $\Gamma$  das Kelleralphabet
- $z_0 \in Z$  der Startzustand
- $\delta : Z \times (\sum \cup \{\epsilon\}) \times \Gamma \rightarrow P_c Z \times \Gamma^*$  die Überföhrungsfunktion

$$\text{es } a \in \sum \cup \{\epsilon\} \text{ gibt mit } w = aw' \text{ und } (z', B_1 \dots B_k) \in \delta(z, a, A)$$

DEFINITION

Sei  $M$  ein PDA. Dann ist die von  $M$  akzeptierte Sprache

SATZ

aus einer kontextfreien Grammatik  $G$  kann eine kontextfreie Grammatik  $G'$  in Greibach Normalform berechnet werden mit

DEFINITION

PDA mit Endzuständen

DEFINITION

ein deterministischer Kellerautomat oder DPDA ist ein PDAE  $M$ ,

SATZ

Ist  $L \subseteq \Sigma^*$  deterministisch kontextfrei,

DEFINITION

eine kontextfreie Grammatik  $G$  ist in Greibach Normalform

SATZ

Sei  $L$  eine Sprache. Dann sind äquivalent

DEFINITION

Sei  $M$  ein PDAE. Die von  $M$  akzeptierte Sprache ist

DEFINITION

eine Sprache  $L$  ist deterministisch kontextfrei,

SATZ

falls alle Produktionen aus P folgende Form haben:  
 $A \rightarrow aB_1B_2\dots B_k$ , mit  $k \in \mathbb{N}$ ,  $A, B_1, \dots, B_k \in V$  und  
 $a \in \Sigma$  Die Greibach Normalform garantiert, dass bei  
 jedem Ableitungsschritt genau ein Alphabetsymbol  
 entsteht.

$$L(M) = \{x \in \Sigma^* \mid \text{es gibt } z \in Z \text{ mit } (z_0, x, \#) \vdash^* (z, \epsilon, \epsilon)\}$$

- L ist kontextfrei
- es gibt einen PDA M mit  $L(M) = L$
- es gibt einen PDA M mit nur einem Zustand und  $L(M) = L$ . Gilt  $\epsilon \notin L$ , so sind diese Aussagen äquivalent zu
- es gibt einen PDA M mit nur einem Zustand und ohne eine  $\epsilon$ -Transitionen, so dass  $L(M) = L$  gilt

$$L(G') = L(G) \setminus \{\epsilon\}.$$

- Jede kontextfreie Sprache L ist Sprache eines PDA M mit nur einem Zustand. Gilt  $\epsilon \notin L$ , so werden keine  $\epsilon$ -Transitionen benötigt
- Ist M ein PDA, so ist  $L(M)$  kontextfrei

$$L(M) = \{w \in \Sigma^* \mid \text{es gibt } e \in E \text{ und } \gamma \in \Gamma^* \text{ mit } (\iota, w, \#) \vdash^* (e, \epsilon, \gamma)\}$$

Ein Kellerautomat mit Endzuständen oder PDAE ist ein 7-Tupel M, wobei  $(Z, \Sigma, \Gamma, \iota, \delta, \#)$  ein PDA und  $E \subseteq Z$  eine Menge von Endzuständen ist

wenn es einen deterministischen Kellerautomaten M gibt mit  $L(M) = L$

so dass für alle  $z \in Z, a \in \Sigma, A \in \Gamma$  gilt:  
 $|\delta(z, a, A)| + |\delta(z, \epsilon, A)| \leq 1.$

aus einem DPDA M kann ein DPDA M' berechnet werden mit  $L(M') = \Sigma^* \setminus L(M)$

so auch  $\Sigma^* \setminus L$

DEFINITION

**das Lemma von Ogden (William Ogden)**

DEFINITION

**Wortproblem für eine kontextfreie Sprache  $L$**

DEFINITION

**Uniformes Wortproblem für kontextfreie Sprachen**

DEFINITION

**Leerheitsproblem für kontextfreie Sprachen**

DEFINITION

**Endlichkeitsproblem für kontextfreie Sprachen**

DEFINITION

**Intuitiver Berechenbarkeitsbegriff**

DEFINITION

**Ein Loop-Programm ist von der Form**

DEFINITION

DEFINITION

DEFINITION

Gegeben  $w \in \Sigma^*$ . Gilt  $w \in L$ ? Ist die kontextfreie Sprache  $L$  durch eine kontextfreie Grammatik in Chomsky-Normalform gegeben, so kann das Wortproblem mit dem CYK-Algorithmus in Zeit  $O(|w|^3)$  gelöst werden. Ist  $L$  durch einen deterministischen PDA gegeben, so kann das Wortproblem für  $L$  sogar in Zeit  $O(n)$  gelöst werden.

Gegeben eine kontextfreie Grammatik  $G = (V, \Sigma, P, S)$ . Gilt  $L(G) = \emptyset$ ? Lösung: Sei  $W = \{A \in V \mid \exists w \in \Sigma^* : A \Rightarrow_G^* w\}$  die Menge aller produktiven Nichtterminale. Dann gilt  $L(G) \neq \emptyset \leftrightarrow S \in W$ . Berechnung von  $W$ :  $W_0 := \{A \in V \mid \exists w \in \Sigma^* : (A \rightarrow w) \in P\}$

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der  $f$  berechnet, d.h.

- das Verfahren erhält  $(n_1, \dots, n_k)$  als Eingabe,
- terminiert nach endlich vielen Schritten
- und gibt  $f(n_1, \dots, n_k)$  aus.

