Diplomprüfung theoretische Informatik bei Professor Indermark, Herbst 2004

Inhalte:

- Logikprogrammierung (Vorlesung bei Prof. Indermark im WS 01/02)
- Compilerbau (Vorlesung bei Prof. Indermark im SS 2004)
- Angewandte Automatentheorie (Skript von Prof. Thomas aus dem SS 2003)

Note: 1.0

Dauer: 45 Minuten

Compilerbau

KI: Man kann die Aufgaben eines Compilers ja in einzelne Phasen gliedern. In der Analysephase gibt es die lexikalische, die syntaktische und die semantische Analyse. Erzählen Sie mal etwas zur lexikalischen Analyse.

Ich: Die Aufgabe der lexikalischen Analyse ist die Zerlegung eines Programms, gegeben als ein Wort, in eine Symbolfolge.

KI: Richtig. Betrachten wir jetzt mal nur das Problem, ob ein Wort zu einer durch einen regulären Ausdruck spezifizierten Sprache gehört. Dafür gibt es die NFA- und die DFA-Methode. Erzählen Sie mal etwas über diese beiden Methoden.

Ich: Bei der DFA-Methode wird der reguläre Ausdruck zunächst per Thompson-Konstruktion in einen NFA umgewandelt. Dieser wird dann per Potenzmengenkonstruktion in einen DFA umgewandelt. Da die Potenzmengenkonstruktion allerdings exponentiellen Zeit- und Platzbedarf hat, ist dieses Vorgehen eher z.B. bei der Erstellung eines Compilers als bei nur einmaliger Verwendung des Automaten zu empfehlen. Dafür kann man die Zugehörigkeit des Eingabewortes zur Sprache des regulären Audrucks in $w+1$ Schritten entscheiden.

KI: Länge von $w+1$ Schritten meinen Sie, oder?

Ich: Ja klar, sorry. Bei der NFA-Methode wird wieder der NFA konstruiert, die Potenzmengenkonstruktion wird dann aber nur für den Lauf des jeweiligen Eingabewortes durch den Automaten durchgeführt. Man kann dann in $o(|\alpha| * |w|)$ Zeit und $o(|\alpha| + |w|)$ Platz das Wortproblem entscheiden.

KI: Gut. Kommen wir nun zur syntaktischen Analyse. Wir haben dort ja bei der Top-Down-Analyse LL(k)-Grammatiken betrachtet. Wie lautet die Definition von LL(k)-Grammatiken?

Ich: Ich zeichne am besten erstmal das Diagramm. *Diagramm mit den Linksableitungsschritten wie in der Vorlesung gezeichnet und erläutert.*

KI: Hierbei muss man ja die Rechtskontexte α mitberücksichtigen. Betrachten wir mal den Fall der LL(1)-Grammatiken. Gibt es ein Kriterium für die Zugehörigkeit zu LL(1), das nur von den Regeln der Grammatik abhängt?

Ich: Ja, und zwar, ob die Lookahead-Mengen zu den einzelnen Regeln mit jeweils gleichem Nichtterminal auf der linken Regelseite disjunkt sind.

KI: Wie sind die Lookahead-Mengen denn definiert?

Ich: $\underline{la}(A \to \beta) = \underline{fi}(\beta \underline{fo}(A))$.

KI: Was sind $\underline{fi}$ und $\underline{fo}$?

Ich: Abkürzungen für $\underline{first}_1$ und $\underline{follow}_1$.

KI: Wie sehen diese Mengen aus?

Ich: schreibe $\subseteq \Sigma_\varepsilon$ *neben die Definition der Lookahead-Mengen*

KI: Wann kann ε in die Lookahead-Mengen reinkommen?

Ich: Wenn $\beta \Rightarrow^* \varepsilon$ und $\varepsilon \in \underline{fo}(A)$.

KI: Was heißt $\varepsilon \in \underline{fo}(A)$ anschaulich?

Ich: $\varepsilon \in \underline{fi}$ der Wörter, die von einer Satzform ableitbar sind, die in einer Linksableitungsfolge rechts von A auftauchen können.

KI: Muss das denn immer passieren?

Ich: Naja, es kann auch gar nichts rechts von dem A stehen. Dann aber steht da ε, und ε ist insbesondere von ε ableitbar.

KI: Und wann kann a in die Lookahead-Mengen reinkommen?

Ich: Entweder wenn $\beta \Rightarrow^* a$ oder wenn $\beta \Rightarrow^* \varepsilon$ und $a \in \underline{fo}(A)$.

