Gedächtnisprotokoll Diplomprüfung Theoretische Informatik

Prüfungsinhalt: Angewandte Automatentheorie (SS 2003)
Automata and Reactive Systems (WS 2002/03)
Compilerbau (WS 2002/03)
Prüfer: Thomas, Indermark (Compilerbau)

Datum: 9.12.2003
Prüfling: Lutz Ißler

Welche Reihenfolge wünschen Sie sich denn?
Angewandte Automatentheorie, Automata and Reactive Systems, und dann Compilerbau.

Angewandte Automatentheorie

Wir haben verschiedene Modelle von Baumautomaten kennengelernt. Kennen Sie eine Aussage über endliche Bäume, welche sich vom Fall der Wörter unterscheidet?
Nein.

Nun, lassen Sie mich das anders formulieren. Wenn Sie ein endliches Wort haben, und dieses herumdrehen, ist das noch immer regulär. Ist das bei Bäumen auch so?
(Worauf will er hinaus? Ich bin etwas verwirrt.)

Na, ich meine, wenn Sie einen Baum aus der anderen Richtung betrachten, gelten dann noch immer alle Aussagen?
(Ach so!) Es Nichtdeterministische und deterministische, Bottom-Up- und Top-Down-Baumautomaten. Deterministische Top-Down-Baumautomaten sind schwächer als die anderen.

Was bedeutet schwächer?
Sie erkennen nicht die gleiche Sprachklasse.

...und das bedeutet, man kann eine Sprache finden, die nicht von einem deterministischen Top-Down-Baumautomaten erkannt wird, aber von einem der anderen. Welche fällt Ihnen denn da ein?
(Ich stehe schon wieder auf dem Schlauch. Wo ist meine Vorbereitung hin? Prof. Thomas hilft mir schrittweise über die Potenzmengenkonstruktion bei Baumautomaten und dem Tipp „vollständige Bäume“ weiter.) Die Sprache der vollständigen Bäume. Denn dort muss bei hinreichend großem Baum (dh. hinreichend langem Pfad) ein Zustand doppelt vorkommen, so dass man dort den Pfad auch „verlängern“ kann.

Wir haben neben den Baumautomaten noch die Pushdown-Systeme und die Pushdown-Graphen gehabt. Wie sieht ein Knoten eines Pushdown-Graphen aus?
Konfiguration eines Pushdown-Systems, also Zustand plus Kellerinhalt.

Und wann sind zwei Zustände durch eine Transition verbunden?
Wenn es eine entsprechende Regel gibt.

Sagen wir, ich habe zwei Konfigurationen, und ich möchte wissen, ob ich von der einen zur anderen komme. Geht das?
Das ist das Erreichbarkeitsproblem für Pushdown-Systeme und das ist entscheidbar.

Sagen wir, wir haben eine Konfiguration, geschrieben als Menge. Welche Notation haben Sie denn da für die Konfigurationen davor kennengelernt? Schreiben Sie das doch bitte mal auf!
Die Menge der Vorgängerkonfigurationen einer Konfigurationsmenge L wird mit $\text{pre}^*(L)$ bezeichnet.

Und können Sie das $\text{pre}^*(L)$ berechnen?
Ja, mit dem (1.) Saturierungsalgorithmus.

Und wie funktioniert der? Beschreiben Sie doch mal grob, wovon Sie ausgehen und was da passiert!
(Ich beschreibe den 1. Saturierungsalgorithmus.)

Und ist das effizient, also in Polynomzeit zu erledigen?
Ja.

Erläutern Sie uns doch bitte mal, wie Sie auf die Polynomzeit kommen!
(Sowas, ich bin im Stress und kann nicht vernünftig nachdenken. Prof. Thomas hilft mir weiter. Endliche Menge von Regeln, man konstruiert einen Automaten, wie viele Zustände kann der Automat haben, wie viele Transitionen, usw. Irgendwann habe ich es.) Maximal $O(kn^2)$ Schritte.

Wir haben auch Petrinetze behandelt. Dort haben Sie Stellen und Transitionen, und Marken auf den Stellen, wenn Sie sich erinnern. Was ist denn eine Markierung?
Eine Zuordnung von Marken zu Stellen.

Können Sie mir ein Entscheidungsproblem zu Petrinetzen nennen?
Zum Beispiel das Beschränktheitsproblem.

Und das ist...?
...die Entscheidung, ob ein Petrinetz beschränkt ist, also ob man eine obere Schranke für die Markenzahl angeben kann.

Und wie entscheiden Sie das Beschränktheitsproblem?
Mittels Konstruktion des Karp-Miller-Baums und Nachsehen, ob Unendlichzeichen auftreten.

Und wann treten Unendlichzeichen auf?
Wenn man bei der Konstruktion eine Markierung erreicht, die größer ist als eine Markierung vorher auf dem Pfad.

Und das bewirkt?
...dass der Erreichbarkeitsbaum endlich wird und die Konstruktion terminiert.