Die modifizierte Subtraktion  $\div$  ist definiert durch  $\div : \mathbb{N}^2 \rightarrow \mathbb{N} : (m, n) \rightarrow \max(0, m - n)$

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  (mit  $k \geq 0$ ) heißt loop-berechenbar, falls es ein  $l \geq k$  und ein Loop-Programm  $P$ , in dem höchstens die Variablen  $\forall n_1, \dots, n_k \in \mathbb{N} : f(n_1, \dots, n_k) = \pi_1^l([P]_l(n_1, \dots, n_k, 0, \dots, 0))$ .  
Loop-Vermutung: Eine Funktion  $\mathbb{N}^k \rightarrow \mathbb{N}$  mit  $k \geq 0$  ist genau dann intuitiv berechenbar, wenn sie loop-berechenbar ist.

Wenn  $L$  eine kontextfreie Sprache ist, dann gibt es  $n \geq 1$  derart, dass für alle  $z \in L$ , in denen  $n$  Positionen markiert sind, gilt: es gibt Wörter  $u, v, w, x, y \in \Sigma^*$  mit

- $z = uvwxy$
- $v$  oder  $x$  enthält wenigstens eine der Markierungen oder
- $uv^iwx^iy \in L$  für alle  $i \geq 0$

Gegeben kontextfreie Grammatik  $G$  und Wort  $w \in \Sigma^*$ . Gilt  $w \in L(G)$ ? Lösung:

- berechne kontextfreie Grammatik  $G'$  in Chomsky Normalform mit  $L(G) = L(G')$
- Wende CYK-Algorithmus auf die Frage  $w \in L(G')$  an

Gegeben eine kontextfreie Grammatik  $G$ . Ist  $L(G)$  endlich? O.E. können wir annehmen, daß  $G$  in Chomsky-Normalform ist. Wir definieren einen Graphen  $(W, E)$  auf der Menge der produktiven Nichtterminale mit folgender Kantenrelation:  $E = \{(A, B) \in W \times W \mid \exists C \in W : (A \rightarrow BC) \in P \text{ oder } (A \rightarrow CB) \in P\}$  Beobachtung:  $(A, B) \in E$  gilt genau dann, wenn es einen vollständigen  $A$ -Ableitungsbaum gibt, so daß  $B$  ein Kind der Wurzel beschriftet.

- $x_i := c, x_i := x_j + c, x_i := x_j \div c$  mit  $c \in \{0, 1\}$  und  $i, j$  (Wertzuweisung) oder
- $P_1; P_2$ , wobei  $P_1$  und  $P_2$  Loop-Programme sind (sequentielle Komposition) oder
- loop  $x_i$  do  $P$  end, wobei  $P$  ein Loop-Programm ist und  $i_1$ .

Für jedes Loop-Programm  $P$ , in dem keine Variable  $x_i$  mit  $i > k$  vorkommt, definieren wir zunächst eine Funktion  $[[P]]_k : \mathbb{N}^k \rightarrow \mathbb{N}^k$  durch Induktion über den Aufbau von  $P$

DEFINITION

DEFINITION

### Hilberts Vermutung (1926)

DEFINITION

DEFINITION

**Die primitiv rekursiven Funktionen  
sind induktiv wie folgt definiert**

DEFINITION

DEFINITION

SATZ

DEFINITION

**Die Ackermann Funktion ist nicht  
berechenbar**

**Ein While Programm ist von der Form**

DEFINITION

DEFINITION

Eine Funktion  $\mathbb{N}^k \rightarrow \mathbb{N}$  mit  $k \geq 0$  ist genau dann intuitiv berechenbar, wenn sie primitiv rekursiv ist.

Seien  $k \geq 0, \mathbb{N}^k \rightarrow \mathbb{N}$  und  $h : \mathbb{N}^{k+2}$ . Die Funktion  $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  mit  $f(0, n_2, \dots, n_{k+2}) = g(n_2, \dots, n_{k+1})$  und  $f(m+1, n_2, \dots, n_{k+1}) = h(f(m, n_2, \dots, n_{k+1}), m, n_2, \dots, n_{k+1})$  entsteht aus  $g$  und  $h$  mittels Rekursion.

Seien  $f, g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  Funktionen mit

•

$$g(m, \bar{n}) = \begin{cases} 1 & \text{falls } \exists i \leq m : f(i, \bar{n}) \geq 1 \\ 0 & \text{sonst} \end{cases}$$

- für alle  $\bar{n} \in \mathbb{N}^k$ . Wir sagen,  $g$  geht durch den beschränkten Existenzquantor aus  $f$  hervor.

Sei  $P$  Loop-Programm mit Variablen  $x_1, x_2, \dots, x_n$ .

Für Anfangswerte  $(n_i)$  seien  $(n'_i)$  die Werte der Variablen bei Programmende.

$$f_p : \mathbb{N} \rightarrow \mathbb{N} : n \rightarrow \max \left\{ \sum_{1 \leq i \leq l} n'_i \mid \sum_{1 \leq i \leq l} n_i \leq n \right\}$$

- $x_i = c; x_i = x_j + c; x_i = x_j - c$  mit  $c \in \{0, 1\}$  und  $i, j \geq 1$  (Wertzuweisung) oder
- $P_1; P_2$ , wobei  $P_1$  und  $P_2$  bereits While Programme sind (sequentielle Komposition) oder
- $\text{while } x_i \neq 0 \text{ do } P \text{ end}$ , wobei  $P$  ein While Programm ist und  $i \geq 1$ .

wie bei Loop Programmen definieren wir zunächst für jedes While Programm  $P$  in dem keine Variable  $x_i$  mit  $i > k$  vorkommt induktiv eine partielle Abbildung  $[[P]]_k : \mathbb{N}^k \rightarrow \mathbb{N}^k$ . Hierfür sei  $\bar{n} \in \mathbb{N}^k$

- $[[x_i = c]]_k(n_1, \dots, n_k) = (m_1, \dots, m_k)$  genau dann, wenn  $m_i = c$  und  $m_l = n_l$  für  $l \neq i$
- $[[x_i = x_j \pm c]]_k(n_1, \dots, n_k) = (m_1, \dots, m_k)$  genau dann, wenn  $m_i = n_j \pm c$  und  $m_l = n_l$  für  $l \neq i$
- $[[P_1; P_2]]_k(\bar{n})$  ist genau dann definiert, wenn  $\bar{m} = [[P_1]]_k(\bar{n}) \in \mathbb{N}^k$  und  $[[P_2]]_k(\bar{m})$  definiert sind. In diesem Falle gilt  $[[P_1; P_2]]_k(\bar{n}) = [[P_2]]_k([[P_1]]_k(\bar{n}))$  sonst undefiniert.