KI: Betrachten wir mal die Sprachklasse von LL(1). Die regulären Sprachen sind darin enthalten. Wie kann man das zeigen?

Ich: Zu einem regulären Ausdruck, der die Sprache definiert, gibt es insbesondere einen DFA. Dieser entspricht einer rechtslinearen Grammatik ...

KI: Ok, Sie kennen den Trick. :-)

Ich: Transition $q_i \xrightarrow{a} q_j$ *gemalt, entsprechende Regel* $Q_i \to aQ_j$ *hingeschrieben.* Wenn q_i jetzt noch ein Endzustand ist, dann gibt es noch die Regel $Q_i \to \varepsilon$.

KI: Wieso ist diese Grammatik LL(1)?

Ich: Da es für jedes q_i durch δ für jedes a nur einen Nachfolger gibt, sind die Lookahead-Mengen für festes Q_i auf der linken Regelseite disjunkt, sie enthalten jeweils das entsprechende a (bzw. ε).

KI: Auch wenn das jetzt nicht mehr in den Prüfungsstoff geht, wo braucht man diesen Zusammenhang noch? Denken Sie mal an die Grundvorlesung.

Ich: Ich habe das beim Nachsehen in meinen ATFS-Unterlagen in einem Beweis gesehen ...

KI: Denken Sie mal an CFL.

Ich: Ja klar, so kann man zeigen, dass jede reguläre Sprache kontextfrei ist.

KI: Wir hatten bei der Top-Down-Analyse auch den Top-Down-Analyseautomaten betrachtet. Was für Schritte kann dieser durchführen?

Ich: *pop*, falls das gleiche Terminal *a* auf der Eingabe und auf der Kellerspitze steht. Dabei wird das *a* von beiden entfernt. *reduce*, falls ein Nichtterminal *A* auf der Kellerspitze steht, dabei wird dieses durch die rechte Regelseite β einer Regel $A \to \beta$ ersetzt.

KI: Wird denn hierbei ein Symbol von der Eingabe entfernt?

Ich: Nein, die Eingabe wird beim Reduktionsschritt nicht verändert.

KI: Was ist hier die Besonderheit bei $\mathcal{G} \in$ LL(1)?

Ich: Hier ist die Entscheidung, nach welcher Regel abgeleitet werden soll, anhand der Lookahead-Mengen festgelegt, so dass der Automat deterministisch arbeitet.

KI: Dieser Automat lässt sich in abgewandelter Form nochmal an anderer Stelle einsetzen. Wissen Sie, was ich meine?

Ich: Ja, die semantischen Analyse von L-Attributgrammatiken.

KI: Richtig. Was für Schritte kann der Automat da auf dem Keller durchführen?

Ich: Zunächst zu den Kellersymbolen: $\underline{\text{LR(0)}}_\pi$ ist die Menge der LR(0)-Auskünfte zu einer Regel π von $\mathcal{G}$. *(Hier hatte ich zunächst versehentlich etwas von LR(0)-Informationen erzählt, obwohl ich die Auskünfte meinte, aber Herr Indermark hat mich schnell wieder auf den richtigen Weg gebracht.)* Hierbei werden die LR(0)-Auskünfte betrachtet, die der Regel entsprechen, aber auf der rechten Regelseite einen Punkt enthalten. Hier hat der Punkt aber eine andere Bedeutung als bei der LR-Analyse, er hat nichts etwa mit Henkeln oder so zu tun, sondern markiert die aktuelle Position bei der Baumreise. Dabei wird jeder Knoten genau zweimal besucht. Der einfachste Schritt ist der Match-Schritt, bei dem der Punkt, wenn er vor einem Terminal *a* steht, beim Lesen dieses Terminals über das *a* gezogen wird.

KI: Dieser Schritt ist genau wie bei dem Automaten vorher.

Ich: Dann gibt es noch den Expand-Schritt, bei dem, gegeben eine Kellerspitze $\underline{|[A \to \alpha \cdot B\gamma]|val|}$ mit einer Valuation *val* für die synthetischen Attribute, die der Regel entsprechen. Jetzt wird analog zum ε-Abschluss bei LR(0) $[B \to \cdot\beta]$ mit einer Valuation $\underline{val'}$ auf den Keller gelegt.

KI: Das hat damit aber nicht viel zu tun ...

Ich: Okay, im Gegensatz zum ε-Abschluss wird hier lediglich *eine* Regel expandiert, die durch die *la*-Mengen bestimmt ist, letztlich ging es mir auch nur um den technischen Aspekt des Punktes vor dem Nichtterminal und dass dann expandiert wird, die Hintergründe sind natürlich anders.

