Prüfungsprotokoll- Theoretische Informatik

Prüfer: Indermark, Noll
Termin: 08.10.2004
Note: 1,3

Fächer: Compilerbau (Indermark SS 04)
Logikprogrammierung (Indermark WS 03/04)
Formular Models of Concurency (Noll WS 03/04)
Modelchecking (Thomas WS 03/04)

Zur Atmosphäre muß ich zunächst mal sagen, dass beide super nett und locker waren. Ich hingegen war doch ziemlich nervös. Herr Indermark wollte mir - glaube ich - mit einigen einfachen Einstiegsfragen helfen, die mich teilweise allerdings etwas verwirrt hatten. Naja, lest selbst:

1. Compilerbau (20 - 25 Minuten)

Indy: Fangen wir doch mit den Compilern an.(Super, wollte eigentlich mit Logik anfangen.) Da haben wir ja verschiedene Analysen kennengelernt. Zunächst einmal die lexikalische. Was übergibt die denn so an die syntaktische Analyse?

Moi: Eine Tokenfolge. (super, die erste Antwort direkt erstmal falsch)... äh, ich meine die Symbole. Symbol= Token, Attribut.

Indy: Und wieso macht man das? Wieso nicht direkt einfach die Attribute?

Moi: Weil es einfacher für den Parser ist, mit z.B. Bezeichnern zu arbeiten als mit allen möglichen Arten von Bezeichnern. Es ist jedenfalls gut, zunächst schon mal eine Unterteilung zu treffen.

Indy (ihm schien die Antwort nicht so ganz zu gefallen): Ja, na schön, es ist dann einfacher. Wie funktioniert denn die lexikalische Analyse?

Moi: Reguläre Ausdrücke für die Symbolklassen, lm-Zerlegung (höchstens eine), flm-Analyse,...

Indy: Was versteht man denn unter longest-match?

Moi: Hab versucht zu definieren, aber leider fällschlicherweise gesagt, dass man nur überprüfen muss, ob das Wort plus dem nächsten Zeichen in einer Symbolklasse liegt. Nachdem Indy mich darauf hinwies, habe ich im dann das Beispiel aus der Vorlesung gebracht, bei dem man auch den quadratischen Aufwand erkennt.

Indy: Und wie arbeitet dann der Automat?

Moi: Mit zwei Köpfen, Normal und Backtrack-Mode,...

Indy: Und wann erkennt der Automat, dass er den längsten Match gefunden hat?

Moi: Immer ein Zeichen weiter gehen, bis er in einem unproduktiven Zustand landet.

Indy: Das genügt soweit. Wann berechnet man denn nun die Attribute?

Moi: Normalerweise bei der semantischen Analyse, bei LAGs und SAGs schon bei der syntaktischen Analyse.

Indy: Okay, wie funktioniert den die Berechnung der Attribute bei den SAGs? (super, das Thema, was ich am wenigsten konnte)

Moi: Bottom-Up, da nur synthetische Attribute.

Indy: Wie sieht das denn beim abstrakten Syntaxbaum aus?

Moi: AST ist die Compilersicht, Funktionen zu jeder Regel...

Indy: Was unterscheidet denn den AST vom normalen Ableitungsbaum?

Moi: (Nach ein wenig Überlegen komm ich schließlich drauf.) Die Terminalsymbole fehlen.

Indy: Okay, wie funktioniert jetzt die Konstruktion auf dem Heap? Was wird da gespeichert?

Moi: Adressen der Kinder eines Knotens werden gespeichert, bei den Blättern nur die Werte, die von dem Parser kommen.

Indy: Sie meinen von dem Scanner.

Moi: Natürlich. (Oh Gott, wenn ich schon das verwechsle.)

Indy: Wie funktioniert denn jetzt konkret die Konstruktion? Wieso klappt das?

Moi: Habe versucht, die Adressen und ihre Funktion anzugeben, hat ihm aber nicht so gut gefallen. Ich stand da ziemlich auf'm Schlauch.

Indy: Also bei Anlegen eines neuen Knotens, was passiert da?

Moi: Ähhh... (Auf'm Schlauch). Es werden die Adressen der Kinder gespeichert.