- Alle konstanten Funktionen der Form  $k_c : \mathbb{N}^0 \rightarrow \mathbb{N} : () \rightarrow c$  (für ein festes  $c \in \mathbb{N}$ ) sind primitiv rekursiv.

- Alle Projektionen der Form  $\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N} : (n_1, \dots, n_k) \rightarrow n_i$  (mit  $1 \geq i \geq k$ ) sind primitiv rekursiv.

- Die Nachfolgerfunktion  $s : \mathbb{N} \rightarrow \mathbb{N} : n \rightarrow n + 1$  ist primitiv rekursiv.

- Wenn  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  und  $g_1, \dots, g_l : \mathbb{N}^l \rightarrow \mathbb{N}$  (mit  $k, l \geq 0$ ) primitiv rekursiv sind, dann ist auch die Funktion  $f(g_1, \dots, g_k) : \mathbb{N}^l \rightarrow \mathbb{N}$  primitiv rekursiv (Substitution).

- Sind  $g : \mathbb{N}^k \rightarrow \mathbb{N}$  und  $h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$  primitiv rekursiv (mit  $k \geq 0$ ) und entsteht  $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  aus  $g$  und  $h$  mittels Rekursion, so ist auch  $f$  primitiv rekursiv (Rekursion).  
Die Funktion  $ack_k : \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $ack(x, y, j) = ack_x(y)$  heißt Ackermann Funktion.

Beweis indirekt: Angenommen  $P$  wäre Loop-Programm, das  $ack$  berechnet. Nach Beschränkungslemma existiert  $k \in \mathbb{N}$  mit

$$f_p(m) < ack_k(m), \text{ damit } ack_k(k) \leq f_p(2k) < ack_k(2k) \text{ im Widerspruch zum Monotonielemma.}$$

Seien  $r \in \mathbb{N}$  und  $D \subseteq \mathbb{N}^r$ . Eine Funktion  $f : D \rightarrow \mathbb{N}$  heißt partielle Funktion von  $\mathbb{N}^r$  nach  $\mathbb{N}$ . Wir schreiben hierfür  $f : \mathbb{N}^r \rightarrow \mathbb{N}$ .

DEFINITION

DEFINITION

## Gödels Vermutung

DEFINITION

DEFINITION

$\mu$ -rekursive Funktion

Die Klasse der  $\mu$ -rekursiven Funktionen  
ist rekursiv definiert

DEFINITION

DEFINITION

Ein GoTo Programm

DEFINITION

DEFINITION

Eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  heißt  
Goto berechenbar,

DEFINITION

DEFINITION

Eine Turingmaschine (TM)

Eine partielle Funktion  $\mathbb{N}^k \rightarrow \mathbb{N}$  ist genau dann intuitiv berechenbar, wenn sie  $\mu$ -rekursiv ist.

- Alle konstanten Funktionen  $k_m : \mathbb{N}^0 \rightarrow \mathbb{N} : () \rightarrow m$ , alle Projektionen  $\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N} : (n_1, \dots, n_k) \rightarrow n_i$  und die Nachfolgerfunktion  $s : \mathbb{N} \rightarrow \mathbb{N} : n \rightarrow n + 1$  sind  $\mu$ -rekursiv.
- Sind  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  und  $g_1, \dots, g_k : \mathbb{N}^r \rightarrow \mathbb{N}$   $\mu$ -rekursiv, so auch  $F : \mathbb{N}^r \rightarrow \mathbb{N}$  mit  $F(n) = f(g_1(\bar{n}), \dots, g_k(\bar{n}))$  (wobei  $F(n)$  genau dann definiert ist, wenn  $g_i(n)$  für alle  $i$  definiert ist und wenn  $f$  auf diesen Werten definiert ist).
- Jede partielle Funktion  $f$ , die durch Rekursion aus  $\mu$ -rekursiven Funktionen entsteht, ist  $\mu$ -rekursiv.

Sei  $P = A_1; A_2; \dots; A_m$  ein Goto-Programm, in dem keine Variable  $x_i$  mit  $i > k$  vorkommt. Eine Konfiguration von  $P$  ist ein  $(k+1)$ -Tupel  $(n_1, n_2, \dots, n_k, p) \in \mathbb{N}^k \times \{0, 1, \dots, m\}$ , wobei  $n_i$  die Belegung der Variablen  $x_i$  und  $p$  den Wert des Programmzählers beschreibt.

falls es ein  $l \geq k$  und ein Goto Programm  $P$ , in dem keine Variable  $x_i$  mit  $i > l$  vorkommt, gibt, sodass für alle  $\bar{n} \in \mathbb{N}^k$  gilt:

