Gedächtnisprotokoll zur Diplomprüfung

Theoretische Informatik

Prüfer	:	1. Prof. Dr. Klaus Indermark (KI)
		2. Prof. Dr. Jürgen Giesl (JG)
Prüfungsgebiete	:	1. Programmschemata (Indermark, SS 2002)
		2. Compilerbau (Indermark, WS 2002/2003)
		3. Termersetzungssysteme (Giesl, SS 2002)
Datum, Uhrzeit	:	10. April 2003, 10:05 Uhr
Dauer	:	ca. 50 Minuten
Note	:	1.0

Bemerkungen : Die Prüfungsatmosphäre war sehr angenehm. Die Anfangsnervosität verschwand schon nach den ersten Fragen. Es gab keinen Beisitzer. Herr Giesl protokollierte, als Herr Indermark dran war, und umgekehrt. Überrascht hat mich vor allem, dass Herr Giesl in der Prüfung auch eine konkrete Berechnungsaufgabe gestellt hat. Ich hatte eher mit abstrakten Beispielen gerechnet. Er meinte, dass gerade dadurch man erkennen kann, ob der Prüfling die Sachverhalte wirklich verstanden oder nur auswendig gelernt hat. Es schadet also nicht, die Übungsaufgaben bei der Vorbereitung durchzugehen.

Hinweis : Der untenstehende Prüfungsverlauf ist nicht wörtlich zu verstehen (d.h. keine genauen Zitate), sondern er soll nur möglichst gut beschreiben, wie die Prüfung ablief.

Viel Spaß beim Lernen und viel Erfolg bei Euren Prüfungen !

Programmschemata

KI: Fangen wir mal mit Programmschemata an. Wie kommt man von Programmen zu Schemata ?

Ich: Man abstrahiert von der Struktur des Zustandraumes. Ein Programm wird in zwei Komponenten zerlegt : das Schema und die zugehörige Interpretation.

KI: Ist eine solche Zerlegung denn eindeutig ?

Ich: Nein, diese Zerlegung ist nicht eindeutig. Hier stellt sich natürlich die Frage, wie weit man abstrahiert. In der Vorlesung haben wir uns hauptsächlich mit einer „starken“ Abstraktion beschäftigt, bei der wir nur einstellige Operationssymbole und Prädikatensymbole zulassen. Daraus ergeben sich dann einfache Schemata. Aber am Ende der Vorlesung haben wir auch eine „schwächere“ Abstraktion kennengelernt, die nicht-monolithische Zustandsräume zulassen. Das sind die LPP-Schemata.

KI: Genau. Wir haben dann gesehen das viele Probleme bei dieser Art von Abstraktion aber nicht mehr entscheidbar sind, während das bei der starken Abstraktion der Fall ist. Wissen Sie wofür LPP steht ?

Ich: Luckham-Park-Paterson.

KI: Richtig. Die LPP-Schemata waren Gegenstand der Dissertation von Paterson. Wir bleiben hier aber doch bei den einfachen Schemata. Erklären Sie mal was eine Interpretation ist.

Ich: Eine Interpretation $\mathcal{A} = \langle A; \alpha \rangle$ besteht aus einer Trägermenge A und der Deu-

tung α der Funktionssymbole $f \in \Omega$ und der Prädikatensymbole $p \in \Pi$.

KI: Schreiben Sie doch mal auf wie $\alpha(f)$ aussieht.

Ich: *(hingeschrieben)* $\alpha(f) : A \to A$ für alle $f \in \Omega$, $\alpha(p) : A \to \{T, F\}$ für alle $p \in \Pi$.

KI: Es gibt da eine spezielle Interpretation ...

Ich: Das sind die freien Interpretationen $\mathcal{F} = \langle \Omega^*; \varphi \rangle$, wobei $\varphi(f)(w) = wf$ für alle $w \in \Omega^*$ ist. Dabei ist die Deutung der Prädikatensymbole beliebig.

KI: Wir haben uns ja mit Ianov-Schemata beschäftigt. Da haben wir eine Charakterisierung der Ianov-Schemata durch Schemasprachen. Wie sieht denn so eine Sprache aus ?

Ich: $L \subseteq (B\Omega)^* B$.

KI: Was ist denn das B ?

Ich: $B = \{0, 1\}^n$ ist die Menge der Bedingungsvektoren, wobei n die Anzahl der Prädikatensymbole ist.

KI: Was bedeutet z.B. eine 1 an der i-ten Stelle ?

Ich: Dass zu diesem Zeitpunkt das Prädikat p_i erfüllt ist.

