
Einführung in die Informatik C

1. Grundlagen

- Syntax: Menge von Vorschriften, Regeln für den formal korrekten Aufbau von Wörtern und Sätzen
- Semantik: Bedeutung eines syntaktisch korrekten Satzes
- Notationen
 - Alphabet A
 - Wort w über A ist endliche Folge von Zeichen von A ; $|w|$ = Länge von w
 - leeres Wort: λ mit $|\lambda| = 0$
 - A^n : Menge der Wörter mit Länge n , A^* : Menge aller Wörter über A , A^+ : Menge aller nichtleeren Wörter
 - Sprache L über $A \Leftrightarrow L \subseteq A^*$
 - L ist formale Sprache \Leftrightarrow falls L durch ein endliches formales System vollständig beschreibbar
- **Chomsky Hierarchie**
 - Chomsky Grammatik: $G = (N, T, P, S)$ mit
N: Nichtterminalsymbole, T: Terminalsymbole, P: Regelmenge (funktionale Zuordnung),
S: Startsymbol
 - v unmittelbar überführbar in w ($v \Rightarrow w$): durch eine Regel aus P erzeugbar
 - v überführbar in w bzw. w aus v ableitbar ($v \Rightarrow^* w$): durch mehrere Regeln aus P erzeugbar
 - durch G erzeugte Sprache: $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$
 - Problem: Man kann zwar jedes Wort der Sprache erzeugen, aber es ist nicht algorithmisch entscheidbar, ob ein Wort zur Sprache der Grammatik gehört.
 - Typ-0, Typ-1, Typ-2, Typ-3 Grammatiken (\Rightarrow jeweilige Kapitel)
 - es gilt: Typ-3 \Rightarrow Typ-2 \Rightarrow Typ-1 \Rightarrow Typ-0

2 Endliche Automaten und reguläre Sprachen

- **Endlicher Automat (EA)** ohne Ausgabe
 - $EA = (E, S, \delta, s_0, F)$ mit E : Eingabealphabet, S : Zustandsmenge, δ : Überföhrungsfunktion, s_0 : Anfangszustand, F : Menge der Endzustände
 - δ^* ist natürliche Fortsetzung von δ : z.B.: $\delta^*(s_0, e_1 e_2) = \delta(\delta(s_0, e_1), e_2)$
 - Sprache eines endlichen Automaten $L(EA)$: EA akzeptiert $w \in E^* \Leftrightarrow \delta^*(s_0, w) \in F$
 - Pumping Lemma für EA-Sprachen:
geg.: EA mit $|S| = n$ dann gilt für jedes Wort $w \in E^*$: $w = xyz$, $|xy| \leq n$, $|y| \geq 1$, $\forall i: xy^i z \in L(EA)$
- EA mit Ausgabe (**Mealy-Maschine**)
 - $M = (E, S, A, \delta, \gamma, s_0)$ mit A : Ausgabealphabet, γ : Ausgabefunktion; keine Endzustandsmenge, aber durch geeignete Ausgaben simulieren
 - Moore-Maschine: Ausgabe hängt nur vom aktuellen Zustand ab: $\gamma(s, a) = \gamma(s, b)$
 \Rightarrow in Pipeline Strukturen,
Transformation Mealy \Rightarrow Moore: neue Zustände $[s, a]$ einföhren, periodische Eingaben föhren immer zu periodischen Ausgaben
 - A_1 äquivalent A_2 ($A_1 \sim A_2$) $\Leftrightarrow L(A_1) = L(A_2)$
 - Minimierung:
 1. nicht erreichbare Zustände entfernen \Rightarrow vereinfachter Automat
 2. äquivalente Zustände zusammenfassen \Rightarrow reduzierter Automat
 - k-äquivalent: $\forall w \in E^*: |w| \leq k \delta(s, w) \in F \Leftrightarrow \delta'(s', w) \in F$
 - äquivalent: k-äquivalent $\forall k \in \mathbb{N}$
 - Zerlegung der Zustandsmenge in Äquivalenzklassen π_k
 - Dreiecksverfahren:
 - Markiere mit x_0 : Zustände, die nicht 0-äquivalent sind
 - Markiere mit x_i : $\exists w \in E (\delta(s, e), \delta(s', e))$ ist bereits markiert
 - Reduktion ist minimal: zu jedem EA gibt es einen eindeutig bestimmten reduzierten EA