- $f(\bar{n})$  definiert  $\leftrightarrow [[P]]_l(\bar{n}, 0, \dots, 0)$  definiert
- Falls  $f(\bar{n})$  definiert ist, gilt  $f(\bar{n}) = \pi_1^l([[P]]_l(\bar{n}, 0, \dots, 0))$

ist ein 7-Tupel  $M = (Z, \sum, \Phi, \delta, z_0, \square, E)$ , wobei

- $\sum$  das Eingabealphabet
- $\Phi$  mit  $\Phi \supseteq \sum$  und  $\Phi \cap Z \neq \emptyset$  das Arbeits- oder Bandalphabet,
- $z_0 \in Z$  der Startzustand,
- $\delta : Z \times \Phi \rightarrow (Z \times \Phi \times \{L, N, R\})$  die Überföhrungsfunktion
- $\square \in \Phi / \sum$  das Leerzeichen oder Blank und
- $E \subseteq Z$  die Menge der Endzustände ist

Eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  heißt while Berechenbar, falls es ein  $l \geq k$  und ein While Programm  $P$ , in dem höchstens die Variablen  $x_1, \dots, x_l$  vorkommen, gibt, sodass für alle  $n_1, \dots, n_k \in \mathbb{N}$  gilt:

- $f(n_1, \dots, n_k)$  definiert  $\leftrightarrow [[P]]_l(n_1, \dots, n_k, 0, \dots, 0)$  definiert
- Falls  $f(n_1, \dots, n_k)$  definiert ist, gilt  $f(n_1, \dots, n_k) = \pi_1^l([[P]]_l(n_1, \dots, n_k, 0, \dots, 0))$ .

Sei  $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  eine partielle Funktion Dann ist  $\mu f : \mathbb{N}^k \rightarrow \mathbb{N}$  definiert durch  $\mu f(n_1, \dots, n_k) = \min\{m \mid f(m, n_1, \dots, n_k) = 0 \text{ und } \forall x < m : f(x, n_1, \dots, n_k) \text{ definiert}\}$ . Dabei ist  $\min \emptyset$  undefiniert. Wir sagen, dass die Funktion  $\mu f$  aus  $f$  durch den  $\mu$ -Operator hervorgeht.

ist eine endliche nichtleere Datei  $P = A_1; A_2; \dots; A_m$  von Anweisungen  $A_i$  der folgenden Form:

- $x_i = c, x_i = x_j + c, x_i = x_j - c$  mit  $c \in \{0, 1\}$  und  $i, j \geq 1$
- goto  $l$  mit  $0 \leq l \leq m$  (unbedingter Sprung)
- if  $x_i = 0$  then  $l$  mit  $i \geq 1$  und  $0 \leq l \leq m$  (bedingter Sprung)

$[[P]]_k(\bar{n})$  ist definiert, falls es  $\bar{n}' \in \mathbb{N}^k$  gibt mit  $(\bar{n}, 1) \vdash_P^* (\bar{n}', 0)$ . In diesem Fall gilt  $[[P]]_k(\bar{n}) = \bar{n}'$

Seien  $P = A_1; A_2; \dots; A_m$ ; ein GoTo Programm und  $(\bar{n}, p), (\bar{n}', p')$  zwei Konfigurationen. Wir setzen  $(\bar{n}, p) \vdash_P (\bar{n}', p')$ , falls  $p > 0$  und eine der folgenden Bedingungen gilt:

- $A_p = (x_i = c), n'_i = c, n'_i = n_i$  für  $l \neq i$  und  $p' = p + 1$
- $A_p = (x_i = x_j + c), n'_i = n_j + c, n'_i = n_i$  für  $l \neq i$  und  $p' = p + 1$
- $A_p = (x_i = x_j - c), n'_i = n_j - c, n'_i = n_i$  für  $l \neq i$  und  $p' = p + 1$
- $A_p = (gotol), \bar{n}' = \bar{n}$  und  $p' = l$

DEFINITION

DEFINITION

DEFINITION

DEFINITION

DEFINITION

DEFINITION

## Mehrband Turingmaschine

SATZ

SATZ

**Zu jeder Mehrband Turingmaschine  $M$   
gibt es**

SATZ

DEFINITION

**Eine Sprache  $L \subseteq \Sigma^*$  hei entscheidbar,**

Sei  $M = (Z, \Sigma, \Phi, \delta, z_o, \square, E)$  eine TM und  $k$  eine Konfiguration. Dann heißt  $k$  Haltekonfiguration falls für alle Konfigurationen  $k'$  gilt:  $k \vdash_M k' \Rightarrow k = k'$  (d.h. ist  $k = uzav$ , so gilt  $\delta(z, a) = (z, a, N)$ ). Die Haltekonfiguration  $k$  ist akzeptierend, wenn zusätzlich  $k \in \square^* E \Sigma^* \square^*$  gilt.

Eine partielle Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  heißt Turing berechenbar, wenn es eine TM  $M$  gibt mit  $g_M = f$ .

- Eine Mehrband-Turingmaschine besitzt  $k (k \geq 1)$  Bänder mit  $k$  unabhängigen Köpfen, aber nur eine Steuereinheit.
- Aussehen der Übergangsfunktion:  $\delta : Z \times \Phi^k \rightarrow (Z \times \Phi^k \times \{L, N, R\}^k)$  (ein Zustand,  $k$  Bandsymbole,  $k$  Bewegungen)
- Die Ein- und Ausgabe stehen jeweils auf dem ersten Band. Zu Beginn und am Ende (in einer akzeptierenden Haltekonfiguration) sind die restlichen Bänder leer.

Sei  $g : \Sigma^* \rightarrow \Sigma^*$  eine Turing-berechenbare partielle Funktion. Dann wird  $g$  von einer TM  $M$  berechnet, für die gilt:  $\forall x \in \Sigma^* \forall k$  Haltekonfiguration:  $z_o x \square \vdash_M^* k \Rightarrow k \in \square^* E \Sigma^* \square^*$ .

falls die charakteristische Funktion von  $L$ , d.h. die Funktion  $\chi_L : \Sigma^* \rightarrow \{0, 1\}$  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.

