Protokoll Diplomprüfung
Theoretische Informatik

10.10.2008

Prüfer: Thomas, Noll
Themen: Angewandte Automatentheorie (SS 08),
Effiziente Algorithmen (SS 08),
Compilerbau (SS 08)
Dauer: ca. 40min
Note: 1.0

Ich durfte mir die Reihenfolge der Themen aussuchen:

1 Angewandte Automatentheorie

T(homas): Was ist denn ein Quotientenautomat?

I(ch): (konnte mich nicht an die genaue Definition erinnern...uah, fing ja gut an!) Habe was von Kongruenzen, und der kanonischen Zustandsäquivalenz erzählt. Und dass man DEAs mittels einem Quotientenautomat zur kanonischen Zustandsäquivalenz minimiert...

T: Wie sehen die Zustände in einem Quotientenautomat aus?

I: Zustände, die im Ausgangsautomaten in einer Äquivalenzklasse bzgl. der betrachteten Relation liegen, werden im Quotientenautomaten zu einem Zustand zusammengefasst. Eine solche Zustandsäquivalenz wird von einem Homomorphismus (h: $\mathcal{A} \rightarrow \mathcal{B}$ aufgeschrieben) induziert.

T: Was kann man über das Verhältnis der Sprachen der beiden Automaten sagen?

I: L($\mathcal{A}$) ist Teilmenge L($\mathcal{B}$). (aufgeschrieben)

T: Welche Rolle spielt der Quotientenautomat denn in der NEA-Minimierung?

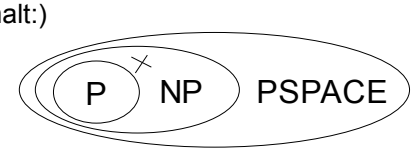
I: Die NEA-Minimierung ist PSPACE schwer, genauso wie das NEA-Äquivalenzproblem, daher kann man hier, anders als im Fall der DEAs, nicht einfach die äquivalenten Zustände zusammenfassen. Man betrachtet stattdessen die Bismulationsäquivalenz und berechnet den Bisimulationsquotienten mit den Blockverfeinerungsalgorithmus. (aufgemalt:)
Bisimulationsquotient ≥ „Zustandsäquivalenz-Quotient“ ≥ minimaler Quotient ≥ minimaler DEA

T: Wie ist den überhaupt die Klasse PSPACE definiert?

I: Das sind alle Probleme, die mit einer polynomial platzbeschränkten TM lösbar sind.

T: Wie verhalten sich P, NP, PSPACE zueinander?

I: (aufgemalt:)



T: (Malt das x in die Zeichnung.) Gibt es den Probleme hier?

I: Ja, ehm, man vermutet, dass P echte Teilmenge von NP, aber das ist ein offenes Problem. Schreibe auf P „Teilmenge“ NP „Teilmenge“ PSPACE

T: Kommen wir zu Baumautomaten. Wie sehen da die Transitionen aus?

I: Man hat ein Rangalphabet und damit Symbole mit bestimmten Stelligkeiten. Die Transitionsfunktion von DBA z.B. hat die Form: (Definition hingeschrieben und erklärt)

T: Kann man testen, zwei Baumautomaten äquivalent sind?

I: Ja, genau wie im Fall der Wörter kann man Baumautomaten minimieren und dann ihre reduzierten Versionen auf Isomorphie testen.

T: Wir hatten auch Baumautomaten mit unbeschränkter Verzweigung.

I: Ja, die XML-Automaten.

T: Wie sieht da die Transitionsrelation aus?

I: Die möglichen Folgezustände sind aus regulären Sprachen über der Zustandsmenge. Die definiert man mittels regulären Ausdrücken oder endlichen Automaten über Q.

T: Wir hatten ja auch unendliche Transitionssysteme mit einigen unentscheidbaren Problemen. Können sie mir welche nennen?

I: PDS. Für 2-PDS ist z.B. das Erreichbarkeitsproblem unentscheidbar. Perti-Netze sind auch unendliche Transitionssysteme, aber hier sind alle Probleme die wir betrachtet haben entscheidbar.

