Prüfungsprotokoll theoretische Informatik

26.06.06

Prüfer: Prof. Thomas, Prof. Vöcking

Angewandte Automatentheorie (AAT)

Compilerbau (CB)

Effiziente Algorithmen (EA)

Note: 1.7

Da mir die Prüfungsprotokolle sehr bei der Prüfungsvorbereitung geholfen haben, habe ich mich entschlossen selber eins zu schreiben. Da ich sehr nervös war kann ich leider nicht so genau wieder geben wie gefragt wurde und was ich darauf geantwortet habe. Aber so grob sollte das passen denke ich.

Am Anfang der Prüfung fragte mich Herr Thomas in welcher Reihenfolge ich geprüft werden möchte. Ich habe mich für die Reihenfolge AAT, CB und EA entschieden.

AAT

WT: Fangen wir mal mit den NEAs über Wörtern an. Da hatten wir ja so eine Äquivalenz Schlange. Damit kann man ja einen Quotientenautomat bauen. Wie sieht eine Transition dieses Quotientenautomats aus? (schreibt auf ([p],a,[q]) ⇔)?

Ich: Ja das ist die Transition im Qutientenautomat. Man kommt mit einem a von der Klasse p in die Klasse q.

WT: Naja, aber wann ist diese Transition im Quotientenautomaten enthalten? Was bedeutet das im Zusammenhang mit dem ursprünglichen NEA?

Ich: Das bedeutet, dass im ursprünglichen NEA es eine Transition geben muss, die von einem Repräsentanten der Schlange-Klasse mit einem a zu einem Zustand aus der Klasse von q kommt.

WT: Gut. Und wenn wir uns jetzt die Sprache anschauen, die dabei heraus kommt, nennen wir sie L' und L wäre die ursprüngliche Sprache. Wie ist die Beziehung zwischen L und L'?

Ich: Im Idealfall sind sie natürlich äquivalent, aber da das ein NEA ist bekommt man ohnehin nicht einen minimalen Automat durch Zustandszusammenfassung.

WT: Ja, aber wie sieht das denn im Allgemeinen aus?

Ich: L' ist eine vergröberte Sicht.

WT: (lacht) Und wie nennt man das dann? Vergröberte Sicht hatten wir ja gar nicht definiert.

Ich: A(L') erkennt mehr Wörter als A(L), L ist also eine Untermenge von L'.

WT: Ja genau. Sie hatten ja vorhin etwas von Minierung von NEAs erzählt. Wie funktioniert das?

Ich: Gar nicht. Oder gar nicht effizient. NEA-Minimerung ist nämlich PSPACE-Schwer

WT: Aha. Und wie zeigt man das?

Ich: Man sucht nach einer Polynomzeitreduktion von PSACE-Schweren Sprachen zum NEA-Nichtuniversalitätsproblem. Das NEA-Nichtunversalitätsproblem ist nämlich ein spezieller Fall der NEA-Minimierung.

WT: Ok, und wie geht man da vor?

Ich: Hab die Reduktion kurz erklärt. Hab etwas rumgestammelt, weil ich den Beweis nicht so richtig gut konnte. Also fing ich an zu erzählen, dass die p-time Reduktion Wörter w über ein Alphabet Sigma in NEAs F(w) überführt. Dann kurz erklärt, dass F(w) genau die Wörter akzeptiert, die keine akzeptierende M,w-Berechnung der polynomial platzbeschränkten TM sind. Er hat mehrmals nach gehakt aber das krieg ich jetzt nicht alles zusammen. Er wollte es schon sehr detailliert wissen.

WT: Gut. Gibt es denn Probleme, die wir effizient lösen konnten?

Ich: Ja, wir hatten das Erreichbarkeitsproblem für 1-PDS. Das war entscheidbar.

WT: 1-PDS?

Ich: Pushdown Systeme mit einem Keller.

WT: Was passiert wenn wir zwei Keller haben? Ist das Erreichbarkeitsproblem dann noch entscheidbar?

Ich: Nein, ist es nicht.

WT: Und wie zeigt man das?