Eine Konfiguration einer Turingmaschine ist ein Wort  $k \in \Phi^* Z \Phi^+$  Bedeutung:  $k = uzv$

- $u \in \Phi^*$  ist Abschnitt des Bandes vor Kopfposition der bereits besucht wurde
- $z \in Z$  ist aktueller Zustand
- $c \in \Phi^+$  ist Abschnitt des Bandes ab Kopfposition, der besucht wurde oder im Bereich des Eingabewortes liegt.

Sei  $M = (Z, \Sigma, \Phi, \delta, z_o, \square, E)$  eine TM. Die von  $M$  berechnete partielle Funktion  $f_M : \Sigma^* \rightarrow \Sigma^*$  erfüllt für alle  $x, y \in \Sigma^* : f_M(x) = y \Leftrightarrow \exists z_e \in E, i, j \in \mathbb{N} : z_o x \square \vdash_M^* \square^i z_e y \square^j$  und  $\square^i z_e y \square^j$  ist Haltekonfiguration.

Sei  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  eine partielle Funktion. Definiere eine partielle Funktion  $F : \{0, 1, \#\}^* \rightarrow \{0, 1, \#\}^*$  durch  $F(w) =$

$\begin{cases} bin(f(n_1, \dots, n_k)) & \text{falls } w = bin(n_1) \# bin(n_2) \# \dots \# bin(n_k) \text{ und} \\ \text{undefiniert} & \text{textsonst} \end{cases}$

Dann heißt  $f$  Turing berechenbar, wenn  $F$  Turing berechenbar ist. (Für  $n \in \mathbb{N}$  sei  $bin(n)$  die Binärdarstellung der Zahl  $n$ )

eine (Einband) Turingmaschine  $M'$  die dieselbe Funktion löst Beweis:

- Simulation mittels Einband-Turingmaschine durch Erweiterung des Alphabets: Wir fassen die übereinanderliegenden Bandeinträge zu einem Feld zusammen und markieren die Kopfpositionen auf jedem Band durch  $*$ . Neues Bandalphabet:  $\Phi' = \Sigma \uplus \{\square\} \uplus (\Phi \times \{*, \diamond\})^k$
- Alphabetsymbol der Form  $(a, *, b, \diamond, c, *, \dots) \in (\Phi \times \{*, \diamond\})^k$  bedeutet: 1. und 3. Kopf anwesend (\* Kopf anwesend,  $\diamond$  Kopf nicht anwesend)

Sind  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  und  $g_1, g_2, \dots, g_k : \mathbb{N}^l \rightarrow \mathbb{N}$  Turing berechenbar, so auch die partielle Funktion  $f(g_1, g_2, \dots, g_k) : \mathbb{N}^l \rightarrow \mathbb{N}$

DEFINITION

das allgemeine Halteproblem ist die  
Sprache

SATZ

SATZ

SATZ

SATZ

DEFINITION

das spezielle Halteproblem ist die  
Sprache

DEFINITION

SATZ

DEFINITION

SATZ

Sei  $L \subseteq \Sigma^*$  eine nichtleere Sprache.  
Dann sind äquivalent

$K = \{w \in L_{TM} \mid M_w \text{ angesetzt auf } w \text{ hält}\}$

$H = \{w\#x \mid w \in L_{TM}, x \in \{0,1\}^*, M_w \text{ angesetzt auf } x \text{ hält}\}$

Seien  $A \subseteq \Sigma^*, B \subseteq \Phi^*$ . Eine Reduktion von A auf B ist eine totale und berechenbare Funktion  $f : \Sigma^* \rightarrow \Phi^*$ , so dass für alle  $w \in \Sigma^*$  gilt:  $w \in A \leftrightarrow f(w) \in B$ . A heißt auf B reduzierbar (in Zeichen  $A \leq B$ ), falls es eine Reduktion von A auf B gibt.

Das spezielle Halteproblem ist unentscheidbar

Das Halteproblem auf leerem Band ist unentscheidbar

Das allgemeine Halteproblem ist unentscheidbar

Eine Sprache  $L \subseteq \Sigma^*$  heißt semi-entscheidbar, falls die "halbe" charakteristische Funktion von L, d.h. die partielle Funktion  $X'_L : \Sigma^* \rightarrow \{1\}$  mit  $x'_L = \begin{cases} 1 & \text{falls } w \in L \\ \text{undef.} & \text{falls } w \notin L \end{cases}$  berechenbar ist.

Satz von Rice: Sei  $R$  die Klasse aller Turing-berechenbaren Funktionen  $\{0,1\}^* \rightarrow \{0,1\}^*$ ,  $\Omega$  die nirgendwo definierte Funktion und sei  $S \subseteq R$  mit  $\Omega \in S$  und  $\Omega \notin R$ . Dann ist die Sprache  $C(S) = \{w \in L_{TM} \mid \phi_w \in S\}$  unentscheidbar.

- L ist semi-entscheidbar
- L wird von einer Turing-Maschine akzeptiert
- L ist vom Typ 0 (d.h. von einer Grammatik erzeugt)
- L ist Bild einer berechenbaren partiellen Funktion  $\Sigma^* \rightarrow \Sigma^*$
- L ist Bild einer berechenbaren totalen Funktion  $\Sigma^* \rightarrow \Sigma^*$
- L ist rekursiv aufzählbar

Ein Problem  $L \subseteq \Sigma^*$  ist genau dann entscheidbar, wenn sowohl L als auch  $\bar{L} = \Sigma^* \setminus L$  semi-entscheidbar sind. 1.  $w \in L$ , dann existiert  $t \in \mathbb{N}$ , so dass  $M_L$  nach t Schritten terminiert. Wegen  $w \notin \bar{L}$  terminiert  $M_{\bar{L}}$  niemals. 2.  $w \notin L$ , dann existiert  $t \in \mathbb{N}$ , so dass  $M_{\bar{L}}$  nach t Schritten terminiert. Wegen  $w \in L$  terminiert  $M_L$  niemals.