T: Welche Probleme waren das?

I: Das Beschränktheits-, das Erreichbarkeits- und das Überdeckungsproblem.

T: Ja richtig. Aber nochmal zu den unentscheidbaren Problemen. Das Erreichbarkeitsproblem für 2-PDS ist unentscheidbar. Kennen sie ein einfacheres Problem, welches dennoch unentscheidbar ist?

I: Das Erreichbarkeitsproblem für 2-Zähler bzw. 2-Registermaschinen. Dies zeigt man durch die Reduktion des Erreichbarkeitsproblems für n-PDS auf das Erreichbarkeitsproblem n+1 Registermaschinen, und die Reduktion des Erreichbarkeitsproblems für n >= 3 Registermaschinen auf das Erreichbarkeitsproblem für 2 Registermaschinen.

T: Wie zeigt man denn den ersten Schritt?

I: Man fasst die Inhalte der n Keller als Dezimalzahlen auf und packt den Inhalt von Keller i in das Register Xi (aufgemalt wie).

T: Wie kann man simulieren, dass die oberste Zahl vom Stack genommen wird?

I: Man teilt durch 10.

T: Und was macht man, wenn man anstatt Zahlen Buchstaben in den Registern stehen hat?

I: Hm,... man könnte den Inhalt der Keller jeweils durch Potenzen der ersten Primzahlen darstellen.

T: Wie würde das für ACC aussehen?

I: $2^1 \cdot 3^0 \cdot 5^2$, ach ne, das geht so nicht... herumprobiert. $2^1 \cdot 3^3 \cdot 5^3$ also erste Stelle steht das Symbol Nummer 1, ein a, an der zweiten Stelle steht Symbol Nummer 3, ein c, an der dritten Stelle steht Symbol Nummer 3.

T: (War damit nicht ganz zufrieden.) Das ist aber ziemlich aufwendig.

I: (Keinen Plan) Man müsste irgendwie in einem anderen Ring rechnen. Modulo 26 oder so...
(Prof. Thomas

T: Sie meinen zur Basis 26. Naja, man hat ja bei Dezimalzahlen die Zahlen 1 bis 9. Warum nicht die 0?

I: Wenn die 0 ganz unten im Keller liegen würde, würde sie vorne an der Zahl im zugehörigen Register stehen und man könnte darauf nicht mehr zugreifen.

T: Ja, und wir rechnen hier zur Basis 10. Dann nehmen wir bei Buchstaben?

I: Basis 27.

2 Effiziente Algorithmen

T: Ja. Gut, machen wir weiter mit Effiziente Algorithmen. Gehen wir zu Randomisierten Algorithmen. Was sind Las Vegas und Monte Carlo Algorithmen?

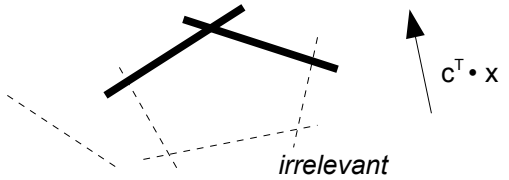
I: Las Vegas Algorithmen haben eine Fehlerwahrscheinlichkeit von 0, d.h. er berechnet immer eine korrekte Lösung. Aber die Laufzeit ist eine Zufallsvariable. Monte Carlo Algorithmen dagegen haben eine positive Fehlerwahrscheinlichkeit, während die Laufzeit deterministisch ist.

T: Ein Beispiel für einen Las Vegas Algorithmus ist ja SeideLP. Wie ist die Problemstellung beim SeideLP?

I: Man will ein LP mit d Variablen x_1 ,..., x_d mit Zielfunktion maximiere $c^T \bullet x$ unter den Nebenbedingungen $A \bullet x \leq b$ lösen.

T: Wie funktioniert der Algorithmus?

