

Einführung in die Theoretische Informatik 2

Prof. Dr. Horst Müller
SS 2004

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Inhalt

- Formale Sprachen
- Grammatiken
- Automaten und Maschinen
- Berechenbarkeit und Entscheidbarkeit
- Halteproblem, Termination

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Einleitung

- Voraussetzungen: Einführung in die theoretische Informatik 1, Algorithmik 1
- (Verweis z.B. auf [Skript Teil I](#))
- [Seite 103 \(Def. Grammatik\)](#) aus dem Skript von Prof. Jablonski
- Eigene [Foliensammlung im Internet](#) unter
 - <http://www3.informatik.uni-erlangen.de/Lehre/EThInfII/SS2004/>

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Formale Sprachen

- **Alphabet Σ** : Menge von Zeichen/Buchstaben
- Σ^* : Menge aller Wörter über Σ
- **Formale Sprache L** : Teilmenge von Σ^*
- Erzeugung/Generierung durch Grammatiken
- Analyse/Erkennung durch Automaten/Maschinen

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Grammatik

- Eine **Grammatik** ist ein Tupel $G = (Var, \Sigma, P, S)$ mit
- Var : endliche Menge (von *Variablen*)
- Σ : endliche Menge (von *terminalen Symbolen*)
- P : endliche Menge (von *Produktionen, Regeln*)
- $P \subseteq (Var \cup \Sigma)^+ \times (Var \cup \Sigma)^*$
- $S \in Var$ (*Startvariable*)

Notierung von Produktionen:

- $liS \rightarrow reS$ statt $(liS, reS) \in P$
- $liS \rightarrow reS_1 | \dots | reS_n$ statt $(liS, reS_i), \dots, (liS, reS_n) \in P$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Ableitung und erzeugte Sprache

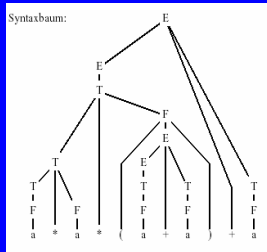
- **Produktionen**: linkeSeite \rightarrow rechteSeite
 - $y \rightarrow y'$ ($y, y' \in (Var \cup \Sigma)^*$)
- **Direkter Ableitungsschritt**:
 - $w \Rightarrow w'$ gdw es gibt x, y, y', z mit
 - $w = x y z, w' = x y' z$ und $y \rightarrow y'$ in P
- **Ableitung**: $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$ ($w_i \in (Var \cup \Sigma)^*$)
- **Erzeugte Sprache**:

$$L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kontextfreie Grammatik (Beispiel)

- $G = (Var, \Sigma, P, S)$
- $G_1 = ((E, T, F), \{ (,), a, +, * \}, P, E)$ mit
- $P = \{ E \rightarrow T, E \rightarrow E + T, T \rightarrow F, T \rightarrow T * F, F \rightarrow a, F \rightarrow (E) \}$
- Kurzform:
 - $E \rightarrow T \mid E + T,$
 - $T \rightarrow F \mid T * F,$
 - $F \rightarrow a \mid (E)$
- Ableitung:
 - $E \Rightarrow E + T \Rightarrow T + T \Rightarrow T * F + T \Rightarrow T * (E) + T \Rightarrow T * (T * F) + T \Rightarrow T * (E + F) + T \Rightarrow T * (E + a) + T \Rightarrow \dots$



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Nichtkürzende Grammatik (kontextsensitive Sprache)

$G_2 = (\{S, B, C\}, \{a, b, c\}, P, S)$ mit

- $P = \{ S \rightarrow aSBC \mid aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc \}$

$L(G_2) = \{ a^n b^n c^n \mid n \geq 1 \}$

Ableitung: $S \Rightarrow aSBC \Rightarrow \dots \Rightarrow a^{n-1}S(BC)^{n-1} \Rightarrow a^n(BC)^n \Rightarrow^* a^n B^n C^n \Rightarrow^* a^n b^n c^n$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Produktions-Typen

- Produktions-Typen: Sei $u, v, w, w_1, w_2 \in (V \cup \Sigma)^*$; $X, Y \in V$
Produktion $u \rightarrow v$

Typ	Bezeichnung	Form von u	Form von v	Nebenbedingung
0	uneingeschränkt	$u \neq \epsilon$		
1	kontextsensitiv	$w_1 X w_2$	$w_1 w w_2$	$w \neq \epsilon$
2	kontextfrei	X	w	
3	rechtslinear (regulär)	X	wY oder w	$w \in \Sigma^*$
	nichtkürzend			$ u \leq v $

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Grammatik-Typen:

Typ	Bedingung
0	Jede Produktion hat Typ 0
1	Jede Produktion hat Typ 1 oder $(P - P' \cup \{ S \rightarrow \epsilon \} \wedge \forall (u \rightarrow v) \in P': (u \rightarrow v) \text{ hat Typ 1} \wedge S \text{ nicht in } v)$
2	Jede Produktion hat Typ 2
3	Jede Produktion hat Typ 3

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

nichtkürzend - kontextsensitiv, ϵ -Produktionen

- Satz:** Nichtkürzende Grammatiken und kontextsensitive Grammatiken ohne $S \rightarrow \epsilon$ erzeugen dieselbe Sprachenklasse.
- Satz:** Zu jeder Typ-i-Sprache L gibt es eine Typ-i-Grammatik G' ohne ϵ -Produktionen (Form: $l|S \rightarrow \epsilon$) mit $L(G') = L - \{ \epsilon \}$ ($i = 0, 1, 2, 3$).

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Chomsky-Hierarchie

- L_i = Klasse der Typ-i-Sprachen
- Chomsky-Hierarchie-Theorem:**
 $L_3 \subset L_2 \subset L_1 \subset L_0$



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Wortproblem für Grammatiken

- Def.:** **Wortproblem** für Grammatiken G vom Typ i :
 - Das Wortproblem für G ist entscheidbar, wenn es einen Algorithmus gibt, der bei Eingabe eines Wortes $w \in \Sigma^*$ entscheidet, ob $w \in L(G)$ gilt oder nicht.
- Satz:** Für Typ-1-Grammatiken ist das Wortproblem entscheidbar.
- Satz:** Es gibt Typ-0-Grammatiken, für die das Wortproblem unentscheidbar ist.
- Satz:** $L_1 \subset$ Klasse der entscheidbaren Sprachen $\subset L_0$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Endlicher erkennender Automat

- $A = (Z, \Sigma, \delta, s, E)$ (bzw. $(Z, \Sigma, \delta, S, E)$) mit
 - Z : endl. Zustandsmenge ; Σ : endl. Eingabealphabet
 - a) deterministisch:
 - $\delta : Z \times \Sigma \rightarrow Z$: partielle Übergangsfunktion
 - $s \in Z$: Startzustand
 - b) vollständig deterministisch:
 - $\delta : Z \times \Sigma \rightarrow Z$: totale Übergangsfunktion
 - c) nicht-deterministisch: Übergangsrelation $\delta \subseteq Z \times \Sigma \times Z$
 - $S \subseteq Z$: Startzustandsmenge
 - $E \subseteq Z$: End-Zustandsmenge, Menge akzeptierender Zustände

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Vom deterministischen Automaten akzeptierte Sprache

- Definition:** Erweiterung von $\delta : Z \times \Sigma \rightarrow Z$ zu $\delta^* : Z \times \Sigma^* \rightarrow Z$
 - $\delta^*(z, \epsilon) = z$; $\delta^*(z, w\sigma) = \delta(\delta^*(z, w), \sigma)$
- Definition:** Der deterministische endliche Automat A akzeptiert die Sprache
 - $L(A) = \{ w \in \Sigma^* \mid \delta^*(s, w) \in E \}$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

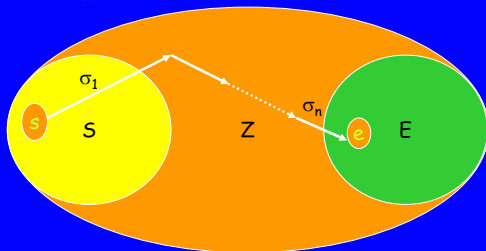
Vom nichtdeterministischen Automaten akzeptierte Sprache

- Definition:** Erweiterung von $\delta \subseteq Z \times \Sigma \times Z$ zu $\delta^* \subseteq Z \times \Sigma^* \times Z$:
 - $(z, \epsilon, z) \in \delta^*$ und
 - $(z_1, w\sigma, z_2) \in \delta^*$ gdw $\exists z_3 (z_1, w, z_3) \in \delta^*$ und $(z_3, \sigma, z_2) \in \delta$
- Definition:** Der nichtdeterministische endliche Automat A akzeptiert die Sprache
 - $L(A) = \{ w \in \Sigma^* \mid \exists s \in S: \exists e \in E: (s, w, e) \in \delta^* \}$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Erfolgreicher Pfad

$w = \sigma_1 \dots \sigma_n \in L(A) \iff$



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Endlicher Automat, Zähler mod 4

$A = (Z, \Sigma, \delta, s, E)$ mit
 $Z = \{z_0, z_1, z_2, z_3\}$, $\Sigma = \{a, b\}$, $s = z_0$, $E = \{z_3\}$

Wertetabelle für δ :

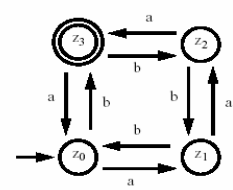
	z	z_0	z_1	z_2	z_3
x	$\delta(z, x)$				
a		z_1	z_2	z_3	z_0
b		z_3	z_0	z_1	z_2

Akzeptierte Sprache:

$L(A) = \{w \mid I_a(w) - I_b(w) \equiv_4 3\}$

$I_a(w) =$ Anzahl der Vorkommen von a in w

Automatengraph:



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel-Lauf des Automaten

- Eingabeband
a a b a a
- $aabaa \in L(A)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel-Lauf des Automaten (Animation)

- Eingabeband:
a a b a a
- Verarbeiteter Teil:
a a b a a
- Also: $aabaa \in L(A)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Endlicher Automat \rightarrow Typ-3-Grammatik, Potenzmengenkonstruktion

- **Satz:** Jede durch einen det. endlichen Automaten akzeptierte Sprache ist regulär (durch eine Typ-3-Grammatik erzeugbar).
- **Satz(Rabin/Scott):** Jede durch einen nicht det. akzeptierbare Sprache ist durch einen det. endlichen Automaten akzeptierbar.
 - Bemerkung: Umkehrung trivial
 - Beweis mit Potenzmengenkonstruktion

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Typ-3-Grammatik in Normalform

- Alle Produktionen haben eine der Formen
 - $X \rightarrow \sigma$
 - $X \rightarrow \sigma Y$ ($X, Y \in V; \sigma \in \Sigma$)
 - $S \rightarrow \epsilon$
- **Satz:** Zu jeder Typ-3-Grammatik G kann man eine äquivalente Typ-Normalform-Grammatik G' konstruieren (d.h. mit $L(G) = L(G')$).

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel zur Potenzmengen-Konstruktion 1

Nichtdeterministischer Automat:

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel zur Potenzmengen-Konstruktion 2

Konstruierter deterministischer Automat:

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Vier Charakterisierungen von „regulär“

Det. endl. erk. Automat Typ 3- Grammatik

Potenzmengen-
konstruktion Reguläre Sprache

Nichtdet. endl. erk. Autom. Typ-3-Normalform

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Reguläre Ausdrücke über Σ

γ	\emptyset	ε	a	$\alpha\beta$	$(\alpha \beta)$	$(\alpha)^*$
$L(\gamma)$	\emptyset	$\{\varepsilon\}$	$\{a\}$	$L(\alpha)L(\beta)$	$L(\alpha) \cup L(\beta)$	$L(\alpha)^*$

$(a \in \Sigma)$

Satz (Kleene): Die regulären Ausdrücke beschreiben genau die regulären Sprachen (Typ-3-Sprachen).

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Pump-Lemma für reguläre Sprachen (pumping-Lemma):

Pumplemma: Zu jeder regulären Sprache L gibt es eine natürliche Zahl p (Pumpzahl) mit

$$\forall x \in L : (|x| \geq p \rightarrow (\exists u, v, w \in \Sigma^* : (x = uvw \wedge |v| \geq 1 \wedge |uv| \leq p \wedge \forall i \in \mathbb{N} : u v^i w \in L)))$$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Langer erfolgreicher Pfad

$x = \sigma_1 \dots \sigma_n \in L(A) \iff$

$x = u v w \mid v \neq \varepsilon$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Satz von Myhill/Nerode

- Definition:** Die von der Sprache L induzierte Äquivalenzrelation R_L (auf Σ^*):
 $x R_L y \iff \forall z \in \Sigma^* : (x z \in L \iff y z \in L)$
- Definition:** Index von R_L : Anzahl der Äquivalenzklassen von R_L
- Satz (Myhill/Nerode):** Die Sprache L ist genau dann regulär, wenn der Index von R_L endlich ist.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Minimalautomat

- Satz/Definition:** Zu jeder regulären Sprache L gibt es einen (bis auf Isomorphie eindeutig bestimmten) vollständigen deterministischen erkennenden Automaten minimaler Zustandszahl (Minimalautomat zu L).

