Theorie Prüfung (22.12.2008) bei Prof. Thomas (zusammen mit Dr. Noll) über:
- Angewandte Automatentheorie (Thomas, SS03)
- Modelchecking (Thomas, WS03/04)
- Compilerbau (Noll, WS06/07)
- Bisimulation (aus Buch: Principles of model checking (Baier,Katoen))

Thomas hat AAT und MC geprüft, Noll hat CB gemacht.

Lernmaterialien:
- Vorlesungsvideos (Thomas,SS03) + Folien(Thomas,WS03/04) + Skript (Thomas) zu AAT
- Vorlesungsvideos (Thomas,WS03/04) + Folien (Thomas,WS03/04) zu MC
- Folien (Noll,WS06/07) + Skript (Indermark,SS04) + Wikipedia + [http://www2.informatik.hu-berlin.de/~kunert/papers/lr-analyse/] zu CB
- entspr. Buch zu Bisimulation

Note: 1.3

Mit dem Lernen habe ich etwa 2 Monate angefangen vor der Prüfung. Keine der Vorlesungen habe ich vorher gehört. Die Vorlesungsvideos vom Thomas sind sehr verständlich und sehr zu empfehlen, danach habe ich im Prinzip alles von AAT und MC bereits verstanden. Zum Festigen bin ich dann die Folien in AAT und MC noch durchgegangen und habe alles grob zusammengefasst. Beim CB kam ich leider mit den Folien vom Noll nicht gut klar und habe ich insgesamt auch recht wenig gelernt. Die Lexikalische Analyse konnte ich noch von dem Grundstudiumseminar (und auch, weil die mit dem AAT Wissen wirklich einfach ist), die Syntaxanalyse (d.h. die LL/LR/SLR/LALR Grammatiken) habe ich vor allem durch den Text aus Berlin (siehe URL oben) richtig verstanden (durch den Text auch direkt auf anhieb, den ich auch erst am vorletzten Tag mir durchgelesen habe). Im Skript vom Indermark habe ich mir die Code Generierung noch einmal genauer angesehen (weil ich die nach den Folien bisher eigentlich gar nicht konnte; wohlgemerkt, das habe ich auch erst am letzten Abend vor der Prüfung gemacht, vor allem aus Panik, damit ich wenigstens überhaupt etwas sagen konnte, wenn das kommt). Bei der Semantikanalyse wusste ich auch nur die Grundlagen (d.h. Attributgrammatiken und dass man da auf Zirkularität aufpassen muss) und die nur aus den Folien. Innerhalb der letzten 3 Tage bin ich außerdem viele der Prüfungsprotokolle von s-inf durchgegangen und habe mir viele Fragen rausgeschrieben.

Verlauf:

Ich sollte eine Reihenfolge vorschlagen. Ich wollte AAT,MC,CB und Thomas war damit einverstanden.

AAT:

Thomas: Wir haben Logiken betrachtet, die gleichmächtig zu Automaten sind.
Ich: Ja, z.B. MSO.
Thomas: Genau. Zu einer MSO-Formel, wie groß ist denn da der Automat?
Ich: *überleg* Also man benutzt für jeden Quantor einen Zustand.
Thomas: Das hört sich ja dann recht gut.
Ich: (nun fällt mir gerade wieder alles ein) Also, aber das macht man nur für die E-Quantoren. Man muss die Formel dann an entsprechenden Stellen negieren, um nur E-Quantoren zu haben. Für die Negation muss man dann jeweils den Komplementautomaten bilden und für die anderen Operationen, also Und/Oder entsprechend. Das Komplement macht man durch Potenzkonstruktion nach DEA und dort dann F umdrehen. Durch die Potenzkonstr. wird der exp. groß.
Thomas: Ja genau. Können sie dann eine obere Grenze angeben, wenn ich ihnen eine Formel gebe?
Ich: Das mit der Potenzkonstr kann in jedem Schritt passieren. Dadurch hat man dann so eine Treppe von 2er-Potenzen, also 2^(2^(2^...n)).
Thomas: Genau. Wie ist das denn bei DEAs? Also wie groß werden die?
Ich: Die müssen natürlich mindestens so groß wie die NEAs werden.
Thomas: Wodran liegt das genau? Wieso kann man die MSO-Formel nicht bei jedem Schritt direkt in einen DEA transformieren?
Ich: Also beim Produktautomat (beim Und/Oder) wird es schon mal komplizierter.
Thomas: Aber das ist ja noch polynomiell, nicht exp.
Ich: (ah, mir fällt es wieder ein) Ah, aber das Problem ist bei dem EM-Quantor, denn wenn man den anwendet auf einen Automaten, dann kann der nichtdet. werden. Und dann muss man doch wieder Potenzmengenkonstr. machen.