I: Man beginnt mit der Menge H der initialen Nebenbedingungen. Zusätzlich hat man noch Box-Bedingungen für jede Variable um sicherzustellen, dass das LP beschränkt ist. Die Idee ist, dass die meisten Nebenbedingungen für die Lösung irrelevant sind. (Male auf:)



(1) Wenn man nur noch eine Variable (d = 1) oder nur eine NB (m = 1) hat, kann man das LP einfach lösen. Bei einer Variablen erfüllt man die strengste NB im Sinne der Zielfunktion möglichst gut. Wenn man nur eine NB hat, entspricht das Problem dem relaxierten Rucksackproblem. Da kann man einfach die Variablen „nach Effizienz bzgl. der Zielfunktion voll machen“.
(2) Ansonsten nimmt man uniform zufällig eine NB h aus H heraus und löst das entstehende LP rekursiv (opt(H \ {h})).
(3) Falls das Ergebnis opt(H \ {h}) die NB h nicht verletzt, gibt man diese Lösung aus.
(4) Wenn die NB h verletzt wird, weiß man, dass das Optimum auf dieser NB-Hyperebene liegt. Daher berechnet man den Schnitt dieser Hyperebene mit dem Lösungspolyhedron. Dann löst man das resultierende LP mit einer Dimension weniger rekursiv.
(5)

T: (unterbricht mich irgendwo) Ja, das reicht. Wie kann man die Laufzeit abschätzen?

I: O (m • d!)

T: (Druckste rum, schien ihm zu wenig zu sein) Wie kann man die einzelnen Schritte abschätzen?

I: Für d > 1 und m >1 kann man die Schritte so abschätzen:
(1) O(1)
(2) T(m, d-1) da man eine NB heraus geschmissen hat
(3) O(d) Test, ob die Lösung, also Belegung der d Variablen, die heraus genommene NB verletzt.
(4) T(m+1, d-1) (+ O(d • m) vergessen, ist aber nicht aufgefallen =))Man hat eine hat zwar eine NB weniger, aber wenn man den Schnitt mit dem Lösungspolyhedron berechnet, kommen auch zwei neue durch umgeformte Box-Bedingungen hinzu, daher hat man m+1 NBen. Durch den Schnitt mit dem Lösungspolyhedron hat man nur noch d-1 Variablen.

Die Wahrscheinlichkeit, dass der 4. Schritt ausgeführt wird, also, dass eine relevante NB getroffen wird, ist nur d / m (am Bild gezeigt).

T: (unterbricht mich) Ja das reicht schon. Was kann man noch machen, wenn man Probleme nicht direkt effizient berechnen kann?

I: Approximationsalgorithmen.

T: Was ist die Approximationsgüte?

I: Der Approximationsfaktor. Wenn $\mathcal{A}$ unser Approximationsalgorithmus ist der Approximationsfaktor auf Eingabeinstanz I $r_{\mathcal{A}}(I) = w_{\mathcal{A}}(I) / opt(I)$ (aufgeschrieben und erklärt).

T: (unterbricht mich) Können sie das anhand eines einfachen Approximationsalgorithmus erklären?

I: Die Approximation des Vertex-Cover Problems ist einfach. Der Approximationsalgorithmus hat Approximationsfaktor 2. D.h. für alle Eingabeinstanzen I gilt: $w_{\mathcal{A}}(I) \leq 2 \bullet opt(I)$ (erklärt, aber kam nicht dazu das zu beweisen).

T: Wie funktioniert der Approximationsalgorithmus?

I: Man berechnet ein inklusions-maximales Matching und gibt die Menge aller Endknoten der Kanten im Matching als Approximation für das minimale Vertex-Cover aus.

T: Gibt es auch Probleme, die man nicht approximieren kann?

I: Ja. Das TSP.

T: Warum kann man das nicht approximieren?

I: Also es ex.keine ploynomialzeitberechenbare Funktion mit der ein Polynomialzeit-Approximationsalgorithmus das TSP approximieren könnte. Das zeigt man dadurch, dass es mit so einem Approximationsalgorithmus möglich wäre das NP-harte Hamiltonkreisproblem in polynomieller Zeit zu lösen. Also man reduziert das Hamiltonkreisproblem auf das TSP.

T: Was betrachtet man stattdessen für ein Problem?