Ist das Beschränktheitsproblem denn auch in Polynomzeit entscheidbar?
Nein. (Meine erste Antwort war im Stress nicht nachgedacht und falsch: Ja. Prof. Thomas hat mich korrigiert und meinte, er würde mir jetzt nicht die Zeit geben, das zu beweisen.)

Ein beschränktes Petrinetz kann man ja als endlichen Automaten auffassen. Was sind denn die Zustände des Automaten?
Die Markierungen.

Und warum sollte man ein Petrinetz statt eines endlichen Automaten verwenden wollen?
(Ja, warum eigentlich? Schnell überlegt:) Weil Petrinetze andere Sprache erkennen können als endliche Automaten.

Ja, aber nur unbeschränkte Petrinetze. Diese jedoch sind nicht mit endlichen Automaten äquivalent. Die meisten in der Praxis auftretenden sind beschränkt. Fällt Ihnen noch was ein?
Möglicherweise aus pragmatischen Gründen: Ein Petrinetz lässt sich intuitiver konstruieren.

Ja, durchaus. Oder prägnanter formuliert: Eine Markierung (2,3,5) liest sich einfach besser als ein q_{4357}.

Automata and Reactive Systems

Kommen wir zu Automaten auf unendlichen Wörtern. Da gibt es zum Beispiel den Büchi-Automaten. Konstruieren Sie doch mal den Automaten über dem Alphabet $\Sigma = \{a, b, c\}$*, der die Sprache erkennt, wo ab einem bestimmten Punkt kein c mehr vorkommt, also jedes Wort nur endlich viele c enthält!*
(Ich entscheide mich für einen nichtdeterministischen Büchi-Automaten und konstruiere diesen.)

Sie haben einen nichtdeterministischen Automaten gewählt. Wie würden Sie den denn deterministisch machen? Nicht Büchi, sondern irgendwas anderes. Überlegen Sie sich eine Akzeptierbedingung!
Zum Beispiel mittels co-Büchi-Akzeptanz.

Oh, ja, richtig. Aber nehmen Sie eine andere Variante!
Dann die Muller-Akzeptanz.

Wie sieht denn da die Akzeptanzkomponente formal aus?
Das ist eine Teilmenge der Potenzmenge der Zustände des Büchi-Automaten.

Und wie würden Sie den deterministischen Muller-Automaten zur genannten Sprache nun konstruieren?
(Ich konstruiere ihn.)

Sie haben eben co-Büchi-Akzeptanz vorgeschlagen. Das ist richtig. Warum denn nicht deterministische Büchi-Akzeptanz? Wie würden Sie das beweisen?
Über die möglichen Läufe.

Ja, in diesem Fall. Und allgemein? Welche Bedingung können Sie prüfen, wenn Sie feststellen wollen, ob Sie die deterministische Büchi-Akzeptanz hinbekommen?
Die Akzeptanzkomponente des Muller-Automaten muss unter Superloops abgeschlossen sein, was sie hier nicht ist (das sieht man deutlich an der Zeichnung).

Und von wem ist der Satz, der diese Eigenschaft beschreibt?
(Das weiss ich nicht mehr, und das sage ich auch. *Landweber.*

Wir haben unendliche Spiele betrachtet. Was bedeutet es, ein Spiel zu lösen?
Gewinnbereiche müssen gefunden und Gewinnstrategien angegeben werden.

Sie kennen die Logik S2S. Worüber macht diese Logik Aussagen?
Über unendliche Bäume. *(Kritischer Blick.)* – Unendliche Bäume mit zwei Nachfolgern, also zweifach verzweigt. *(Noch kritischerer Blick.)* – Über binäre Bäume.

Genauer: Über die Struktur des binären Baumes. Erst wenn Sie einzelne Bäume betrachten, stimmt Ihre Aussage dann auch. Kennen Sie einen Satz über Bäume, bei dessen Beweis Spiele eine Rolle spielen?
Den Beweis des Komplementabschlusses von Paritäts-Baumautomaten. Dort überträgt man die Problemstellung auf ein Spiel zwischen Automaton und Pathfinder, und man braucht eine Gewinnstrategie für Pathfinder an der „Wurzel“.

Und was braucht man dazu?
(Ich zögere, und er antwortet:) *Die Determiniertheit von Spielen. Und was ist das?* Man kann für jeden Knoten eine Gewinnstrategie angeben.

Nicht angeben – es genügt, dass man weiss, dass es eine gibt. Und was ist das für eine Gewinnstrategie bei Parity-Spielen? Und bei Büchi-Spielen?

Eine positionale bei Parity-Spielen und eine Automatenstrategie bei Büchi-Spielen.

Compilerbau

Wir haben Formalismen zur Syntaxanalyse kennengelernt. Welche sind das?
Die kontextfreien Grammatiken.

Und reichen diese aus zur Beschreibung der Eigenschaften von Programmiersprachen?
Nein, zum Beispiel Deklariertheit von Bezeichnern ist nicht ausdrückbar.

