# Prüfungsprotokoll
# zur
# Theoretischen Diplomprüfung

Carsten Kern
06.10.2004

| | | | |
|---|---|---|---|
| Prüfer | : | Prof. K. Indermark | |
| Beisitzer | : | Benedikt Bollig | |
| Inhalt | : | 1. Logikprogrammierung | (WS 03-04) |
| | | 2. Compilerbau | (SS 04) |
| | | 3. Effiziente Algorithmen | (nach Buch von Prof. Hromkovic) |
| Dauer | : | 55 Minuten | |
| Note | : | 1.0 | |

## 1 Logikprogrammierung

- K.I.: Zum Einstieg etwas Einfaches : Was ist ein Prologprogramm ?
  Ich : Eine endliche Menge definiter Hornklauseln. *Dann habe ich noch kurz erklärt, was überhaupt eine Hornklausel ist.*
- K.I.: Geben Sie mal ein Programm zum Invertieren einer Liste an (*ich musste aber nur die naive Version des reverse aufschreiben*)
  Ich :
  ```
  rev([],[]).
  rev([X|Xs],Zs):- rev(Xs,Ys), append(Ys,[X],Zs).
  ```
- K.I.: Die SLD-Resolution...Wie wird denn da resolviert ?
  Ich : Habe die Definition der SLD-Resolution genannt.
- K.I.: SLD-Resolution ist ja vollständig. Was bedeutet das ?
  Ich : Das bedeutet, dass zu einer unerfüllbaren Hornklauselmenge $\mathcal{K}$ und einer negativen Klausel $N \in \mathcal{K}$ die leere Klausel $\Box$ aus $N$ in $\mathcal{K}$ ableitbar ist.
- K.I.: Na das muss aber noch was präzisiert werden...
  Ich : Achja...natürlich muss das eine lineare Resolution sein.
- K.I.: Ja genau.
  So ... das haben wir zwar nicht bewiesen, aber das Problem die Vollständigkeit der SLD-Resolution zu beweisen kann man ja zurückführen auf ein anderes Problem.
  Ich : Ja, auf die Vollständigkeit der linearen Resolution.
- K.I.: Genau. Was ist denn überhaupt lineare Resolution ?
  Ich : Definition aufgeschrieben.
- K.I.: So, kommen wir mal zum SLD-Baum. Was ist ein SLD-Baum ?
  Ich : Definition gegeben und gesagt, dass in Datalog unendliche Äste erkennbar sind.
- K.I.: Warum sind die in Datalog erkennbar ?
  Ich : Weil es nur eine endliche Anzahl von Datalogfakten gibt und keine Funktionen (der Stelligkeit $n > 0$).

- K.I.: Können Sie sich denn vorstellen **wie** man unendliche Berechnungen erkennen kann ?
  Ich : Hab ein bisschen rumgerätselt und versucht einen Ansatz zu finden. Hab dann auch irgendetwas gesagt, was ok war, aber Prof. Indermark hat dann eingelenkt und mir erklärt, wie man das machen kann. Hab aber leider kein Wort davon verstanden, weil da Sachen ins Spiel kamen, von denen ich noch nie in meinem Leben gehört hatte.

- K.I.: Wir haben ja einige Semantiken für Logikprogramme betrachtet. Unter anderem die Fixpunktsemantik. Dazu hatten wir ja eine Funktion $\underline{\text{trans}}_\pi$ definiert. Wie sah die aus und was kann man sich anschaulich darunter vorstellen ?
  Ich : *Definition der Funktion* $\underline{trans}_\pi$ *angegeben*

- K.I.: Diese Funktion hatte ja einige Eigenschaften wie z.B. Monotonie und Stetigkeit.
  Ich : Ja! Außerdem, dass $\langle \underline{Pot}(\underline{Prim}_\Sigma), \subseteq \rangle$ ein vollständiger Verband ist.

- K.I.: Was bedeutet denn eigentlich das "vollständig" in "vollständiger Verband" ?
  Ich : Hmmm... (hatte keine Ahnung) Vielleicht, dass ...
  K.I.: *Hat irgendwas erklärt, wovon ich absolut keine Ahnung hatte. Aber das schien für die Note nicht relevant gewesen zu sein.*

- K.I.: Definieren Sie mir mal den kleinsten Fixpunkt, wie er in der Fixpunktsemantik auftritt.
  Ich : Definition aufgeschrieben.