KI: Und was sagt ein solches Wort aus ?

Ich: Ein solches Wort beschreibt die Berechnung eines Programms. Man macht zunächst Test-Schritte und anschließend einen Operationsschritt, und wieder Testschritte, usw. und endet dann mit einer Bedingungsvektor.

KI: Wenn wir zwei freie Schemata haben, d.h. Schemata, bei denen jeder Pfad ein Berechnungspfad ist, ist dann die Verkettung dieser beiden Schemata wieder frei ? (*Ich antwortete : Nein*) Warum ?

Ich: Jeder Pfad in S_1 ist noch ein Berechnungspfad, aber beim Verlassen (d.h. STOP-Befehl) von S_1 nur bestimmte Bedingungen (d.h. nicht alle) gelten. Nun sind in S_2 nur noch Berechnungspfade möglich, die mit diesen Bedingungen anfangen. Es könnte also Pfade in S_2 geben, die nicht von einer Berechnung durchlaufen werden.

KI: Und welche Operation auf Schemasprachen entspricht der Verkettung zweier Schemata ?

Ich: Das ist das bedingte Produkt : $L_1 \circ L_2 = \{w\beta v \mid w\beta \in L_1 \text{ und } \beta v \in L_2\}$

KI: Darüber hinaus gibt es noch die bedingte Vereinigung und den bedingten Stern. Welcher Schemaklasse entspricht denn die reguläre Teilklasse der Schemasprachen ? (*An dieser Stelle war ich etwas verwirrt wegen des Wortes „regulär“. Ich dachte, er meinte mit „regulär“ wirklich die regulären Sprachen, und wollte gerade antworten „die Ianov-Schemata, deren Schemasprachen durch DFA's definiert sind“ als KI, der meine Verwirrung wohl merkte, weiter sagte :*) Man kann ja die Operationen auf Schemasprachen als reguläre Operationen betrachten ...

Ich: Ach so, dann sind es die Schemasprachen der Dijkstra-Schemata. Dies ist die kleinste Teilklasse, die BfB für jedes Operationssymbol f enthält, und abgeschlossen ist unter dem bedingten Produkt, der bedingten Vereinigung und dem bedingten Stern.

(*Ich weiß nicht mehr genau, wonach KI an dieser Stelle gefragt hat. Jedenfalls ging es darum, dass die Ianov-Schemata mächtiger sind als die Dijkstra-Schemata*)

KI: Wenn man jetzt zu den Dijkstra-Schemata zusätzliches Hilfsmittel habe, nämlich

boolesche Variablen, kann man dann die Ianov-Schemata simulieren ? (*Ich antwortete : Ja*) Was ist dabei die Idee ?

Ich: Die Idee ist, jeder Marke eines Ianov-Schemas eine boolesche Variable zuzuordnen, und je nach dem, welche Marke gerade „aktiv“ ist, wird genau die entsprechende boolesche Variable auf 1 gesetzt.

KI: Die boolesche Variablen sind nicht ganz natürlich. Deshalb versucht man, eine andere Konstrukt zu finden, die induktiv aufgebaut ist, und dennoch zu den Ianov-Schemata äquivalent ist. Welche Konstrukt ist das ?

Ich: Die Repeat-Exit-Konstrukt. (*KI meinte, dass Exit alleine nicht ausreichend ist*) Für Exit haben wir noch einen Index n, also `exit n`.

KI: Was bedeutet das n ?

Ich: Das bedeutet, dass n Repeat-Schleifen verlassen werden müssen.

(*Hier kann ich mich auch nicht mehr erinnern, was genau gefragt wurde. Es ging um die Blocknormalform (Sprünge nur zu Vorgängerknoten) von Ianov-Schemata und wie man das erreicht (durch sukzessives Abwickeln von Schleifen).*)

KI: Kennen Sie die Sprache Java ? Da gibt es ja auch Sprachkonstrukte, die Sprünge ermöglichen. Wissen Sie welche Konstrukte das sind ?

Ich: (*Hmm ... , welche Konstrukt könnte das sein ? Ich überlegte und überlegte. Doch ich konnte mich nicht daran erinnern. KI schien es zu merken und sagte*)

KI: Na ja, lassen wir das mal sein und kommen gleich zu Compilerbau.

Compilerbau

KI: Steigen wir direkt in die Syntaxanalyse ein. Wir haben da die LR(0)-Grammatiken kennengelernt und festgestellt, dass diese nicht ausreichend sind. Deshalb haben wir dann die LR(1)-Grammatiken betrachtet. Was ist denn eine LR(1)-Auskunft ? Wie sieht sie aus ?