Ich: Man kann das zeigen in dem man das Halteproblem für TM mit einem 2-PDS simuliert. (Hab dann aufgemalt wie die Keller aufgebaut werden)

WT: Und wo steht jetzt die Kellerspitze?

Ich: Hab ihm aufgemalt wo die Kellerspitzen sind. Auf dem Band welches vom Lesekopf rechts alles beinhaltet ist die Kellerspitze links und auf dem anderen Keller ist die Kellerspitze rechts.

WT: Wie sieht das denn bei Zählersystemen aus?

Ich: Wenn wir nur einen Zähler haben können wir einen 1-PDS simulieren. Damit ist das Entscheidbarkeitsproblem für 1-ZS entscheidbar.

WT: Wie funktioniert das denn?

Ich: Nun, wir können ja Wörter auch als Zahlen repräsentieren (Stichwort k-adische Darstellung)

WT: Und wo ist die Kellerspitze?

Ich: An meiner Zeichnung dann erklärt wo die Kellerspitzen sind.

WT: Und wieso macht man das so?

Ich: Damit man besser damit rechnen kann.

WT: (lacht) Ja, besser Rechnen kann man damit das stimmt! Aber was passiert da genau? Was passiert wenn sie die Zahl 395 haben und die 3 Streichen?

Ich: Ja im ZS muss man dann durch 10 dividieren ohne Rest.

WT: Richtig. Man braucht das also damit man die arithmetischen Operationen durchführen kann. Kommen wir zu Petri-Netzen. Kennen sie da Probleme, die wir entscheiden konnten?

Ich: Ja, das Beschränktheitsproblem.

WT: Was ist das?

Ich: Man versucht heraus zu finden ob von einer Anfangsmarkierung eines Petrinetzes man eine obere Schranke K existiert, sodass jede Markierung kleiner ist als K.

WT: Malen sie mal bitte ein Beispiel auf. Wie funktioniert das denn mit den Markierungen? Und wie kommt man von einer in die Nächste?

Ich: Hab ein recht einfaches Petri-Netz mit 3 Stellen und einer Transition aufgemalt und ihm erklärt, dass die Stellen im Vorbereich der Transition alle mindestens eine Markierung haben müssen damit die Transition schalten kann.

WT: Richtig. Wenn die Transition schaltet dann zieht man eine Markierung von jeder Stelle des Vorbereichs von t ab und addiert auf alle Stelle des Nachbereichs eine Markierung hinzu. Und wie funktioniert das jetzt genau mit der Beschränktheit?

Ich: Karp-Miller-Baum erstellen und gucken ob in irgendeiner Komponente ein Unendlichkeitssymbol steht.

WT: Wie sind denn die Abbruchbedingungen für den Karp-Miller-Baum?

Ich: Deadlock und Wiederholung. Daher ist der Karp-Miller-Baum endlich, weil sobald man eine Stelle findet, die echt größer ist als eine andere auf dem Pfad zur Wurzel, so wird in der Komponente, die echt größer ist ein Unendlichkeitssymbol eingetragen, so dass bei der nächsten Wiederholung der Algo abbricht.

(Herr Thomas hat noch weitere Fragen zu Karp-Miller gestellt und wieso das Ding jetzt endlich ist. Hab nicht so ganz gerafft was er von mir wollte und hab einfach mal ein paar Stichworte erwähnt. Königs-Lemma zum Beispiel. Das war's aber dann nicht wirklich)

WT: Ok, machen wir weiter. Petri-Netze sind ja so eine Art Zählersystem. Können wir von 2-ZS in Petri-Netze transformieren?

Ich: Hatte kein Plan was er von mir wollte. Hab laut überlegt und erzählt, dass 2-ZS ja irgendwie von den PDS abstammen und die von den PDAs. PDAs erkennen ja bekanntlich die kontextfreien Sprachen. Petri-Netze erkennen aber nicht alle CFLs, also dürfte das nicht gehen. Letzten Endes stellte sich aber heraus, dass es doch irgendwie geht. Wie das genau passiert wurde nicht in der Vorlesung behandelt. Er wollte auch keinen Beweis oder ähnliches.

