# Diplomprüfung Theoretische Informatik

**Datum:** 11. Oktober 2005
**Prüfer:** Prof. Thomas, Prof. Vöcking
**Themen:**
Effiziente Algorithmen (WS 2004/05, Vöcking)
Automaten auf unendlichen Wörtern (WS 2004/05, Thomas)
Baumautomaten und Anwendungen (WS 2004/05, Löding)
Compilerbau (SS 2005, Indermark)
**Dauer:** ca. 40 Minuten
**Note:** 1,0
*Keine Garantie für Korrektheit oder Vollständigkeit dieses Gedächtnisprotokolls.*

## 1 Effiziente Algorithmen

V: Fangen wir mal vorne an. Was sind Flussprobleme?

I: Ein Flussnetzwerk ist ein gerichteter, gewichteter Graph mit ausgezeichneter Quelle und Senke. Ein Fluss ist eine Funktion von den Kanten auf natürliche Zahlen, wobei Kapazitätsbeschränkung und Flusserhaltung (in jeden Knoten geht soviel rein, wie rauskommt, ausgenommen Quelle und Senke) gewährleistet sein müssen. Wir suchen einen maximalen Fluss, also einen mit maximaler Summe der aus der Quelle gehenden Flüsse.

V: Dann hatten wir den Ford-Fulkerson-Algorithmus.

I: Wir bezeichnen unseren Graphen als 0. Restnetzwerk. Dann suchen wir über Breiten- oder Tiefensuche einen flussvergrößernden Weg. Den ziehen wir dann von unserem Restnetzwerk ab und fügen ihn in entgegengesetzte Richtung ein; die Idee ist hier, Flüsse 'zurückpumpen' zu können. Da wir einen fv-Weg in $O(m)$ finden können und sich der Fluss stets um mindestens 1 erhöht, ist die Laufzeit beschränkt durch $O(m \cdot C)$, wobei $C$ die Summe der Kapazitäten ist. Also pseudopolynomiell.

V: Wie können wir das besser machen?

I: Indem wir stets kürzeste fv-Wege suchen, also über Breitensuche. Für jeden Knoten v gilt, dass sich der Abstand von der Quelle zu v während der Laufzeit nie verringert. Die Schwierigkeit war ja, dass Kanten gelöscht und wieder eingefügt werden. Wir hatten aber gesagt, dass sich durch Löschen und Wiedereinfügen von (u, v) der Abstand von q nach v um mindestens zwei erhöht.

V: Und wieviele Iterationen brauchen wir dann?

I: Wir haben $2m$ Kanten, jede kann $\frac{n}{2}$ mal gelöscht und wieder eingefügt werden, also $O(m \cdot n)$. Insgesamt also $O(n^5)$.

V: Kannst du die $O(n^5)$ genauer beschreiben? Also nach Kanten und Knoten aufgeteilt.

I: $O(m^2 \cdot n)$.

V: OK. Gehen wir mal zu randomisierten Algorithmen. Wir hatten da Min-Cuts. Weißt du, wie man das Problem ohne Randomisierung über Flussprobleme löst?

I: Bei diesem Min-Cut-Problem haben wir ja keine feste Quelle und Senke. Wir fixieren einfach einen Knoten als Quelle. Dann setzen wir nacheinander alle anderen Knoten als Senken und wählen das Minimum der Cuts. Das hat dann Laufzeit $O(n) \cdot O(n^3)$, also $O(n^4)$.

V: (nickt)

I: Wir hatten dann einen randomisierten Algorithmus, der arbeitete allerdings auf Multigraphen statt auf gewichteten Graphen: das erleichtert die Rechnung. Wir wählen uniform zufällig eine Kante und kontrahieren sie. (Kontraktionsbeispiel aufgemalt, erklärt.) Eine Kante kann in $O(n)$ kontrahiert werden.

V: Gut, und welche Erfolgswahrscheinlichkeit hat unser Algorithmus dann?

I: Mindestens $\frac{2}{n^2}$. Soll ich das vorrechnen?

V: Nein, das ist nicht nötig. Wir hatten da eine Methode, um die Erfolgswahrscheinlichkeit zu erhöhen, und zwar die Wahrscheinlichkeitsamplifikation. Wie ging das?

I: Wenn wir den Algorithmus $n^2$ mal wiederholen, haben wir bereits konstante Erfolgswahrscheinlichkeit. Wenn wir dann noch einen konstanten Faktor ergänzen, also $t \cdot n^2$ mal wiederholen, geht das t exponentiell ein.