- K.I.: Und definieren Sie jetzt mal mit Hilfe dieses kleinsten Fixpunktes die Fixpunktsemantik eines Logikprogrammes $\pi$ unter der Anfrage $G$.
  Ich : *Definition hingeschrieben und dabei erklärt, was das bedeutet, was ich da aufschreibe.*

- K.I.: Und jetzt wenden Sie mal die Fixpunktsemantik auf das *reverse*-Programm an, das wir eben definiert haben.
  Ich : *Hab mich dabei etwas schwer getan, aber mit der Hilfe von Professor Indermark ging es dann letztendlich doch.*

  *Dann bemerkte Professor Indermark, dass die Zeit schon stark fortgeschritten war und fragte mich, welches Fach ich denn gerne als nächstes hätte. Da ich natürlich Compilerbau viel intensiver gelernt hatte als Effiziente Algorithmen, entschied ich mich dann für CB.*

## 2 Compilerbau

- K.I.: Was ist die syntaktische Analyse ?
  *Weil ich nicht richtig zugehört hatte, hab ich mit der Erklärung der lexikalischen Analyse angefangen. Da hat er mich aber schnell wieder auf die richtigen Bahn gebracht.*

- K.I.: Wir haben ja da in der syntaktischen Analyse Top-down und Bottom-Up Analysen mit Linksableitungen und Rechtsableitungen bzw. gespiegelten Rechtsableitungen. Und außerdem haben wir dann gesehen, wie man aus dem Ableitungsbaum (mit der konkreten Syntax) einen abstrakten Syntaxbaum erzeugen kann mit Hilfe von S-Attributgrammatiken.
  Ich : Ja! Das kann man machen, indem man Bottom-Up diesen $AST$ aufbaut...
  *Professor Indermark wollte das wohl noch genauer erläutert haben, aber als er dann merkte, dass ich da wohl nicht so fit drin war, kam er zum nächsten Thema.*

- K.I.: Wir haben in der Vorlesung ja gesehen, dass Grammatiken mehrdeutig sein können. Wann ist denn überhaupt eine Grammatik mehrdeutig ?
  Ich : Definition der Mehrdeutigkeit einer Grammatik angegeben.

- K.I.: Kommen wir mal zur Bottom-Up Analyse und nehmen mal an, dass in LR(0) Konflikte auftreten, die auch durch SLR(1) nicht behoben werden können... Welche Konfliktarten gibt es überhaupt?
  Ich : Es können Shift-/Reduce- und Reduce-/Reduce-Konflikte auftreten.

- K.I.: Wie sähe denn ein Shift-/Reduce- bzw. ein Reduce-/Reduce-Konflikt in LR(1) aus ?
  Ich : Also... ein Shift-/Reduce-Konflikt in LR(1) liegt vor, wenn in einer LR(1)-Menge das Zeichen hinter dem Punkt einer Shift-Auskunft und das Zeichen hinter dem Komma einer Reduce-Auskunft gleich sind.
  Ein Reduce-/Reduce-Konflikt liegt dann vor, wenn die Zeichen hinter den Kommas zweier Reduce-Auskünfte dieser LR(1)-Menge gleich sind.

- K.I.: Kommen wir zur Codeerzeugung. Da hat man ja Übersetzungsfunktionen und dabei unterscheidet man ja bei der Übersetzung von booleschen Ausdrücken die strikte und nicht-strikte Semantik. Was ist denn die Idee beim Jumpingcode (der ja in den Übersetzungsfunktionen der nicht-strikten Semantik boolescher Ausdrücke verwendet wird) ?
  Ich : Man hat 2 zusätzliche Adressen, nämlich $a_t$, $a_f$, über die man im Code an die richtigen Stellen springt. Z.B. bei der Verzweigung ( $\underline{\text{nbt}}(\underline{\text{if}}$ BE $\underline{\text{then}}$ $\Gamma_1$ $\underline{\text{else}}$ $\Gamma_2$ ), springt man mit $a_t$ an die Codeadresse von $\Gamma_1$ und mit $a_f$ an die Codeadresse von $\Gamma_2$.