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

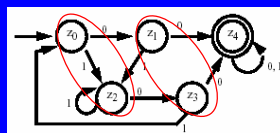
Minimierung

- Definition:**
 - $z \equiv_n z' \iff \forall x \in \Sigma^* : (|x| \leq n \rightarrow (\delta^*(z, x) \in E \leftrightarrow \delta^*(z', x) \in E))$
 - $z \equiv z' \iff \forall x \in \Sigma^* : (\delta^*(z, x) \in E \leftrightarrow \delta^*(z', x) \in E)$
- Algorithmus zur Minimierung det. erk. Automaten**
 - Eingabe: vollständiger deterministischer erkennender Automat
 - Voraussetzung: alle Zustände sind vom Startzustand erreichbar
 - Stelle eine Tabelle aller $\{z, z'\}$ mit $z \neq z'$ auf.
 - Markiere alle $\{z, z'\}$ mit $z \in E$ und $z' \notin E$ (oder umgekehrt)
 - Für jedes unmarkierte $\{z, z'\}$ und jedes $\sigma \in \Sigma$ teste, ob $\{\delta(z, \sigma), \delta(z', \sigma)\}$ markiert ist. Wenn ja, markiere auch $\{z, z'\}$.
 - Wiederhole Schritt 3 bis sich keine Änderung mehr ergibt.
 - Für unmarkierte $\{z, z'\}$ werden z und z' identifiziert.
- Lemma:** $\{z, z'\}$ markiert $\iff \exists n : \neg z \equiv_n z'$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel zur Minimierung

Zu minimierender Automat:



Paar-Übergänge:

z, z'	$\delta(z, 0)$	$\delta(z', 0)$	$\delta(z, 1)$	$\delta(z', 1)$
0,1	1,1	1,1	2,2	2,2
0,2	1,3	1,3	2,2	2,2
0,3	1,4	1,4	2,0	2,0
1,2	4,3	4,3	2,2	2,2
1,3	4,4	4,4	2,0	2,0
2,3	3,4	3,4	2,0	2,0

(In-)Äquivalenz-Tabelle:

z_1	*			
z_2	*	*		
z_3	*	*	*	
z_4	*	*	*	*
	z_0	z_1	z_2	z_3

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Abschluss-Eigenschaften der Klasse regulärer Sprachen

- Definition:** Eine Klasse K von Sprachen ist abgeschlossen unter der zweistelligen Operation op genau dann, wenn gilt:
 - $L_1, L_2 \in K \rightarrow op(L_1, L_2) \in K$
- Satz:** Die Klasse der regulären Sprachen ist abgeschlossen unter Vereinigung, Durchschnitt, Komplement, Produkt und Sternoperation.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Entscheidungsprobleme für reguläre Sprachen 1

- Wort-Problem** (für eine Typ-3-Grammatik über Σ):
 - Gegeben: $x \in \Sigma^*$
 - Entscheide: $x \in L$
- Leerheits-Problem** (für Typ-3-Grammatiken über Σ):
 - Gegeben: Typ-3-Grammatik G
 - Entscheide: $L(G) = \emptyset$
- Endlichkeits-Problem** (für Typ-3-Grammatiken über Σ):
 - Gegeben: Typ-3-Grammatik G
 - Entscheide: L(G) ist endlich

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

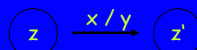
Entscheidungsprobleme für reguläre Sprachen 2

- Schnitt-Problem** (für Typ-3-Grammatiken über Σ):
 - Gegeben: Typ-3-Grammatiken G_1 und G_2
 - Entscheide: $L(G_1) \cap L(G_2) = \emptyset$
- Äquivalenz-Problem** (für Typ-3-Grammatiken über Σ):
 - Gegeben: Typ-3-Grammatiken G_1 und G_2
 - Entscheide: $L(G_1) = L(G_2)$
- Inklusions-Problem** (für Typ-3-Grammatiken über Σ):
 - Gegeben: Typ-3-Grammatiken G_1 und G_2
 - Entscheide: $L(G_1) \subseteq L(G_2)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Mealy-Automaten, Moore-Automaten

- Endliche Automaten mit Ein- und Ausgabe
- $A = (X, Y, Z, \delta, \lambda)$
- X : endliche Menge der Eingaben
- Y : endliche Menge der Ausgaben
- Z : endliche Menge der Zustände
- $\delta : Z \times X \rightarrow Z$: Übergangsfunktion
- (Mealy-A.) $\lambda : Z \times X \rightarrow Y$: Ausgabefunktion
- (Moore-A.) $\lambda : Z \rightarrow Y$: Ausgabefunktion



Für $\delta(z, x) = z', \lambda(z, x) = y$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kara

- Kara kann als Mealy-Automat aufgefasst werden.
- Eine Beschreibung von Kara findet sich unter:
 - [http://www.educath.ch/informatik/koneto.java/ \(lskl\)](http://www.educath.ch/informatik/koneto.java/)
- Beispiele:
 - Kara und die Blätter (1. Aufgabe)
 - **Kara** kann den Automaten „Zähler mod 4“ simulieren.
 - Etwas für Geduldige: Chaotische Ameise (letzte Aufgabe)

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Endlicher Automat, Zähler mod 4

$A = (Z, \Sigma, \delta, s, E)$ mit
 $Z = \{z_0, z_1, z_2, z_3\}$, $\Sigma = \{a, b\}$, $s = z_0$, $E = \{z_3\}$

Wertetabelle für δ :

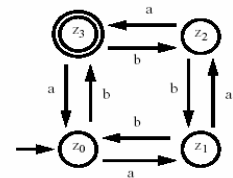
	z	z ₀	z ₁	z ₂	z ₃
x	$\delta(z, x)$				
a		z ₁	z ₂	z ₃	z ₀
b		z ₃	z ₀	z ₁	z ₂

Akzeptierte Sprache:

$$L(A) = \{w \mid I_a(w) - I_b(w) \equiv_4 3\}$$

$I_a(w)$ = Anzahl der Vorkommen von a in w

Automatengraph:



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 3 Kontextfreie Sprachen und Kellerautomaten

Friedrich-Alexander-Universität Erlangen-Nürnberg

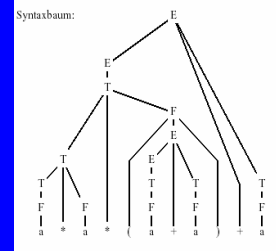
Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Kontextfreie Grammatik (Beispiel)

- $G = (Var, \Sigma, P, S)$
- $G_1 = ((E, T, F), \{ (,), a, +, * \}, P, E)$ mit
- $P = \{ E \rightarrow T, E \rightarrow E + T, T \rightarrow F, T \rightarrow T * F, F \rightarrow a, F \rightarrow (E) \}$
- Kurzform:
 - $E \rightarrow T \mid E + T$,
 - $T \rightarrow F \mid T * F$,
 - $F \rightarrow a \mid (E)$
- Ableitung:
- $E \Rightarrow E + T \Rightarrow T + T \Rightarrow T * F + T \Rightarrow T * (E) + T \Rightarrow T * (E + T) + T \Rightarrow T * (E + T) + T \Rightarrow T * (E + a) + T \Rightarrow \dots$

Syntaxbaum:



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Erweiterte Backus-Naur-Form

Erweiterte Backus-Naur-Form	Äquivalente kontextfreie Form
$X \rightarrow v_1 \mid \dots \mid v_n$	$X \rightarrow v_1, \dots, X \rightarrow v_n$
$X \rightarrow u (v_1 \mid \dots \mid v_n) w$	$X \rightarrow u v_1 w \mid \dots \mid u v_n w$
$X \rightarrow u \{ v \} w$	$X \rightarrow u w \mid u Y w,$ $Y \rightarrow v \mid v Y$
$X \rightarrow u [v] w$	$X \rightarrow u w \mid u v w$
Statt " → " wird auch " ::= " verwendet.	

Satz: Auch durch EBNF werden genau die kontextfreien Sprachen erzeugt.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel: LOOP in EBNF-Darstellung

- $\langle \text{Konstante} \rangle ::= 0 \mid \langle \text{Pos. Ziffer} \rangle \{ \langle \text{Ziffer} \rangle \}$
- $\langle \text{Pos. Ziffer} \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- $\langle \text{Ziffer} \rangle ::= \langle \text{Pos. Ziffer} \rangle \mid 0$
- $\langle \text{Variable} \rangle ::= x \langle \text{Konstante} \rangle$
- $\langle \text{Wertzuweisung} \rangle ::=$
 $\langle \text{Variable} \rangle ::= \langle \text{Variable} \rangle (+ \mid -) \langle \text{Konstante} \rangle$
- $\langle \text{Programm} \rangle ::= \langle \text{Wertzuweisung} \rangle$
 $\mid \langle \text{Programm} \rangle ; \langle \text{Programm} \rangle$
 $\mid \text{LOOP} \langle \text{Variable} \rangle \text{ DO} \langle \text{Programm} \rangle \text{ END}$

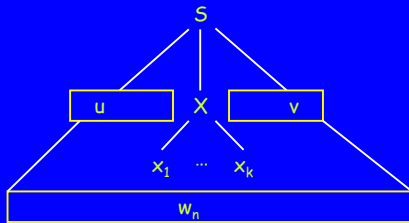
Beispiel-Programm:

- $x0 ::= x1 + 0 ; \text{LOOP } x2 \text{ DO } x0 ::= x0 + 1 \text{ END}$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Syntaxbaum einer Ableitung

- Einer Ableitung (in einer kontextfreien Grammatik)
 - $S = w_0 \Rightarrow \dots \Rightarrow w_i = u X v \Rightarrow w_{i+1} = u x_1 \dots x_k v \Rightarrow \dots \Rightarrow w_n$
- wird ein *Syntaxbaum* zugeordnet:



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Chomsky-Normalform (CNF)

- Definition:** Eine Grammatik ist in *Chomsky-Normalform*, wenn alle Produktionen die Form $X \rightarrow YZ$ oder $X \rightarrow \sigma$ ($X, Y, Z \in V; \sigma \in \Sigma$) haben.
- Satz:** Zu jeder kontextfreien Sprache L mit $\epsilon \notin L$ gibt es eine L erzeugende Grammatik in CNF.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Greibach-Normalform (GNF)

- Definition:** Eine Grammatik ist in *Greibach-Normalform*, wenn alle Produktionen die Form $X \rightarrow \sigma Y_1 \dots Y_k$ ($k \geq 0; X, Y_1, \dots, Y_k \in V; \sigma \in \Sigma$) haben.
- Satz:** Zu jeder kontextfreien Sprache L mit $\epsilon \notin L$ gibt es eine L erzeugende Grammatik in GNF.
- Bemerkung:** $X \rightarrow \sigma \mid \sigma Y_1 \mid \sigma Y_1 Y_2$ (also $0 \leq k \leq 2$) genügt

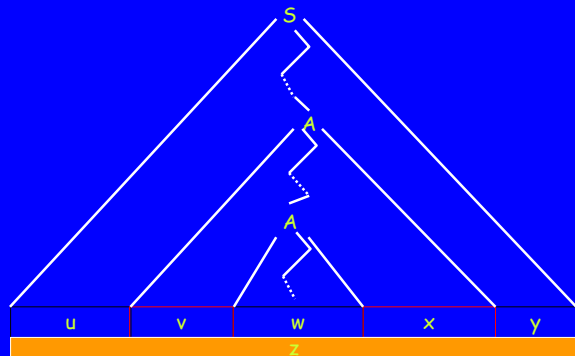
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Pump-Lemma für kontextfreie Sprachen

- Satz:** Für jede kontextfreie Sprache L gibt es eine natürliche Zahl (Pumpzahl) p , so dass sich jedes Wort $z \in L$ mit $|z| \geq p$ zerlegen lässt in $z = uvwx y$ mit
 - $|vx| \geq 1$
 - $|vwx| \leq p$
 - $\forall i (i \geq 0 \rightarrow u v^i w x^i y \in L)$

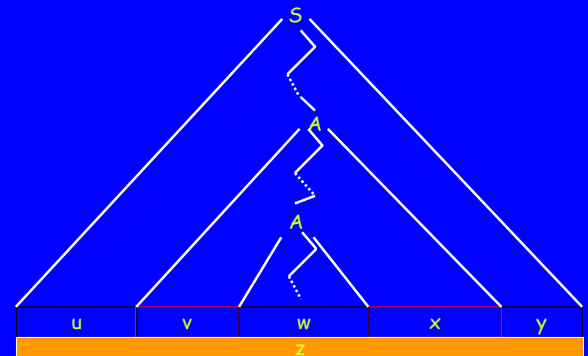
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beweis-Skizze zum Pump-Lemma 1



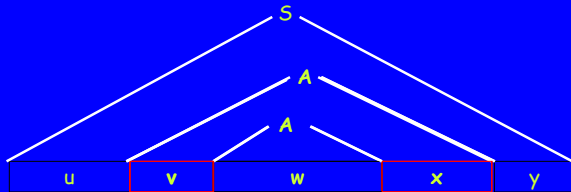
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beweis-Skizze zum Pump-Lemma 2



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beweis-Skizze zum Pump-Lemma 3



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kontextsensitive, aber nicht kontextfreie Sprache kontextfrei über einelementigem Alphabet

- **Satz** : $L = \{a^n b^n c^n \mid n \geq 1\}$ ist nicht kontextfrei (, aber kontextsensitiv).
- **Beweis-Skizze:**
 - $a \dots a \ b \dots b \ c \dots c$
 - $u \ v \ w \ x \ y$
- **Satz** : Jede kontextfreie Sprache über einelementigem Alphabet ist regulär.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

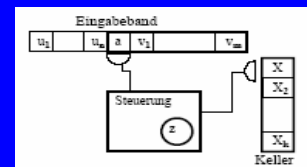
Abschluss-Eigenschaften der Klasse kontextfreier Sprachen

- Die Klasse kontextfreier Sprachen ist abgeschlossen unter Vereinigung, Produkt und Sternoperation.
- Die Klasse kontextfreier Sprachen ist **nicht** abgeschlossen unter Durchschnitt.
- Die Klasse kontextfreier Sprachen ist **nicht** abgeschlossen unter Komplementbildung.