Ich: Wenn $S' \Rightarrow^* \alpha Aaw \Rightarrow \alpha\beta_1\beta_2 aw$, dann ist $[A \to \beta_1 \cdot \beta_2, a]$ eine LR(1)-Auskunft für $\alpha\beta_1$. Das a gibt an, dass man nach einer Reduktion zu A ein a in der Eingabe haben muss.

KI: Wie berechnet man die LR(1)-Auskünfte ?

Ich: Zunächst $\underline{\mathrm{LR(1)}}(\varepsilon) : [S' \to \cdot S, \$]$ ...

KI: Was ist denn \$?

Ich: Das Eingabeende. Weiter zu obiger Berechnung : Wenn $[A \to \gamma \cdot B\delta, x] \in \underline{\mathrm{LR(1)}}(\varepsilon)$, $B \to \beta$ in G und $y \in \underline{\mathrm{fi}}(\delta x)$, dann ist $[B \to \cdot\beta, y] \in \underline{\mathrm{LR(1)}}(\varepsilon)$.

KI: Warum $\underline{\mathrm{fi}}(\delta x)$?

Ich: Weil nach einer Reduktion zu B das erste Symbol von δ als Look-ahead vorkommen kann. Falls zusätzlich $\delta \Rightarrow^* \varepsilon$ gilt, dann kann auch x als Look-ahead vorkommen.

KI: Den Rest der Berechnungsregeln brauchen Sie jetzt nicht aufzuschreiben. Bei der Berechnung der LR(1)-Auskünfte liegt die Komplexität natürlich in der Menge $\underline{\mathrm{fi}}(\delta x)$. Wie haben wir dann die LR(1)-Analyse vereinfacht ?

Ich: Indem wir LR(1)-Mengen zusammenfassen, die den gleichen LR(0)-Kern besitzen. Sie bilden dann eine LALR(1)-Menge.

KI: Kommen wir nun zur semantischen Analyse. Da haben wir den Begriff der Zir-

kularitäten. Zeichnen Sie doch mal den Abhängigkeitsgraphen der Regel $A \rightarrow BCD$ auf. Jedes Nichtterminalsymbol hat ein inherites und ein synthetisches Attribut. Und zeichnen Sie alle möglichen Abhängigkeitskanten zu dem inheriten Attribut von C auf.

Ich: (*zeichne den Abhängigkeitsgraphen gemäß Vorlesung auf. Konvention :* $\bullet$ A $\circ$ *, wobei* $\bullet$ *inherit und* $\circ$ *synthetisch ist.*) Wir haben die semantischen Regeln (Gleichungen) so definiert, dass man keine Zirkularität innerhalb einer Regel hat. Das erreicht man dadurch, dass man die formalen Variablen in zwei disjunkte Mengen zerlegt : die Innenvariablen für die linke Seite einer Gleichung und die Außenvariablen für die rechte Seite. Zirkularitäten treten beim Verkleben der Regeln zu einem Ableitungsbaum auf.

KI: Um Zirkularitäten zu vermeiden, haben wir uns auf spezielle Attributgrammatiken beschränkt, z.B. die L-Attributgrammatiken. Wie sind sie definiert ?

Ich: In einer Regel gilt : falls ein inherites Attribut von einem synthetischen Attribut abhängt, dann muss dieses synthetische Attribut in dem Abhängigkeitsgraphen links von dem inheriten Attribut liegen.

KI: Wie kann man Zirkularitäten in einer Attributgrammatik erkennen ?

Ich: Die Zirkularität im Abhängigkeitsgraphen eines Ableitungsbaums bestimmt einen Teilgraphen folgender Form (*siehe Vorlesungsmitschrift*). Also berechnet man zunächst die Attributabhängigkeitsmengen (Stichwort: „unterhalb abhängig“) zu jedem Nichtterminalsymbol und überprüft, ob es eine Regel gibt, die mit Hilfe der Abhängigkeitsmengen einen solchen Teilgraphen „induzieren“ kann.

KI: Was muss man bei der Berechnung der Abhängigkeitsmengen beachten ?

Ich: Man muss darauf achten, dass Abhängigkeiten von verschiedenen Ableitungsbäumen getrennt behandelt werden.