V: Warum?

I: Aufgeschrieben: $\left(1 - \frac{2}{n^2}\right)^{n^2 \cdot t} \leq \left(\frac{1}{e}\right)^{2t}$. Gesagt, wie die Formel aussieht, aus der man sich das hergeleitet hat.

V: OK, aber die $O(n^4)$ sind ja noch nicht zufriedenstellend. Wir hatten da noch einen weiteren Algorithmus, nämlich Fastcut. Wie funktioniert der denn?

I: Wir bestimmen zwei unabhängige Kontraktionssequenzen der Länge n-t, wobei wir t definieren als (hingeschrieben) $\lceil (1 - \frac{n}{\sqrt{2}}) \rceil$. Dann lösen wir die Teilprobleme rekursiv und wählen den kleineren der beiden Cuts.

V: Da stimmt noch was nicht, mit dem Minus.

I: Muss natürlich ein Plus sein.

V: Und in welcher Größenordnung sind dann die Teilprobleme?

I: Äh... (Gehirnblockade) halb so groß?

V: Nein, natürlich um den Faktor $\frac{1}{\sqrt{2}}$ kleiner.

I: (D'Oh!)

V: Und wie war dann die Erfolgswahrscheinlichkeit?

I: Wir haben gezeigt, dass die Wahrscheinlichkeit, dass der Min-Cut durch eine Kontraktionssequenz zerstört wird, höchstens $\frac{1}{2}$ ist. Die Rechnung lief wie beim vorherigen Algorithmus. Für die Erfolgswahrscheinlichkeit gilt dann: (aufgeschrieben und erklärt) $P(G) = 1 - \left(1 - \frac{P(H_A)}{2}\right) \cdot \left(1 - \frac{P(H_B)}{2}\right)$. Wir können dann zeigen, dass gilt: $P(D) = 1 - \left(1 - \frac{P(D-1)}{2}\right)^2$. Dann haben wir gezeigt, dass gilt: $P(D) \leq \frac{1}{P(D+1)}$. Nein, $P(D) \leq \frac{1}{D+1}$.

V: Und wie zeigt man, dass diese Rekursionsgleichung gilt?

I: *Äh*

V: Ich will jetzt gar nicht die Rechnung sehen, aber wie macht man sowas allgemein? Wenn man das gewünschte Ergebnis schon kennt?

I: *Öh*

V: (Noch mehr Tipps gegeben)

I: *Üh* (Blackout und nicht kapiert, worauf er hinauswollte)

V: Na, per Induktion!

I: (D'Oh!)

V: OK, dann gehen wir mal noch zu Online-Algorithmen. Wir hatten da einen Algorithmus, dem wir mehr Speicher gegeben haben.

I: Ja, wir haben RANDOM doppelt so viel Speicher gegeben wie OPT. RANDOM ist (2, 2)-competitive. Wir haben das über eine Potentialfunktion gezeigt, die kann man sich als Konto vorstellen, in das RANDOM einzahlen kann, wenn es in einer Phase unter dem Faktor 2 geblieben ist, und das dann aufbrauchen konnte, wenn er später mehr Seitenfehler hat.

V: Und wie hatten wir die Potentialfunktion hier gewählt?

I: Die Anzahl der Seiten, die OPT im Cache hat und RANDOM nicht.

V: Und wie ging das dann?

I: (hingeschrieben) $E[C] + 2 \cdot E[\Phi] \leq 2 \cdot C^*$

V: Warum folgt dann, was wir zeigen wollen?

I: Weil die Potentialfunktion immer positiv ist, ist $E[C] \leq 2 \cdot C^*$. Wir haben dann die Invariante gezeigt: $E[\Delta C] + 2 \cdot E[\Delta\Phi] \leq 2 \cdot \Delta C^*$. Das ging über eine Fallunterscheidung mit vier Fällen, z. B. wenn beide Algorithmen die Seite im Cache haben, ist alles 0.

V: OK, das war's dann von mir.

## 2 Automaten auf unendlichen Wörtern

T: Wir hatten ja Automaten für Omega-Sprachen mit verschiedenen Akzeptanzbedingungen: Büchi-, Rabin- und Mullerautomaten. Erklären Sie doch mal, wie die Bedingungen für Rabin- und Mullerautomaten aussehen.

I: (aufgeschrieben, erklärt.)

T: Rabin- und Mullerautomaten sind ja gleichmächtig. In welche Richtung ist die Umwandlung einfacher?

