Gedächtnisprotokoll

Niklas Hoppe

30. März 2010

Allgemeines

Dauer: ca. 45 Minuten
Note: 1.3

Die Prüfungsatmosphäre bei Prof. Giesl und Dr. Unger ist angenehm und sachlich. Insbesondere Dr. Unger ist ein sehr lockerer und freundlicher Zeitgeselle. Der Prüfungsteil mit Dr. Unger ist zwar kein Gespräch, aber durch ständige Einwürfe, Nachfragen und Tips entspricht es zumindest nicht dem üblichen Frage-Antwort-Spiel.

1 Effiziente Algorithmen

Dr. Unger wollte locker anfangen und fragte mich nach dem Min-Cut-Max-Flow-Theorem. Er fragte mich was das Theorem besagt und wie man den Schritt von der zweiten zur dritten Aussage beweist. Es ging sofort weiter zur Funktionsweise des Algorithmus von Dinitz. Dann sagte er, dass man ja irgendwo im Algorithmus eine Schlange benutzt und wollte wissen warum man denn gerade eine Schlange benutzt und nicht zum Beispiel einen Keller. Ich stand bei dieser Frage total auf dem Schlauch und so sind wir schrittweise durch die Laufzeitanalyse gegangen. Ich konnte ihm am Ende zwar immer noch nicht so genau sagen was er hören wollte, aber damit gab er sich dann auch zufrieden. Dann ging es weiter mit Approximationsalgorithmen: Dr. Unger wollte wissen wie man das Zentrumsproblem approximiert. Ich erklärte also die Funktionsweise des Algorithmus und begründete warum es eine 2-Approximation ist. Außerdem erklärte ich wie man vom Entscheidungs- zum Optimierungsverfahren kommt: Durchprobieren aller Distanzwerte. Dazu wollte er wissen warum man denn eine lineare Suche benutzt anstatt einer Binärsuche. An dieser Stelle wusste ich wieder nicht so recht weiter bis er auflöste: Die Ergebnisse des Entscheidungsverfahrens sind nicht monoton. Zu guter Letzt fragte er mich nach dem (2,2)-kompetetiven Marking-Algorithmus Random. Dazu erklärte ich die Potentialfunktion und wie man diese für den Beweis nutzt. Bevor ich jedoch in den richtigen Beweis einsteigen konnte war die Zeit auch schon um und Dr. Unger gab an Prof. Giesl weiter.

2 Logikprogrammierung

Giesl: Wir haben ja die aussagenlogische Resolution kennengelernt. Was bedeutet denn zunächst mal, dass dieses Verfahren vollständig und korrekt ist?
Ich: Vollständigkeit bedeutet, dass bei jeder unerfüllbaren Klauselmenge auch die leere Klausel aus dieser Menge resolviert werden kann. Korrektheit bedeutet, dass wenn ich die leere Klausel aus einer Klauselmenge resolvieren kann, dann ist die Klauselmenge auch unerfüllbar.
Giesl: Gut, wie würden Sie die Vollständigkeit der aussagenlogischen Resolution denn beweisen?
Ich: Ich mache strukturelle Induktion über der Klauselmenge.
Giesl: Über was genau machen sie da die Induktion?
Nach einer kleinen Diskussion kam heraus: Es ist natürlich eine Induktion über die Anzahl der

verschiedenen atomaren Formeln in der Klauselmenge.
Giesl: Dann machen sie mal weiter.
Ich schreibe den Beweis komplett auf.
Giesl: Kann man mit diesem Beweis denn jetzt auch die Vollständigkeit der Input-Resolution beweisen?
Kurzes Gespräch über die Funktionsweise der Input-Resolution.
Ich: Nein, weil die beiden atomaren Formeln A, $\neg A$ die ich am Ende miteinander resolviere erfüllen nicht zwangsläufig die Kriterien der Input-Resolution.
Giesl: Gut, dann machen wir doch mal etwas mit Prolog. Schreiben sie mir mal ein Prädikat, dass das Maximum einer Liste bestimmt.
Ich:

```
max([],0).
max([X|XS],X) :- max(XS,Z), X > Z.
max([X|XS],Z) :- max(XS,Z), X <= Z.
```