KI: Genau. Sonst hat man starke Nichtzirkularität, was keine notwendige Bedingung ist. Das ist Knuth früher auch nicht erkannt. Also, die letzte Frage. Bei der Zwischencodeerzeugung haben wir Programmiersprachen mit Prozedurkonzept betrachtet, einmal ohne Prozedurparameter und einmal mit Prozedurparameter. Nehmen wir mal den Fall ohne Parameter. Für die Erzeugung des Aktivierungsblocks haben wir den `CALL`-Befehl. Wie sieht der aus ?

Ich: `CALL`(ca, dif, loc) wobei ca die Codeadresse, dif die Leveldifferenz und loc die Anzahl der lokalen Variablen ist (*dabei habe ich plötzlich vergessen, was die dritte Komponente ist. KI hat mir eine Hilfestellung gegeben*).

KI: Erklären Sie dif genauer. Warum braucht man es ? Wie wird es berechnet ?

Ich: Wir haben das „static scope“-Prinzip, d.h. beim Aufruf einer Prozedur ist die Deklarationsumgebung und nicht die Aufrufumgebung gültig. Deshalb muss man mit Hilfe der statischen Verweises auf den Aktivierungsblock der Deklarationsumgebung zugreifen. dif wird bei der Berechnung des statischen Verweises benötigt. dif = Aufruflevel − Deklarationslevel.

Termersetzungssysteme

JG: Was war unser Hauptanliegen in der Vorlesung ?

Ich: Das Wortproblem für ein Gleichungssystem zu lösen.

JG: Wie ist das Problem definiert ?

Ich: Man hat ein Gleichungssystem $\mathcal{E}$, zwei Terme s und t und möchte untersuchen,

ob $s \equiv_{\mathcal{E}} t$ gilt. D.h. ob jede Algebra A, die $\mathcal{E}$ erfüllt, auch die Gleichung $s \equiv t$ erfüllt.

JG: Das ist ein semantischer Begriff. Nun wollen wir das Wortproblem syntaktisch lösen. Wie geht das ?

Ich: Wir haben die Ersetzungsrelation $\rightarrow_{\mathcal{E}}$ definiert und dann deren transitiv-reflexiv-symmetrische Hülle $\leftrightarrow^*_{\mathcal{E}}$ als Beweisrelation verwendet. Es gilt dann $\leftrightarrow^*_{\mathcal{E}} = \equiv_{\mathcal{E}}$.

JG: Welcher Satz ist das ? (*Satz von Birkhoff*) Welche Richtung ist schwieriger zu zeigen ? (*die Vollständigkeit*) Was ist dabei die Idee ?

Ich: Man konstruiert eine Algebra $A = (\mathcal{T}(\Sigma, \mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}, \alpha)$, wobei $\alpha(f)([t_1]_{\leftrightarrow^*_{\mathcal{E}}}, \ldots, [t_n]_{\leftrightarrow^*_{\mathcal{E}}}) = [f(t_1, \ldots, t_n)]_{\leftrightarrow^*_{\mathcal{E}}}$ für alle $f \in \Sigma_n$, und zeigt, dass aus $A \models s \equiv t$ $s \leftrightarrow^*_{\mathcal{E}} t$ folgt. Anschließend zeigt man $A \models \mathcal{E}$, woraus dann $A \models s \equiv t$ wegen $s \equiv_{\mathcal{E}} t$ folgt.

JG: Also, wie kann man jetzt das Wortproblem lösen ?

Ich: Das Wortproblem ist im Allgemeinen nicht entscheidbar, aber semi-entscheidbar. Für die Grundidentitäten ist das Wortproblem entscheidbar, und zwar mit dem Kongruenzabschlussmethode. Für den allgemeinen Fall versuchen wir es mit TE-Sen.

JG: Was heißt semi-entscheidbar ?

Ich: Man kann ein Semi-Entscheidungsverfahren für das Wortproblem angeben. Ein Semi-Entscheidungsverfahren für das Wortproblem terminiert und ist korrekt, falls $s \equiv_{\mathcal{E}} t$ tatsächlich gilt, aber falls dies nicht der Fall ist, terminiert das Verfahren nicht.

JG: Wie löst man das Wortproblem mit der Kongruenzabschlussmethode ?

Ich: Die Idee ist, man ergänzt $\mathcal{E}$ um Gleichungen, die für die Transitivität, Reflexivität, Symmetrie und Monotonie notwendig sind (*Mit dieser Antwort war JG nicht ganz zufrieden. Ich habe dann die Definition von* $CC(\mathcal{E})$ *hingeschrieben*).

JG: Gut. $CC(\mathcal{E})$ könnte aber auch unendlich sein.