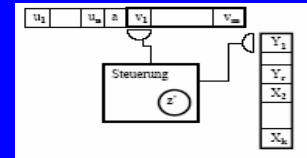
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

(Nichtdet.) Kellerautomat

- $A = (Z, \Sigma, \Gamma, \delta, s, E, \#)$
- (endliche) Zustandsmenge: Z
- Eingabealphabet: Σ
- $u_1, \dots, u_n, v_1, \dots, v_m, a \in \Sigma$
- Kelleralphabet: Γ
- $X_1, X_2, \dots, X_k, Y_1, \dots, Y_r \in \Gamma$
- Überföhrungsfunktion: $\delta : Z \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \wp_{\neq \emptyset}(Z \times \Gamma^*)$
- Anfangszustand: $s \in Z$;
- Endzustandsmenge: $E \subseteq Z$
- Kelleralphabetsymbol: $\# \in \Gamma$
- Konfiguration: $(z, a, v, X, X_2, \dots, X_k) \in Z \times \Sigma^+ \times \Gamma^*$



Folgekonfiguration für $(z', Y_1 \dots Y_r) \in \delta(z, a, X)$:



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Rechenschrittrelation |— des Kellerautomaten Vom Kellerautomaten akzeptierte Sprache

- Lesender Übergang:
 $(z', Y_1 \dots Y_r) \in \delta(z, a, X) \leftrightarrow (z, a, v, X, kw) \mid\text{---} (z', v, Y_1 \dots Y_r, kw)$
- Spontaner Übergang:
 $(z', Y_1 \dots Y_r) \in \delta(z, \epsilon, X) \leftrightarrow (z, v, X, kw) \mid\text{---} (z', v, Y_1 \dots Y_r, kw)$
- Die von A durch leeren Keller akzeptierte Sprache:
 $N(A) := \{ w \in \Sigma^* \mid (s, w, \#) \mid\text{---}^* (z, \epsilon, \epsilon) \text{ f\u00fcr ein } z \in Z \}$
(N von 'Null-Keller')
- Die von A durch Endzustand akzeptierte Sprache:
 $L(A) := \{ w \in \Sigma^* \mid (s, w, \#) \mid\text{---}^* (e, \epsilon, k) \text{ f\u00fcr ein } e \in E \text{ und ein } k \in \Gamma^* \}$
- (Eingabe-Rest muss in beiden F\u00e4llen ϵ sein.)

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Deterministischer Kellerautomat

- **Definition:** Ein Kellerautomat A ist *deterministisch* genau dann, wenn f\u00fcr alle $z \in Z, a \in \Sigma, Y \in \Gamma$ gilt:
 $|\delta(z, a, Y)| + |\delta(z, \epsilon, Y)| \leq 1$
- **Bemerkung:** Die beiden Formen der Akzeptierung sind f\u00fcr nichtdeterministische Kellerautomaten gleichwertig, jedoch nicht gleichwertig f\u00fcr deterministische Kellerautomaten.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel Kellerautomaten für Palindrome

(deterministischer) Kellerautomat A_1 für $L_1 = \{w \$ w^{rev} \mid w \in \{a,b\}^*\}$

$(a_1 \dots a_n)^{rev} = a_n \dots a_1, A_1 = (Z = \{s, z_1\}, \Sigma = \{a, b, \$\}, \Gamma = \{a, b, \#\}, \delta, s, E = \{z_1, \#\})$

(1) $\delta(s, x, Y) = \{(s, xY)\} \quad (x \in \{a,b\}, Y \in \Gamma)$

(2) $\delta(s, \$, Y) = \{(z_1, Y)\} \quad (Y \in \Gamma)$

(3) $\delta(z_1, x, x) = \{(z_1, \epsilon)\} \quad (x \in \{a,b\})$

(4) $\delta(z_1, \epsilon, \#) = \{(z_1, \epsilon)\}$

Akzeptierende Berechnung für $w = ba\$ab$:

$s, ba\$ab, \# \xrightarrow{-s} s, a\$ab, b\# \xrightarrow{-s} s, \$ab, ab\# \xrightarrow{-z_1} z_1, ab, ab\# \xrightarrow{-z_1} z_1, b, b\# \xrightarrow{-z_1} z_1, \epsilon, \# \xrightarrow{-z_1} z_1, \epsilon, \epsilon$

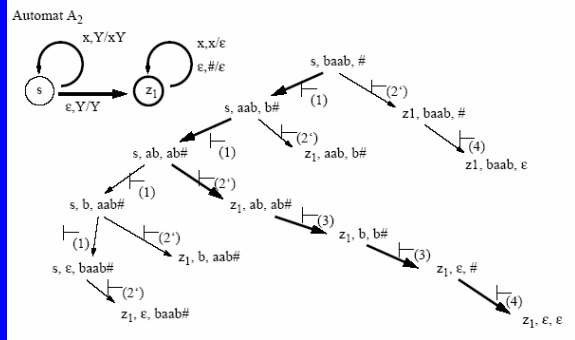
$N(A_1) = L_1; L(A_1) = \{a,b\}^* L_1$

(nichtdeterministischer) Kellerautomat A_2 mit $N(A_2) = L_2 = \{w w^{rev} \mid w \in \{a,b\}^*\}$

ersetze (2) durch $(2') \delta(s, \epsilon, Y) = \{(z_1, Y)\} \quad (Y \in \Gamma)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel Berechnungsbaum



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Deterministisch-kontextfreie Sprachen

- **Definition:** Eine Sprache heißt *deterministisch kontextfrei* genau dann, wenn sie von einem deterministischen Kellerautomaten durch Endzustand akzeptiert wird.
- **Satz:**
 - a) Die Klasse der deterministisch-kontextfreien Sprachen ist unter Komplementbildung abgeschlossen.
 - b) Die Klasse der deterministisch-kontextfreien Sprachen ist nicht unter Durchschnitt, Vereinigung, Produkt, Stern-Operation abgeschlossen.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Äquivalenz von kontextfreien Grammatiken und (nichtdeterministischen) Kellerautomaten

- **Satz:** Eine Sprache L ist kontextfrei genau dann, wenn sie von einem (nichtdeterministischen) Kellerautomaten akzeptiert wird.
- Beweis-Skizze: i) \Rightarrow : Aus einer kfr. Grammatik G wird ein äquivalenter Kellerautomat A konstruiert
 - Lemma:
 - a) $X \Rightarrow^* u \leftrightarrow \forall u', \gamma: (s, u u', X \gamma) \vdash^* (s, u', \gamma)$
($u, u' \in \Sigma^*; v \in (V \cup \Sigma)^*$; $X, Y \in V$)
 - b) $X \Rightarrow^* u v \leftrightarrow \forall u', \gamma: (s, u u', X \gamma) \vdash^* (s, u', v \gamma)$
- ii) \Leftarrow : Aus einem Kellerautomaten wird eine äquivalente Grammatik konstruiert.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

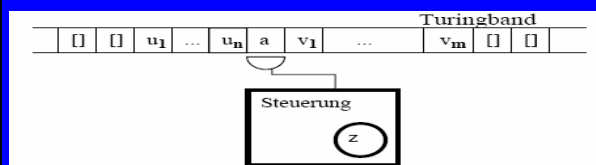
§ 4 Kontextsensitive Sprachen und linear beschränkte Automaten, Typ-0-Sprachen und Turingmaschinen

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
Martensstraße 3 • 91058 Erlangen



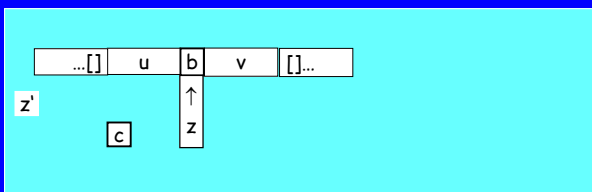
Turing-Maschine



- $TM = (Z, \Gamma, \delta, s, E, \square)$
- (endliche) Zustandsmenge Z
- Band-Alphabet $\Gamma: u_1, \dots, u_n, v_1, \dots, v_m, a \in \Gamma; \quad Z \cap \Gamma = \emptyset$
- deterministische TM:
 - Überföhrungsfunktion: $\delta: (Z - E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$
- nichtdeterministische TM:
 - Überföhrungsrelation: $\delta \subseteq (Z \times \Gamma) \times (Z \times \Gamma \times \{L, R, N\})$
- (L: links, R: rechts, N: keine Kopfbewegung)
- Anfangszustand $s \in Z$; Endzustandsmenge $E \subseteq Z$
- Blanksymbol $\square \in \Gamma$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Schrittrelation der TM



- **Definition: Konfiguration:** $u z' b v \in \Gamma^* Z \Gamma \Gamma^*$
- **Schrittrelation** \vdash : $(u, v \in \Gamma^*, a, b, c \in \Gamma, z' \in Z)$
- (1) $z u b v \vdash u z' c v$, wenn $((z, b), (z', c, N)) \in \delta$
- (2) $u z b v \vdash u c z' v$, wenn $((z, b), (z', c, R)) \in \delta \wedge v \neq \epsilon$
- (3) $u z b \vdash u c z' []$, wenn $((z, b), (z', c, R)) \in \delta$
- (4) $u a z b v \vdash u z' a c v$, wenn $((z, b), (z', c, L)) \in \delta$
- (5) $z b v \vdash z' [] c v$, wenn $((z, b), (z', c, L)) \in \delta$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Von Turingmaschinen akzeptierte Sprachen

- **Definition:** Die von der Turingmaschine TM akzeptierte Sprache: $(\Sigma \subseteq \Gamma - \{[]\})$
 $- L_2(TM) := \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* : \exists e \in E : s w [] \vdash^* u e v\}$
- **Satz:** Die von (nichtdeterministischen) Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen.
- **Satz:** Jede nichtdeterministische Turingmaschine ist durch eine deterministische Turingmaschine simulierbar.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Übersetzung von Turingmaschinen in Grammatiken

	Überführungsrel.	Schrittrelation	Produktionen P _{Sim}
(1)	$(z', c, N) \in \delta(z, b)$	$uzbv \vdash uz'cv$	$zb \rightarrow z'c$
(2)	$(z', c, R) \in \delta(z, b)$	$uzbb'v \vdash uc z' b'v$	$zbb' \rightarrow cz'b'$
(3)	$(z', c, R) \in \delta(z, b)$	$uzb \vdash uc z' []$	$zb\$ \rightarrow cz'[]\$$
(4)	$(z', c, L) \in \delta(z, b)$	$uazbv \vdash uz'acv$	$azb \rightarrow z'ac$
(5)	$(z', c, L) \in \delta(z, b)$	$zbv \vdash z' [] cv$	$\&zb \rightarrow \&z' [] c$

- & kennzeichnet den linken Rand des benutzten Turingbandteils
- \$ kennzeichnet den rechten Rand des benutzten Turingbandteils
- (6) Lemma: $u z v \vdash^* u' z' v' \leftrightarrow \& u z v \$ \Rightarrow^*_{P_{Sim}} \& u' z' v' \$$
- (7) Lemma: $s w [] \vdash^* u e v \leftrightarrow \& u e v \$ \Rightarrow^*_{(P_{Sim}^{-1})} \& s w [] \$$
- $P_{Start} = \{S \rightarrow \&X \$, X \rightarrow \gamma X \mid e \gamma \mid \gamma \in \Gamma, e \in E\}$
- $P_{Schluss} = \{\&s \rightarrow \epsilon, []\$ \rightarrow \epsilon\}$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Typ-0-Grammatik \rightarrow Turing-Maschine

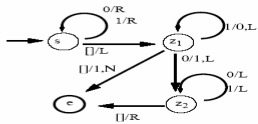
- Gegeben sei eine Grammatik G.
- Informelle Beschreibung einer TM, die Ableitungen von G simuliert: Bei Eingabe von w, also Startkonfiguration s w [] (und später einer Konfiguration der Form z [] w' []^m, die nach Simulation von k inversen Ableitungsschritten mit der Grammatik G erreicht ist), wählt TM (nichtdet.) eine Produktion $u \rightarrow v$ und sucht ein Vorkommen von v in w. Falls ein v gefunden wird, wird es durch u ersetzt (Für $|v| \neq |u|$ ist die Verschiebung der einen „Bandhälfte“ erforderlich). Dies wird iteriert. Wird dabei z [] S []^m erreicht, so akzeptiert TM.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

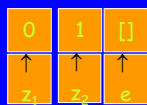
Beispiel TM für die Nachfolgerfunktion

TM = (Z = {s, z₁, z₂, e}, $\Gamma = \{0, 1, []\}$, $\delta, s, E = \{e\}, []$)

1. $\delta(s, 0) = (s, 0, R)$
2. $\delta(s, []) = (z_1, [], L)$
3. $\delta(z_1, 0) = (z_2, 1, L)$
4. $\delta(z_1, 1) = (z_1, 0, L)$
5. $\delta(z_1, []) = (e, 1, N)$
6. $\delta(z_2, 0) = (z_2, 0, L)$
7. $\delta(z_2, 1) = (z_2, 1, L)$
8. $\delta(z_2, []) = (e, [], R)$



- $s1101[] \vdash 1s101[] \vdash {}^3 1101s[] \vdash 110z_11[] \vdash 11z_100[]$
 $\vdash 1z_2110[] \vdash {}^2 z_2[]1110[] \vdash []e1110[]$



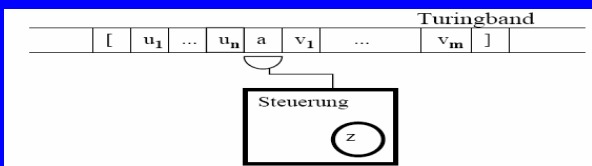
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Turing-Kara

- Kara kann Turingmaschinen simulieren
- Turing-Kara - Turingmaschinen anschaulich
- **Beispiele:**
 - Turingmaschine für die Sprache $\{0^n 1^n \mid n > 0\}$
 - Turingmaschine für die Sprache $\{a^n b^n c^n \mid n > 0\}$
 - TM zur Berechnung der Nachfolger-Funktion succ
 - Zweidimensionale Turingmaschine: Binäre Addition
 - Fleißiger Biber mit vier Zuständen
 - (2-dimensionaler) fleißiger Turing-Kara

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Linear Beschränkter Automat (LBA)



- $LBA = (Z, \Sigma, \Gamma, \delta, s, E, [,])$
- (endliche) Zustandsmenge Z ; Eingabe-Alphabet Σ ;
- Band-Alphabet Γ ; $u_1, \dots, u_n, v_1, \dots, v_m, a, [,] \in \Gamma$
- Anfangszustand $s \in Z$; Endzustandsmenge $E \subseteq Z$
- Linkes Rand-Symbol $[\in \Gamma$; Rechtes Rand-Symbol $] \in \Gamma$
- Schrittrelation $\dashv\vdash$; wie bei einer nichtdeterministischen Turingmaschine, jedoch mit *Randrespektierung*:
 - $(z', b, r) \in \delta(z, [) \rightarrow b = [\wedge r \in \{R, N\}$
 - $(z', b, r) \in \delta(z,]) \rightarrow b =] \wedge r \in \{L, N\}$

