Zusammenfassung Grundzüge der Informatik 4

Sommersemester 04

Thorsten Wink

21. September 2004

Version 1.2
Dieses Dokument wurde in LaTeX 2_ε geschrieben.
Stand: 21. September 2004

Inhaltsverzeichnis

Kapitel 1

Automatentheorie und formale Sprachen

1.1 Grammatiken

Definition: Eine Grammatik ist ein 4-Tupel G = (V, Σ, P, S) mit
V = Menge der Variablen
Σ = Terminalalphabet
P = Regeln bzw. Produktionen
S = Startvariable

Man kann alle Grammatiken in die folgende Hierarchie einordnen:
TYP 0: Alle Grammatiken
TYP 1: Eine Grammatik ist *kontextsensitiv*, falls für alle Regeln $w_1 \rightarrow w_2$ aus P $|w_1| \leq |w_2|$ gilt.
TYP 2: Eine Grammatik ist *kontextfrei*, wenn sie von TYP 1 ist und für alle Regeln aus P gilt dass w_1 eine einzelne Variable ist.
TYP 3: Eine Grammatik ist *regulär*, wenn sie von Typ 2 ist und die rechte Seite der Regeln entweder ein einzelnes Terminalzeichen oder ein Terminalzeichen gefolgt von einer Variable ist.

1.2 Reguläre Sprachen

1.2.1 DFA

Defintion: Ein deterministischer endlicher Automat (DFA) M wird angegeben durch ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, E)$ mit
Z = Menge der Zustände
Σ = Eingabealphabet
δ = Überführungsfuntion
z_0 = Startzustand
E = Menge der Endzustände

Endliche Automaten werden entweder mithilfe der Übergangsfunktion oder als Zustandsgraph

angegeben. Mit einem DFA kann man genau die Menge aller TYP 3 Sprachen erkennen.

1.2.2 NFA

Ein nichtdeterministischer endlicher Automat (NFA) wird formal wie ein DFA angegeben, nur ist hier S die Menge der Startzustände statt z_0. Ein NFA kann zu einem Eingabezeichen mehrere mögliche Übergänge haben, zum Aktzeptieren eines Wortes muss es mindestens eine mögliche Zustandsfolge geben.
DFA und NFA sind ineinander überführbar, die Modelle sind demnach gleichmächtig.
ev. TODO: Algo NFA → DFA

1.2.3 Minimalautomat

Folgender Algorithmus liefert aus einem DFA den Minimalautomaten:

1. Stelle eine Tabelle aller Zustandspaare $\{z, z'\}$ mit $z \neq z'$ auf.
2. Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$ (oder umgekehrt)
3. Für jedes noch unmarkierte Paar $\{z, z'\}$ und für jedes $a \in \Sigma$ teste, ob $\{\delta(z, a), \delta(z', a)\}$ bereits markiert ist. Wenn ja markiere auch $\{z, z'\}$
4. Wiederhole den letzten Schritt, bis sich keine Änderungen mehr an der Tabelle ergeben
5. Alle jetzt noch unmarkierten Paare können zu jeweil einem Zustand verschmolzen werden.

1.2.4 Reguläre Ausdrücke

Mit regulären Ausdrücken kann man alle regulären Sprachen definieren. Die reg. Ausdrücke sind wie folgt definiert:

- $\emptyset$ ist ein regulärer Ausdruck
- ϵ ist ein regulärer Ausdruck
- Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck
- wenn α und β reguläre Ausdrücke sind, dann auch $\alpha\beta$, $(\alpha|\beta)$ sowie $(\alpha)^\star$

1.2.5 Pumping Lemma

Das Pumping Lemma ist ein Hilfsmittel um von einer Sprache zu zeigen, dass sie NICHT regulär ist.
Sei L eine reguläre Sprache. Dann gibt es eine Zahl n, so dass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, so dass folgende Eigenschaften erfüllt sind:

- $|v| \geq 1$
- $|uv| \leq n$
- für alle i = 0,1,2,... gilt: $uv^iw \in L$.

1.2.6 Abschlusseigenschaften/Entscheidbarkeit

Die regulären Sprachen sind abgeschlossen unter: Vereinigung, Schnitt, Komplement, Produkt, Stern-Operator.
Entscheidbar für reguläre Sprachen sind das Wortproblem, Leerheitsproblem, Schnittproblem, Äquivalenzproblem.

1.3 Kontextfreie Sprachen

Die kontextfreien Sprachen sind in der Informatik besonders wichtig, da sich mit Ihnen die meisten Programmiersprachen beschreiben lassen. Einen Parser für Klammerstrukturen oder IF-THEN-ELSE Strukturen kann man mithilfe eines Kellerautomaten implementieren.