Ich: Deshalb haben wir noch den Begriff des Kongruenzabschlusses $CC(\mathcal{E})^S$ bezüglich einer Menge S eingeführt, wobei als S wir die (endliche) Menge der Teilterme in $\mathcal{E}$, s und t. $CC(\mathcal{E})^S$ ist dann endlich.

JG: Gegeben sei folgendes Gleichungssystem :

$$\begin{aligned} \text{plus}(x, \mathcal{O}) &\equiv x \\ \text{plus}(\text{succ}(x), y) &\equiv \text{succ}(\text{plus}(y, x)) \end{aligned}$$

Schreiben Sie doch mal eine Gleichung für die Assoziativität.

Ich: $\text{plus}(\text{plus}(x, y), z) \equiv \text{plus}(x, \text{plus}(y, z))$

JG: Folgt diese Gleichung aus dem obigen Gleichungssystem ?

Ich: Zunächst versuchen wir, ein konvergentes TES aus dem obigen Gleichungssystem zu bekommen. Dazu richten wir die Gleichungen, und zwar hier beide von links nach rechts, mit RPO.

JG: Ist die Multimengenrelation einer fundierten Relation auch fundiert ?

Ich: Ja.

JG: Vorsicht. Das gilt nicht immer.

(*Das hat mich verwirrt, denn aus der Vorlesung wissen wir, dass das immer gilt. Ich habe dann die Definition der Multimengenrelation einer Relation hingeschrieben und das hat gereicht. Nach der Prüfung meinte JG, dass das sein Fehler gewesen*

war. Bei Multimengenrelation gilt das immer, aber bei lexikographischer Kombination muss man darauf achten, dass die Länge des Tupels festbleibt.)

JG: Kann man hier LPO verwenden ? Wie ist denn die Beziehung zwischen LPO und RPO ?

Ich: Nein, hier nicht, da die Argumente der zweiten Regel vertauscht werden. Es gibt aber auch Beispiele, wo RPO scheitert, während LPO erfolgreich ist, nämlich wenn díe Argumente „größer“ werden.

JG: Nun ist die Terminierung dieses TES bewiesen. Was fehlt noch ?

Ich: Die Konfluenz. Diese zeigt man, indem man untersucht, ob alle kritischen Paare zusammenführbar sind. In dem Fall ist das TES lokal konfluent und wegen Terminierung auch konfluent.

JG: Gibt es Beispiele, wo aus der lokalen Konfluenz die Konfluenz nicht folgt ?

Ich: $a \leftarrow b \rightleftharpoons c \rightarrow d$ ist ein solches.

JG: Was ist ein kritisches Paar ?

Ich: (*Definition hingeschrieben*)

JG: Ist jetzt das obige TES konfluent ?

Ich: Ja, denn es gibt kein kritisches Paar. Also sind trivialerweise alle kritischen Paare zusammenführbar.

JG: Folgt also die obige Assoziativitätsgleichung ?

Ich: Nein, die beiden Terme sind bereits Normalformen und dennoch syntaktisch nicht gleich.

JG: Wir wissen aber, dass die Addition assoziativ ist. Das obige Gleichungssystem beschreibt ja die Addition, trotzdem folgt aus dem Gleichungssystem die Assoziativität nicht. Haben Sie eine Ahnung, woran das liegen kann ?

Ich: Das liegt daran, dass die „normale“ Addition in unserem Sinne eigentlich nur eine Algebra bzw. Interpretation ist, die das Gleichungssystem erfüllt. Es könnte aber auch andere Algebren geben, die das Gleichungssystem erfüllen und nicht assoziativ sind.

JG: Angenommen, das obige TES wäre nicht konfluent. Was würden Sie tun ?

Ich: Das TES um neue Regeln ergänzen bzw. vervollständigen.

JG: Was ist die Idee bei den verbesserten Vervollständigungsverfahren ?

Ich: Man gibt eine Menge von Transformationsregeln für Vervollständigung an. Dabei hat man zwei Vorteile. Zum Einen kann kann man bereits eingefügte Regeln verändern bzw. löschen, und zum Anderen beschreibt man damit eine Menge von Vervollständigungsprozeduren, die sich nur in der Strategie unterscheiden.

JG: Wie garantiert man die Korrekheit solcher Vervollständigungsprozeduren ?

Ich: Ein hinreichendes Kriterium ist, dass jede nicht-fehlschlagende Transformationsfolge fair ist.

JG: Was heißt fair ?

Ich: Dass jedes kritische Paar der persistenten Regeln auch einmal als Gleichung auftritt.

– E N D E –