- **Nichtdeterministische Endliche Automaten (ndet-EA)**
 - Folgezustand ist nicht eindeutig \wedge Menge von Folgezuständen kann leer sein
 - A akzeptiert $w \in E^* \Leftrightarrow \delta^*(s_0, w) \cap F \neq \emptyset$ (d.h. wenn es für w einen Weg gibt, der zu einem Endzustand aus F führt)
 - Konstruktion ndet-EA aus EA: trivial
 - Konstruktion EA aus ndet-EA: Bilde Vereinigung von Zuständen als neue Zustände
 - Vorteile ndet-EA: einfache Darstellung + einfache Konstruktion in vielerlei Beweisen
 - Aufwand für Entscheidung, ob $w \in L(A)$: EA: $O(|w|)$ bzw. ndet-EA $O(|s| |w|)$ $\hat{=}$ linear
- **reguläre Sprachen $L_{\text{reg}}(E)$**
 - $+ \sim \vee ; \cdot \sim \wedge ; * \sim *$
 - Menge der L_{reg} ist kleinste Menge von Sprachen über E die endliche Sprachen enthält und abgeschlossen ist bzgl. Produkt, Vereinigung und Iteration
 - Menge der regulären Ausdrücke $RA(E)$
- **Typ-3 (rechtslineare) Sprachen $L_3(E)$**
 - Typ-3-Grammatik (**rechtslineare Grammatik**)
 $\Leftrightarrow A \hat{=} \lambda, A \hat{=} a, A \hat{=} aB$ ($A, B \in N; a \in T$)
 - Konstruktion Typ-3-Grammatik aus EA:

Endlicher Automat (EA)	Typ-3-Grammatik L_3
2 Zustände S ($ S = 2$)	2 Nonterminalsymbole N
Eingabealphabet E	Terminalsymbole T
Anfangszustand s_0	Startsymbol σ (vorher: S)
Endzustand s_1	Symbol A ($A \hat{=} \lambda$ Abbruch)
$\delta(s_0, 1) = s_1$	$\sigma \hat{=} 1A$
- linkslineare Grammatik: Umwandlung in rechtslinear:
 $S \hat{=} Aa$ wird $A \hat{=} a$
 $A \hat{=} a$ wird $S \hat{=} aA$
 $A \hat{=} Ba$ wird $B \hat{=} aA$
- $L_3(E) = L_{EA}(E) = L_{\text{ndet-EA}}(E) = L_{\text{reg}}(E) = RA(E)$

3. Kellerautomaten und kontextfreie Sprachen

- **Kellerautomat (KA)**
 - $KA = (E, S, K, \delta, s_0, k_0, F)$ mit K : Kellularphabet, $k_0 \in K$: Kellerstartzeichen
 - falls $\delta(s, e, k)$ definiert $\Rightarrow \delta(s, \lambda, k)$ undefiniert
 - KA hält an, falls δ weder definiert für (s, λ, k) noch (s, e, k)
 - $\delta(s, e, k) = (s', v)$: k wird ersetzt durch v
 - w wird akzeptiert, falls LK rechts von w und KA in Endzustand
 - Konfiguration (s, w, v) mit s : aktueller Zustand, w : Restwort, v : Kellerinhalt
 - Aufwand für Spracherkennung: $O(n)$
- **nicht deterministischer Kellerautomat (ndet-KA)**
 - akzeptiert auch Palindrome
 - Aufwand für Spracherkennung $O(n^3)$
 - $L_{\text{det-KA}}(E) \subset L_{\text{ndet-KA}}(E)$
- **Typ-2-Sprachen (kontextfreie Sprachen)**
 - Typ-2-Grammatik (kontextfreie Grammatik)
 $\Leftrightarrow A \hat{=} \psi$ ($A \in N; \psi \in (N \cup T)^+$), $S \hat{=} \lambda$ erlaubt, S kommt nichts rechts vor
 - lineare Grammatik: auf jeder rechten Seite ist höchstens ein Variable
 - Ableitungsbäume $AB(G)$
Rechtsableitung: ersetze die rechteste Variable
Linksableitung: Ersetze die linkeste Variable
 - $L_2(E) = L_{\text{ndet-KA}}(E)$