1.3.1 Kellerautomaten

Das Problem des DFA ist es, dass er keinen Speicher hat. Diese Einschränkung wird nun durch einen Kellerspeicher aufgehoben.
Definition: Ein Kellerautomat (PDA) M wird wie folgt angeben:
M = (Z, $\Sigma, \Gamma, \delta, z_0, \#$) mit
Z = Menge der Zustände
Σ = Eingabealphabet
Γ = Kelleralphabet
δ = Überführungsfuntion
z_0 = Startzustand
\# = unterstes Kellerzeichen

Bei einem Zustandsübergang kann nur das oberste Zeichen des Kellers verändert werden. So ist es möglich das oberste Zeichen zu entfernen oder weitere Zeichen in den Keller zu legen.
Ein Kellerautomat akzeptiert ein Wort durch leeren Keller. Mit einem nichtdeterministischen Kellerautomat kann man die Menge aller kontextfreien Sprachen erkennen.

1.3.2 CNF

Zu jeder kontextfreien Grammatik gibt es eine entsprechende Grammatik in Chomsky Normalform (CNF). Dafür muss gelten, dass alle Regeln eine der beiden folgenden Formen haben:

- $A \rightarrow BC$
- $A \rightarrow a$

Um eine Grammatik in CNF zu transformieren, muss sie zuerst ϵ-frei gemacht werden. Danach werden sukzessive neue Variablen eingeführt und die Produktionen aufgespalten, bis alle Regeln obiges Format haben.

1.3.3 CYK-Algorithmus

Der CYK-ALgorithmus entscheidet, ob ein Wort x durch eine Grammatik G erzeugt werden kann, also ob $x \in L(G)$. Er hat die Komplexität $O(n^3)$. Die Grammatik muss in CNF vorliegen.
TODO: Erklärung

1.3.4 Pumping-Lemma

Ebenso wie bei dem Pumping-Lemma für reguläre Sprachen benutzt man das folgende um zu zeigen, dass eine Sprache nicht kontextfrei ist.
Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl n, so dass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$, so dass folgende Eigenschaften erfüllt sind:

- $|vx| \geq 1$
- $|vwx| \leq n$
- für alle i = 0,1,2,... gilt: $uv^iwx^iy \in L$.

1.3.5 Abschlusseigenschaften/Entscheidbarkeit

Die kontextfreien Sprachen sind abgeschlossen unter: Vereinigung, Produkt, Stern-Operator. Sie sind nicht abgeschlossen unter Schnitt und Komplement
Entscheidbar für reguläre Sprachen sind Wortproblem und Leerheitsproblem. Nicht entscheidbar sind Schnittproblem und Äquivalenzproblem.

1.3.6 deterministisch kontextfreie Sprachen

Ein deterministischer Kellerautomat DPDA hat für jede mögliche Konfiguration höchstens einen Folgezustand. Desweiteren akzeptiert ein det. Kellerautomat per Endzustand und nicht per leerem Keller.
Ein deterministisch kontextreier Kellerautomat erkennt die Menge der deterministisch kontextfreien Sprachen. Dies sind die LR(k)-Sprachen, die im Compilerbau eine wichtige Rolle spielen.
Wichtig: PDA und DPDA sind nicht gleichwertig, ein DPDA ist in einen PDA überführbar, die Richtung PDA ⇒ DPDA gilt jedoch nicht.

1.4 Übersicht

Typ 3	reguläre Grammatik DFA NFA reguläre Ausdrücke
Det. kf.	LR(k)-Grammatik
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 1	kontextsensitive Grammatik linear beschränkter Automat
Typ 0	Typ 0 Grammatik Turinmaschine

nichtdet. Automat	deterministischer Automat	äquivalent?
NFA	DFA	Ja
PDA	DPDA	nein
LBA	DLBA	?
TM	DTM	Ja

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Typ 3	ja	ja	ja	ja	ja
det. kf	nein	nein	ja	nein	nein
Typ 2	nein	ja	nein	ja	ja
Typ 1	ja	ja	ja	ja	ja
Typ 0	ja	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheitsproblem	Äquivalenzproblem	Schnittproblem
Typ 3	ja	ja	ja	ja
det. kf	ja	ja	ja	nein
Typ 2	ja	ja	nein	nein
Typ 1	ja	nein	nein	nein
Typ 0	nein	nein	nein	nein

Kapitel 2

Berechenbarkeit

Der Begriff der Berechnebarkeit konnte bisher noch nicht eindeutig formal definiert werden. Intuitiv ist jede Funktion, für die man eine Berechnungsvorschrift (einen Algorithmus) angeben kann, berechenbar.
Die **Churchsche These** besagt: Die durch die formale Definition der Turing- Berechenbarkeit (die äquivalent ist zur WHILE-, GOTO-Berechenbarkeit und μ-Rekursivität) erfasste Klasse von Funktionen stimmt genau mit der Klasse der im intuitiven Sinne berechenbaren Funktionen überein.