Dieses letzte Argument heißt mitunter "Schwalbenschwanz-Argument".

DEFINITION

DEFINITION

DEFINITION

SATZ

SATZ

SATZ

SATZ

DEFINITION

SATZ

SATZ

Eine Sprache  $L \subseteq \Sigma^*$  heißt rekursiv aufzählbar, falls  $L \neq \emptyset$  oder es eine totale und berechenbare Funktion  $f: \mathbb{N} \rightarrow \Sigma^*$  gibt mit  $L = \{f(n) | n \in \mathbb{N}\} = \{f(0), f(1), f(2), \dots\}$ .

Sei  $M$  eine Turing Maschine. Die von  $M$  akzeptierte Sprache ist  $L(M) = \{w \in \Sigma^* | \text{es gibt akzept. Haltekonf. mit } z_0 w \sqsupset \vdash_M^* k\}$ .

Es gibt eine universelle Turing Maschine Beweis: eine Turing Maschine mit drei Bändern. - 1.Band: Kode  $w$  der zu simulierenden Turing Maschine  $M_w$  - 2.Band: aktueller Zustand der zu simulierenden Turing Maschine  $M_w$  - 3.Band: augenblicklicher Bandinhalt der Turing Maschine  $M_w$  1. Initialisierung: auf 1.Band steht  $w000x$  mit  $w \in L_{TM}$ . Kopiere  $x$  auf 3.Band und lösche  $000x$  auf erstem, schreibe  $010$  auf 2.Band 2. Simulation: stehen auf 2.Band  $01^{i+1}0$  und auf 3. an Kopfposition  $j$ , so suche auf 1.Band Anweisung  $(z_{i'}, a_{j'}, y) = \delta(z_i, a_j)$  und schreibe  $01^{i'+1}0$  auf 2.Band; ersetze  $j$  an Kopfposition auf 3.Band durch  $j'$ ; bewege 3.Kopf entsprechend  $y$  nah rechts, links oder aber auch nicht. 3. Aufräumen: bei Erreichen einer akzeptierenden Haltekonfiguration auf 3.Band, es gibt eine Grammatik  $G$ , deren Wortproblem  $L(G)$  unentscheidbar ist. Folgerung: es gibt eine Typ-0 Sprache, die nicht vom Typ 1 ist.

Eine Turing Maschine  $U$  heißt universelle Turing Maschine, wenn sie die folgende partielle Funktion berechnet.  $\{0, 1\}^* \rightarrow \{0, 1\}^*$

$$y \rightarrow \begin{cases} \phi_w(x) & \text{falls } y = w000x, w \in L_{TM}, x \in \{0, 1\}^* \\ \text{undef.} & \text{sonst} \end{cases}$$

-  $U$  hält bei Eingabe  $w000x$  genau dann, wenn  $M_w$  bei Eingabe  $x$  hält -  $U$  akzeptiert  $w000x$  genau dann, wenn  $M_w$  das Wort  $x$  akzeptiert

das spezielle Halteproblem  $K = \{w \in L_{TM} | M_w \text{ angesetzt auf } w \text{ hält}\}$  ist semi-entscheidbar.

1. Ein Korrespondenzsystem ist eine endliche Folge von Paaren  $K = ((x_1, y_1), (x_2, y_2), \dots, (x_k, y_k))$  mit  $x_i, y_i \in \Sigma^+$  für alle  $1 \leq i \leq k$  (dabei ist  $\Sigma$  ein beliebiges Alphabet) 2. Eine Lösung von  $K$  ist eine endliche Folge von Indizes  $i_1, i_2, \dots, i_n \in \{1, 2, \dots, k\}$  mit  $n \geq 1$  und  $x_{i_1}x_{i_2}\dots x_{i_n} = y_{i_1}y_{i_2}\dots y_{i_n}$ . 3. MPCP ("modifiziertes PCP") ist die Menge der Korrespondenzsysteme, die eine Lösung mit  $i_1 = 1$  besitzen 4. PCP ist die Menge der Korrespondenzsysteme, die eine Lösung besitzen

das allgemeine Wortproblem  $A = \{(G, w) | G \text{ ist Grammatik mit } w \in L(G)\}$  ist unentscheidbar.

PCP ist semi-entscheidbar.

Satz (Emil Post, 1947): PCP ist unentscheidbar. (T. Neary 2015: 5 Paare reichen hierfür.)

SATZ

SATZ

SATZ

DEFINITION

**Church-Turing These**

DEFINITION

DEFINITION

**Unentscheidbarkeit**

**Intuitiver Effizienzbegriff**

DEFINITION

DEFINITION

**Deterministische Zeitklassen**

**REACH**

DEFINITION

SATZ

**Euler-Kreise**

**Satz (Euler 1707-1783, 1736)**

Satz (Stearns 1967): Das Regularitätsproblem für DPDAs  $Reg_{DPDA} = \{P \mid P \text{ DPDA mit } L(P) \text{ regulär}\}$  ist entscheidbar.

Das Regularitätsproblem für PDAs  $Reg_{PDA} = \{P \mid P \text{ PDA mit } L(P) \text{ regulär}\}$  ist nicht semi-entscheidbar.

Die Funktionen, die durch Turingmaschinen bzw. While/Goto-Programme berechnet werden können, sind genau die intuitiv berechenbaren Funktionen.

Das Schnittproblem für DPDAs  $Schn_{DPDA} = \{(P_1, P_2) \mid P_1, P_2 \text{ DPDAs mit } L(P_1) \cap L(P_2) = \emptyset\}$  ist nicht semi-entscheidbar.