I: Das metrische TSP. Hier sucht man eine minimale TSP-Tour auf Graphen auf denen die Dreiecksungleichung gilt.

T: Ja richtig. Wie kann man das nun approximieren?

I: Wir hatten zwei Algorithmen. Zuerst die Berechnung der TSP-Tour via MST. Das ist eine 2-Approximation. Dann gibt es noch den Cristofides Algorithmus. Der liefert eine 3/2-Approximation.
Beim ersten Algorithmus berechnet man zunächst einen MST, dann verdoppelt man die Kanten darin, so dass ein Eulergraph entsteht. Die Eulertour darauf bereinigt man um wiederholt verkommende Knoten, bzw. man gibt die Knoten in der Reihenfolge der Eulertour als TSP-Tour aus.

T: Welcher Schritt ändert sich beim Cristofides Algorithmus?

I: Der 2. Schritt. Man berechnet die Menge V' der Knoten im MST mit ungeradem Grad. Dann sucht man ein Min-Cost Matching auf V' aus Kanten im Ausgangsgraphen. Schließlich sucht man wieder eine Eulertour, diesmal auf dem Graphen aus MST und Min-Cost Matching.

T: Ja. Gut. Kommen wir zu Online-Algorithmen. Wie ist die Güte vom solchen Algorithmen definiert?

I: Das ist der Competitive-Faktor. Wenn wir einen Online-Algorithmus $\mathcal{A}$ haben mit Competitive-Faktor **c**, dann gilt für jeden Anfragesequenz σ die dem Online-Algorithmus nur nach und nach aufgedeckt wird: $C_{\mathcal{A}}(\sigma) \leq \mathbf{c} \bullet C^*(\sigma) + a$ (aufgeschrieben und erklärt:) $C_{\mathcal{A}}(\sigma)$ sind die Kosten vom Online-Alg. auf der Anfragesequenz, $C^*(\sigma)$ sind die Kosten des optimalen Offline-Alg. und a ist eine Konstante.

T: Was gab es da für Algorithmen?
I: Wir hatten das File Allocation Problem und das Paging-Problem. Beim File Allocation Problem geht es um das Verwalten von Dateien in Netztwerken. Wobei wir hier einen sehr vereinfachten Fall von zwei Computern

und einer Datei betrachten (aufgemalt). Es gibt drei mögliche Konfigurationen, die Datei kann entweder auf einem der Computer a und b liegen, oder beide halten die Datei. Unser Online-Algorithmus muss nun entscheiden wo die Datei zu einem Zeitpunkt liegt um möglichst wenig Kosten bei Lese- und Schreibzugriffen zu haben. Beim kopieren der Datei entstehen Migrationskosten...

T: (unterbricht mich) Wie ist die Güte vom Online-Algorithmus für FAP?

I: Der ist 3-competitive.

T: Ok. Ich würde sagen wir gehen rüber zum Compilerbau.

3 Compilerbau

N(oll): Fangen wir mal von hinten an. Codegenerierung. Auswertung von Booleschen Ausdrücken. Was gibt es da für unterschiedliche Herangehensweisen?

I: Man kann boolesche Ausdrücke strikt und sequentiell auswerten. (aufgeschrieben und erklärt:)

strikt:	sequentiell:
$b \wedge / \vee \perp = \perp$	$\text{true} \vee \perp = \text{true}$
	$\text{false} \wedge \perp = \text{false}$
$\perp \wedge / \vee b = \perp$	$\perp \wedge / \vee b = \perp$

N: Wie kann eine solche Unbestimmtheit entstehen?

I: Hm, wenn bei der Berechnung des Booleschen Wertes ein Fehler auftritt. Die Berechnung nicht fertig wird.

N: (hat noch irgendein Beispiel dazu genannt). Wie sehen die jeweiligen Übersetzungsfunktionen dazu aus?