Thomas: Ja genau, das ist nämlich das Prob.
Thomas: Zu einem Automaten, wie groß ist da denn die MSO-Formel?
Ich: Also man benutzt da einen EM Quantor für jeden einzelnen Zustand, deshalb also so groß wie der Automat.
Thomas: Geht das auch besser?
Ich: Ja, durch spezielle Kodierung kann man das auch mit einer logarithmischen Anzahl machen.
Thomas: Also z.B. bei 100 Zuständen?
Ich: Für 128 Zustände geht das mit 2^7 oder 2^8 oder so.
Thomas: Ja genau. Geht es denn noch besser?
Ich: *überleg* Ja, wir hatten dazu etwas gesagt (wusste aber nicht mehr ganz genau was das Ergebnis war)
Thomas: Geht es denn z.B. nur mit einem E-Mengenquantor?
Ich: *etwas zweifelnd* Ja, ich glaube schon.
Thomas: Sie glauben? Also ist auch nicht so schlimm, das wurde in einer Übung gemacht. Also ja, das geht auch.

Thomas: Kommen wir zu etwas anderem. Die NEA-Minimerung, wie effizient lässt sich die denn ausführen?
Ich: Das Problem ist in PSPACE.
Thomas: Wodran liegt das denn? Oder wie kann man das zeigen?
Ich: Man geht von einer PSPACE-Sprache L aus und für jedes Wort w in L konstruiert man einen entsprechenden NEA ... (hier in etwa den kompletten Beweis beschrieben)
Thomas: Über welcher Sprache geht denn der NEA?
Ich: Über den Konfigurationen der TM. Weil die Sprache in PSPACE ist, haben wir für ein Wort w die maximale Größe einer Konfig beschränkt durch p(|w|). Jede Konfig, deren Anzahl damit endlich ist, ist ein Element im Alphabet.
Thomas: Und was erkennt der Automat dann?
Ich: Also, der checkt gerade die Läufe auf der TM, also d.h. ob die Fehler enthalten.
Thomas: Ah genau, ja der geht über die Läufe.

Thomas: Kommen wir mal zu Nichtentscheidbarkeitsproblemen. Bei den PDS, was kennen sie denn da für Nichtentscheidbarkeitsprobleme? Wie sieht es z.B. mit der Erreichbarkeit aus?
Ich: Also, wir wissen, dass die Erreichbarkeit in 2-PDS nicht entscheidbar ist.
Thomas: OK, kennen sie andere einfachere Systeme, wo das auch noch gilt.
Ich: Ja, man kann das auf 2-CS überführen und dort ist das auch noch unentscheidbar.
Thomas: Und bei 1-CS?
Ich: Da ist das dann entscheidbar.

Thomas: Wie sieht das denn mit der Erreichbarkeit bei PDS aus?
Ich: Ja, die ist entscheidbar.
Thomas: Wie genau macht man das? Also wenn man jetzt c_1 und c_2 hat, wie kann man das entscheiden ob c_2 von c_1 erreichbar ist?
Ich: Also, wir haben das allgemeiner gemacht. Wir betrachten da die Menge pre*(C), wobei C regulär ist. Für C haben wir einen NEA gegeben. In dem Beispiel wäre einfach C = {c_2}.
Thomas: OK, und wie bildet man das pre*(C)?
Ich: Man erweitert den Automaten von C entsprechend so lange, bis sich nichts mehr ändert.
Thomas: Was genau fügt man denn hinzu? Auch Zustände?
Ich: Nein, man fügt nur Transitionen hinzu.

Thomas: Noch kurz etwas zu Petri-Netzen. Die sind ja sehr ähnlich wie CS, die zählen ja auch. Kann man dann nicht mit einem Petrinetz ein CS simulieren?
Ich: Also, das geht grundsätzlich erstmal nicht, denn in Petrinetzen ist die Erreichbarkeit ja entscheidbar.
Thomas: Aber wodran scheitert es denn?
Ich: (Ja, gute Frage, mal überlegen...) Hm, also bei Petrinetzen kann man keinen Test auf Leerheit machen, dann kann man einfach nicht schalten, wenn eine Stelle leer ist. Bei CS kann man aber eine Transition für =0 (Kellerboden) machen. Thomas: Ah, ja genau das ist das Problem.

Thomas: Die Sprachen, die von den Petrinetzen erkannt werden, verglichen mit den kontextfreien Sprachen, was können sie dazu sagen?
Ich: Bis auf eine gemeinsame Teilmenge, unter anderem die reg. Sprachen, kann ma

n die nicht direkt vergleichen. Man hat z.B. die Sprache a^n b^n c^n, die nicht kontextfrei ist, aber von Petrinetzen erkannt wird. Und dann die Sprache w w^R, die von Petrinetzen nicht erkannt wird, aber die kontextfrei ist.