- Chomsky Normalform: nur Regeln der Form $A \rightarrow BC$; $A \rightarrow a$
 - Konstruktion:
 1. mache G λ -frei Bsp.: $A \rightarrow (A)\lambda \Rightarrow A \rightarrow (A)()$
 2. eliminiere reine Umbenennungen ($A \rightarrow B$)
 3. Umformung so, daß jede rechte Seite eine Konstante oder beliebig viele Variablen
 \Rightarrow falls rechts Konstante und Variablen: ersetze Konstante durch Variablen
 \Rightarrow evtl. Einführung neuer Variablen
 4. Ersetze $A \rightarrow B_1 \dots B_n$ durch $A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2, \dots, D_{n-2} \rightarrow B_{n-1} B_n$
 - Greibach Normalform: $A \rightarrow a\phi$
 Vorteil: $S \rightarrow^* w$ in genau n -Abhängigkeitsschritten
 - Pumping-Lemma für L_2 -Sprachen
 $\exists k \in \mathbb{N}: z \in L_2$ mit $|z| \geq k \Rightarrow \exists u, v, w, x, y \in T^*$ mit
 (a) $z = uvwxyz$ (b) $|vwx| \leq k$ (c) $vx \neq \lambda$ (d) $\forall i \in \mathbb{N}_0: u v^i w x^i y \in L_2$
- **LR(k)- Grammatik**, eindeutige Grammatik
 - einmaliges Lesen von links nach rechts und Konstruktion Rechtsableitung
(k Symbole vorausschauen erlaubt)
 - für jedes Wort darf es nur einen Ableitungsbaum geben
 - $L_{LR(k)}(E) = L_{\text{det-KA}}(E)$