KI: Gut, technisch ist der Vorgang ähnlich.

Ich: *val* ist dabei die Valuation für die synthetischen Attribute in $A \to \alpha B\gamma$, und in $\underline{val'}$ werden den inheriten Attributen dann die Werte zugewiesen.

KI: Es gibt dabei aber eine Sache, auf die man achten muss. Was meine ich da?

Ich: Den Indexshift beim Übergang zur neuen Kellerspitze.

KI: Gut, bleiben wir bei der semantischen Analyse. Wann ist eine Attributgrammatik zirkulär?

Ich: Es gibt einen Ableitungsbaum t, dessen Abhängigkeitsgraph $\underline{DG_t}$ einen Zyklus aufweist.

KI: Dazu hatten wir in der Vorlesung einen Zirkularitätstest gesehen. Können Sie beschreiben, wie dieser funktioniert?

Ich: Dazu betrachtet man induktiv die unterhalb-Abhängigkeiten eines synthetischen Attributs von einem inheriten Attribut an einem Nichtterminal A: $\bullet \overset{A}{\smile} \circ$. Diese liegen vor, wenn in einem Ableitungsbaum mit A als Wurzel in der transitiven Hülle der Kantenrelation $(\bullet, \circ)$ enthalten ist. Dabei konstruiert man induktiv die Abhängigkeiten für die zwar unendlich vielen, aber endlich erzeugten möglichen Ableitungsbäume. Man fängt bei Regeln der Form $A \to w$ an und arbeitet sich dann induktiv hoch.

KI: Welcher Art sind denn diese Abhängigkeitsmengen?

Ich: Es handelt sich um Mengen von Mengen, man muss die einzelnen Ableitungsbäume getrennt betrachten.

KI: Was passiert denn, wenn man das nicht tut?

Ich: Dann erkauft man sich die Effizienzsteigerung zur Polynomzeit gegenüber dem exponentiellen Zeitaufwand durch ein zu hartes Kriterium für die Zirkularität, so dass nicht mehr genau die zirkulären Grammatiken als solche erkannt werden.

KI: Gut, eine letzte Frage noch zum Compilerbau. Wir hatten bei der Codeerzeugung eine Sprache mit rekursiven Prozeduren betrachtet. Bei der Erzeugung eines neuen Aktivierungsblocks mit dem CALL-Befehl, welche Aktivierungsblöcke auf dem Prozedurkeller sind da sichtbar? Definition: Ein Aktivierungsblock ist auf dem aktuellen Prozedurkeller *sichtbar*, wenn man auf ihn mit LOAD oder STORE zugreifen kann.

Ich: Das sind diejenigen, die man über den statischen Verweis oder durch wiederholtes Entlanghangeln am statischen Verweis erreichen kann.[1]

Logikprogrammierung

KI: Kommen wir jetzt zur Logikprogrammierung. Kennen Sie Datalog?

Ich: Ja, natürlich.

KI: Was ist da der wesentliche Unterschied zu Prolog?

Ich: Es gibt keine Funktionssymbole außer den Konstanten.

KI: Richtig. Kann man denn unendliche Berechnungen im SLD-Baum erkennen?

Ich: Ja, das geht, und zwar kann man beim Versuch, ein Literal zu beweisen, bei Wiederholung des Literals auf dem Berechnungspfad erkennen, dass dieser Pfad unendlich wird. Das geht, weil *Pred* und *Const* endlich sind. Ohne Konstruktoren kann man nur endlich viele Terme erzeugen.

[1] Der aktuelle natürlich auch.

KI: So habe ich es in der Vorlesung gesagt, aber die Zusammenhänge sind etwas komplizierter. Kommen wir zur Fixpunktsemantik. Wie ist diese definiert?

Ich: hingeschrieben

KI: Wie ist $\underline{trans}_{\mathcal{P}}$ definiert?[2]

Ich: Die $\underline{trans}_{\mathcal{P}}$-Funktion dient quasi dazu die logische Implikation darzustellen. *Definition hingeschrieben.*

KI: Was ist $\underline{Prim}_{\tau}$ dabei?[3]

Ich: Die Menge der Primformeln über τ, also der Formeln, die nur aus einem Prädikatsssymbol mit Termen als Argumenten besteht.

KI: Können dabei alle Terme vorkommen?

Ich: Nein, nur Terme ohne Variablen. Letztendlich handelt es sich also um Grundprimformeln.