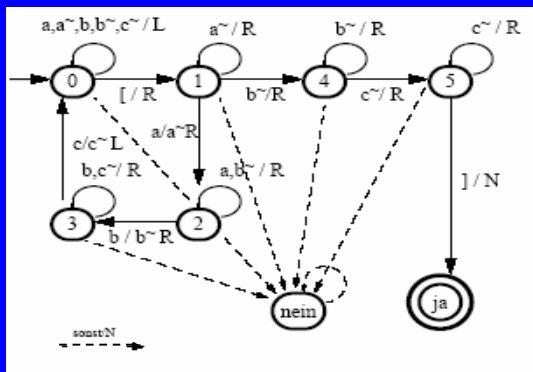
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Von LBA akzeptierte Sprachen

- **Definition:** Die vom linear beschränkten Automaten LBA akzeptierte Sprache:
 - $(\Sigma \subseteq \Gamma - \{[,]\})$
- $L(LBA) :=$
 - $\{w \in \Sigma^* \mid \exists u, v \in \Gamma^* : \exists e \in E : s[w] \dashv\vdash^* u e v\}$
- **Satz :** Die von (nichtdet.) Linear Beschränkten Automaten akzeptierten Sprachen sind genau die Typ-1-Sprachen.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LBA für $L = \{a^n b^n c^n \mid n > 0\}$



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LBA \rightarrow nichtkürzende Grammatik

- **Modifikation der Konstruktion für Turingmaschinen**
- $P_{Start} = \{S \rightarrow [X \mid e [Y], X \rightarrow vX \mid e \mid Xv, Y \rightarrow vY \mid v \mid v \in \Gamma - \{[,]\}, e \in E\}$
- **Tupel-Codierung:** Statt $\Gamma \cup Z$ wird (mit $\Gamma := \Gamma - \{[,]\}$) $\Gamma \cup (Z \cup \{()\} \times \Gamma) \cup (\Gamma \times \{()\}) \cup (Z \times \{[,]\}) \times \Gamma \times \{1, 2, 3\}$ verwendet.
- Aus $\&s \rightarrow \epsilon$ wird $(s, [, v) \rightarrow v$ und $(s, [, v, 1) \rightarrow v$
- Aus $] \$ \rightarrow \epsilon$ wird $(a,]) \rightarrow v \quad (v \in \Gamma)$
- Aus $[u_1 \dots u_m z v_1 \dots v_n]$ wird $([, u_1) u_2 \dots u_m (z, v_1) v_2 \dots v_{n-1} (v_n,])$
- Fall (3) und (5) von P_{sim} entfallen wegen der Randrespektierung

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Nichtkürzende Grammatik \rightarrow LBA

- Informelle Beschreibung eines LBA: Bei Eingabe von w , also Startkonfiguration $s[w]$ (und später einer Konfiguration der Form $z[w]$, die nach Simulation von k Ableitungsschritten mit der Grammatik G erreicht ist), wählt LBA (nichtdet.) eine Produktion $u \rightarrow v$ und sucht ein Vorkommen von v in w . Falls ein v gefunden wird, wird es durch $u H^k$ ersetzt (Hilfssymbol H , $k = |v| - |u|$) und anschließend werden die H 's an's rechte Ende verschoben. Dies wird iteriert. Wird dabei $z[S H^m]$ erreicht, so akzeptiert LBA.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 5 Berechenbarkeit und (Un-)Entscheidbarkeit

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Intuitive Berechenbarkeit

- $f(n) =$ if n Anfangsstück von π then 1 else 0 fi
 □ $\pi = 3,141593\dots$; $f(314) = 1$; $f(453) = 0$
- $g(n) =$ if n kommt in π vor then 1 else 0 fi
 - $g(415) = 1$; $g(777) = ?$; $\forall n : g(n) = 1 ?$
- $h(n) =$ if „7“ⁿ Teilwort der Dezimaldarstellung von π then 1 else 0
- (Quelle: Schöning: Theoretische Informatik - kurz gefaßt)

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Church'sche These

- Die Klasse der Turing-berechenbaren Funktionen (äquivalent: die Klasse der
- WHILE-berechenbaren Funktionen,
- GOTO-berechenbaren Funktionen,
- μ -rekursiven Funktionen)
- stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.
- (Vgl. Jablonski: ThI1-1 Grundbegriffe, Seite 75: Church'sche These: alle Formalisierungen von „Algorithmus“ sind gleichwertig.)

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 6 Turing-Berechenbarkeit

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
 Martensstraße 3 • 91058 Erlangen



Zahlendarstellungen

- Sei n eine natürliche Zahl, von 0 verschieden
- **Unäre** Darstellung: $unär(n) = a^n$ (a Symbol)
- **Binäre (Dual-)Darstellung:**
 - $bin(n) = a_k \dots a_0 \leftrightarrow n = \sum \{a_i \cdot 2^i \mid 0 \leq i \leq k\} \wedge a_k \neq 0$
- **Dezimal-Darstellung:**
 - $dez(n) = a_k \dots a_0 \leftrightarrow n = \sum \{a_i \cdot 10^i \mid 0 \leq i \leq k\} \wedge a_k \neq 0$
- **Spezialfall $n = 0$:**
 - $unär(0) = \epsilon$; $bin(0) = „0“$; $dez(0) = „0“$

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

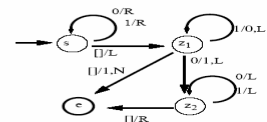
Turing-Berechenbarkeit

- **Definition:** Eine (partielle) Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ ($k > 0$) heißt **Turing-berechenbar** genau dann, wenn es eine (deterministische) Turingmaschine M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:
 - $f(n_1, \dots, n_k) = m \leftrightarrow \exists e \in E : \exists \alpha, \beta \in []^* :$
 $s \ bin(n_1) \# \ bin(n_2) \# \dots \# \ bin(n_k) \ [] \ \vdash^* \ \alpha \ e \ \bin(m) \ \beta$
- **Spezialfall $k=0$:** $f: \mathbb{N}^0 \rightarrow \mathbb{N}$, für alle $m \in \mathbb{N}$
 - $f() = m \leftrightarrow \exists e \in E : \exists \alpha, \beta \in []^* :$
 $s \ [] \ \vdash^* \ \alpha \ e \ \bin(m) \ \beta$

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel TM für die Nachfolgerfunktion

1. $\delta(s, \epsilon) = (s, \epsilon, R)$
2. $\delta(s, 1) = (z_1, 1, L)$
3. $\delta(s, 1) = (z_1, 1, L)$
4. $\delta(z_1, \epsilon) = (z_2, 1, L)$
5. $\delta(z_1, 1) = (z_1, 0, L)$
6. $\delta(z_1, 1) = (e, 1, N)$
7. $\delta(z_2, \epsilon) = (z_2, 0, L)$
8. $\delta(z_2, 1) = (z_2, 1, L)$
9. $\delta(z_2, 1) = (e, 1, R)$



- $s1101[] \vdash 1s101[] \vdash 1101s[] \vdash 110z_11[] \vdash 11z_100[]$
 $\vdash 1z_2110[] \vdash z_2[]1110[] \vdash []e1110[]$
- Diese TM berechnet die Nachfolgerfunktion $succ$ mit $succ(n) = n + 1$.

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Turingberechenbare Wortfunktionen

- **Definition:** Eine (partielle) Funktion $f: \Sigma_1^* \rightarrow \Sigma_2^*$ heißt **Turing-berechenbar** genau dann, wenn es eine (deterministische) Turingmaschine M gibt, so dass für alle $x \in \Sigma_1^*, y \in \Sigma_2^*$ gilt:
 - $f(x) = y \iff \exists e \in E : \exists \alpha, \beta \in \Sigma^* : s \times [] \vdash^* \alpha e y \beta$
- **Variante:** $\{0, 1\}$ statt Σ_2^*
- **Bemerkung:** $f(x)$ undefiniert \iff Es gibt keine in einen Endzustand führende Berechnung

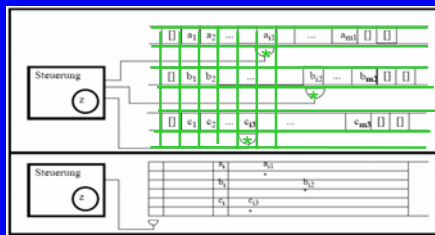
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Typ-0-Sprachen und Definitionsbereich Turing-berechenbarer Funktionen

- **Satz:** Ist L eine Typ-0-Sprache über Σ , so ist (die „halbe“ charakteristische Funktion von L) $\chi'_L = \langle x \mapsto 1 \mid x \in L \rangle : \Sigma^* \rightarrow \{0, 1\}$ Turing-berechenbar.
- **Bemerkung:** $\text{dom}(\chi'_L) = L$
- **Satz:** Ist $f: \Sigma_1^* \rightarrow \Sigma_2^*$ Turing-berechenbar, so ist $L := \text{dom}(f)$ eine Typ-0-Sprache.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

k-Band-Turing-Maschine und simulierende 1-Band-Turing-Maschine

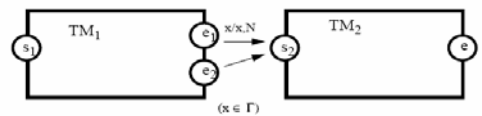


- **Satz:** Zu jeder k-Band-Turing-Maschine M gibt es eine M simulierende (gewöhnliche) Turing-Maschine M' mit gleicher Leistung.

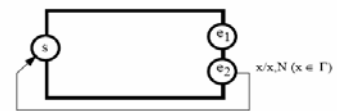
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Komposition und Rückkopplung von Turing-Maschinen

Sequentielle Komposition $TM_1 : TM_2$



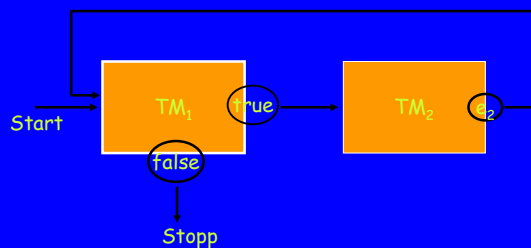
Rückkopplung



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Schleifenbildung bei TM

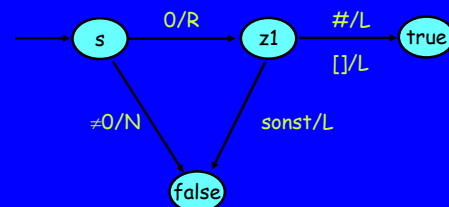
- **WHILE TM_1 DO TM_2**



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Prüf-TM „ $x = 0$?“

- $u \text{ s bin}(0) \# v \vdash^* u \text{ true bin}(0) \# v$
- $x \neq 0 \rightarrow u \text{ s bin}(x) \# v \vdash^* u \text{ false bin}(x) \# v$



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 7 LOOP-, WHILE- und GOTO-Berechenbarkeit

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Beispiel: LOOP in EBNF-Darstellung

- $\langle \text{Konstante} \rangle ::= 0 \mid \langle \text{Pos. Ziffer} \rangle \{ \langle \text{Ziffer} \rangle \}$
- $\langle \text{Pos. Ziffer} \rangle ::= 1|2|3|4|5|6|7|8|9$
- $\langle \text{Ziffer} \rangle ::= \langle \text{Pos. Ziffer} \rangle \mid 0$
- $\langle \text{Variable} \rangle ::= x \langle \text{Konstante} \rangle$
- $\langle \text{Wertzuweisung} \rangle ::=$
 $\langle \text{Variable} \rangle := \langle \text{Variable} \rangle (+ \mid -) \langle \text{Konstante} \rangle$
- $\langle \text{Programm} \rangle ::= \langle \text{Wertzuweisung} \rangle$
 $\mid \langle \text{Programm} \rangle ; \langle \text{Programm} \rangle$
 $\mid \text{LOOP } \langle \text{Variable} \rangle \text{ DO } \langle \text{Programm} \rangle \text{ END}$

Beispiel-Programm:

- $x_0 := x_1 + 0 ; \text{LOOP } x_2 \text{ DO } x_0 := x_0 + 1 \text{ END}$

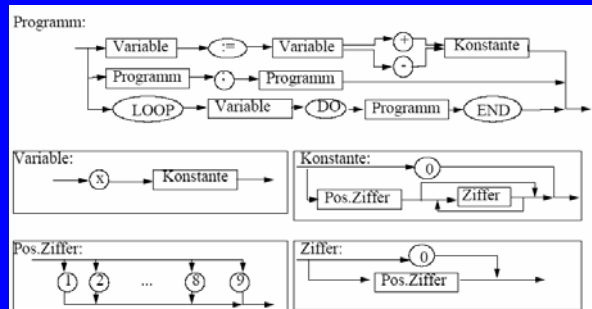
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LOOP-Programme (1):

- (1) $x_i := x_j + c$ ($i, j, c \in \mathbb{N}$)
- (2) $x_i := x_j - c$ ($i, j, c \in \mathbb{N}$)
 - ($x - y := \text{if } x > y \text{ then } x - y \text{ else } 0 \text{ fi}$)
- (3) $P_1 ; P_2$
 - (Komposition: Erst P_1 , dann P_2)
- (4) **LOOP** x_i **DO** P **END**
 - (Führe P x_i -mal aus.)
- **Definition:** Das Programm P *berechnet* $f^{(k)} : \mathbb{N}^k \rightarrow \mathbb{N}$, wenn die P -Berechnung von der Startsituation mit „ $x_i = n_i$ ($i=1, \dots, k$)“ zur Endsituation mit „ $x_0 = f^{(k)}(n_1, \dots, n_k)$ “ führt.
- **Definition:** $f^{(k)} : \mathbb{N}^k \rightarrow \mathbb{N}$ ist *LOOP-berechenbar*, wenn es ein LOOP-Programm gibt, das $f^{(k)}$ berechnet.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Programmiersprache LOOP in Syntax-Diagramm-Form



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LOOP-Programme (2)