Es wurden noch verschiedene Berechenbarkeitsdefinitionen angegeben, die wie folgt zusammenhängen:

$$primitiv{-}rekursiv \Longleftrightarrow LOOP \Longrightarrow WHILE \Longleftrightarrow GOTO \Longleftrightarrow TM \Longleftrightarrow RAM \Longleftrightarrow \mu{-}rekursiv$$

2.1 Turing-Maschinen

Eine Turing-Maschine besteht aus einem (oder auch mehreren) auf einer Seite unendlich langen Band und einem Schreib/Lesekopf. In jedem Rechenschritt kann das Zeichen gelesen werden, und abhängig vom Zeichen und dem aktuellen Zustand ein neues Zeichen geschrieben werden. Danach kann der Kopf entweder nach Links/Rechts oder garnicht bewegt werden.
Noch ein paar Konventionen: Leere Zellen sind mit einem $\emptyset$ bedruckt, die Eingabe steht am Beginn des 1.Bandes, die Ausgabe beginnt an der Stelle auf Band 1 wo die TM stoppt.

Formale **Definition** der TM τ (nach Brandt):
$\tau = (A_{(\tau)}, S_{(\tau)}, an_{(\tau)}, \delta_\tau)$ mit
$A_{(\tau)}$ = Bandalphabet der TM
$S_{(\tau)}$ = Zustandsmenge der TM
$an_{(\tau)}$ = Anfangszustand
δ_τ = Programm

Das Programm schreibt man in folgender Art: $(s_0,\ 0) : (s_1,\ (0,\ \mathrm{S}))$, wobei die TM in diesem Beispiel in Zustand s_0 beim Lesen einer 0 in den Zustand s_1 wechselt, eine 0 druckt und den Kopf nicht bewegt.
Eine Mehrband-TM ist immer auch mit einer Einband-TM realisierbar. Dazu teilt man das Band in mehrere Spuren auf und markiert auf jeder Spur, wo sich der entsprechende Kopf befinden würde.

Eine TM berechnet für jede r-Stellige Eingabe die Wort-Funktion

$$f_\tau^r : (\Sigma^\star)^r \longrightarrow \Sigma^\star$$

Um auch Zahl-Funktionen mit einer TM darstellen zu können, decodiert man die natürlichen Zahlen bin2ar auf das Eingabeband. Dann berechnet die TM die Zahl-Funktion

$$\vartheta_\tau^r : (N)^r \longrightarrow N$$

2.2 RAM-Programme

Assembler-like

2.3 Loop-While-GOTO

wie die Programmier-Konstrukte

2.4 Primitive Rekursion

Die Menge der primitiv rekursiven Funktionen $Prim$ ist folgendermaen definiert:
Alle folgenden Ausgangsfunktionen sind in $Prim$:

- die Nachfolgerfunktion
- die Projektion
- die konstante Funftion

Aus primitiv rekursiven Funktionen können neue Funktionen gebildet werden, die ebenfalls in $Prim$ sind durch:

- Substitution
- Primitive Rekursion
- beschränkte Summation
- beschränkte Produktbildung
- Definition durch Fallunterscheidung
- Iteration
- beschränkter μ-Operator
  Der beschränkte μ-Operator ist definiert durch

$$(\mu^b f)(y, x_1, ..., x_r) = \begin{cases} min\{z \leq y | f(z, x_1, ..., x_r) = 0\} & \text{,falls dieses Minimum existiert} \\ y + 1 & \text{,sonst} \end{cases}$$

Beispiele für primitiv rekursive Funktionen:
Addition, Multiplikation, Vorgängerfunktion, Modifizierte Differenz, Signumsfunktion, Ganzzahldivision.
Ein Beispiel für eine Funktion, die total und intuitiv berechenbar ist, die aber nicht primitiv rekursiv ist: die Ackermannfunktion:
$a(0, y) = y + 1)$
$a(x + 1, 0) = a(x, 1)$
$a(x + 1, y + 1) = a(x, a(x + 1, y))$

2.5 μ-Rekursion

Die Menge $\mu - Rek$ der μ-rekursiven Funktionen ist die kleinste Menge von (eventuell partiellen) Funktionen, die die Ausgangsfunktionen

- Nachfolgerfunktion
- Projektion
- konstante Funktion

enthält und abgeschlossen ist unter

- Substitution
- primitiver Rekursion
- Anwendung des μ-Operators

Der μ-Operator ist definiert durch

$$(\mu f)(x_1, ..., x_r) = \begin{cases} min\{z \in N | f(z, x_1, ..., x_r) = 0\} & \text{,falls dieses Minimum existiert} \\ & \text{und} \forall y \leq z_{min} : f(y, x_1, ..., x_r) \text{terminiert.} \\ \text{undefiniert} & \text{,sonst} \end{cases}$$

2.6 Entscheidbarkeit: Gödelisierung, Satz von Rice, Diagonalisierung