Angewandte Automatentheorie
Automata and Reactive Systems
Programmanalyse und Compileroptimierung

18. Oktober 2004

Zusammenfassung

Dieses Protokoll beinhaltet Gedächtnisprotokolle zu zwei Diplomprüfungen (Theoretische Informatik) über folgende Vorlesungen:

- `Automata and Reactive Systems` im Wintersemester 2002/03 (Prof. Dr. W. Thomas)
- `Angewandte Automatentheorie` im Sommersemester 2003 (Prof. Dr. W. Thomas)
- `Programmanalyse und Compileroptimierung` im Sommersemester 2003 (Prof. Dr. K. Indermark)

Außerdem ist noch eine Prüfungsprotokoll über eine Scheinprüfnung in der Vorlesung `Programmanalyse und Compileroptimierung` beigefügt.

Inhaltsverzeichnis

1 Diplomprüfung Nr. 1

Die Diplomprüfung wurde in einer gemeinsamen mündlichen Prüfung der beiden Professoren W. Thomas und K. Indermark durchgeführt. Es war eine sehr faire und angenehme Atmosphäre, bei der meine anfängliche Nervosität schnell wich. Die Professoren haben gegenseitig Prototkoll geführt, so dass im Geschäftszimmer von Herrn Professor Dr. W. Thomas nur drei Personen waren.

Auf diese Diplomprüfung habe ich mich ca. 2 Monate mit Kollegen vorbereitet und vorher aktiv an den Vorlesungen teilgenommen (Übungen bearbeitet und in den Automaten Vorlesungen auch Scheine gemacht).

Die folgenden Fragen habe ich aus dem Gedächtniss aufgeschrieben. Sie wurden so oder in ähnlicher Form gestellt. Die Fragen wurden falls ich eine Antwort gegeben habe, die anders war als die gewünchte umformuliert. Es ist also nicht ganz so schlimm wenn man nicht gleich weiß was die Professoren von einem wollen.

Insgesamt wurde die Prüfung mit 1,0 bewertet.

1.1 Automata and Reactive Systems

- Wie akzeptiert ein Büchi Automat?
  Ein Büchi Automat akzeptiert gdw. er bei einem Lauf auf einem unendlichen Wort unendlich oft einen akzeptierenden Zustand erreicht. Also einen Zustand aus seiner akzeptierenden Menge F.

- Wie ist ein reguläre ω Sprache aufgebaut?
  Sie besteht aus der endlichen Vereinigung dieser Konkatenation: $U * V^\omega$ (Ich habe diese Formel auf dass Papier geschrieben, dass mir zur Verfügung stand)

- Wie kann man aus einem Büchi Automaten diese Struktur ablesen?
  Die regulären Ausdrücke können im Automaten an den möglichen akzeptierenden Läufen abgelesen werden. Die endlichen Läufe, die zu einer Schleife im Automaten führen bilden dabei die regulären Ausdrücke zu den endlichen Sprachen und die Schleifen mit akzeptierendem Zustand bilden dann den regulären Ausdruck zu den ω Sprachen.

- Welche Sprache kann ein nicht det. Büchi Automat erkennen im Vergleich zu einem det. Büchi?
  Ich habe hierbei drei verschiedene Sprachen angegeben, die ihm aber alle zu schwer waren. Schließlich haben wir uns dann auf $(0+1)^*1^\omega$ geeinigt.

- Warum kann der det. Büchi diese Sprache nicht erkennen?
  Man kann ihm Einsen geben, bis der Automat einen akz. Zustand erreicht und anschließend eine 0. Diesen Vorgang wiederholt man.

- Drücke die Sprache 0^*1^ω in S1S aus! (Sprache ist nicht zählend)
  $\phi = \exists y \forall z (z < y \rightarrow P_0(z) \wedge z \geq y \rightarrow P_1(z))$

- Hat die Formel in dieser Form die volle Ausdruckstärke?
  Nein, denn es fehlen die Variablen zweiter Ordnung. Mit ihnen kann man das modulo Zählen realisieren.

- Wie kann dass modulo zählen realisiert werden?
  Man kann Abhängigkeit so zwischen den Mengen realisieren, dass jedes modulo Zählen nachgebildet wird. (Ich habe dazu noch eine Beispiel Formel aufgeschrieben um es besser zu erläutern)

- Wie sind Büchi spiele definiert?
  Spiele sind in der Automatentheorie Spiele zwischen zwei Spielern, die ziehen können. Es ist festgelegt, wo welcher Spieler zieht. In Büchi Spielen gewinnt der Spieler 0 gdw. er garantieren kann, dass einen Zustand aus der akz. Menge unendlich oft besucht. Wobei die akz. Menge eine Menge von Spielzuständen ist. (Ich kann mich leider nicht mehr genau erinnen, aber ich habe es noch etwas deutlicher erklärt)

- Was für eine Eigenschaft hat ein Büchi spiel?
  Ein Büchi Spiel ist determiniert.