Compilerbau

WT: Gut. Kommen wir zum Compilerbau. Wie funktioniert die lex. Analyse?

Ich: Vom einfachen Matching-Problem erzählt. Man hat nur einen regulären Ausdruck und zerlegt das Quellprogramm anhand dieses Ausdrucks...

WT: Und wie sieht das im allgemeinen Fall aus?

Ich: Da hatten wir n reguläre Ausdrücke. Aber Zerlegung und Analyse sind nicht Eindeutig, da die regulären Ausdrücke nicht notwendigerweise disjunkte Sprachen definieren. Deshalb sucht man nach dem längsten Match, was die Zerlegung eindeutig

macht. Damit die Analyse ebenfalls eindeutig ist wählt man den ersten regulären Ausdruck, der das (Teil)-Wort matcht => flm-Analyse.

WT: Genau. Und wie macht man das genau?

Ich: Reguläre Ausdrücke mit Thompson Konstruktion in NEAs überführen. NEAs mit der Potenzmengenkonstruktion deterministisch machen. Produktautomat bauen. Diesen dann zum Backtrack-Automat erweitern.

WT: Wie ist die Komplexität?

Ich: Hmmm, müsste irgendwas exponentielles sein. Wir haben ja die Potenzmengenkonstruktion.

WT: Ja. Wenn sie jetzt mal r als die Anzahl der regulären Ausdrücke und w als die Länge des Eingabewortes bezeichnen. Wie sieht die Komplixität in Abhängigkeit von w und r aus?

Ich: Hmmm.. Weiss nicht so genau.

WT: Gut. Kommen wir zu nächsten Phase. Welche ist das?

Ich: Die syntaktische Analyse.

WT: Richtig. Da hatten wir ja zwei Ansätze. Wissen sie noch welche?

Ich: Wir hatten den Top-Down und den Buttom-Up Ansatz. Bei TD haben wir LL(1)-Grammatiken, bei BU LR(0), LR(1) betrachtet.

WT: Schreiben sie mal bitte die Definition von LL(k) auf.

Ich habe angefangen die Definition aufzuschreiben und hatte den totalen Blackout. Ich wusste nicht mehr so genau welche der first-Mengen denn jetzt gleich sind. Es war einfach nicht mehr da. Ich habe alle Kombinationen probiert. Herr Thomas war nicht sehr zufrieden damit und hat mir Tipps gegeben. Ich habe es auch mit Hilfe nicht auf die Kette gekriegt.

WT: Machen wir weiter. Wie kann man denn testen ob eine Grammatik LL(1) ist?

Ich: Man berechnet die la-Mengen und überprüft Regelalternativen auf Disjunktheit der la-Mengen.

WT: la-Mengen?

Ich: Look-ahead-Mengen.

WT: Ja. Nun wir hatten ja in der semantischen Analyse Attribute benutzt. Können sie mir da etwas drüber erzählen?

Ich: Aufteilung in synthetische und inherite Attribute. Aufteilung in Innen- und Außenvariablen. Synthetische Attribute werden standardmäßig BU berechnet, inherite TD.

WT: Können sie mir mal ein Beispiel zeigen?

Ich: Eine Regel A -> BC aufgemalt und erklärt. Wichtig war für ihn noch zu hören, dass inherite Attribute der Unterseite von synthetischen Attributen der Unterseite abhängen können und dass dadurch Zirkularitäten auftreten können.

WT: Erzählen sie mir etwas mehr zu Zirkularitäten.

Ich: Im Abhängigkeitsgraphen der Regeln treten keine Zirkularitäten. Erst durch das Verkleben der Regeln zu einem Abhängigkeitsgraphen eines Ableitungsbaum t können sich Zirkularitäten ergeben.

WT: Kann algorithmisch Testen ob Zirkularität auftritt?

Ich: Ja, das geht. Man verklebt ja die Abhängigkeitsgraphen aus endlich vielen Bausteinen. Also kann man in endlich vielen Schritten Zirkularität feststellen.