- **add** ist LOOP-berechenbar (Makro $x_i := x_j + x_k$)
- **Weitere Makros:**
 - $x_i := x_j$ für $x_i := x_j + 0$
 - $x_i := 0$ für $\text{LOOP } x_i \text{ DO } x_i := x_i - 1 \text{ END}$
 - $x_i := c$ für $x_i := 0 ; x_i := x_i + c$
 - **IF** $x_i = 0$ **THEN** P **END** für
 $y := 1 ; \text{LOOP } x_i \text{ DO } y := 0 \text{ END} ; \text{LOOP } y \text{ DO } P \text{ END}$
 - **mult** ist LOOP-berechenbar (Makro $x_0 := x_1 \cdot x_2$):
 $x_0 := 0 ; \text{LOOP } x_2 \text{ DO } x_0 := x_0 + x_1 \text{ END}$
 - **minus** ist LOOP-berechenbar (Makro $x_0 := x_1 - x_2$):
 $x_0 := x_1 ; \text{LOOP } x_2 \text{ DO } x_0 := x_0 - 1 \text{ END}$
 - Statt „**IF** $x_i = 0$ “ auch „**IF** $x_i \neq 0$ “, „**IF** $x_i = c$ “, „**IF** $x_i = x_j$ “ möglich

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

WHILE -Programme

- (1) $x_i := x_j + c$ ($i, j, c \in \mathbb{N}$)
- (2) $x_i := x_j - c$ ($i, j, c \in \mathbb{N}$)
- (3) $P_1 ; P_2$
- (4) **LOOP** x_i **DO** P **END**
- (5) **WHILE** $x_i \neq 0$ **DO** P **END** (**WHILE-Schleife**)
- **Definition:** Das Programm P *berechnet* $f^{(k)} : \mathbb{N}^k \rightarrow \mathbb{N}$, wenn die P -Berechnung von der Startsituation mit „ $x_i = n_i$ ($i=1, \dots, k$)“ zur Endsituation mit „ $x_0 = f^{(k)}(n_1, \dots, n_k)$ “ führt.
- **Definition:** $f^{(k)} : \mathbb{N}^k \rightarrow \mathbb{N}$ ist *WHILE-berechenbar*, wenn es ein WHILE-Programm gibt, das $f^{(k)}$ berechnet.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LOOP vs. WHILE

- Jede LOOP-berechenbare Funktion ist WHILE-berechenbar.
- Es gibt (totale) WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.
- LOOP-berechenbare Funktionen sind total.
- WHILE-berechenbare Funktionen können echt partiell sein.
- Auf LOOP-Schleifen (4) kann man in WHILE-Programmen verzichten.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

WHILE-Programm für div

- $x \text{ div } y = z \iff y \cdot z \leq x < y \cdot (z + 1)$
 $\iff \exists r(x = y \cdot z + r \wedge 0 \leq r < y)$
 (wobei $r = x \text{ mod } y$)
- P_1 : $r := x; z := 0;$
WHILE $r \geq y$ **DO** $z := z + 1; r := r - y$ **END**
 $r \geq y \iff r+1-y \neq 0$; $h = r+1-y$
- P_2 : $r := x; z := 0; h := r+1; h := h-y;$
WHILE $h \neq 0$ **DO** $z := z + 1; r := r-y; h := h-1; h := h-y$ **END**
- P_3 : $z := 0; x := x+1; x := x-y;$ *kann entfallen*
WHILE $x \neq 0$ **DO** $z := z + 1; x := x-y$ **END**

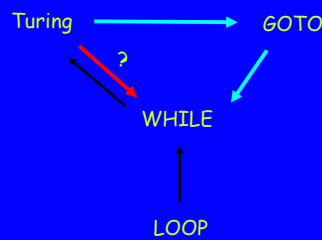
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

WHILE-berechenbar \rightarrow Turing-berechenbar

- **Satz:** Jede WHILE-berechenbare Funktion ist Turing-berechenbar.
- **Beweis-Skizze:** WHILE-Berechnungen werden auf einer Mehrband-Turing-Maschine simuliert. Für jede im Programm vorkommende Variable x_i wird ein Band verwendet. WHILE-Schleifen lassen sich durch Komposition und Rückkopplung von TM implementieren.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Berechenbarkeit - Übersicht (1)



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

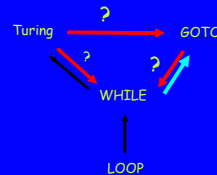
GOTO-Programme

- **GOTO-Programme :**
 - Marke₁: Anweisung₁ ; ... ; Marke_m: Anweisung_m
- **Anweisungen:**
 - Wertzuweisungen:
 - (1) $x_i := x_j + c$ ($i, j, c \in \mathbb{N}$),
 - (2) $x_i := x_j - c$ ($i, j, c \in \mathbb{N}$),
 - Unbedingte Sprünge: GOTO Marke ,
 - Bedingte Sprünge: IF $x_i = c$ THEN GOTO Marke ,
 - Stop-Anweisung: HALT
- **Marken:** (Zahl-) Wörter

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

WHILE-Programme \rightarrow GOTO-Programme

- **Satz:** Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.
- **Folgerung:** Jede WHILE-berechenbare Funktion ist GOTO-berechenbar.



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

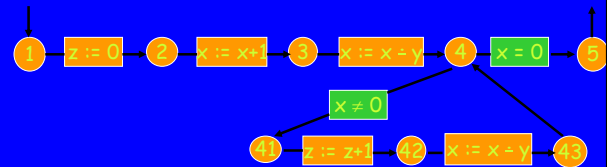
Beispiel für „WHILE → GOTO“

- **WHILE-Programm P_3 :**
 - $z := 0 ; x := x + 1 ; „x := x - y“ ;$
 - **WHILE $x \neq 0$ DO $z := z + 1 ; „x := x - y“$ END**
- **GOTO-Programm:**
 - 1: $z := 0 ; 2: x := x + 1 ; 3: „x := x - y“ ;$
 - 4: **IF $x = 0$ THEN GOTO 5 ;**
 - 41: $z := z + 1 ; 42: „x := x - y“ ; 43: GOTO 4 ;$
 - 5: HALT

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

GOTO-Programme als gerichtete Graphen

- 1: $z := 0 ; 2: x := x + 1 ; 3: „x := x - y“ ;$
- 4: **IF $x = 0$ THEN GOTO 5 ;**
- 41: $z := z + 1 ; 42: „x := x - y“ ; 43: GOTO 4 ;$
- 5: HALT



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

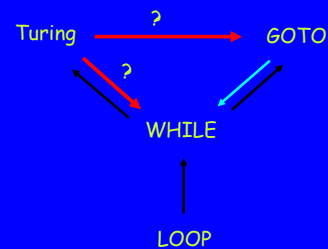
GOTO → WHILE

- **Satz:** Jedes GOTO-Programm kann durch ein WHILE-Programm (mit nur einer WHILE-Schleife) simuliert werden.
- **Beweis-Skizze:**
 - GOTO-Programm P:
 - 1: $A_1 ; \dots ; m: A_m$
 - WHILE-Programm P':
 - $bz := 1 ;$
 - WHILE $bz \neq 0$ DO**
 - IF $bz = 1$ THEN A'_1 END ; ...**
 - IF $bz = m$ THEN A'_m END**
 - END**

A_i	A'_i
Wert-zuweisung	$A_i ; bz := bz + 1$
$GOTO M_j$	$bz := j$
IF $x_j = c$ THEN $GOTO M_k$	IF $x_j = c$ THEN $bz := k$ ELSE $bz := bz + 1$
HALT	$bz := 0$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Berechenbarkeit - Übersicht (2)



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kleene's Normalform für WHILE-Programme

- **Satz (Kleene):** Jede WHILE-berechenbare Funktion kann durch ein WHILE-Programm mit nur einer WHILE-Schleife berechnet werden.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Turing-Maschinen → GOTO-Programme(1)

- **Satz:** (det.) Turing-Maschinen sind durch GOTO-Programme simulierbar.
- **Satz:** Jede Turing-berechenbare Funktion ist GOTO-berechenbar.
- **Beweis-Skizze:**
 - Für $\Gamma = \{a_0, \dots, a_{m-1}\}$ sei $c: \Gamma \rightarrow \mathbb{N}$ gegeben durch $c(a_i) = i$ (speziell sei $c(\text{blank}) = 0$) und
 - für $b \geq m$ sei $\text{code}: \Gamma^* \rightarrow \mathbb{N}$ gegeben durch
 - $\text{code}(x_n \dots x_0) = (c(x_n) \dots c(x_0))_b = \sum \{c(x_i) * b^i \mid i = 0, \dots, n\}$
 - Codierung($u z v$) = (code(u), z , code(v^{rev}))

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Turing-Maschinen → GOTO-Programme(2)

	Turingmaschine	GOTO-Programm
Konfiguration	$u z v (u, v \in \Gamma^*)$	Marke M_z , $x = \text{code}(u)$, $y = \text{code}(v^{\text{reverse}})$
Links-Schritt	$\delta(z, a_i) = (z', a_j, L)$	$P(z, a_i)$: $y := y - i + j$; $y := y * b + x \text{ MOD } b$; $x := x \text{ DIV } b$
Rechts-Schritt	$\delta(z, a_i) = (z', a_j, R)$...(analog)
Schritt ohne Kopfbewegung	$\delta(z, a_i) = (z', a_j, N)$...(analog)

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Turing-Maschinen → GOTO-Programme(3)

- P_1 : Berechne aus (x_1, \dots, x_k) die Codierung der Startkonfig.: $(0, M_s, \text{code}(\text{bin}(x_1)\#\dots\#\text{bin}(x_k)[\])^{\text{rev}})$

P_2 : Simulations-GOTO-Programm:
(Startmarke M_1 sei M_s , wobei s der Startzustand der Turing-Maschine ist.)

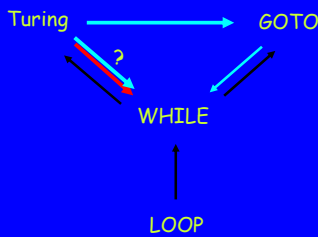
```

Mz: a := y MOD b;      (für alle z ∈ Z-E)
  IF a=0 THEN GOTO Mz,0;
  IF a=1 THEN GOTO Mz,1;
  ....
  IF a=m-1 THEN GOTO Mz,m-1;
Mz,i: P(z,ai);      (für alle z ∈ Z-E und i=0,...,m-1)
  GOTO Mz;
Me: HALT              (für alle e ∈ E)
    
```

- P_3 : Decodiere aus $(0, M_e, \text{code}(\text{bin}(r)^{\text{rev}}))$ das Resultat $x_0 = r$.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Berechenbarkeit - Übersicht (3)



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 8 Primitiv-rekursive Funktionen

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Primitiv-rekursive Funktionen

- Abkürzung: $\underline{x} = x_1, \dots, x_n$

$h = \text{Komposition}(f_1, \dots, f_m, g)$:
$h(\underline{x}) = g(f_1(\underline{x}), \dots, f_m(\underline{x}))$
$f = \text{Primitive_Rekursion}(g, l)$:
$f(\underline{x}, 0) = g(\underline{x})$
$f(\underline{x}, y+1) = h(\underline{x}, y, f(\underline{x}, y))$
Basisfunktionen:
0 $= \langle x \mapsto 0 \rangle$
succ $= \langle x \mapsto x + 1 \rangle$
$\text{pr}_k^{(n)}$ $= \langle x \mapsto x_k \rangle$ ($1 \leq k \leq n$)

- Definition:** Die Klasse der *primitiv-rekursiven Funktionen* enthält die Basisfunktionen und ist abgeschlossen unter *Komposition* und *primitiver Rekursion*.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiele primitiv-rekursiver Funktionen (1)

- Konstante Funktion $c^{(n)} = \langle x_1, \dots, x_n \rightarrow c \rangle : \mathbb{N}^n \rightarrow \mathbb{N}$
- n -stellige Nullfunktion $Z_n = \langle x_1, \dots, x_n \rightarrow 0 \rangle$
 - Z_0 Basisfunktion ; $Z_1(0) = 0$; $Z_1(y+1) = Z_1(y)$
 - $Z_n = \text{Komposition}(pr_1^{(n)}, Z_1)$
 - $Z_n(x_1, \dots, x_n) = Z_1(pr_1^{(n)}(x_1, \dots, x_n))$ ($n > 1$)
- $\text{add} = \text{Primitive_Rekursion}(pr_1^{(1)}, \text{Komposition}(pr_3^{(3)}, \text{succ}))$
 - $\text{add}(x, 0) = x$; $\text{add}(x, y+1) = \text{succ}(\text{add}(x, y))$
- $\text{mult} = \text{Primitive_Rekursion}(Z_1, \text{Komposition}(pr_3^{(3)}, pr_1^{(3)}, \text{add}))$
 - $\text{mult}(x, 0) = 0$; $\text{mult}(x, y+1) = \text{add}(\text{mult}(x, y), x)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiele primitiv-rekursiver Funktionen (2)

- fak = Primitive_Rekursion(1⁽⁰⁾, Komposition(pr₁⁽²⁾, Komposition(pr₁⁽²⁾, succ), mult))
 - fak(0) = 1; fak(y+1) = mult(fak(y), y+1)
- Signum-Funktion sg = (x → if x > 0 then 1 else 0) : N → N
 - sg(0) = 0; sg(y+1) = 1 (Primitive_Rekursion)
- Vorgängerfunktion pred = (x → if x = 0 then 0 else x-1)
 - pred(0) = 0; pred(y+1) = y
- Modifizierte Differenz
 - monus = (x → if x > y then x-y else 0) : N² → N;
 - monus(x, y) = x - y
 - x - 0 = x; x + (y+1) = pred(x - y)

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiele primitiv-rekursiver Funktionen (3)

- sumbis(n) = 1 + 2 + ... + n (= n(n+1)/2)
 - sumbis(0) = 0; sumbis(n+1) = sumbis(n) + (n+1)
- Paarfunktion paarcod : N² → N
- paarcod ist bijektiv.
- Es gibt primitiv-rekursive Funktionen q₁ : N → N und q₂ : N → N mit
 - q_i(paarcod(x₁, x₂)) = x_i (i=1,2)
 - paarcod(q₁(x), q₂(x)) = x

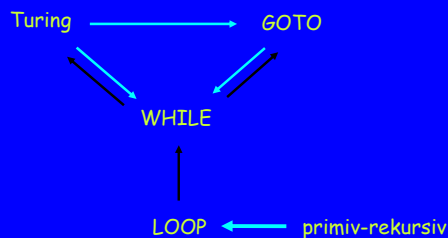
- paarcod⁻¹ = (q₁, q₂)
- paarcod(x, 0) = sumbis(x)
- paarcod(x, y) = sumbis(x+y) + y = (x+y)(x+y+1)/2 + y

y \ x	0	1	2	3
0	0	1	3	6
1	2	4	7	
2	5	...		
3				

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

primitiv-rekursiv → LOOP

- Satz:** Jede primitiv-rekursive Funktion ist LOOP-berechenbar.