KI: Wieso existiert denn überhaupt ein kleinster Fixpunkt?

Ich: Das liegt daran, dass $\langle \underline{Pot}(\underline{Prim}_{\tau}), \subseteq \rangle$ ein vollständiger Verband ist.

KI: Und was folgt daraus?

Ich: Insbesondere, dass für $M_0 \subseteq M_1 \subseteq \ldots$ eine kleinste obere Schranke existiert: $\bigcup_{n=0}^{\infty} M_n$.

KI: Das alleine reicht aber nicht.

Ich: Dazu kommt noch, dass $\underline{trans}_{\mathcal{P}}$ monoton und stetig ist, also *Definitionen mündlich genannt*.

KI: Was heißt denn die Stetigkeit von $\underline{trans}_{\mathcal{P}}$ anschaulich?

Ich: Da wäre mir Analysis jetzt aber lieber ...

KI: Es geht aber auch hier, versuchen Sie es mal. Gehen Sie mal von der Definition aus.

Ich: Naja, ob man zuerst vereinigt und dann $\underline{trans}_{\mathcal{P}}$ berechnet oder erst $\underline{trans}_{\mathcal{P}}$ berechnet und dann vereinigt, das kommt aufs gleiche raus. In irgendeiner Menge sind die B_i'[4] schließlich alle enthalten.

KI: Wie ist der Fixpunkt definiert?

Ich: $\underline{fix}(\underline{trans}_{\mathcal{P}}) = \bigcup_{n=0}^{\infty} \underline{trans}_{\mathcal{P}}^{n}(\emptyset)$

KI: Vorsicht, das ist aber nicht die Definition!

Ich: Ja klar, das ist eher ein Satz zur Berechnung des kleinsten Fixpunktes, die Definition ist M mit $\underline{trans}_{\mathcal{P}}(M) = M$, also Anwenden von $\underline{trans}_{\mathcal{P}}$ ändert nichts mehr an der Menge.

---

[2] Hierbei ist $\mathcal{P}$ ein Logikprogramm. Im WS 2003/2004 wurde stattdessen üblicherweise die Bezeichnung π gewählt.

[3] τ war im WS 2001/2002 in Logikprogrammierung die übliche Bezeichnung für die betrachtete Signatur. Im WS 2003/2004 war hingegen von Σ die Rede.

[4] meine Grundprimformeln aus M, aus denen per $\underline{trans}_{\mathcal{P}}$ das A_i' (kommt aus $\underline{trans}_{\mathcal{P}}(M)$ dazu) gewonnen wird

KI: Exkurs: Wissen Sie, wie der Unterschied zwischen einem kleinsten und einem minimalen Element einer Halbordnung definiert ist?
Wie es sich herausstellte, ist ein minimales Element einer Halbordnung eines, zu dem keines existiert, das darunter läge. Davon kann es auch durchaus mehrere geben. Ein kleinstes Element einer Halbordnung ist eines, das mit allen anderen in Relation steht. Aufgrund der Eigenschaften der Halbordnung ist ein kleinstes Element eindeutig bestimmt. (Solche Fragen sind wohl ein gutes Zeichen für den Prüfling, denn sie werden vermutlich eher gestellt, wenn man eine ziemlich gute Note zu erwarten hat ...)

Angewandte Automatentheorie

KI: Es wurden unter anderem Pushdown-Systeme vorgestellt, können Sie dazu etwas sagen?

Ich: Es handelt sich um Pushdownautomaten ohne Eingabe, Endzustandsmenge und hervorgehobenes Kellersymbol Z_0.

KI: Können Sie etwas zum Erreichbarkeitsproblem erzählen?

Ich: mündlich $pre^*(C)$ und $post^*(C)$ erläutert, erwähnt, dass es für reguläres C sowohl rückwärts als auch vorwärts lösbar ist

KI: Wer hat dies bewiesen?

Ich: Uff, der Name des Wissenschaftlers fällt mir gerade nicht ein.

KI: Büchi. Wie geht der Beweis?

Ich: Man betrachtet P-Automaten, die die Konfigurationen erkennen, also NFAs mit einer Menge P von Startzuständen, die keine eingehenden Kanten haben dürfen. Diese können durch Hinzufügen von Transitionen zu Automaten erweitert werden, die $pre^*(C)$ bzw. $post^*(C)$ erkennen.

KI: Ich weiß nicht, bei wem Sie die Grundvorlesung gehört haben, aber wir hatten da etwas analoges für CFGs.