Das Wortproblem einer Sprache L ist effizient entscheidbar, wenn es einen Algorithmus gibt, der die Antwort auf die Frage "Gehört das Wort w zu L?" mit geringen Ressourcen (Zeit, Speicherplatz) bestimmt. "mit geringen Ressourcen" heißt hier, daß die benötigten Ressourcen nur moderat mit der Eingabelänge  $|w|$  wachsen.

Probleme, die nicht durch Turing-Maschinen gelöst werden können, sind damit prinzipiell unlösbar (wenn auch u.U. semi-entscheidbar). Beispiele:

- die verschiedenen Versionen des Halteproblems
- Posts Korrespondenzproblem
- das Schnitt- und verwandte Probleme über kontextfreie Sprachen

ist die Menge der gerichteten Graphen mit zwei ausgezeichneten Knoten s und t, in denen es einen Pfad von s nach t gibt. REACH ist in P. (Beweis: z.B. mit Dijkstras Algorithmus)

Sei  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine monotone Funktion. Die Klasse  $TIME(f)$  besteht aus allen Sprachen L, für die es eine 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 eine akzeptierende Haltekonfiguration (und gibt 0 oder 1 aus, je nachdem ob  $w \notin L$  oder  $w \in L$  gilt).

Ein Graph  $(V, E)$  enthält einen Eulerkreis genau dann, wenn er höchstens eine Zusammenhangskomponente mit  $> 1$  Knoten enthält und jeder Knoten geraden Grad hat (d.h. jeder Knoten hat eine gerade Anzahl von Nachbarn).  
Folgerung: EC ist in P, denn die genannten Bedingungen lassen sich in polynomieller Zeit prüfen.  
Die erweiterte Church-Turing These: P umfaßt die Klasse der effizient lösbaren Probleme.

EC ist die Menge der ungerichteten Graphen, die einen Eulerkreis (d.h. einen Kreis, der jede Kante genau einmal durchläuft) enthalten.

DEFINITION

## Deterministische Platzklassen

DEFINITION

## Definition

DEFINITION

## SAT

DEFINITION

## Hamilton-Kreise HC

DEFINITION

## 3-Färbbarkeit 3C

DEFINITION

Sei  $M$  NTM. Die von  $M$  akzeptierte Sprache ist

SATZ

## Determinisierbarkeit von NTM

DEFINITION

## Nichtdeterministische Zeitklassen

DEFINITION

## Nichtdeterministische Platzklassen

SATZ

## Satz von Kuroda (1964)

$$PSPACE = \bigcup_{f \in Poly} SPACE(f)$$

$$EXSPACE = \bigcup_{f \in Poly} SPACE(2^f)$$

$$2EXSPACE = \bigcup_{f \in Poly} SPACE(2^{2^f}) \dots$$

ist die Menge der ungerichteten Graphen, die einen Hamiltonkreis (d.h. einen Kreis, der jeden Knoten genau einmal besucht) enthalten.

Sei  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine monotone Funktion. Die Klasse  $SPACE(f)$  besteht aus allen Sprachen  $L$ , für die es eine Turingmaschine  $M$  gibt mit:

- $M$  berechnet die charakteristische Funktion von  $L$ .
- Für jede Eingabe  $w \in \Sigma^*$  hat jede von der Startkonfiguration  $z_0 w \square$  aus erreichbare Konfiguration höchstens die Länge  $f(|w|)$ .

ist die Menge der erfüllbaren aussagenlogischen Formeln.

Beobachtung: SAT  $\in$  PSPACE

$$L(M) = \{w \in \Sigma^* \mid \text{es gibt akzept. Haltekonf. } k \text{ mit } z_0 w \square \vdash_M^* k\}.$$

: 3C ist die Menge der ungerichteten Graphen, deren Knoten sich mit drei Farben färben lassen, so daß benachbarte Knoten unterschiedliche Farben haben.

Beobachtung: 3C  $\in$  PSPACE

Sei  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine monotone Funktion. Die Klasse  $NTIME(f)$  besteht aus allen Sprachen  $L$ , für die es eine nichtdeterministische Turingmaschine  $M$  gibt mit:

- $M$  akzeptiert  $L$ .
- Für jede Eingabe  $w \in \Sigma^*$  hält  $M$  auf jeden Fall nach  $f(|w|)$  vielen Schritten.

Definition

$$NP = \bigcup_{f \in Poly} NTIME(f)$$

• Sei  $L$  eine Sprache. Dann sind äquivalent

• 1.  $L$  ist kontextsensitiv (d.h. vom Typ 1)

• 2.  $L \in NSPACE(n)$

$$2NEXPTIME = \bigcup_{f \in Poly} NTIME(2^{2^f}) \dots$$

Zu jeder nichtdeterministischen Turingmaschine gibt es eine Turingmaschine, die dieselbe Sprache akzeptiert.

Sei  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine monotone Funktion. Die Klasse  $NSPACE(f)$  besteht aus allen Sprachen  $L$ , für die es eine nichtdeterministische Turingmaschine  $M$  gibt mit: -  $M$  akzeptiert  $L$ . - Für jede Eingabe  $w \in \Sigma^*$  folgt  $|k| \leq f(|w|)$  aus  $z_0 w \square \vdash_M^* k$ .

Lemma:  $NP \subseteq PSPACE, NEXPTIME \subseteq EXSPACE, 2NEXPTIME \subseteq 2EXSPACE \dots$

SATZ

DEFINITION

Satz von Savitch (1970)

NP-Vollständigkeit

SATZ

DEFINITION

SAT Vollständigkeit

3-SAT

SATZ

SATZ

3-SAT vollständigkeit

SAT und 3-SAT vollständigkeit

SATZ

SATZ

ist eine Formel  $\phi$  in KNF mit höchstens  
zwei Literalen pro Klausel erfüllbar?