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Primitiv-rekursive Relationen

- Definition:** Eine Relation $R \subseteq \mathbb{N}^n$ ist primitiv-rekursiv genau dann, wenn ihre charakteristische Funktion χ_R primitiv rekursiv ist.
- Definition:** S entsteht durch beschränkte Quantifizierung aus R, wenn gilt:
 - $S(x, z) \leftrightarrow \exists y \leq z : R(x, y, z)$ ($Q \in \{\forall, \exists\}$)
- Satz:** Die Klasse der primitiv-rekursiven Relationen ist abgeschlossen unter Durchschnitt, Vereinigung, Komplement und beschränkter Quantifizierung.
- Satz:** Die Klasse der primitiv-rekursiven Funktionen ist abgeschlossen unter Fallunterscheidung.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Simultane primitive Rekursion

$f_1, \dots, f_k : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ (Abkürzung: $\underline{x} = x_1, \dots, x_n$)

entstehen durch simultane primitive Rekursion aus

$g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$ und $h_1, \dots, h_k : \mathbb{N}^{n+k+1} \rightarrow \mathbb{N}$, wenn gilt:

$$f_i(\underline{x}, 0) = g_i(\underline{x}) \quad (i = 1, \dots, k)$$

$$f_i(\underline{x}, y+1) = h_i(\underline{x}, y, f_1(\underline{x}, y), \dots, f_k(\underline{x}, y)) \quad (i = 1, \dots, k)$$

Satz: Die Klasse der primitiv-rekursiven Funktionen ist abgeschlossen unter simultaner primitiver Rekursion.

Codierung: $C_k : \mathbb{N}^k \rightarrow \mathbb{N}$; Decodierung: $D_{k,i} : \mathbb{N} \rightarrow \mathbb{N}$ ($i = 1, \dots, k$)

$$C_k(D_{k,1}(x), \dots, D_{k,k}(x)) = x; D_{k,i}(C_k(x_1, \dots, x_k)) = x_i \quad (i = 1, \dots, k)$$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LOOP → primitiv-rekursiv (1)

- Satz:** Jede LOOP-berechenbare Funktion ist primitiv rekursiv.
- Beweis-Skizze: (Induktion über den Aufbau von LOOP-Programmen)
 - Sei $m \geq \max \{ i \mid x_i \text{ kommt in } P \text{ vor} \}$,
 - $\underline{x} = x_0, \dots, x_m$
 - Seien Funktionen $f_{p,i}$ definiert durch:

$$\underline{x} \vdash_p^* \underline{x}' \leftrightarrow \forall i : x'_i = f_{p,i}(\underline{x})$$
 - $f^{(k)}_P(x_1, \dots, x_k) = f_{p,0}(0, x_1, \dots, x_k, 0, \dots, 0)$

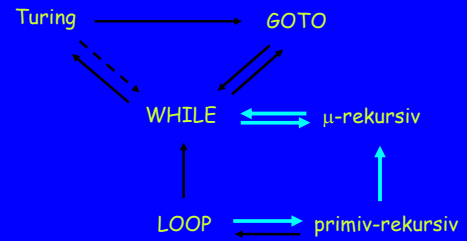
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

LOOP → primitiv-rekursiv (2)

- 4. Alle $f_{p,i}$ sind primitiv-rekursiv.
 - $P = "x_i := x_j + c"$: $f_{p,i}(x) = x_j + c$; $f_{p,r}(x) = x_r$ ($r \neq i$)
 - $P = "x_i := x_j - c"$: $f_{p,i}(x) = x_j - c$; $f_{p,r}(x) = x_r$ ($r \neq i$)
 - $P = "P1 ; P2"$:
 $f_{(P1;P2),i}(x) = f_{P2,i}(f_{P1,0}(x), \dots, f_{P1,m}(x))$
 - $P = "LOOP x_i DO P' END"$:
 $f_{p,i}(x) = g_i(x, x_i)$
 $g_i(x, 0) = x_i$; $g_i(x, y+1) = f_{p,i}(g_0(x, y), \dots, g_m(x, y))$
- Anmerkung: $(g_0(x, y), \dots, g_m(x, y))$ beschreibt die Registerinhalte nach y Durchläufen von P' .

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Berechenbarkeit - Übersicht (4)



Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 9 μ-Operator, μ-rekursive Funktionen

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
 Martensstraße 3 • 91058 Erlangen



μ-Operator

- Kleinste Nullstelle von $f : \mathbb{N} \rightarrow \mathbb{N}$:
 $\min \{ y \mid f(y) = 0 \}$
- Kleinste Nullstelle im 1. Argument von $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$:
 $\min \{ y \mid f(y, x_1, \dots, x_k) = 0 \}$,
 - abhängig von den Parametern $\underline{x} = x_1, \dots, x_k$
- **Definition:** Sei $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ eine (partielle) Funktion. Die Funktion $g : \mathbb{N}^k \rightarrow \mathbb{N}$ entsteht durch Anwendung des (unbeschränkten) μ -Operators auf f , wenn gilt (kurz $g = \mu f$):
 $\forall \underline{x} \in \mathbb{N}^k : g(\underline{x}) = \min \{ y \mid f(y, \underline{x}) = 0 \wedge \forall y' < y : (y', \underline{x}) \in \text{dom}(f) \}$
 - Schreibweise: $(\mu f)(\underline{x}) = \mu y [f(y, \underline{x}) = 0]$

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beschränkter μ-Operator

- **Definition:** Sei $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ eine (partielle) Funktion. Die Funktion $g : \mathbb{N}^k \rightarrow \mathbb{N}$ entsteht durch Anwendung des beschränkten μ -Operators auf f , wenn gilt (kurz $g = \# f$):
 $\forall (z, \underline{x}) \in \mathbb{N}^{k+1} : g(z, \underline{x}) = \text{if } M \neq \emptyset \text{ then } \min M \text{ else } z+1$, wobei
 $M = \{ y \mid y \leq z \wedge f(y, \underline{x}) = 0 \wedge \forall y' < y : (y', \underline{x}) \in \text{dom}(f) \}$
- **Schreibweise:** $(\# f)(\underline{x}) = \mu y \leq z [f(y, \underline{x}) = 0]$
- **Satz:** Die Klasse der primitiv-rekursiven Funktionen ist abgeschlossen unter Anwendung des beschränkten μ -Operators.

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Umkehrfunktion zu paarcod

- $\text{paarcod}(x,y) = \text{sumbis}(x+y) + y$
 $= (x+y)(x+y+1)/2 + y$
- Es gibt primitiv-rekursive Funktionen $q_1 : \mathbb{N} \rightarrow \mathbb{N}$ und $q_2 : \mathbb{N} \rightarrow \mathbb{N}$ mit
 $q_1(\text{paarcod}(x_1, x_2)) = x_1$ ($i=1,2$)
 $\text{paarcod}(q_1(z), q_2(z)) = z$
- **Hilfssatz:** $x \leq \text{paarcod}(x,y)$, $y \leq \text{paarcod}(x,y)$
- $q_1(z) = \mu x \leq z [\exists y \leq z : \text{paarcod}(x,y) = z]$
- $q_2(z) = \mu y \leq z [\exists x \leq z : \text{paarcod}(x,y) = z]$

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

μ -rekursive Funktionen (1)

- Definition:** Die Klasse der μ -rekursiven Funktionen wird induktiv definiert durch:
 - Basisfunktionen sind die primitiv-rekursiven Basisfunktionen.
 - Die Klasse ist abgeschlossen unter Komposition, primitiver Rekursion und Anwendung des (unbeschränkten) μ -Operators.
- Satz:** Die Klasse der μ -rekursiven Funktionen ist identisch mit der Klasse der WHILE-berechenbaren Funktionen.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

μ -rekursive Funktionen (2)

- Satz (Kleene-Normalform):** Für jede μ -rekursive Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ gibt es zwei primitiv-rekursive Funktionen $p : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$, $q : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ mit
 - $\forall \underline{x} \in \mathbb{N}^k : f(\underline{x}) = p(\underline{x}, (\mu q)(\underline{x}))$
- Beweis-Skizze:** Einmalige Anwendung von WHILE liefert eine Anwendung des μ -Operators.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 10 Ackermann-Funktionen

Ackermann-Funktion $ack: \mathbb{N}^2 \rightarrow \mathbb{N}$

$$ack(x, y) = \begin{cases} y + 1 & \text{wenn } x = 0 \\ ack(x - 1, 1) & \text{wenn } x > 0 \text{ und } y = 0 \\ ack(x - 1, ack(x, y - 1)) & \text{sonst (also } x > 0 \text{ und } y > 0) \end{cases}$$

Ackermannfunktion $a: \mathbb{N}^2 \rightarrow \mathbb{N}$

$$a(x, y) = \begin{cases} 1 & \text{wenn } x = 0 \text{ oder } y = 0 \\ 3y + 1 & \text{wenn } x = 1 \text{ (und } y > 0) \\ \frac{a(x-1, a(x-1, \dots, a(x-1, y) \dots))}{y \text{ mal } 'a(x-1, \dots, x > 1} & \text{sonst (also } x > 1 \text{ und } y > 0) \end{cases}$$

- Beispiel:** $a(2,3) = a(1,a(1,a(1,3))) = a(1,a(1,10)) = a(1,31) = 94$
- Bemerkung:** ack und a sind total und intuitiv berechenbar.
- Lemma:** Für jedes LOOP-Programm P , das nur die Variablen x_0, \dots, x_m benutzt, gibt es ein $k \in \mathbb{N}$ mit $f_P(n) \leq a(k, n)$ für alle $n \geq k$.
 - Dabei ist $f_P(n)$ die maximale Summe der Endwerte in x_0, \dots, x_m für Berechnungen von P mit Anfangswerten, deren Summe höchstens n ist.
- Satz:** ack und a sind nicht LOOP-berechenbar (also auch nicht primitiv-rekursiv).

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

WHILE-Programm mit Stapeloperationen für die Ackermannfunktion a

```

INPUT(x, y); (* Eingabe der Argumente x = a, y = b *)
INIT(stack); (* Erzeuge leeren Stapel: stack = [] *)
PUSH(x, stack) (* stack = [a] *); PUSH(y, stack) (* stack = [b, a] *);
WHILE size(stack) ≠ 1 DO
    (* stack = [xk-1, ..., x1] entspricht a(x1, ..., a(xk-1, xk), ...) *)
    y := POP(stack) (* y = xk *); x := POP(stack) (* x = xk-1 *);
    IF (x = 0) OR (y = 0) THEN PUSH(1, stack)
    ELSE IF x=1 THEN PUSH(3*y+1, stack)
    ELSE LOOP y DO PUSH(x-1, stack) END;
    PUSH(y, stack) (* stack = [y, x-1, ..., x-1, xk-2, ..., x1] entspricht
    a(x1, ..., a(xk-2, a(x-1, a(x-1, ..., a(x-1, y) ...)) = a(x1, ..., a(xk-2, a(xk-1, xk), ...) *)
    END
    END
END; result := POP(stack); OUTPUT(result)
    
```

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Simulation von Stapeln durch WHILE-Programme

- Stapel $[x_0, \dots, x_{l-1}]$ entsprechen endliche Folgen über \mathbb{N} , also \mathbb{N}^* .
- $c = \langle x_1, x_2 \rightarrow \text{paarcod}(x_1, x_2) + 1 \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$
- $c_i = \langle z \rightarrow q_i(z \div i) \rangle : \mathbb{N} \rightarrow \mathbb{N}$ ($i = 1, 2$)
- $\text{stack} = [x_0, \dots, x_{l-1}]$
- $\text{cod}(\text{stack}) = st$
- $\text{cod}([x_0, \dots, x_{l-1}]) = c(x_l, c(\dots, c(x_1, 0) \dots))$

INIT(stack);	st := 0
PUSH(x, stack)	st := c(x, st)
POP(stack)	result := c ₁ (st); st := c ₂ (st)
size(stack) ≠ 1	c ₂ (st) ≠ 0

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 11 Halteproblem, (Un-)Entscheidbarkeit, Reduktion

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Martensstraße 3 • 91058 Erlangen



(semi-)entscheidbar

Charakteristische Funktion ($M \subseteq G$)

$$\chi_M(x) = \begin{cases} 1 & \text{wenn } x \in M \\ 0 & \text{wenn } x \notin M \end{cases}$$

„halbe charakteristische Funktion“:

$$\chi'_M(x) = \begin{cases} 1 & \text{wenn } x \in M \\ \text{undefiniert} & \text{wenn } x \notin M \end{cases}$$

- **Definition:**
- M ist *entscheidbar*, wenn χ_M berechenbar ist.
- M ist *semi-entscheidbar*, wenn χ'_M berechenbar ist.

Beispiele entscheidbarer Mengen

- $G = \mathbb{N}$: $M_1 =$ Menge aller Primzahlen
- $G = \Sigma^*$: $M_2 = L(\text{Gram})$ mit kontextfreier Grammatik Gram (Wortproblem)
- $G = \mathbb{N}^2$: $M_3 = \{ (x, y) \mid x < y \}$ (Kleiner-Relation)
- **Anmerkung:** M_1 und M_3 sind sogar primitiv-rekursiv.