- Was bedeutet determiniert?
  Dass von jeder Position des Spielfeldes feststeht, welcher der beiden Spieler gewinnt.

- Wie sind die Gewinnregionen für Büchi Spiele definiert?
  Die Gewinnregionen für Spieler 0 sind die Regionen, in denen Spieler 0

garantieren kann, dass er unendlich oft einen Zustand aus der akz. Menge sieht. Bestimmt wird diese Menge durch den $Attr_0(Recur_0(Q))$

- Was steckt hinter dem Recur Set?
  Definition von Recur

- Gibt es auch nicht determinierte Spiele?
  Ja. Zum Beispiel gibt es auf dem Binärbaum (unendlich) ein Spiel bei dem der Spieler Null eine ω Sprache erfüllen muß.

- Warum ist dieses Spiel nicht determiniert?
  Bei diesem Spiel auf dem unendlichen Binärbaum, kann zu jeder Strategie die einer der beiden Spieler verfolgt eine Gegenstrategie entworfen werden, und entsprechend die Mengen aufgebaut werden. So ist esmöglich, das von der Wurzelposition keiner der beiden Spieler eine Gewinnstrategie hat.

- Welches ordnungstheoretische Konstrukt braucht man dafür? Die Ordinalzahlen um die reelen Zahlen aufzählen zu können.

1.2 Angewandte Automatentheorie

- Wie sind Baumautomaten definiert?
  Haben eine endliche Zustandsmenge und eine Transitionsstruktur, die es ihnen ermöglicht Bäume mit endlichem Verzweigungsgrad zu verarbeiten. Die Struktur der Transitionen korelliert also mit dem Rangalphabet.

- Kann man die Sprache, die nur Einsen links an der Front und Nullen rechts hat erkennen?
  Ja!

- Welche und wieviel Zustände braucht man dafür?
  Man braucht die folgenden Zustände:

  - q_1 drückt aus, dass an der Front nur Einsen gesehen wurden
  - q_0 drückt aus, dass an der Front nur Nullen gesehen wurden
  - q_{10} drückt aus, das an der Front links nur Einsen und rechts nur Nullen gesehen wurden
  - q_{error} Baum ist nicht in der Sprache

- Kann man die Sprach die gleich viele Nullen wie Einsen an der Front hat erkennen?
  Nein

- Warum kann man diese nicht erkennen?
  Man bräuchte unendlich viele Zustände um die unendlich vielen verschiedene Differenzen zu codieren. Angenommen es wäre möglich, dann muß der Automat verschiedene Differenzen auf den gleichen Zustand abbilden. Durch geschicktes verketten von Bäumen ist es nun möglich zwei Bäume zu bauen, von denen einer in der Sprache ist und der andere außerhalb. Da es allerdings durch den gleichen Zustand codiert wurde muß der Automat akzeptieren oder verwefen. Beides ist nicht möglich. Widerspruch.

- Ist das Leerheitsprobleme über Baumautomaten effizient lösbar?
  Ja. Man kann sich einfach die erreichbaren Mengen konstruieren, indem man mit die Transitionsfunktion zur Hilfe nimmt! Wird ein Zustand aus F erreicht ist der Automat nicht leer.

- In welcher Komplexität ist es lösbar? Das Problem ist in Polynomzeit in der Anzahl der Zustände plus Anzahl der Transitionen lösbar. (Ich bin mir nicht mehr ganz sicher wie es war.)

- Welche stärkeren Automatenmodelle kennen sie?
  CSFM, Pushdownautomaten usw.

- Warum ist das Ereichbarkeitsprolem über der CSFM unendscheidbar? Das Ereichbarkeitsproblem ist die Frage ob eine Konfiguration erreicht werden kann oder nicht. Ich habe es auf das Halteproblem von Turingmaschinen reduziert. (Relativ genau. Mit Konfiguration der CSFM und Aufbau)

- Was ist wenn der Kanal der CSFM beschränkt ist?
  Dann kann die CSFM durch eine NEA simuliert werden. Dann folgt, dass alle Probleme wie beim NEA abgehandelt werden können.