Ich: Bei Ihnen. $pre^*(L)$ ... Da habe ich aber nur dunkle Erinnerungen dran.

KI: Und wie ist es mit 2 Stacks?

Ich: Unentscheidbar: Das Halteproblem für Turing-Maschinen ist auf das Erreichbarkeitsproblem für 2-PDS reduzierbar: Man nimmt das Band der Turingmaschine und ... *mache eine Handbewegung, als würde ich das Band der Turingmaschine in der Mitte überbrechen und dann die beiden Stücke in je einen Stack stecken*

KI: Okay, reicht mir. Und wie ist es bei 2-Zählern?

Ich: Hier hat das Alphabet nur 2 Symbole, wobei eines nur als unterstes auftritt, aber trotzdem ist eine Reduktion 2-PDS $\rightarrow$ 4-Zähler $\rightarrow$ 2-Zähler möglich: Zu 2-PDS $\rightarrow$ 4-Zähler: Das Alphabet Γ ist per Definition endlich, damit kann es auch aufgefasst werden als $\{1,\ldots,k\}$ (k ist die Mächtigkeit von Γ). Dann kann der Kellerinhalt als $(k+1)$-adische Zahl aufgefasst werden. Eine pop-Operation entspricht dann eine Division durch 10, ...

KI: Ok, da war doch noch was mit Primzahlen ...

Ich: Ja, bei der Reduktion von 4-Zählern zu 2-Zählern: Jede natürliche Zahl hat die Eigenschaft, dass ...

KI: Ok! Kommen wir nun zu Baumautomaten: Geben Sie mal einen Automaten für boolesche Ausdrücke ohne Variablen an, also boolesche Grundausdrücke.

Ich: Mit oder ohne Klammern? D.h., generalisierte oder normale boolesche Ausdrücke?

KI: Was würde man für einen Automaten für generalisierte Ausdrücke brauchen?

Ich: Einen XML-Automaten.

KI: Welche Problematik stellt sich beim Verarbeiten von XML?

Ich: Wir haben kein Rangalphabet mehr, sondern stattdessen einen unbeschränkten Branchingfaktor, also Verzweigungsgrad.

KI: Geben Sie mal einen Automaten für Ausdrücke mit Klammern an.

Ich: Automat aus dem Skript hingeschrieben, erläutert

KI: Wie sind δ und δ^* definiert?

Ich: δ ist von der Form $\bigcup_{i=0}^{m} Q^i \times \Sigma \to Q$. *Habe zunächst* Σ *und* Q *verwechselt; erläutert, dass* m *der maximale vorkommende Rang ist und dass man per Konvention für* $i = 0$ *nur* $\Sigma \to Q$ *betrachtet.*
Zu δ^*: $\delta^*(a) = \delta(a)$ *und* $\delta^*(f(t_1,\ldots,t_n)) = \delta(\delta^*(t_1),\ldots,\delta^*(t_n), f)$ *hingeschrieben und erläutert*

KI: Wenn Sie mal für eine Minute auf den Flur gehen würden ...

Wie üblich gilt auch hier: Dieses Protokoll gibt im Wesentlichen nur die behandelten Themen wieder, im Detail wird es sicherlich einige Auslassungen und Abweichungen vom tatsächlichen Prüfungsverlauf geben. Mir gegenüber saß der Beisitzer, schräg rechts von mir saß Prof. Indermark. Im Hintergrund war eine Wanduhr zu sehen, auf die gelegentliche Blicke aus dem Augenwinkel möglich waren. Während der Prüfung herrschte eine sehr angenehme Atmosphäre, in der auch kleinere Fehler nicht sonderlich ins Gewicht fielen. Es hat sich bewährt, unmittelbar vor einer Prüfung einen halben Liter Wasser zu trinken, um sich besser konzentrieren zu können. Zum Lernen habe ich jeweils parallel die (ggf. studentischen) Skripte zu den Vorlesungen und meine eigenen Aufzeichnungen benutzt (dabei war AAT aus dem Sommer 2004). Das berühmte "Drachenbuch" zum Compilerbau von Aho, Sethi und Ullman fand ich zumindest in der deutschen Übersetzung wegen der von der zu prüfenden Vorlesung abweichenden Terminologie nicht sehr nützlich, die Vorlesung nebst den Videos aus dem SS 2004 (sehr hilfreich!) hat zur Prüfungsvorbereitung aber auch ausgereicht. Nach der Logikprogrammierungs-Mitschrift ließ sich sehr gut lernen, auch wenn es schon etwas länger her war, dass ich die Vorlesung gehört hatte.