Indy: Das sagten Sie bereits... Na, man muß ja noch die Adresse für den neuen Knoten vergeben und speichern. Themenwechsel. Bei der syntaktischen Analyse hatten wir ja unteranderem SLR(1) kennengelernt. Wieso reichte uns das nicht?

Moi: Weil man dabei die follow-Mengen betrachtet, also alle möglichen Rechtskontexte. In der konkreten Ableitung sind die aber gar nicht alle möglich. Also nimmt man LR(1).

Indy: Definieren Sie doch mal die LR(1)- Auskünfte.

Moi: Definition hingeschrieben (dafür lohnt sich das Auswendiglernen.)

Indy: Und wie werden die LR(1)-Mengen berechnet?

Moi: Ebenfalls hingeschrieben. Habe noch erklärt, wozu man das look-ahead-Symbol braucht, was $ bedeutet,etc.

Indy: Und wie sieht jetzt ein Shift-Reduce-Konflikt aus?

Moi: Beispiel hingeschrieben. Wichtig war Indy wohl, dass beim Shiften das nächste Symbol und nicht das la-Symbol beachten muß.

Indy: Wieso hat man dann überhaupt das la-Symbol in der Shift-Auskunft?

Moi: Für die weitere Berechnung. Man kommt ja später nach zu einer Reduceauskunft.

Indy: Gut. Was ist schließlich das Ergebnis der Syntaxanalyse?

Moi: Ableitungsbaum, bzw. l-Analyse oder r-Analyse.

Indy: Wann ist eine Grammatik zirkulär?

Moi: Wenn ein Attribut im Abhängigkeitsgraphen von sich selbst abhängt.

Indy: Was ist denn **der** Abhängigkeitsgraph?

Moi: Es gibt unendlich viele, weil es unendlich viele Ableitungsbäume gibt und dazu dann jeweils einen Abhängigkeitsgraph. Sobald aber einer zirkulär ist, ist die Grammatik zirkulär.

Indy: Nehmen wir doch mal A -> BC. Wie sieht dann der Abhängigkeitsgraph aus?

Moi: Hingemalt. Erklärt, wieso es keine Schleife geben kann. Erst beim Verkleben treten diese auf.

Indy: Wie geht man denn mit den unendlich vielen Graphen um? Wieso können wir trotzdem den Test durchführen?

Moi: Da sich die Ableitungsbäume aus den Regeln zusammen setzen. Davon gibt es nur endlich viele. Also betrachtet man diese als Bausteine. Man berechnet die Unterhalbabhängigkeiten zu den Nichtterminalen und sucht dann später noch nach geeigneten Deckeln.

Indy: Wo stecken diese Deckel?

Moi: In den Regeln.

Indy: Und wo genau?

Moi: Bei den Nichtterminalen der linken Regelseite.

Indy: Welche? Malen sie's dochmal auf.

Moi: Jetzt dämmerte mir, dass es die NT-Symbole der rechten waren. Indy stellte noch klar, dass der Deckel nicht immer so einfach aussieht, wie ich es dargestellt hatte. Dannach hat er noch nach Komplexität gefragt und ich hab auch noch zwischendrin die stark nicht-zirkulären erwähnt.

2. Logikprogrammierung (10 Minuten)

Indy: Kommen wir zur Logikprogrammierung. Schreiben Sie dochmal append auf!

Moi: append aufgeschrieben und kurz erläutert. Gesagt, dass es mit Differenzlisten besser geht.
Indy: Schön. Sie kennen ja die prozedurale Semantik. Wenden Sie diese mal bei dem Beispiel an!

Moi: Beispielanfrage zu dem Programm erstellt. Bei der Gelegenheit noch kurz die beiden Nichtdeterminismen genannt, Prologauswertungsstrategie und Backtracking erläutert.

Indy: Wie kann man denn CFG in Prolog definieren?

Moi: Mit Definite Claus Grammars. Ich wollte schon anfangen, die Differenzlisten-Struktur hinzuschreiben. Die hatte ich mal gelernt.

Indy: Stop. Erläutern Sie doch zunächst mal das Grundprinzip!

Moi: Hier hatte ich ziemliche Probleme. Indy hat mir ein wenig geholfen, so dass ich irgendwann eine Art Definition der DCGs mit append hingeschrieben hatte.

Indy: Und wie geht das jetzt mit Differenzlisten?