- Welche andere unbeschränkte Automaten kennen sie?
  Petrinetze (Hierbei habe ich wieder alle möglichen Automaten aufgezählt und brauchte einen kleinen Wink um auf dass Netz zu kommen)

- Was ist eine Konfiguration in einem Petri Netz?
  Eine Markierung ist ein Konfiguration, also eine Belegung der Stellen mit Marken.

- Wie kann man diese Konfiguration mit einem Wort ausdrücken?
  Durch einen Vektor von natürlichen Zahlen. (Hier wurde ich noch nach dem Lemma von Dixons und seiner Aussage gefragt. Allerdings kann ich mich nicht mehr an die Frage erinnern, ging irgendwie fließend über.)

- Warum ist der Karp Miller Baum endlich?
  Weil er nur eine endliche Verzweigung hat. (Leider kenne ich auch hier

meine Antwort nicht mehr genau. Hatte was mit dem Dixon Lemma, der Markierungswiederholung und dem ∞ Zeichen zu tun)

- Welche Aussage steckt hinter dem ∞ Zeichen im Baum?
  Dass in dem Baum eine Schaltungsfolge existiert in der sich die Markierung erhöht. Die Markierungsfolge ist also nicht beschränkt.

- Wann sind die Konfigurationen im Petrinetz beschränkt?
  Wenn im Karp Miller Baum keine ∞ Zeichen auftreten.

1.3 Programmanalyse und Compileroptimierung

- Was für eine mathematische Modellierung verwenden wir in der Programmoptimierung?
  Wir verwenden den Verband.

- Was ist der Unterschied zwischen einem Verband und einer Halbordnung?
  Eine vollständige Halbordnung besitzt ein kleinstes Element und jede aufsteigende ω Kette hat eine kleinste obere Schranke. In einem vollständiger Verband besitzt jede Teilmenge aus dem Verband eine größte untere und kleinste obere Schranke.

- Wofür braucht man die Halbordung?
  Um Fortsetzungsfunktionen zu definieren. Diese braucht man um die Programmsemantik zu modellieren.

- Warum gibt es bei partiellen Funktionen nicht immer ein Maximum?
  Man kann sich die Menge der Funktionen über den natürlichen Zahlen vorstellen. Dort kann es zu den fast komplett undefinierten Funktionen $g(3) = 4$ und $f(3) = 5$ kein Maximum geben, da die Funktion dann die Eigenschaft haben müsste: $h(3) = 4$ und $h(3) = 5$. Dies ist ein Widerspruch.

zurück zur Programmanalyse

- Woraus besteht das monotone Framework?
  Das monotone Framework besteht aus einem Datenflußgraph und einer abstrakten Interpretation. Die abstrakte Interpretation besteht aus dem vollständingen Verband, einer Abbildung und einer Startinformation. (usw.)

- Was ist wenn die Abbildungen monoton sind?
  f ist monoton falls. aus $a \leq b \rightarrow f(a) \leq f(b)$. Falls dies zutrifft hat der vollständige Verband einen kleinsten Fixpunkt.

- Warum braucht man Stetigkeit?
  Um dem kleinsten Fixpunkt berechnen zu können.

- Was ist bei acc?
  Die acc Eigenschaft sagt aus, dass jede aufsteigende ω Kette nach endlich vielen Schritten stationär wird.
- Warum ist bei acc Stetigkeit gleich Monotonie?
  Weil jede monotone Funktion nach endlich vielen Schritten stationär wird gilt die Eigenschaft der Stetigkeit. (Ich habe es ausführlicher erläutert, weiß aber nicht mehr wie.)
- Welches ist das Standardverfahren um einen Fixpunkt zu bestimmen?
  Die Fixpunktiteration
- Wie geht sie vor?
  Sie rechnet parallel ausgehend vom Bottom Element auf allen Gleichungen bis der Fixpunkt erreicht wird.
- Welche Vereinfachungen gibt es?
  Den Worklist- Algorithmus.
- Wie arbeitet der Worklistalgorithmus anschaulich?
  Er folgt in unserem Fall der Struktur des Datenflußgraphen und rechnet immer nur auf einem Teil des Gleichungssytems weiter.
- Wie ist eine Datenflußgeleichung in einem MFP System definiert?
  Sie ist ein Join über mehrere verschiedene Abbildungen der abstrakten Interpretation.
- Welche Analyse würde sie verwenden, um das mehrmalige ausrechenen gleicher Ausdrücke zu vermeiden?
  Common Subexpression Elimination
- Wie erfolgt der Übergang (die Modellierung) der Transformationen in diesem Verband?
  Die Tranformation ist eine Konkatenation von kill und gen Funktion.
- Wie arbeiten die Kill und gen Funktion? Hier habe ich an einem konkreten Beispiel gezeigt wie die Kill und gen Funktion definiert sind und zusammenspielen.
- Was braucht man um Schleifen in `GOTO` Programmen zu erkennen?
  Die Dominator Relation. Die Dominator Relation sagt aus, dass jeder Pfad zu dem dominierten Kanten über den dominierenden führt.
- Warum hat die Menge der Rückwärtskanten die Schleifeneigenschaften?
  Weil die Rückwärtskanten mit Hilfe der Dominator Relation defniert wurden stellt sie die Bedingung des starken Zusammenhangs und des eindeutigen Eingangs sicher.