Giesl: Das könnte man ja jetzt auch noch abkürzen.
Ich: Ja stimmt. Man könnte einen Cut setzen und sich dann den Rest des zweiten Prädikats sparen:

```
max([],0).
max([X|XS],X) :- max(XS,Z), X > Z, !.
max([X|XS],Z) :- max(XS,Z).
```

Giesl: Kann man mit logischer Programmierung denn auch alles berechnen was man mit anderen Programmiersprachen berechnen kann?
Ich: Ja natürlich.
Giesl: Und wie macht man das?
Ich: Man zeigt wie man die μ-rekursiven Funktionen in Prolog implementieren kann.
Giesl: Dann zeigen sie mir doch mal wie man die Minimalisierung in Prolog realisiert.
Ich: Wir wollen ein Prolog-Prädikat welches folgendes kann:

$h(k_1, ..., k_n) = k$ gdw. $f(k_1, ..., k_n, k) = 0$
und für alle $1 \leq k' \leq k$ ist $f(k_1, ..., k_n, k)$ definiert und größer als 0.

Das würden wir in Prolog dann folgendermaßen lösen:

```
h(X1,...,Xn,Z) :- f'(X1,...,Xn,0,Z).
f'(X1,...,Xn,Y,Y) :- f(X1,...,Xn,Y,0).
f'(X1,...,Xn,Y,Z) :- f(X1,...,Xn,Y,s(U)), f'(X1,...,Xn,s(Y),Z).
```

Giesl: Gut, dann gehen wir mal zur funktionalen Programmierung über.

3 Funktionale Programmierung

Giesl: Wir haben ja gerade schon ein Prädikat geschrieben um das Maximum einer Liste zu bestimmen. Lösen sie das doch mal in Haskell.
Da ich mit Prüfungsprotokollen gelernt hatte, hatte ich sofort eine elegante Lösung parat. Allerdings war es Prof. Giesl auch bei mir nicht genug Haskell*, weswegen eine zweite Programmieraufgabe folgte.* Ich:

```
foldr max 0
```

Giesl: Ja, das ist tatsächlich eine elegante Variante. Dann hätte ich aber gerne noch was zu Ein- und Ausgabe. Schreiben sie mal eine Funktion `echo`, die einen Integer als Argument nimmt und entsprechend viele Zeichen des Benutzers einliest um diese dann auszugeben.

Ich: *Nach ein wenig Rumgedruckse von mir und dem Hinweis, dass ich noch eine Abbruchbedingung brauche.*

```
gets :: Int -> IO [Char]
gets 0 = return []
gets n = getChar >>= \x -> gets (n-1) >>= \xs -> return (x:xs)

echo :: Int -> IO ()
echo n = gets n >>= putStr
```

Giesl: Ich sehe, dass sie die do-Notation nicht mögen und es lieber unleserlich machen, aber das ist in Ordnung. Wie legt man denn jetzt die Semantik dieser Funktion fest?
Ich: Wir bauen eine neue Funktion, die unsere Funktion als Argument nimmt und setzen die Semantik der ursprünglichen Funktion gleich dem kleinsten Fixpunkt der konstruierten Funktion.
Giesl: Warum kann ich denn davon ausgehen, dass es einen solchen Fixpunkt gibt?
Ich: Naja, wir können die Funktion in einer Programmiersprache beschreiben, deswegen ist sie berechenbar. Wenn sie berechenbar ist, dann ist sie stetig. Und wenn sie stetig ist, dann gibt es auch einen kleinsten Fixpunkt.
Giesl: Woher wissen wir, dass die Existens eines kleinsten Fixpunkts aus der Stetigkeit folgt?
Ich: Das sagt uns der Fixpunktsatz. Soll ich den beweisen?
Giesl: Ja, bitte.
Ich fing an mit Aussage und Beweis des Fixpunktsatzes.
Giesl: Sie gehen da ja jetzt davon aus, dass die Funktion auch monoton ist. Warum kann man das machen?
Ich: Weil aus der Stetigkeit die Monotonie folgt.
Giesl: Zeigen sie mir das doch mal.
Mit einigen Tips von Prof. Giesl habe ich dann irgendwie diesen Beweis hinbekommen.
Giesl: Gut, das war es schon. Warte sie bitte kurz draußen.