Das gehört nicht unbedingt zur Vorlesung, aber was glauben Sie, wie man mithilfe endlicher Wörter beweist, dass die Deklariertheit von Bezeichnern nicht kontextfrei ist?
(Ich bin ratlos. Dabei ist es ganz einfach, und nach ein paar Tipps ist es klar – wobei Prof. Indermark die Antwort in dem Moment selbst gibt, als es mir einfällt:) Deklariertheit bedeutet mindestens zweifaches Vorkommen, das ist die Sprache der Wiederholungen. Diese ist nicht regulär und damit auch nicht kontextfrei.

Welchen Formalismus kennen Sie, um semantische Eigenschaften zu beschreiben?
Attributgrammatiken.

Dort gibt es inherite und synthetische Attribute. Zeichnen Sie mir doch mal den Graphen für die Regel $A \longrightarrow BC$ auf, je ein inherites und ein synthetisches Attribut!
(Ich zeichne das auf.)

Zeichnen Sie mir mal eine Abhängigkeit ein, die bei einer L-attributierten Grammatik nicht vorkommen darf.
(Ich erläutere L-attributierte Grammatiken und zeichne die verbotene Abhängigkeit ein. Es gibt zwei, und deshalb kommt gleich die nächste Frage: *Gibt es denn noch weitere verbotene Abhängigkeiten?* Ich zeichne auch die zweite noch ein.)

Und wo ist der Vorteil, wenn man eine L-attributierte Grammatik hat?
Baumreise mit zwei Knotenbesuchen schafft die komplette Attributauswertung.

Wir haben Attributgrammatiken ja auch zur Codeerzeugung genutzt. Zum Beispiel für Stackcode für arithmetische Ausdrücke, was würden Sie da für Attribute verwenden?
Nur synthetische, da Bottom-Up-Auswertung reicht.

Und wie sähe ihre Attributgleichung zum Beispiel für die Regel $E \longrightarrow E + E$ aus, wenn c das Attribut wäre, das denn Code beinhaltet?
$c.0 = c.1; c.3;$ ADD.

Können Sie mir bei der Codeerzeugung eine semantische Eigenschaft nennen, wo Sie nicht mehr nur mit synthetischen Attributen auskommen?
Die Deklariertheit von Bezeichnern. (Jedenfalls fällt mir gerade nur das ein.)

Na ja, das ist ja nicht bei der Codeerzeugung. Zum Beispiel beim Aufruf einer Prozedur, da gibt es den CALL-Befehl. Welche Parameter braucht der?
Codeadresse, Differenz zwischen Aufruf- und Deklarationslevel, Anzahl der lokalen Variablen.

Und wie berechnen Sie diese Differenz?
Aus Aufruflevel und Deklarationslevel?

Ja, aber wo speichern Sie das Deklarationslevel?
(Nach einem kurzen verwirrten Ausflug zur base-Funktion fällt es mir ein:) In der Symboltabelle.

Richtig, und dafür brauchen Sie dann die inheriten Attribute. Was sind denn Zirkularitäten?
Wenn ein Attribut von sich selbst abhängt.

Und wann passiert das?
Innerhalb von Ableitungsbäumen.

Und kann man in Polynomzeit testen, ob Zirkularitäten auftreten?
Nein.

Aber Knuth hat doch damals in seiner ersten Arbeit zum Thema einen Polynomzeitalgorithmus für den Test veröffentlicht?
Ja, aber er hat vergessen, dass man beim Testen zwischen verschiedenen Ableitungsbäumen differenzieren muss.

Ja, das gibt das starke Zirkularität. Und wie funktioniert der Zirkularitätstest nun?
(Ich stottere etwas herum, spreche von Ableitungsbäumen, die man betrachten muss – wovon es unendlich viele gibt, also sind es dann wohl doch eher die Regeln, die man betrachten muss, denn davon gibt es nur endlich viele. Prof. Indermark merkt mein Zögern und erklärt mir kurz, dass man in der Tat die Regeln betrachten muss.)

Fazit:

Meine ersten Fragen waren direkt völlig daneben, und ich wusste nicht so recht, wie diese Prüfung denn weitergehen sollte. Mit der Zeit wurde ich jedoch ruhiger. Beide Prüfer halfen mir gut weiter, wenn ich stockte, und setzten auch mal anders mit einer Frage an, so dass mir die Antwort dann wieder einfiel. Allein beim Zirkularitätstest hatte ich wirklich keine Ahnung – wenig ruhmreich, merkte Prof. Indermark an (zumal man davon ausgehen kann, dass der Zirkularitätstest garantiert gefragt wird). Ich hatte wegen meines schlechten Starts und einigem Nachfragen, das nötig ware, schon die schlimmsten Befürchtungen, aber wie Prof. Thomas nachher sagte: „Von so kleinen Unzulänglichkeiten abstrahieren wir, und darin sind wir ja Meister“.