Erfüllbarkeitsprobleme vollständigkeit

DEFINITION

SATZ

kC

3C vollständigkeit

Eine Sprache  $B$  ist NP-hart, falls für alle  $A \in NP$  gilt:  $A \leq_P B$  ( $A$  ist mindestens so schwer wie jedes Problem in NP). Eine Sprache ist NP-vollständig, falls sie zu NP gehört und NP-hart ist.

3-SAT ist die Menge der erfüllbaren aussagenlogischen Formeln in konjunktiver Normalform mit höchstens drei Literalen pro Klausel.

Die Probleme SAT und 3-SAT sind NP-vollständig.

- Die Erfüllbarkeitsprobleme SAT und 3-SAT sind NP-vollständig.
- Das Erfüllbarkeitsproblem 2-SAT ist in P.

Das Problem 3C ist NP-vollständig.

Für jede super-lineare monotone Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  gilt  $NSPACE(f(n)) \subseteq SPACE((f(n))^2)$ .  
Damit haben wir die folgende Struktur der Komplexitätsklassen:

1. P
2. NP
3. PSPACE = NPSPACE
4. EXPTIME
5. NEXPTIME
6. EXPSPACE = NEXPSPACE

Stephen Cook & Leonid Levin: SAT ist NP-vollständig.

Das Problem 3-SAT ist NP-vollständig.

Es ist in Polynomialzeit entscheidbar, ob eine Formel  $\phi$  in KNF mit höchstens zwei Literalen pro Klausel erfüllbar ist. Beweisidee: konstruieren gerichteten Graphen  $G$ :

- Für jede atomare Formel  $x$  aus  $\phi$  gibt es die Knoten  $x$  und  $\neg x$ .
- Für jede Klausel  $\alpha \vee \beta$  in  $\phi$  gibt es Kanten  $\sim \alpha \rightarrow \beta$  und  $\sim \beta \rightarrow \alpha$ , wobei  $\sim x = \neg x$  und  $\sim \neg x = x$  gelte.

$kC$  ist die Menge der ungerichteten Graphen, die sich mit  $k$  Farben färben lassen.

Ein Graph ist genau dann 2-färbbar, wenn er bipartit ist. Das Problem 2C ist also in P.

DEFINITION

**DHC - Gerichteter Hamiltonkreis**

**DHC**

SATZ

**DHV vollständigkeit**

**HC - Ungerichteter Hamiltonkreis**

DEFINITION

SATZ

**HC**

**HC vollständigkeit**

SATZ

**TSP - Travelling Salesman**

**Das Problem TSP**

**Church-Turing These**

**Unentscheidbarkeit**

DHC ist die Menge der gerichteten Graphen, die einen Hamiltonkreis enthalten.

- EINGABE: ein ungerichteter Graph  $G = (V, E)$  mit Knotenmenge  $V$  und Kantenmenge  $E \supseteq \binom{V}{2} = \{X \subseteq V \mid |X| = 2\}$ .
- FRAGE: Besitzt der Graph  $G$  einen Hamiltonkreis, d.h. kann man den Graphen so durchlaufen, dass jeder Knoten genau einmal besucht wird?

Das Problem HC ist NP-vollständig.

Das Problem HC ist NP-vollständig.

- Beweis:  $TSP \in NP$  ist einfach zu sehen, da eine Indexfolge geraten und in polynomieller Zeit überprüft werden kann, ob sie die Bedingungen erfüllt.
- Für die NP-Härte zeigen wir  $HC \leq_P TSP$ : Sei  $G = (V, E)$  ein ungerichteter Graph, o.E.  $V = \{1, \dots, n\}$ . Wir konstruieren dazu folgende Matrix:  $M_{i,j} = \begin{cases} 1 & \text{falls } \{i, j\} \in E \\ 2 & \text{falls } \{i, j\} \notin E \end{cases}$
- Außerdem setzen wir  $d = n$ .

Probleme, die nicht durch Turing-Maschinen gelöst werden können, sind damit prinzipiell unlösbar (wenn auch u.U. semi-entscheidbar).

- EINGABE: ein gerichteter Graph  $G = (V, E)$  mit Knotenmenge  $V$  und Kantenmenge  $E \supseteq V \times V$ .
- FRAGE: Besitzt der Graph  $G$  einen Hamiltonkreis, d.h. kann man den Graphen so durchlaufen, dass jeder Knoten genau einmal besucht wird?

Das Problem DHC ist NP-vollständig.

Das Problem HC ist NP-vollständig.

- EINGABE: eine  $n \times n$ -Matrix  $M = (M_{i,j})$  von Entfernungen zwischen  $n$  Städten und eine Zahl  $d$ .
- FRAGE: Gibt es eine Tour durch alle Städte, die maximal die Länge  $d$  hat? Das heißt, gibt es eine Indexfolge  $i_1, \dots, i_m$ , so dass gilt:
- $\{i_1, \dots, i_m\} = \{1, \dots, n\}$  (jede Stadt kommt vor)
- $M_{i_1, i_2} + M_{i_2, i_3} + \dots + M_{i_{m-1}, i_m} + M_{i_m, i_1} \leq d$  (die Länge Tour ist höchstens  $d$ )

Die Church-Turing These besagt, dass die Funktionen, die durch Turingmaschinen bzw. While-/Goto-Programme berechnet werden können, genau die intuitiv berechenbaren Funktionen sind.

**Erweiterte Church-Turing These**

**Turing Maschinen für NP und darüber**

Probleme, die durch Turing-Maschinen nicht in Polynomialzeit gelöst werden können, sind damit prinzipiell nicht effizient lösbar.

Die erweiterte Church-Turing These besagt, dass die Funktionen, die durch Turingmaschine bzw. While-/Goto-Programme in Polynomialzeit berechnet werden können, genau die intuitiv und effizient berechenbaren Funktionen sind.