- K.I.: Ja, zeigen sie das doch mal explizit an der nicht-strikten Übersetzung des *AND*.
  Ich : Ok. Also ... beim *AND* der nichtstrikten Semantik ist ja die Auswertung schon *false*, wenn der erste Parameter zu *false* ausgewertet wird. Darum erhält man folgendes :
  $$\begin{aligned}\underline{\text{nbt}}(\text{BE1 } \underline{\text{and}} \text{ BE2, st, } a, a_t, a_f, \text{l}) := \;& \underline{\text{nbt}}(\text{BE1, st, } a\text{ , } a', a_f, \text{l})\\ & \underline{\text{nbt}}(\text{BE2, st, } a', a_t, a_f, \text{l})\end{aligned}$$

- K.I.: Welche Befehle gibt es denn bei der nicht-strikten Semantik nicht ?
  Ich : *Nach einigem nutzlosen Herumargumentieren mit den Übersetzungsfunktionen kam ich dann endlich auf die Antwort :* Es gibt dort keine Booleschen Operatoren wie *AND*, *OR*, *NOT*.

- K.I.: Richtig. So ... ui es ist ja schon sehr spät. Lassen Sie uns mal zu den Effizienten Algorithmen kommen.

# 3 Effiziente Algorithmen

- K.I.: Beginnen wir mal mit dem Travelling Salesman Problem ! Definieren Sie das mal.
  Ich : *Bekomme erstmal einen Schock, weil das erstens das schwerste aller Probleme ist, über das man sich auslassen kann und ich mir das zweitens eher weniger angeschaut habe. Aber habe dann das Problem erklärt :* Gegeben ist ein Graph G=(V,E) mit dem Ziel jeden Knoten genau einmal zu besuchen, und dabei minimale Kosten zu benötigen.

- K.I.: Aber was ist denn das für ein Graph ?
  Ich : *Ach ja ... das hatte ich ja garnicht erwähnt...* Der Graph muss natürlich vollständig sein.

- K.I.: Kommen wir mal zu den Methoden, wie man solch ein Problem lösen kann. Es gibt ja da den naiven Ansatz mit backtracking. Was benötigt man denn da ?
  Ich : Man muss sich eine Struktur in der Menge aller zulässigen Lösungen schaffen, um diese zu durchsuchen. Das macht man mit Hilfe eines Baums.

- K.I.: Und wie sieht dieser Baum aus ?
  Ich : *Da ich dachte, dass Professor Indermark sich auf die allgemeine Gestalt des Lösungsbaums $T_{\mathcal{M}(x)}$ beziehen würde, fing ich an $T_{\mathcal{M}(x)}$ zu definieren. Das war aber eben genau nicht, was er wollte. Er wollte den direkten Bezug auf das TSP. Ich habe ihm dann gesagt, dass ich eigentlich nur die durch Branch-and-Bound optimierte Lösung von TSP kennen würde, und er wollte dann, dass ich ich ihm die Idee dort erläutere. Das habe ich dann auch getan und er war zufrieden.*

- K.I.: Kommen wir mal zum Rucksackproblem. Können Sie mir das mal beschreiben ?
  Ich : Man hat einen Rucksack der Größe $b$, und $n$ Objekte mit den Gewichten $w_1, \ldots, w_n$ und den

Werten $c_1, \ldots, c_n$. Ziel des KSP ist es dann, den Wert der eingepackten Objekte zu maximieren unter der Bedingung, das Fassungsvermögen des Rucksacks nicht zu übersteigen.

- K.I.: Und womit kann man das KSP lösen ?
  Ich : Mit Hilfe dynamischer Programmierung. Das hatten wir im Zusammenhang mit pseudopolynomiellen Algorithmen.

- K.I.: Ok. Kommen wir mal zu Heuristiken. Da gibt es ja Simulated Annealing und Genetische Algorithmen. Was haben die beiden denn gemeinsam ?
  Ich : *Da ich erst nicht wusste, worauf Professor Indermark hinaus wollte, gab ich die triviale Antwort :* Sie basieren beide auf Heuristiken.
  *Da ihm das aber nicht auszureichen schien, überlegte ich weiter und kam schließlich zu dem Schluss :* Ah, die beiden Ansätze basieren auf zufälligen Entscheidungen.