I: Aus einem Rabin-Automaten kann man einen Muller-Automaten machen, indem man alle Zustandsteilmengen wählt, bei denen es ein Akzeptanzpaar gibt, so dass eben die Rabin-Bedingung für die Teilmenge erfüllt ist.

T: Richtig. Und die Gegenrichtung?

I: (Etwas gestockt, weil ich zunächst überlegt habe, wie man die Konstruktion direkt machen könnte.) Ach ja, wir können über NBAs gehen. Die Umwandlung von DMA nach NBA geht über das 'Aufsammeln von Zuständen', und dann kann man über die Muller-Schupp-Konstruktion einen DRA berechnen.

T: Und wie war die Komplexität?

I: Der NBA hat exponentiell viele Zustände, weil wir ja Zustandsteilmengen in die Zustände aufnehmen müssen. Bei der Muller-Schupp-Konstruktion erhalten wir dann $2^{O(n\cdot\log(n))}$ viele Zustände.

T: Wieso $2^{O(n\cdot\log(n))}$?

I: Wir können die Muller-Schupp-Bäume durch drei Funktionen beschreiben. Erst einmal haben wir gesagt, dass ein$3n$großes Namensreservoir ausreicht. Die Funktionen... (wurde hier unterbrochen)

T: Zu den $3n$. Die folgten aus einer bestimmten Eigenschaft der Muller-Schupp-Bäume.

I: Die MS-Bäume sind endlich.

T: Noch etwas.

I: Sie sind strikt binär.

T: Richtig. Na gut, den Rest lassen wir dann jetzt mal. Es gibt noch eine andere Akzeptanzbedingung, nämlich beim Paritätsautomaten. Wie sieht die aus?

I: Funktion $c : Q \rightarrow N$, akzeptiert, wenn höchste unendlich oft auftretende Priorität gerade ist.

T: Priorität... sagen wir Farbe. Wie würde man denn einen Rabin-Automaten in einen Paritätsautomaten umwandeln?

I: *überleg* (etwas irritiert, Paritätswortautomaten waren AFAIR gar nicht Vorlesungsstoff)

T: Wenn wir also z. B. Prioritäten 1, 2, ..., 6 haben.

I: Wir können Akzeptanzpaare bilden, bei denen $F_i$ einen Zustand gerader Farbe enthält und $E_i$ alle Zustände mit höherer Farbe.

T: Genau richtig (Dann noch etwas erläutert, ich weiß nicht mehr genau was). Kommen wir zu Logik. Wir hatten eine Logik, die genauso stark ist wie NBAs.

I: Das ist die Logik S1S.

T: Wie zeigt man denn die eine Richtung? Also von Büchi-Automaten zu S1S. Schreiben Sie das doch mal hin.

I: Aufgeschrieben:

$$\varphi(T) = \exists X_1 ... \exists X_n Partition(X_1, ..., X_n)$$

Habe mich dann mit dem Anfangszustand verhaspelt, Prof. Thomas meinte dann, ich sollte ein Beispiel für die Behandlung der Transitionen beschreiben. Da habe ich mich dann nochmal verhaspelt: $\forall t((X_1(t) \wedge T(a) \wedge X_2(t+1)) \vee \ldots$ . Dass $T(a)$ Quatsch ist, ist mir direkt aufgefallen, Prof. Thomas hat mich dann darauf hingewiesen, dass ich die Eingabe als Wort über Nullen und Einsen betrachten soll. Ich habe dann $\neg T(t)$ daraus gemacht. Allerdings musste Prof. Thomas mich erst auf den richtigen Weg führen, weil ich das ganze nicht gut erklären konnte. (T ist eine Menge natürlicher Zahlen, und der Automat ist nach Lesen der i-ten Stelle im Zustand $q_j$ gdw. die Menge $X_j$ die Zahl i beinhaltet).

T: Und die andere Richtung?

I: Wir haben die S1S-Formel zunächst in $S1S_0$ transformiert, so dass es nur noch Mengenvariablen vorkommen.

T: Ja, und dann?

I: Für Succ, Sing etc. haben wir NBAs konstruiert. Dann konnten wir für die Negation den Komplementautomaten bilden usw. Für die Existenzquantoren hatten wir bei den Baumautomaten das Projektionslemma...

T: Ja, also per Induktion über den Formelaufbau. Was können Sie über die Komplexität sagen?

I: Hyperexponentiell, denn der Komplementschritt ist exponentiell, und da wir über diese iterieren, erhalten wir den 'Tower of Twos'.