I: Ich nehme einfach mal das Beispiel B_1 and B_2. (aufgeschrieben und Idee erklärt:)

$bt(B_1 \text{ and } B_2, st, a, l)$
$= bt(B_1, st, a, l)$
$bt(B_2, st, a', l)$
a'': AND;

$sbt(B_1 \text{ and } B_2, st, a, \mathbf{a_t}, \mathbf{a_f}, l)$
$= sbt(B_1, st, a, a', a_f, l)$
$sbt(B_2, st, a', a_t, a_f, l)$

zwei zusätzliche Adressparameter für jumping code

N: Ja richtig. Wie kann man diese Information jeweils für die Codegenerierung in einer attributierten Grammatik auswerten?

I: (keine Ahnung was er meint, komische Frage) Hm?... (Baum aufgemalt, immer noch keine Ahnung gehabt)... (Dann bin doch auf etwas gekommen:) Die Symboltabelle wird als inherites Attribut von der Wurzel zu den Blättern durchgereicht. (Nach irgendeinem Hinweis.) Bei der sequentiellen Auswertung hat man noch die beiden zusätzlichen Adressparameter a_t und a_f als inherite Attribute.

N: Was gab es noch für eine Möglichkeit zur Auswertung von Booleschen Ausdrücken, die wir nicht so ausführlich behandelt haben?

I: (aufschrieben:)

$\text{true} \vee \perp = \text{true}$
$\text{false} \wedge \perp = \text{false}$ } wie im sequentiellen Fall
$\perp \vee \text{true} = \text{true}$
$\perp \wedge \text{false} = \text{false}$

Die beiden Booleschen Teilausdrücke werden also parallel ausgewertet.

N: Ja genau. Parallele Auswertung. Ok, kommen wir zur Syntaxanalyse. Wie funktioniert die Bottom-Up Syntaxanalyse?

I: Mit shift und reduce. Beim shift legt man das aktuelle Eingabesymbol auf den Keller bzw. in der LR(0)-Analyse die LR(0)-Menge die in der goto-Funktion von der aktuell obersten LR(0)-Menge und dem

Eingabesymbol. Beim reduce ersetzt man die rechten Seite eine Produktion durch ihre linke Seite.
Im nichtdeterministischen Bottom-Up Analyse Automaten hat man die Konfigurationsübergänge:
(aufgeschrieben:)
reduce: (w, αβ, z) ⊢ (w, αA, zi) wobei π(i) = A → β
shift: (aw, α, z) ⊢ (w, αa, z)

N: Welche Fälle von Nichtdeterminismus gibt es?

I:
Bei $(aw, \alpha\beta, z) \vdash \begin{cases} (w, \alpha\beta a, z) \\ (aw, \alpha A, zi) \end{cases}$ falls π(i) = A → β weiß man nicht ob man shiften oder reduzieren soll.

Bei (w, αa, z) und π(i) = A → a, π(j) = B → a, weiß man nicht mit welcher linken Seite man reduzieren soll.

Bei (w, αβ, z) ⊢ (w, αA, zi) wobei π(i) = A → β
=
(w, γδ, z) ⊢ (w, γB, zj) wobei π(j) = B → δ
weiß man nicht welchen Henkel β oder δ man zum reduzieren nehmen soll.

N: Was gilt für dieses β und δ?

I: δ ist Präfix von β oder umgekehrt.

N: Welchen Fall von Nichtdeterminismus konnten wir sehr einfach auflösen?

I: Den Terminierungs-Nichtdeterminismus. Einfach dadurch, dass wir die Grammatik start-separieren.

N: Nehmen wir mal als Beispiel die Grammatik: S → A, A → Aa | a. Kann man die Grammatik mit einem LR(0)-Parser analysieren?

I: Dazu muss man die LR(0)-Mengen berechnen und gucken, ob es dort Konflikte gibt. (aufgeschrieben:)
LR(0)(ε) = { [S → • A], [A → • Aa], [A → • a] }
LR(0)(A) = { [S → A •], [A → A • a] }
accept (bzw. red 0) / shift Konflikt !

=> Die Grammatik ist nicht in LR(0).

N: Wie könnte man den Konflikt am einfachsten lösen?

I: Mit SLR(1). Man shiftet falls das aktuelle Eingabesymbol a ist und man akzeptiert bei ε, da fo(S) = { ε }.