- K.I.: Kommen wir mal zu Divide-and-Conquer. Was ist die grundlegende Idee bei dieser Algorithmenentwurftstechnik ?
  Ich : Die Idee liegt darin, ein gegebenes Problem rekursiv in Teilprobleme zu zerlegen, bis diese direkt zu lösen sind und anschließend aus den Lösungen der Teilprobleme die Lösung für das Gesamtproblem zusammenzusetzen.

- K.I.: Wie ist das denn z.B. bei Fibonacci ?
  Ich : Das Problem bei der Lösung von Fibonacci mittels Divide-and-Conquer ist, dass man hier zur Lösung exponentiell-viele Zwischenlösungen erzeugen muss.

- K.I.: Womit geht das denn besser ?
  Ich : Mit dynamischer Programmierung, weil man sich hier Zwischenergebnisse merkt, um später wieder drauf zugreifen zu können.

- K.I.: Wie ist das denn z.B. mit Sortierverfahren wie Quicksort? Da ist Divide-and-Conquer doch gut oder ?
  Ich : Ja genau. Da läuft es in $O(n \log(n))$.

- K.I.: Woran liegt denn das, dass Divide-and-Conquer für diese Art von Algorithmen gut läuft, aber z.B. für Fibonacci so schlecht ?
  Ich : Hmm... weil Fibonacci oft die gleichen Teillösungen mehrere Male neu berechnen muss. Z.B. fib (3) muss bei fib (n) für $n \gg 3$ sehr oft berechnet werden. Da man diese Zwischenergebnisse bei dynamischer Programmierung aber zwischenspeichert, ist dieser Ansatz für Fibonacci wesentlich effizienter (nämlich in $O(n)$).

- K.I.: Was ist das SAT-Problem ?
  Ich : Das ist das Problem, zu einer gegebenen aussagenlogischen Formel (in KNF), eine erfüllende Belegung zu finden.

- K.I.: Zu welcher Klasse von Problemen gehört es ?
  Ich : Zu den NP-schweren.

- K.I.: Warum stört uns das in der Logikprogrammierung nicht ?
  Ich : Weil wir dort Hornformeln betrachten.

- K.I.: Und wie schwer ist aussagenlogische Resolution da ?
  Ich : Die kann man effizient in Linearzeit durchführen.

- K.I.: Es gibt bei den Optimierungsproblemen ja auch ein Problem, bei dem man sich für die Erfüllbarkeit von aussagenlogischen Formeln interessiert. Kennen Sie das ?
  Ich : Ja. Das ist das Max-SAT Problem.

- K.I.: Was ist denn dort das Ziel ?
  Ich : Das Ziel ist, eine Belegung zu finden, so dass mit dieser Belegung eine maximale Anzahl von Klauseln erfüllt wird.
- K.I.: Und wie kann man es angehen ?
  Ich : Genau wie wir eben das TSP angegangen sind, nämlich mit Branch-and-Bound.
- K.I.: Genau. Und dabei vereinfacht sich auch noch der Lösungsbaum.

  *Aber dazu kommen wir jetzt nicht mehr, denn die Zeit ist um. Würden Sie bitte einen Moment draußen warten!*

## 4 Fazit :

Auch ich war, wie die meisten anderen, super nervös vor und auch während der ersten 5 Minuten der Prüfung. Aber durch die Art von Professor Indermark die Fragen zu stellen, beruhigt man sich recht schnell wieder. Aus meiner Sicht ist Professor Indermark ist ein sehr zu empfehlender Prüfer. Auch wenn man mal "ein Brett vor dem Kopf" hat, nimmt er einem das nicht übel und versucht Hilfestellungen zu geben, um einen in die richtige Richtung zu lenken.
Wie man an meinem Prüfungsverlauf sieht, braucht man bei weitem nicht alle Fragen korrekt, bzw. direkt zu beantworten sondern kann sich auch etwas Zeit nehmen, über die Problemstellung, die einem mit der aktuellen Frage gegeben wurde, nachzudenken (vor allem, wenn die Frage über den in der Vorlesung behandelten Stoff hinausgeht).

An alle, die dieses Protokoll lesen :

Ich hoffe, ich konnte euch damit weiterhelfen und wünsche euch viel Erfolg bei euren Prüfungen