- Welche Transformationen sind vorstellbar?
  Code Motion und Elimination von Induktionsvariablen

- Was passiert bei Code Motion?
  Code der bei jedem Durchlauf durch die Schleife das selbe bewirkt wird vor der Schleife schon ausgeführt. Schleifeninvarianter Code wird unter Umständen vor betreten der Schleife ausgeführt.

- Was sind die Bedinungen bei Code Motion?

  - Die verschobene Zeile dominiert jeden Ausgang
  - Die definierte Variable wird nur einmal in der Schleife definiert.
  - Die Variable wird falls sie in der Schleife benutzt wird eindeutig benutzt. Es gibt keine weiteren Definitionen von ihr die innerhalb der Schleife Einfluß haben.

- Welche weitere Optimierung gibt es bei der Schleifenoptimierung? Elimination von Induktionsvariablen.

- Was ist der Vorteil dieser Optimierung? Langsame Operationen werden durch einfacherer ersetzt. (Da nun schon ungefähr etwar 45 Minuten rum waren mußt ich nichts weiter erklären)

2 Diplompüfung Nr. 2

Inhalt:	Angewandte Automatentheorie (WS 2002/03) Automata and Reactive Systems (SS 2003) Programmanalyse und Compileroptimierung (SS 2003) (je 4 SWS)
Prüfer:	Prof. Thomas und Prof. Indermark
Datum:	10.10.2003
Dauer:	etwa 40 Minuten
Note:	1.0

2.1 Automata and Reactive Sytems

- Wie kann man einen Müller-Automaten (über omega-Wörter) in einen Büchi-Automaten umwandeln?

- Wie ist der Zusammenhang zwischen Müller-Automaten und deterministischen Büchi-Automaten? (Müller ist ein boolesche Kombination aus deterministischen Büchis)

- Was gehört zur "Lösung" eines Spiels?

- Strategien:
  - Wie sieht eine Gewinnstrategie für ein Müller-Spiel aus? (Welche Form?)
  - Wie groß kann der Strategie-Automat werden?
  - Womit kann man diese untere Schranke zeigen? (DJW)
  - Wie ist das DJW-Spiel aufgebaut und definiert?
- Müller-Automaten:
  - Wie kann man erkennen, ob eine von einem Müller-Automaten erkannte Sprache auch von einem deterministischem Büchi-Automaten erkannt wird?
  - Was bedeutet "closed under superloops"?
  - Wie geht das bei co-Büchi?
- Bäume:
  - Wie kann man die Frage "$t \in T(A)$?" für einen Baumautomaten A als Spiel darstellen? (Automaton vs. Pathfinder)
  - Wer gewinnt wann?
  - Wann ist ein Baum regulär?

2.2 Angewandte Automaten Theorie

- DEAs:
  - Wie kann man DEAs minimieren?
  - Mit welcher Komplexität geht das?
- NEAs:
  - Wie kann man NEAs minimieren?
  - Wie ist Bisimilärität definiert?
  - Wie funktioniert das Bisimilaritätsspiel? Wer gewinnt wann?
  - Mit welcher Komplexität kann man BiSim-Relationen ausrechnen?
  - Warum nimmt man nicht die Zustandsäquivalenz? (nicht effizient berechenbar; unter Umständen ist kein Quotienten-Automat minimal)
- Petrinetze:
  - Was ist der Zustand eines Petrinetz? Wie nennt man einen solchen Zustand?

  - Wie schaltet eine Transition?
  - Welchen Sprachen können Petri-Netze erkennen?
  - Was ist eine kontextfreie Sprache, die nicht von einem Petri-Netz erkannt werden kann? (Spiegelsprache)
  - Welche Probleme sind über Petrinetzen entscheidbar?

- Pushdown-Systeme:
  - Bei welchen Automaten ist das Erreichbarkeitsproblem außerdem entscheidbar? (PDAs)
  - Wie heisst das Verfahren dazu? (Saturierungsalgorithmus)
  - Was wird beim P-Automaten verändert? (bei post*(C) auch neue Zustände, bei pre*(C) nur Transitionen)
  - Wie kann die Terminierung garantiert werden?
  - Welche Komplexität hat das Verfahren?