3. Kontextsensitive und allgemeine Sprachen und Turing- Maschinen

- **Typ-1-Sprachen**
 - Typ-1-Grammatik (kontextsensitive Grammatik)
 \Leftrightarrow alle Produktionen der Form $\phi_1 A \phi_2 \rightarrow \phi_1 \psi \phi_2$ ($A \in N$; $\phi, \psi \in (N \cup T)^*$; $\psi \neq \lambda$) und $S \rightarrow \lambda$, wenn S nie rechts
 - monotone Grammatik, der Form $\phi \rightarrow \psi$ mit $|\phi| \leq |\psi|$
 - monotone Grammatik \sim kontextsensitive Grammatik
 - Verfahren monoton \rightarrow kontextsensitiv
 - Transformiere durch Einführung neuer Variablen, so daß nur $A \rightarrow a$, $\phi \rightarrow \psi$ mit $|\phi| \leq |\psi|$
(a ersetzen durch C_a)
 - Transformiere $A_1 A_2 \dots A_p \rightarrow B_1 B_2 \dots B_q$:
 - neue Variablen $V_1 \dots V_p$ einführen
 - $A_1 \dots A_p \rightarrow V_1 A_2 \dots A_p$
 $V_1 A_{i+1} \dots A_p \rightarrow V_i V_{i+1} A_{i+2} \dots A_p$
 $V_i V_{i+1} \rightarrow B_i V_{i+1}$
 $V_{p-1} A_p \rightarrow V_{p-1} V_p$
 $B_{p-1} V_p \rightarrow B_{p-1} B_p \dots B_q$
 - typische Sprache: $a^n b^n c^n$
 - Aufwand für Entscheidung, ob $w \in L(G)$ ist exponentiell \rightarrow eignen sich nicht für Programmiersprachen
(kontextfreie geeignet)
- **Typ-0-Sprachen**
 - Typ-0-Grammatik $\Leftrightarrow P$ beliebig: $\phi \rightarrow \psi$ mit $\phi \neq \lambda$ (allgemeine Chomsky Grammatik, Phasenstrukturgrammatik)
- **Turingmaschine** (hier: 1-Band-TM)
 - $T = (E, S, B, \delta, s_0, F)$ mit $B =$ Bandalphabet $E \cup \{*\} \subseteq B$, δ zusätzliche L (links), R (rechts), N (nicht)
 - Konfiguration: (v, s, w) mit v : Bandinschrift links von SL - Kopf, w : Bandinschrift ab SL - Kopf nach rechts, s : aktueller Zustand
 - Initialkonfiguration: (λ, s_0, w)
 - Endkonfiguration: $(v_1, s, v_2) \wedge s \in F \wedge \neg \exists$ Übergang
 - folgende Situationen sind möglich
 - T hält nie an
 - T hält an, aber nicht in s_f
 - SL Kopf ist vom linken Zeichen nicht nach links bewegbar
- nichtdeterministische Turing- Maschine
 - Erreichen der Finalkonfiguration evtl. mit exponentiell höherer Laufzeit
 - $L_{\text{ndet-TM}}(E) = L_{\text{det-TM}}(E) = L_0$
- Linear beschränkter Automat (LBA)
 - TM mit Länge der Bandinschrift stets in $O(|w|)$, $L_{LBA} = L_1$

4. Berechenbarkeit und Komplexität

- **Berechenbarkeit:** Funktion $f: M_1 \rightarrow M_2$ heißt berechenbar, falls es einen Algorithmus $A: M_1 \rightarrow M_2$ gibt mit $A(w) = f(w) \forall w \in M_1$
- **Abzählbare Menge:** Menge M heißt abzählbar, wenn sich M injektiv auf \mathbb{N} abbilden läßt.
Andernfalls heißt M überanzählbar
Satz: Jede endliche Menge ist abzählbar
 - Bsp.: A^* mit A Alphabet, $\mathbb{N} \times \mathbb{N}, \mathbb{Q}$
 - Diagonalisierungsverfahren: Suche Funktion, die in jeder Spalte unterschiedlich zu je einer Funktion ist
- **entscheidbare Menge:** M_1 heißt entscheidbar relativ zu M_2 , wenn es einen Algorithmus $A_{M_1, M_2}: M_2 \rightarrow \{\text{false}, \text{true}\}$ gibt, mit dessen Hilfe man zu jedem Element $e \in M_2$ feststellen kann, ob es zu M_1 gehört oder nicht
 - M entscheidbar $\Rightarrow E^* \setminus M$ entscheidbar
- **charakteristische Funktion:** es seien $M_1 \subseteq M_2 \subseteq E^*$. Dann heißt die Funktion $\chi_{M_1, M_2}(w) = \text{true}$, falls $w \in M_1 \wedge \text{false}$, falls $w \in M_2 - M_1$
 - M_1 entscheidbar relativ zu $M_2 \Leftrightarrow \chi$ berechenbar
- **aufzählbare Mengen:** Menge $M \subseteq E^*$ heißt aufzählbar, wenn es eine Funktion $f: \mathbb{N}_0 \rightarrow M$ gibt, die surjektiv und berechenbar ist
- Sätze:
 - M entscheidbar $\Rightarrow M$ aufzählbar $\Rightarrow M$ abzählbar
 - Menge der berechenbaren Funktionen ist abzählbar
 - M entscheidbar \Rightarrow charakteristische Funktion berechenbar
- Turing- Maschine
 - f Turing-berechenbar $\Leftrightarrow \exists$ TM T , die f berechnet
 - Church'sche These: Jede berechenbare Funktion ist auch Turing- berechenbar
 - Turing- aufzählbar, Turing- entscheidbar analog
 - universelle TM U (Simulation einer TM)
Band: Arbeitsweise von T (=Codierung aller Überföhrungsfunktionen)* Initialkonfiguration
- Satz: Es gibt keine TM, die $L_{NA} = \{w_i \mid w_i \notin L(T_i)\}$ akzeptieren kann
- Halteproblem: Es ist unentscheidbar, ob es eine TM gibt, die für jede TM T entscheidet, ob T bei Eingabe von w hält oder nicht
- Komplexität
 - Zeitkomplexität $T_{TM}(n) = \max$ der Längen der kürzeseten Konfigurationsfolgen für w mit Länge n , so daß TM hält
 - Platzkomplexität, analog jedoch Anzahl der gelesenen Zellen
 - $P = NP$ Problematik
 - P heißt N_p hart $\Leftrightarrow Q \in NP$ polynomialzeit-reduzierbar auf P d.h. $Q \rightarrow P \rightarrow L_p \rightarrow L_Q$ (Trafo in polynomieller Zeit)
 - Lösungsansätze
 - zusätzliche Randbedingungen \Rightarrow Verinfachung
 - suboptimale Lösung ausreichend ?
 - geeignete systematische Entwurfsmethodik
 - randomisierte Verfahren
 - Berechnungsaufwand bei TM und RAM höchstens polynomiell unterschiedlich