T: Kann es also sein, dass z. B. nie mehr als zehn mal potenziert wird?

I: Nein, wenn in unserer Formel k Negationen vorkommen, haben wir k mal 'zwei hoch'.

T: Ja, also nicht elementar (Das wollte er wohl hören). Wir hatten noch eine andere Logik kennengelernt.

I: Wir können LTL-Model-Checking mithilfe von NBAs lösen. Dazu erstellen wir zunächst einen Automaten für die Kripke-Struktur.

T: Überspringen wir den Schritt mal. Wie funktioniert der Automat für die LTL-Formel?

I: Der Automat simuliert eine $\varphi$-Expansion, d. h. wir berücksichtigen alle Teilformeln von $\varphi$.

T: Und wie sehen die Zustände dann aus?

I: Das sind Vektoren über 0 und 1. 1, wenn die entsprechende Teilformel an der Stelle wahr ist. Die letzte Stelle steht für die Formel $\varphi$.

## 3 Baumautomaten und Anwendungen

T: Na gut. Kommen wir zu den Baumautomaten. Wann akzeptiert ein BBA?

I: (erklärt)

T: Dann haben wir ja noch Muller- und Paritätsbaumautomaten kennengelernt. Was können Sie zur Beziehung von BBAs zu diesen sagen?

I: BBAs sind echt schwächer, weil sie unter Komplement nicht abgeschlossen sind.

T: Können Sie eine BBA-erkennbare Sprache nennen, deren Komplement nicht BBA-erkennbar ist?

I: Die Sprache der Bäume, bei denen alle Pfade... nein, bei denen ein Pfad unendlich viele b enthält, ist BBA-erkennbar. Aber 'in jedem Pfad nur endlich viele b' ist nicht BBA-erkennbar, das haben wir mit dem Pumping-Lemma gezeigt.

T: Ja, mit einer Pumping-Bedingung. Ist diese Sprache denn deterministisch MBA-erkennbar?

I: (Musste etwas überlegen, deterministische Baumautomaten haben wir ja in der Vorlesung nicht behandelt. Habe mir dann aber überlegt, wie ich den MBA konstruieren würde und dass er dann deterministisch wäre.) Ja. Soll ich ihn aufmalen?

T: Nein, das reicht mir. Was wissen wir denn über das Komplement von MBAs und PBAs?

I: Wir können PBAs mithilfe der Spieltheorie komplementieren. Wir hatten da Automatenspieler und Pfadfinder; der Automatenspieler hat genau dann keine Gewinnstrategie, wenn der Pfadfinder eine hat, da Paritätsspiele determiniert sind. Die Pfadfinderstrategie können wir raten und...

T: Gut, das genügt. Warum ist sind denn Paritätsspiele determiniert? Ist das immer so?

I: Es gibt auch Spiele, die nicht determiniert sind.

T: Aber bei Paritätsspielen?

I: *überleg* Wir können positionale Gewinnstrategien angeben.

T: Ja, und warum gibt es diese positionalen Gewinnstrategien?

I: Das haben wir per Induktion über die Prioritäten bewiesen.

T: OK. Wir hatten dann noch die Logik PDL. Wie sind denn die PDL-Formeln aufgebaut?

I: Da sind zunächst die atomaren Formeln. Dann Programme, außerdem Tests über Formeln. Reguläre Ausdrücke über Tests und Programme sind wieder Programme.

T: Na gut. Wie hängt das jetzt mit Baumautomaten zusammen?

I: Wir können die Erfüllbarkeit einer Formel über BBAs testen.

# 4 Compilerbau

T: Was ist denn die Aufgabe bei der Syntaxanalyse?

I: Gegeben ein Wort und eine kontextfreie Grammatik G, ermittle, ob das Wort in L(G) liegt, und finde eine Analyse, also eine Folge von Ableitungsschritten.

T: Im Grundstudium hatten wir dazu ja den CYK-Algorithmus. Warum reicht der uns hier nicht?

I: Der CYK-Algorithmus hat Laufzeit $O(n^3)$. Wir können aber z. B. für LL(1)-Grammatiken einen Parser konstruieren, der in $O(n)$ läuft.

T: Und warum kann man das nicht mit jeder kontextfreien Grammatik so machen?

I: Es gibt kontextfreie Grammatiken, die nicht in LL(1) sind, z. B. mehrdeutige Grammatiken. (Dann ein kleiner Ausflug zur Klasse der deterministisch kontextfreien Sprachen, die in CFL echt enthalten ist, aber mit $\mathcal{L}$(LR(1)) übereinstimmt.)