WT: Nun, das ist nicht immer so der Fall, aber hier stimmt das mit der Beschränktheit. Letzte Frage: Wie ist die Komplexität des Zirkularitätstest?

Ich: Exponentiell.

WT: Ok, das war’s dann von mir.

Effiziente Algorithmen

BV: Ja, fangen wir am besten mal mit Min-Cuts an. Können Sie mir erklären was Min-Cuts sind?

Ich: Wir haben einen Multi-Graphen, dessen Knoten wir in disjunkte Mengen Q und S unterteilen. Ein Cut ist die Menge der Kanten von Q nach S.

BV: Ja genau. Und wir suchen den kleinsten Schnitt also die Menge mit den wenigsten Kanten. Nun hatten wir ja zwei randomisierte Algorithmen für diese Problem. Können sie mir zuerst den einfachen Ansatz erklären?

Ich: Ja, der hatte ne Laufzeit von $O(n^2)$ und heisst Contract. Solange der Graph mehr als zwei Knoten hat wird uniform zufällig eine Kante kontrahiert. Wenn dann nur noch zwei Knoten übrig sind gibt man die überlebenden Kanten aus.

BV: Genau. Wissen Sie die Erfolgswahrscheinlichkeit noch?

Ich: $2/n^2$

BV: Und wie kann man die Erfolgswahrscheinlichkeit erhöhen?

Ich: W'keitsamplifikation. Wir Iterieren dieses Verfahren und wählen den kleinsten Schnitt aus. Um eine konstante Erfolgswahrscheinlichkeit zu bekommen iterieren wir n^2 mal. (Hab die Formel aufgeschrieben mit der man die konstante Fehlerwahrscheinlichkeit $(1/e)^{2t}$ bekommt.)

BV: Ja. Also haben wir insgesamt eine Laufzeit von $O(n^4)$. Damit haben wir uns ja nicht zufrieden gegeben. Wir möchten ja was schnelles fast lineares haben. Wissen Sie wie das ging?

Ich: Ja, der Algo Fastcut war flotter. Hatte eine Laufzeit von $O(n^2 \log n)$.

BV: Ja. Wie arbeitet Fastcut?

Ich: Fastcut aufgesagt. Hatte Probleme mich an den Faktor zu erinnern. Er wollte lediglich die Größenordnung hören. Hab's aber letztlich doch noch geschafft.

BV: Schreiben Sie bitte die Rekursionsgleichung auf.

Ich: $T(n) = 2\ T(n/\text{Wurzel}(2)) + O(n^2)$. Rekursionstiefe ist log(n), es gibt 2^l Teilprobleme der Größenordnung $O(n/\text{Wurzel}(2)^l)$. Aufwand pro Teilproblem $O(n_l^2)$. Also ergibt sich eine Gesamtlaufzeit von $O(n^2 \log n)$.

BV: Ja. Was für eine Erfolgsw'keit haben wir dann?

Ich: Omega von 1 durch log n.

BV: Können sie mir dafür die Gleichung aufschreiben?

Ich: Ähm... (Ich wusste es einfach nicht, ich hab's mir nämlich gar nicht erst angeguckt)

BV hilft mir noch ein bisschen, sieht aber ein, dass ich das nicht peile und klärt mich auf.

BV: Und wenn wir da was konstantes haben wollen?

Ich: Dann müssen wir halt log n oft den Algo ausführen.

BV: Ja. Wir bekommen also eine Laufzeit $O(n^2 \log^2 n)$. Machen wir mal einen Sprung zu Approximationsalgorithmen. Wir hatten uns ja mit Makespan beschäftigt, zuerst auf allgemeinen Maschinen und dann haben wir uns den allgemeinen Fall angeschaut. Aber zuerst der einfache Fall. Was hatten wir da für Ansätze?

Ich: Wir hatten LL und LPT. LL verteilt die Jobs wie sie gerade kommen auf die Maschine mit der geringsten Last. Das war eine 2-Approximation.

BV: Man konnte den Faktor ja recht anschaulich zeigen. Wissen sie noch wie das geht?

Ich: Ja, da gab's ja dieses Beispiel mit...