Anhang: Übersicht Chomsky- Hierarchie und Automaten

	Grammatik $G = (N, T, P, S)$	Automat	Sonstiges, Bsp.	Aufwand
L₃ (reguläre Sprachen)	<u>rechtslineare Grammatik:</u> $A \Rightarrow \lambda, A \Rightarrow a, A \Rightarrow aB$	<i>ohne Ausgabe:</i> <u>endlicher Automat</u> (det/ndet): $EA = (E, S, \delta, s_0, F)$	Pumping-Lemma, Reguläre Ausdrücke (RA)	O(n)
		<i>mit Ausgabe:</i> <u>Mealy-Maschine:</u> $M = (E, S, A, \delta, \gamma, s_0)$		
L₂ (kontextfreie Sprachen)	<u>LR(k) Grammatik:</u> einmaliges Lesen von links nach rechts und Konstruktion Rechtsableitung	<u>det. Kellerautomat:</u> $KA = (E, S, K, \delta, s_0, k_0, F)$	Bsp.: $a^n b^n$	O(n)
	<u>kontextfreie Grammatik:</u> $A \Rightarrow \psi$ ($A \in N, \psi \neq \lambda$) und $S \Rightarrow \lambda$ erlaubt, S nie rechts	<u>nichtdet. Kellerautomat:</u> $KA = (E, S, K, \delta, s_0, k_0, F)$	Pumping Lemma, Chomsky NF, Bsp.: Palindrom	O(n ³)
L₁ (kontextsensitive Sprachen)	<u>kontextsensitive Grammatik:</u> $\varphi_1 A \varphi_2 \Rightarrow \varphi_1 \psi \varphi_2$ ($A \in N; \psi \neq \lambda$) und $S \Rightarrow \lambda$, wenn S nie rechts <u>monotone Grammatik:</u> $\varphi \Rightarrow \psi$ mit $ \varphi \leq \psi $	nichtdet. Linear Beschränkter Automat (LBA)	Bsp.: $a^n b^n c^n$	O(2 ⁿ)
L₀ (allgemeine Sprachen)			entscheidbare Sprachen	
	<u>allgemeine Grammatik:</u> $\varphi \Rightarrow \psi$ mit $\varphi \neq \lambda$	<u>Turing-Maschine</u> (det/ ndet): $TM = (E, S, B, \delta, s_0, F)$	aufzählbare Sprachen	bel.
$\bar{p}(E^*)$			$L_{NA} = \{w_i w_i \notin L(T_i)\}$	