Moi: Ich setzte an, es aufzuschreiben, aber jetzt fiel mir nicht mehr die Definition ein. Indy unterbrach mich schließlich und Herr Noll übernahm die Prüfung.

3. FMC (10 Minuten)

Noll: Wir haben ja CCS kennengelernt. Wie sieht dafür die Semantik aus?

Moi: Es gibt Ableitungsregeln. Habe dann die Regeln für die Parallelоperation angeschrieben.

Noll: Wir haben verschiedene Äquivalenzen behandelt. Welche?

Moi: Starke Bismulation, schwache Bisimulation und Beobachtungskongruenz.

Noll: Definieren Sie die starke Bisimulation!

Moi: Definition hingeschrieben.

Noll: Wie sieht der Unterschied zur schwachen Bisimulation aus und wieso hat uns die nicht gereicht?

Moi: Schwache Bisimulation berücksichtigt Tau-Schritte nicht, ist aber keine Kongruenz. Unsere Kriterien für Äquivalenz waren aber: Berücksichtigung von Tree Äquivalenz, Trace Äquivalenz, Deadlock-Sensitivität und Kompositionalität, also Kongruenz.

Noll: Definieren Sie die Beobachtungskongruenz!

Moi: Definition und außerdem Charakterisierung hingeschrieben. Dann wollte er noch das Gegenbeispiel, welches zeigt, dass schwache Bisimulation kein Kongruenz ist.

4. Modelchecking (5 Minuten)

Noll: Kommen wir noch zum Modelchecking. Was ist das eigentlich?

Moi: Ein Verfahren, um Programme automatisch zu verifizieren.

Noll: Und was betrachtet man da genau?

Moi: Kripke-Strukuren und Formeln.

Noll: Aus welchen Logiken?

Moi: CTL, LTL und CTL*.

Noll: Wenn wir zwei in FMC äquivalente Prozesse haben, sagen wir stark bisimilare, kann man die dann in Modelchecking unterscheiden?

Moi: (Das war jetzt außerhalb des Vorlesungsstoffs. Herr Noll sagte später, dass er einfach nur sehen wollte, wie ich reagiere.) Nun ja, in LTL kann man das vermutlich schon.

Noll: Und kann man sie in CTL unterscheiden?

Moi: Habe hier ein bischen herumgedacht. Kurz gesagt, wie überhaupt Äquivalenz bei Modelchecking aussieht und schließlich geantwortet: Ja!

Noll: Nein!

Moi: (gedanklich und im Homer Simpson-Tonfall) NEIN!

Noll: CTL kann gerade bisimilare Prozesse nicht unterscheiden. Oder haben Sie ein Gegenbeispiel parat? Wie funktioniert denn das Modelchecking für CTL?

Moi: Habe gesagt, auf welche Operatoren man sich beschränken kann und erläutert, wie die Markierung funktioniert.

Als ich draussen warten musste, habe ich mich natürlich sehr über meine Fehler geärgert und schon mit einer wesentlich schlechteren Note gerechnet. Von der tatsächlichen Bewertung war ich sehr überrascht. Indy sagte, dass ich die Zusammenhänge sehr gut erläutert hätte. Bei den S-Attributgrammatiken hatten wohl so ziemlich alle Prüflinge Probleme.

Insgesamt kann man Indy sehr empfehlen, bei manchen Fragen ist es nicht ganz leicht, zu erkennen, worauf er hinaus will. Wenn man dann einfach anfängt, lenkt er einen aber schon in die richtige Richtung. Fangfragen stellt er aber keine.

Herrn Noll ist ebenfalls absolut empfehlenswert. Seine Fragen waren sehr präzisse und gut aufeinander aufbauend. Er war sehr an Zusammenhängen interessiert, womit er mich bei der Transferfrage zwar etwas überfordert hatte, aber er wollte wie gesagt, vor allem sehen, wie ich geantwortet habe.

Am meisten hat mir in der Vorbereitung geholfen, dass ich mit anderen zusammengelernt habe und wir uns dabei gegenseitig immer wieder geprüft haben. Wenn man lange genug den Zirkularitätstest erklären muß, kann man dass auch in der Prüfung. Deshalb möchte ich vor allem Matthias, Basti und Michi danken, die mich zwar manchmal gequält, aber mir damit sehr geholfen haben.