2.3 Programmanalyse und Compileroptimierung

- Dead Code Elimination bei SLC:
  - Was macht Dead Code Elimination bei SLC? Was ist das Ziel?
  - Welche Form hat die Analyse-Information?
  - Wie verändert eine Anweisung (z.B. Wertzuweisung) diese Information? Wie sieht die Transformationsfunktion aus?
  - Wann kann man eine Anweisung weglassen?
- Was gibt es sonst noch für Analysen / Optimierungen für SLC?
- Wie sieht es bei IC aus?
- Wann kann man den Fixpunkt eines Gleichunssystems bestimmen? (Fixpunktsätze)
- Fortsetzungssemantik:
  - Was ist die Fortsetzungssemantik? (Semantik des Restprogramms vom Knoten aus)
  - Welche Eigenschaft haben die phi's? (endrekursiv)
  - Was bedeutet endrekursiv?
- Wie ist die MOP-Lösung definiert?

- MOP - MFP:
  - Wie ist der Zusammenhang zwischen MOP und MFP?
  - Wann sind diese gleich?
  - Was bedeutet Distributivität? (Definition)
- Common Subexpression Elimination auf IC:
  - Wie ist D aufgebaut? (der Bitvektor)
  - Warum "1 = nicht verfügbar"? (wg. join und gewünschtem Ergebnis)
  - Umgekehrt (also "1 = verfügbar") auch möglich? (ja: duale Sicht)
- Schleifen:
  - Wie findet man Schleifen in Programmen? (nat. Schleifen der Rückwärtskanten)
  - Was ist eine Rückwärtskante?
  - Wie ist der Dominator definiert?
  - Warum ist die natürliche Schleife eine Schleife?
- Induktionsvariablen:
  - Welche Optimierung kann man auf Induktionsvariablen anwenden?
  - Welches Ziel hat das?
  - Wie wird das gemacht? (temporäre Variablen, ..)

3 Scheinprüfung Programm Analyse und Compileroptimierung

- Welche programmoptimierungen für SLC gibt es und wie funktionieren diese?
  dead code elimination, konstantenfaltung, common subexpression elimination
- Wie geht man das für IC an?
  Basisblöcke, Fixpunktiteration
- Welche besondere Eigenschaft müssen die Funktionen im Gleichungssystem haben?
  distributiv, monoton, fixpunkt

- Wieso müssen sie distributiv sein?
  für den Join

- Wieso terminiert die Fixpunktiteration?
  ACC Eigenschaft

- Was heisst MFP, MOP?

- Warum rechnet man nicht einfach die Berechnungspfade aus? Reduktion auf Halteproblem - nicht berechenbar

- Welche Analyse wäre noch denkbar?
  MOP

- Warum sollten die Funktionen im Gleichungssystem distributiv sein?
  Damit MFP Lösung = MOP Lösung

- Was passiert bei MFP?
  Betrachte Join statt ganzer Pfade

- Was ist eine schleife? Zusammenhängende Knoten, ein Eingang

- Welche Optimierungen gibt es im Zusammenhang mit Schleifen?
  code motion, strength reduction mit induktionsvariablen

- Was ist die Voraussetzung für code motion?
  invariante variablen

- Wie stellt man fest, ob eine Variable invariant ist?
  ? und Konstantenpropagation

- Wie kann man algorithmisch eine Schleife erkennen?
  $a \to b$ Rückwärtskante und $a \ \underline{dom}\ b$

- Welche Bedingungen müssen für code motion erfüllt sein?
  - Zeile muss alle Ausgänge dominieren.
  - Variable darf innerhalb der Schleife nur einmal definiert werden.
  - falls Variable in Schleife benutzt wird, darf sie nur in der Schleife definiert sein.

- Ist code motion immer eine Verbesserung?
  ja

- Wie funktioniert strength reduction mit Induktionsvariablen?

- Warum ist das eine Verbesserung?
  $+$ einfacher als $*$
- Ist strength reduction immer eine Verbesserung?
  nein, Berechnungspfad kann vor der Optimierung an Anweisung vorbei gehen
- Wie ist der einfache Ansatz bei interprozeduraler Analyse?
  im Flussdiagramm Kanten vom Funktionsaufruf an Funktionsstart und Von funktionsende an Aufrufende
- Wieso funktioniert das nicht gut?
  Bei rekursiven Funktionen oder bei zwei Aufrufen einer Funktion entstehen unmögliche Pfade
- Was macht man?
  kontextfreie grammatiken einführen