T: Und wie entscheidet man bei der LL(1)-Analyse?

I: Man bestimmt die First- und Follow-Mengen. Daraus kann man die Lookahead-Mengen bestimmen, anhand derer man entscheidet.

T: Wie sind denn die First- und Follow-Mengen definiert?

I: Die First-Menge eines Terminalwortes w beinhaltet den k langen Wortanfang von w, oder das ganze w, falls $|w| < k$.

T: Und die Follow-Mengen?

I: Moment. Die First-Menge eines Wortes mit Nichtterminalsymbolen ist die Vereinigung aller First-Mengen der daraus ableitbaren Terminalwörter.

T: Ach ja, das fehlte noch.

I: Die Follow-Menge eines Nichtterminalsymbols besteht dann aus First-Mengen der Wörter, die in einer Linksableitung rechts davon stehen können. Außerdem ε, wenn es ganz rechts steht. (Der zweite Satz wäre wohl nicht nötig gewesen.)

T: Wann ist denn z. B. eine Sprache mehrdeutig?

I: Wenn sie linkszirkulär, äh, linksrekursiv ist (Beispiel aufgeschrieben). In diesem Fall kann man sie einfach in Greibach-Normalform bringen.

T: Einfach?

I: Die Grammatik ist danach nicht notwendigerweise in LL(1), weil man ja jede kontextfreie Sprache in GNF bringen kann.

T: Ja, aber die Transformation in GNF ist auch nicht so einfach. Wie macht man das denn in einfachen Fällen? Z. B. hier: (aufgeschrieben) $A \to Aa|b$

I: $A \to bA', A' \to aA'|\varepsilon$

T: Kommen wir zu den LR(0)-Grammatiken. Wie sind LR(0)-Mengen definiert?

I: (Zunächst die LR(0)-Auskünfte definiert wie im Skript. Daraufhin kam dann auch schon die nächste Frage.)

T: Neben der Action-Funktion wird noch eine weitere Funktion benötigt.

I: Die Goto-Funktion entscheidet, welche Information bei Shift-Schritten auf den Stack gelegt wird. Auch bei Reduce-Schritten kommt sie zum Einsatz: dabei werden so viele Informationen vom Stack genommen, wie die rechte Regelseite lang ist, dann eine Information gemäß Goto-Funktion auf den Stack gelegt.

T: Kommen wir zur semantischen Analyse. Wozu brauchen wir die?

I: Wir können damit Dinge überprüfen, die mittels CFG nicht überprüfbar sind, z. B. die Deklariertheit von Bezeichnern: das geht mit CFG nicht, weil die Sprache der Wiederholungen nicht kontextfrei ist.

T: Wir haben die Attribute in zwei Klassen eingeteilt.

I: Die synthetischen Attribute werden bottom-up berechnet, die inheriten top-down.

T: Erzählen Sie mal was zu Zirkularitäten!

I: Durch die Einteilung in Innen- und Außenvariablen und die Festlegung, dass Innen- nur von Außenvariablen abhängen dürfen, können Zirkularitäten nicht in den Abhängigkeitsgraphen der Regeln entstehen, erst, wenn man diese zu einem Ableitungsbaum 'verklebt'.

T: Knuth hatte da in einer der ersten Arbeiten zum Thema Attribute ein Beispiel.

I: Binärzahlen. Mit Komma.

T: Und was hat er damit gemacht?

I: *Öhm* Er hat den Wert der Binärzahl berechnet. Sie also geparst.

T: Richtig, das ging mithilfe synthetischer Attribute. Kommen wir zum Schluss noch zur Zwischencodegenerierung. Wie sah denn da der Laufzeitstack aus?

I: Der Prozedurkeller?

T: Ja.

I: Auf dem Prozedurkeller liegen Frames der Form sl:dl:ra:loc. Bei jedem Prozeduraufruf wird ein Frame auf den Keller gelegt. Der Dynamic Link zeigt dann auf das davorliegende Frame, der Static Link auf das Frame der Deklarationsumgebung - so kann sichergestellt werden, dass wir in der Variablenumgebung der Deklarationsumgebung arbeiten - Lokale Variablen sind ja klar, und die Rücksprungadresse...

T: auch.

I: Ja, die gibt die Codezeile an, bei der wir nach dem Prozeduraufruf weitermachen.

T: Gut, dann warten Sie mal draußen.