BV: Naja, das Beispiel zeigt halt, dass der Faktor scharf ist. Aber wie beweist man ihn?

Ich: Ähmm... Öhhhhmmm

BV: Da hatten wir was mit unteren Schranken... haben davon das Maximum genommen.

Ich: Ja. Richtig... Entweder die Laufzeit des längsten Jobs oder die durchschnittliche Laufzeit aller Jobs.. Davon das Maximum.

BV: Genau. Und dann kommt man mit ein paar Umformungen auf 2 als Approximationsfaktor. Und wie war das mit LPT?

Ich: LPT steht für Longes Processing Time. LPT sortiert die Laufzeiten der Jobs absteigend und arbeitet dann wie LL.

BV: Und welchen Approximationsfaktor hat LPT?

Ich: ¾.

BV: Ja... ähm... Nä. 4/3.

WT: Ich hab mich schon gewundert. Kann ja irgendwie nicht sein.

Ich: Klar. 4/3. Hab mich natürlich nur verplappert.

BV: Wie haben wir das bei allgemeinen Maschinen gemacht? Das war ja etwas komplizierter und wir hatten eine Matrix, deren Einträge i,j die Laufzeit von Job i auf Maschine j war.

Ich: (hatte mir das nur kurz angeguckt, war also recht aufgeschmissen) Also ein ominöses Orakel hat uns einen optimalen Makespan T angegeben.

BV: So ominös ist das gar nicht. Wir machen einfach Breitensuche und gucken ob das jetzt ein optimaler Makespan sein kann...

Ich: Gut. Dann ist es halt nicht so ominös! Jedenfalls hatten wir das als ILP formuliert und mussten da was mit einem speziellen Bin-Packing lösen. Wir hatten das vorher ohne Orakel und T probiert aber da war der Integrality Gap zu groß.

BV (war etwas unzufrieden mit meiner schwammigen Antwort): Ja und hatten es dann relaxiert, mussten dann aber auf ganzzahlige Wert zurück. Wie haben wir das gemacht?

Ich: ähhhhh

BV: Da hat sich der Approx-Faktor ergeben. Wie groß war der noch mal?

Ich: Ähhhhh.... 2.

BV: Genau. Und wie funktioniert das?

Ich: Da muss ich passen.

Herr Voecking hat mir dann noch weitere Tipps gegeben hat aber eingesehen, dass es zwecklos ist und es mir und Prof. Thomas erklärt.

BV: Gut, dann machen wir noch einen Sprung. Letzte Frage: Wir hatten ja den Algorithmus von Cristofides. Das ist ja ein Klassiker. Können sie mir sagen wie der geht?

Ich: MST T bestimmen. Knoten ungeraden gerades heraus nehmen und ein min-cost Matching M suchen. Dann Euler-Tour bestehend aus Kanten von T und M ausgeben. Optimales Matching existiert, weil wir gradzahlig viele Knoten mit ungeradem Grad und einen vollständigen Graphen haben. Approxfaktor ist 1.5.

BV: Können sie den Approximationsfaktor beweisen?

Ich: Ja. Wir nehmen eine optimale TSP-Tour t mit kosten opt, streichen die Knoten heraus, die im MST ungeraden Grad haben und erhalten einen Kreis t' mit Kosten <= cost(t). Der Kreis besteht aus zwei perfekten Matchings. Da wir nun das kostengünstigere Matching nehmen hat das Matching M Kosten höchstes 0.5 * opt.

BV: Ja. Und die Kosten eines MST sind nicht größer als die einer TSP-Tour als erhalten wir dann zusammen Kosten 1.5*opt. Gut, das war's von mir.

WT: Warten Sie bitte draußen.

Resümee

Herr Thomas und Herr Voecking haben sich entschlossen mir ein Gut mit einem Plus dahinter also eine 1,7 zu geben. Für eine Eins hatte ich zu viele Lücken, aber Prof.

Thomas hat es als positiv empfunden, dass ich im AAT-Teil mir einige Sachen überlegen konnte. Schlecht war, dass ich die LL(k) Definition nicht hin bekommen habe.