Form von Entscheidungsproblemen

- **Gegeben:** Form der Argumente x ; Beschreibung der Grundmenge G
- **Entscheide:** eigenschaft $E(x)$; Beschreibung der Menge $M = \{ x \in G \mid E(x) \}$
- **Beispiel:** (Schnitt-Problem für Grammatiken)
 - Gegeben: Paar von Grammatiken (G_1, G_2) eines Typs i
 - Entscheide: $L(G_1) \cap L(G_2) = \emptyset$

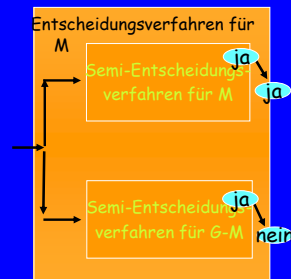
entscheidbar - semi-entscheidbar(1)

- **Bemerkung:** Jede entscheidbare Menge ist semi-entscheidbar.



entscheidbar - semi-entscheidbar(2)

- **Satz:** Die Klasse der entscheidbaren Mengen ist abgeschlossen unter Durchschnitt, Vereinigung und Komplementbildung.
- **Satz:** M ist entscheidbar genau dann, wenn sowohl M als auch $(G - M)$ semi-entscheidbar sind.



rekursiv-aufzählbar

- **Definition:** M ist *rekursiv-aufzählbar*, wenn $M = \emptyset$ oder M Bildmenge einer totalen μ -rekursiven Funktion ist.
- **Satz:** Eine Menge $M \subseteq \mathbb{N}$ ist genau dann rekursiv-aufzählbar, wenn sie semi-entscheidbar ist.
- **Satz:** Die Bildmenge $\{ f(x) \mid x \in \text{dom}(f) \}$ einer (partiellen) berechenbaren Funktion f ist semi-entscheidbar/rekursiv-aufzählbar.

Charakterisierungen von Typ-0-Sprachen

- Satz:** Für Sprachen L sind folgende Aussagen äquivalent:
 - L ist semi-entscheidbar,
 - L ist rekursiv-aufzählbar,
 - L ist Typ-0-Sprache,
 - L wird durch eine Turingmaschine akzeptiert,
 - L ist Definitionsbereich einer μ -rekursiven Funktion,
 - L ist Bildmenge einer (partiellen) μ -rekursiven Funktion

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Wort-Codierung (Gödelisierung) von Turing-Maschinen

- O.B.d.A. sei M eine Turing-Maschine mit $Z = \{z_0, \dots, z_n\}$, $\Gamma = \{a_0, \dots, a_k\}$ mit $a_0 = \square$, $a_1 = 0$, $a_2 = 1$, $a_3 = \#$ und Startzustand $s = z_0$
- Codierung der Kopfbewegungen: $r_0 = L, r_1 = R, r_2 = N$
- Codierung von Zahlen-tupeln:
 - $cod_3(x_1, \dots, x_n) = \# \# \text{bin}(x_1) \# \dots \# \text{bin}(x_n)$
- Codierung von δ : $cod_3(\delta) =$ Konkatenation aller $cod_3(i, j, i', m)$ mit $\delta(z_i, a_j) = (z_{i'}, a_j, r_m)$
- Codierung von $E = \{z_{e_1}, \dots, z_{e_s}\}$: $cod_3(E) := cod_3(e_1, \dots, e_s)$
- Codierung der Turing-Maschine M über $\{0, 1, \#\}$:
 - $cod_3(M) := cod_3(E) cod_3(\delta)$
- Codierung der Turing-Maschine M über $\{0, 1\}$: $cod(M) := cod_2(cod_3(M))$
 - mit $cod_2(0) = 00, cod_2(1) = 01, cod_2(\#) = 11,$
 $cod_2(x_1 \dots x_n) = cod_2(x_1) \dots cod_2(x_n)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Gödelnummer, M_w, φ_w

- $Gödelnummer(M) = g \leftrightarrow cod(M) = \text{bin}(g)$
- Definition:** Jedem $w \in \{0, 1\}^*$ wird eine Turingmaschine M_w zugeordnet. Ist $w = cod(M)$ so ist $M_w = M$, sonst ist M_w eine fest gewählte Turingmaschine M_0 .
- Definition:** Mit $\varphi_w : \{0, 1\}^* \rightarrow \{0, 1\}^*$ wird die von M_w berechnete Wortfunktion bezeichnet

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel: Gödelisierung der TM für die Nachfolgerfunktion

TM = $(Z = \{s, z_1, z_2, e\}, \Gamma = \{0, 1, \square\}, \delta, s, E = \{e\}, \square)$

- $\delta(s, 0) = (s, 0, R)$
- $\delta(s, 1) = (s, 1, R)$
- $\delta(s, \square) = (z_1, \square, L)$
- $\delta(z_1, 0) = (z_1, 0, L)$
- $\delta(z_1, 1) = (z_1, 1, L)$
- $\delta(z_1, \square) = (z_2, \square, L)$
- $\delta(z_2, 0) = (z_2, 0, L)$
- $\delta(z_2, 1) = (z_2, 1, L)$
- $\delta(z_2, \square) = (e, \square, R)$

$s = z_0, e = z_3$
 $a_0 = \square, a_1 = 0, a_2 = 1$

- $cod_3(\delta 4) = cod_3(1, 1, 2, 0) = \# \# 1 \# 1 \# 10 \# 10 \# 0$
- $cod_3(\delta) = cod_3(\delta 1) \dots cod_3(\delta 4) \dots cod_3(\delta 9)$
 $= \# \# \dots \# \# 1 \# 1 \# 10 \# 10 \# 0 \# \# \dots \# \# 10 \# 0 \# 11 \# 0 \# 1$
- $cod_3(E) = \# \# 11$
- $cod_3(TM) = \# \# 1 1 \# \# \dots \# \# 1 \# 1 \# 10 \# 10 \# 0 \# \# \dots \# \# 10 \# 0 \# 11 \# 0 \# 1$
- $cod(TM) = 11 11 01 01 11 11 \dots \dots 11110100110011010111001101$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Selbstanwendbarkeitsproblem

- Selbstanwendbarkeitsproblem:**
 - $K := \{w \in \{0,1\}^* \mid M_w \text{ , angesetzt auf } w, \text{ terminiert}\}$
- Satz:** K ist unentscheidbar.
- Beweisskizze: Angenommen, K sei entscheidbar mit Turingmaschine M_K . Konstruiere M' mit $cod(M') = w'$:

```

    Start → MK → Prüfe auf „0“ → ja → Halt
                    ↖      ↗
                    nein
    
```

Für alle $w \in \{0,1\}^*$:
 $M' - \text{angesetzt auf } w - \text{terminiert} \leftrightarrow M_w - \text{angesetzt auf } w - \text{terminiert nicht}$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Reduktion

- Definition:** Seien A, B Sprachen über Σ . A heißt auf B **reduzierbar** (kurz: $A \leq B$) genau dann, wenn es eine totale berechenbare Funktion $f: \Sigma^* \rightarrow \Sigma^*$ gibt, so dass für alle $x \in \Sigma^*$ gilt:

$$x \in A \leftrightarrow f(x) \in B$$
- Satz:** Ist A auf B reduzierbar, so gilt:
 - Ist B entscheidbar, so ist auch A entscheidbar.
 - Ist B semi-entscheidbar, so ist auch A semi-entscheidbar.
- Folgerung:** Ist A unentscheidbar und auf B reduzierbar, so ist auch B unentscheidbar.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Halteproblem

- **Definition: (allgemeines) Halteproblem:**
 $H := \{ w \# x \mid M_w, \text{ angesetzt auf } x, \text{ terminiert} \}$
 $(w, x \in \{0, 1\}^*)$
- **Satz:** Das allgemeine Halteproblem H ist unentscheidbar.
- **Beweisskizze: Reduktion von K auf H ($K \leq H$) mit**
 $f = \langle w \rightarrow w \# w \rangle : \{0, 1, \#\}^* \rightarrow \{0, 1, \#\}^*$
 $x \in A \leftrightarrow f(x) \in B$
 $w \in K \leftrightarrow f(w) \in H$

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Halteproblem auf leerem Band

- **Definition: Halteproblem auf leerem Band:**
 $H_0 := \{ w \mid M_w, \text{ angesetzt auf leerem Band, terminiert} \}$
- **Satz:** H_0 ist unentscheidbar.
- **Beweisskizze: Reduktion von H auf H_0 ($H \leq H_0$) mit**
 $f = \langle y \rightarrow \text{if } y = w \# x \text{ then } w' \text{ else } \text{cod}(M_w) \text{ fi} \rangle : \{0, 1, \#\}^* \rightarrow \{0, 1, \#\}^*$
 $(w, x \in \{0, 1\}^*)$, wobei
 - $w' = \text{cod}(M')$ Codewort einer Turingmaschine M' ist, die folgendes leistet: M' schreibt zunächst x auf das (zu Beginn leere) Band und verhält sich anschließend wie M_w (angesetzt auf x)
 - M_w eine Turingmaschine ist, die für keine Eingabe terminiert

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Theorem von Rice

- **Satz:** Sei R die Klasse aller Turing-berechenbaren Funktionen und S eine nichtleere, echte Teilmenge von R ($\emptyset \neq S \subset R$). Dann ist die Sprache $C(S) = \{ w \in \{0, 1\}^* \mid \varphi_w \in S \}$ unentscheidbar.
- $C(S)$ ist die Menge der Codewörter von Turingmaschinen, die Funktionen aus S berechnen

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Anwendungen des Theorems von Rice

1. $S = \{ \Omega \}$ ($\text{dom}(\Omega) = \emptyset$) Es ist unentscheidbar, ob eine Turingmaschine für keine Eingabe terminiert.
2. Das Äquivalenz-Problem für Turingmaschinen
 $\bar{A} = \{ u \# v \mid \varphi_u = \varphi_v \}$ ist unentscheidbar, da $C(\{ \Omega \}) \leq \bar{A}$ mit $f(w) = w \# \omega$ und $\Omega = \varphi_\omega$
3. Es ist unentscheidbar, ob eine Turingmaschine für alle Eingaben das gleiche Resultat liefert.
 - $S = \{ f \mid \forall u, v \in \{0, 1\}^* : f(u) = f(v) \}$
 - $C(S) =$ Menge der Codewörter von Turingmaschinen, die konstante Funktionen berechnen
4. Es ist unentscheidbar, ob die von einer Turingmaschine berechnete Funktion primitiv-rekursiv ist.

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Universelle Turingmaschine

- **Definition/Satz:** Es gibt eine universelle Turingmaschine U , die jede Turingmaschine M simulieren kann, d.h. U berechnet eine (partielle) Funktion $f_U : \Sigma^* \rightarrow \Sigma^*$ mit
 - $\forall w \in \{0, 1\}^* : \forall x \in \Sigma^* : f_U(w \# x) = \varphi_w(x)$
- **Satz:** Das Halteproblem ist semi-entscheidbar.
- **Folgerung:** H ist eine Typ-0-Sprache, die nicht kontextsensitiv ist.

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 12 Post'sches Korrespondenzproblem (PCP) und Entscheidungsprobleme für Sprachen-Klassen

Friedrich-Alexander-Universität Erlangen-Nürnberg
 Institut für Informatik
 Martensstraße 3 • 91058 Erlangen



Post'sches Korrespondenz-Problem

- Definition: Post'sches Korrespondenz-System (PCS):**
 $\langle (x_i, y_i) \mid i = 1, \dots, k \rangle \quad (x_i, y_i \in \Sigma^*)$
- Definition:** $\langle i_1, \dots, i_n \rangle$ ist **Lösung** von PCS \Leftrightarrow
 $n \geq 1 \wedge x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$
- Definition: Post'sches Korrespondenz-Problem (PCP):**
 - Gegeben: PCS
 - Entscheide: PCS hat eine Lösung
- Beispiel:** PCS = $\langle (1, 101), (10, 00), (011, 11) \rangle$ hat $\langle 1, 3, 2, 3 \rangle$ als Lösung, denn

$$x_1 x_3 x_2 x_3 = 1 \mid 01 \mid 1 \mid 1 \mid 0 \mid 0 \mid 11$$

$$= 1 \mid 01 \mid 1 \mid 1 \mid 0 \mid 0 \mid 11 = y_1 y_3 y_2 y_3$$
- Satz:** PCP ist nicht entscheidbar, aber semi-entscheidbar.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Unentscheidbarkeit von PCP

- Beweisskizze:**
 - **Definition: Modifiziertes PCP (MPCP):**
 - Gegeben: PCS
 - Entscheide: PCS hat eine Lösung mit $i_1 = 1$
 - **Lemma 1:** MPCP \leq PCP
 - **Lemma 2:** H \leq MPCP
 - **Beweis von Lemma 2** durch Konstruktion eines PCS = $f(M, w)$ für eine beliebige Turingmaschine M und ein beliebiges Eingabewort w mit
 - M, angesetzt auf w, terminiert \Leftrightarrow PCS hat eine Lösung mit $i_1 = 1$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Reduktion des Halteproblems auf MPCP(1)

Terminierende Berechnung mit Turingmaschine TM für Eingabe $w \in \Sigma^*$:

$s w [] = k_0 \quad \dots, k_i = u z \sigma v \quad \text{---} \quad u z' \sigma' v = k_{i+1} \quad \dots, k_n = u e \sigma v$

$x : \# \# s w [] \# \dots \# \# u z \sigma v \# \dots \# \# u e \sigma v \# \dots \# \# e \# \#$
 $y : \# \# s w [] \# \dots \# \# u z \sigma v \# \# u z' \sigma' v \# \dots \# \# u e \sigma v \# \dots \# \# e \# \#$

Definition des PCS: 1. Paar: $(\# \# \# s w [])$
 Kopieren: $(\sigma, \sigma) \ (\sigma \in \Gamma \cup \{\#\})$
 Überführung: $(z \sigma, z' \sigma')$, wenn $\delta(z, \sigma) = (z', \sigma', N)$
 $(z \sigma b, \sigma' z' b)$, $(z \sigma \#, \sigma' z' \#)$, wenn $\delta(z, \sigma) = (z', \sigma', R) \ (b \in \Gamma, b \neq \#)$
 $(\# z \sigma, \# z' \# \sigma)$, $(b z \sigma, z' b \sigma')$, wenn $\delta(z, \sigma) = (z', \sigma', L) \ (b \in \Gamma, b \neq \#)$
 Löschen: $(\sigma e, e), (e \sigma, e) \ (\sigma \in \Gamma, e \in E)$ Abschluss: $(e \# \#, \#) \ (e \in E)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Reduktion des Halteproblems auf MPCP(2)

Terminierende Berechnung mit Turingmaschine TM für Eingabe $w \in \Sigma^*$:

$s w [] = k_0 \quad \text{---} \quad k_i = u z a v \quad \text{---} \quad k_{i+1} = u' e a' v'$
 $(u, v, u', v' \in \Gamma^*; a, a' \in \Gamma; z \in Z; s \text{ Startzustand}; e \in E)$

Lösung des zugeordneten Post'schen Korrespondenzsystems PCS(TM, w):

$x : \# \# s w [] \# \dots \# \# u z a v \# \dots \# \# u' e a' v' \# \dots \# \# e \# \#$
 $y : \# \# s w [] \# \dots \# \# u z' a' v' \# \dots \# \# u' e a' v' \# \dots \# \# e \# \#$

Definition des PCS: 1. Paar: $(\# \# \# s w [])$
 Weitere Paare (Regeln):
 Kopieren: $(a, a) \quad (a \in \Gamma \cup \{\#\})$
 Überführung: $(z a, z' a')$, wenn $\delta(z, a) = (z', a', N)$
 $(z a b, a' z' b)$, $(z a \#, a' z' \#)$, wenn $\delta(z, a) = (z', a', R) \ (b \in \Gamma, b \neq \#)$
 $(\# z a, \# z' \# a')$, $(b z a, z' b a')$, wenn $\delta(z, a) = (z', a', L) \ (b \in \Gamma, b \neq \#)$
 Löschen: $(a e, e), (e a, e) \quad (a \in \Gamma, e \in E)$
 Abschluss: $(e \# \#, \#) \quad (e \in E)$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Schnittproblem für kontextfreie Grammatiken (1)

- Definition: Schnitt-Problem für kontextfreie Grammatiken (SchnittKF):**
 - Gegeben: Paar von kontextfreien Grammatiken (G_1, G_2)
 - Entscheide: $L(G_1) \cap L(G_2) = \emptyset$
- Satz:** Das Schnittproblem für kontextfreie Grammatiken ist unentscheidbar.
- Beweisskizze:** Reduktion von PCP auf das Komplement von SchnittKF. Zu (G_1, G_2) wird ein PCS konstruiert mit:
 - PCS hat eine Lösung $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset$

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Schnittproblem für kontextfreie Grammatiken (2)

- Sei PCS = $\langle (x_i, y_i) \mid i = 1, \dots, k \rangle \ (x_i, y_i \in \{0, 1\}^*)$
- Konstruktion von $G_1, G_2: \Sigma := \{0, 1, a_1, \dots, a_k\}$
- $P_1 := \{ S_1 \rightarrow a_i [S_1] x_i \mid i = 1, \dots, k \}$
- $P_2 := \{ S_2 \rightarrow a_i [S_2] y_i \mid i = 1, \dots, k \}$
- $L(G_1) = \{ a_{i_1} \dots a_{i_n} x_{i_1} \dots x_{i_n} \mid n > 0 \}$
- $L(G_2) = \{ a_{i_1} \dots a_{i_n} y_{i_1} \dots y_{i_n} \mid n > 0 \}$
- PCS hat Lösung $\langle i_1, \dots, i_n \rangle$ genau dann, wenn
- $a_{i_1} \dots a_{i_n} x_{i_1} \dots x_{i_n} = a_{i_1} \dots a_{i_n} y_{i_1} \dots y_{i_n} \in L(G_1) \cap L(G_2)$
- **Bemerkung:** $L(G_1)$ und $L(G_2)$ sind sogar deterministisch-kontextfrei.

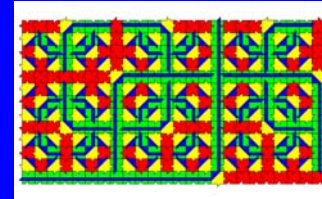
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Weitere (Un-)Entscheidbarkeitssätze

- **Satz:** Das Schnittproblem für deterministisch-kontextfreie Sprachen (gegeben durch det. Kellerautomaten) ist unentscheidbar.
- **Satz:** Das Äquivalenzproblem für kontextfreie Grammatiken ist unentscheidbar.
- **Satz (Senizergues, 1997):** Das Äquivalenzproblem für deterministisch-kontextfreie Sprachen (gegeben durch det. Kellerautomaten) ist **entscheidbar**.
- **Satz:** Das Leerheitsproblem für kontextsensitive Grammatiken ist unentscheidbar.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

§ 13 Domino-Probleme



Horst Müller

Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Informatik
Martensstraße 3 • 91058 Erlangen



Vertiefungs-Hinweise

- <http://www3.informatik.uni-erlangen.de/Lehre/PerlenTheoInf/Domino-Problem/Skript.html>
- <http://www3.informatik.uni-erlangen.de/Lehre/PerlenTheoInf/Domino-Problem/Domino-Literatur.txt>

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kurzfassung (1)

- Das Domino-Problem ist ein besonders anschauliches Beispiel eines unentscheidbaren Problems. Dabei sind Dominos quadratische Spiel-Steine, die durch die Diagonalen in vier gefärbte Teilflächen (rechtwinklig-gleichschenklige Dreiecke) zerlegt sind.



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Kurzfassung (2)

- Zwei Dominos dürfen aneinandergelegt werden, wenn die aneinander stoßenden Teildreiecke gleich gefärbt sind. Die Dominos haben achsenparallele Kanten und dürfen nicht gedreht werden.
- Das (uneingeschränkte) **Domino-Problem** fragt nach der Existenz einer Parkettierung (lückenlosen Belegung) der gesamten Ebene.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Pass-Bedingung



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

(Uneingeschränktes) Domino-Problem

- **Gegeben:** Eine endliche Menge D von Dominotypen (farbe_nord , farbe_west , farbe_süd , farbe_ost).
- **Entscheide:** Es gibt eine (zulässige) Parkettierung, d. h. eine Funktion $\text{parkett}: \text{integer} \times \text{integer} \rightarrow D$.
- **Definition:** parkett ist **zulässig** \Leftrightarrow
 - Für alle Positionen (i, j) : $\text{parkett}(i, j)$ und $\text{parkett}(i+1, j)$ dürfen aneinandergelegt werden:
 - $\text{parkett}(i, j).\text{farbeOst} = \text{parkett}(i+1, j).\text{farbeWest}$
 - und $\text{parkett}(i, j)$ und $\text{parkett}(i, j+1)$ dürfen aneinandergelegt werden
- **Satz:** Das (uneingeschränkte) Domino-Problem ist unentscheidbar.

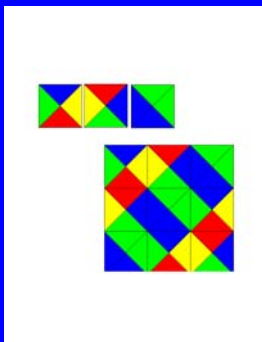
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel Domino-Menge, Parkett

$$D = \{ \text{Domino 1}, \text{Domino 2}, \text{Domino 3} \}$$

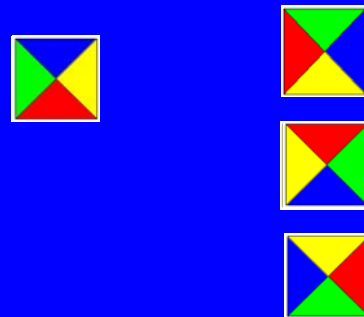
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel



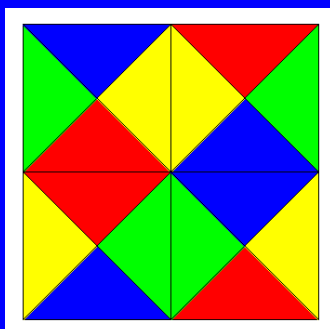
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Drehen der Dominos ist unzulässig



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Triviale Variante für drehbare Dominos



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

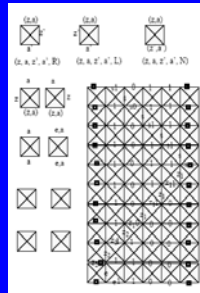
Varianten des Domino-Problems

- D : endliche Domino-Menge
- **Quadranten-Domino-Problem**
 - gegeben : D
 - entscheide: Es gibt ein zulässiges $N \times N$ -Parkett P
- **Ursprungs-Domino-Problem**
 - gegeben : D , Domino d_0 aus D
 - entscheide: Es gibt ein $Z \times Z$ -Parkett P mit $P(0,0) = d_0$
- **Ursprungs-Quadranten-Domino-Problem**
 - gegeben : D , Domino d_0 aus D
 - entscheide: Es gibt ein $N \times N$ -Parkett P mit $P(0,0) = d_0$
- **Satz:** Alle drei Varianten sind unentscheidbar.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Prinzip der Reduktion des Halteproblems H_0 auf das Ursprungs-Quadranten-Domino-Problem

- Jeder Turingmaschine TM kann man eine Domino-Menge $D(TM)$ zuordnen, mit deren Parketten TM-Berechnungen simuliert werden können. Eine TM-Konfiguration "u z a v" (Bandinhalt uav, Zustand z) wird durch einen Farb-Vektor "u (z,a) v" codiert, wobei als Farben Bandsymbole und Paare von Zuständen und Bandsymbolen zulässig sind (und eine triviale Farbe "weiß"). Jedem Rechenschritt der TM entspricht ein Streifen von Dominos. Eine terminierende TM-Berechnung entspricht dann einem Parkett auf einem Rechteck, dessen Höhe die Anzahl der Rechenschritte ist. Ein $N \times N$ -Parkett gibt es nur dann, wenn die TM-Berechnung nicht terminiert.



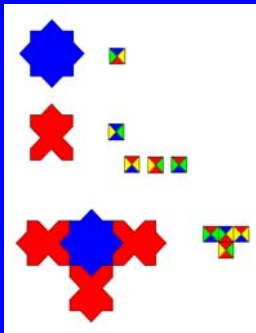
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Polygon-Rand-Dominos

- Statt Dominos kann man auch durch **Polygone berandete Puzzle-Teile (Fliesen)** betrachten, bei denen die Farben-Pass-Eigenschaft durch das Passen der aneinander gelegten Teile ersetzt wird. Diese Puzzle-Teile dürfen allerdings auch gedreht werden.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Beispiel Polygon-Rand-Dominos



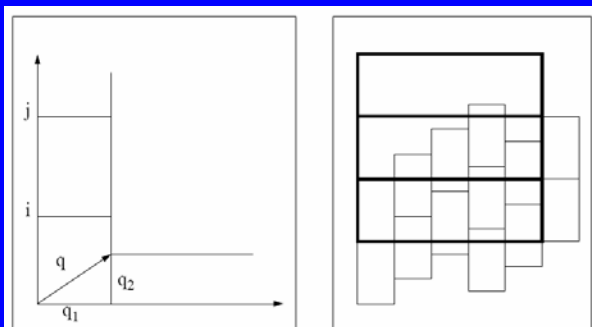
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Bemerkungen zum Unentscheidbarkeitsbeweis

- Für den Unentscheidbarkeitsbeweis des uneingeschränkten Domino-Problems wird eine Überlagerung einer TM-Berechnungen simulierenden Dominomenge $D(TM)$ und einer Domino-Menge D_{np} , die nur *nicht-periodische* Parkette besitzt, benutzt.
- Definition:** Ein $\mathbb{Z} \times \mathbb{Z}$ -Parkett P ist *periodisch*, wenn es eine **Periode** $q \in \mathbb{Z} \times \mathbb{Z}$ gibt, so dass für alle $x \in \mathbb{Z} \times \mathbb{Z}$ gilt : $P(x) = P(x+q)$

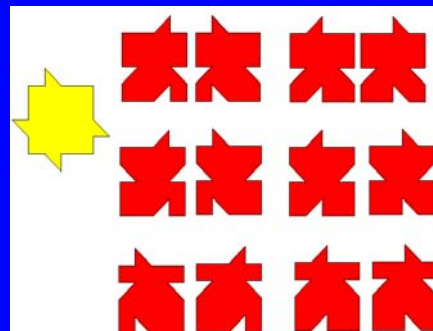
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Periodische Parkette



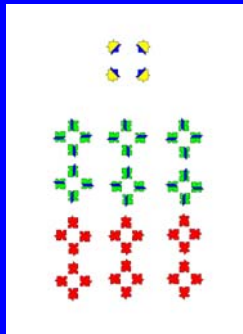
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Eine Domino-Menge, die nur nicht-periodische Parkette hat



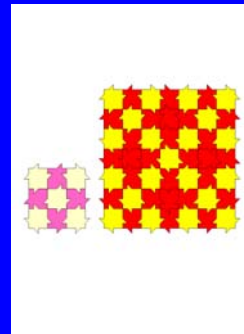
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Dieselbe Dominomenge mit
„Informationsverzerrungen“



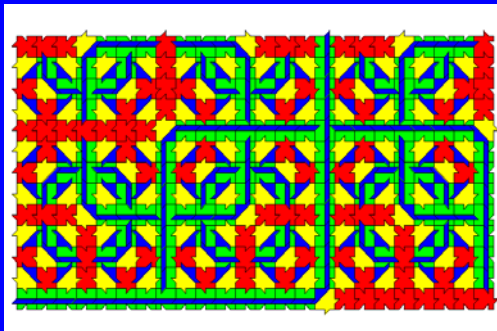
Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Konstruktion eines Parketts dazu



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004

Ausschnitt eines Parketts



Friedrich-Alexander-Universität Erlangen-Nürnberg
Horst Müller: Einführung in die Theoretische Informatik 2, SS 2004