MC:

Thomas: Kommen wir dann mal zum MC. Da haben wir LTL kennen gelernt. Das MC Problem ist ja, dass man eine LTL−Formel hat und ein Transitionssys. und man will nun entscheiden, ob die Formel eingehalten wird. Wie geht man da vor?
Ich: Man will einen Automaten finden, der äquivalent zur negierten Formel ist. Dann bildet man den Produktaut. davon mit dem Transitionssys. Das Umformen von LTL−>BA ist dabei das schwierige. Da geht man in mehreren Schritten vor, nämlich erstmal LTL−>ATA. Das geht noch relativ kanonisch.
Thomas: Wie schnell/gut geht das.
Ich: Das geht noch linear, d.h. man hat für jede Teilformel einen Zustand.
Thomas: Wie sehen denn die Schleifen in dem Automat aus?
Ich: Also, die Schleifen können überhaupt nur bei dem R und U auftreten. Und dort sind die Schleifen trivial, also d.h. nur auf sich selbst. So hat man eine partielle Ordnung auf dem Baum.
Thomas: Ah ja, genau. OK, wie geht es dann weiter?
Ich: Danach kommt ABA−>VBA.
Thomas: Was ist denn ein VBA?
Ich: Das ist ein BA mit spezieller Akzeptanzbedingung. Und zwar hat man da ein Mengensystem \T = {T_1,...,T_n} und jedes T_i muss unendlich oft besucht werden.
Thomas: *erstaunt,zweifelnd* Moment, wie meinen sie das mit unendlich oft?
Ich: *zweifelnd* (überleg, das war doch korrekt so), also für jedes i..., also wenn \phi ein Lauf ist, dann: \forall i: \exists k: \phi_k = T_i. Und dann noch ein \infty über das \exists, dort hat man das unendlich oft.
Thomas: Ah, so wird es klar. Ja genau. Das war nicht ganz so, wie sie das am Anfang gesagt haben.
Thomas: OK, kommen wir mal zum CTL. Z.B. hat man da die Formel E(pUq). Was sagt die denn aus?
Ich: Das sagt, dass es einen Pfad gibt, auf dem erstmal p und dann irgendwann q gilt.
Thomas: Sie meinen also z.B. p***q?
Ich: Nein, das p muss die ganze Zeit vor dem q gelten.
Thomas: Ah, ja genau. Wie führt man denn nun das CTL−MC aus?
Ich: Also, man geht da induktiv über die Teilformeln und markiert die Zustände.
Thomas: Wie genau meinen sie das?
Ich: Also, p/q ist ja schon markiert. Nun betrachtet man pUq. (das war falsch)
Thomas: Moment, aber das bezieht sich ja nicht auf einen Zustand sondern auf einen Pfad.
Ich: *etwas stockend, überlegend* Ah, also man geht von der ganzen Formel jetzt weiter aus. Da markiert man jeden Zustand, ob er die Formel erfüllt oder nicht.
Thomas: Und wie geht man da für das Beispiel vor? Also was würden sie jetzt als erstes markieren?
Ich: Also, erstmal kann man die ganzen q markieren, denn da wird die Formel auf jeden Fall erfüllt.
Thomas: Und wie sieht nun der nächste Schritt aus?
Ich: Nun kann man von den markierten Zuständen ausgehen und entsprechende p markieren. (Nun wird mir grad alles wieder genau klar.) Das kann man so häufig wiederholen, bis man auf die Methode keine weiteren Zustände mehr markieren kann. Dann weiß man, dass man alle gefunden hat und kann aufhören.
Thomas: Ja genau. Kommen wir zum symbolischen CTL. Da hat man das ja auch ähnlich, dass man etwas immer wiederholt, bis sich nichts mehr ändert. Wie geht das? Oder erstmal, wie funktioniert symb. CTL genau?
Ich: Also, man kodiert da die Kriptestruktur mit OBDDs. Die CTL−Formel baut man dann auch wieder induktiv auf über die OBDDs. Für den Fall E(pUq) definiert man sich auch nun wieder rekursiv die Formel, also f^0 := q, f^i := ... und hat so eine Folge von Formeln. Wenn sich die nicht mehr ändert, kann man aufhören.
Thomas: Was heißt das genau?
Ich: Wenn man ein k gefunden hat mit f^k+1 = f^k.
Thomas: Was ist Gleichheit hier? Äquivalenz? Oder Isomorphie bei den OBDDs?
Ich: Ah, also bei jedem Schritt minimiert man den OBDD auch immer. Und weil minimierte OBDDs zu äquivalenten Formeln bis auf Isomorphie gleich sind, reicht es hier einfach auf Isomorphiegleichheit zu testen.
Thomas: Ah, ja genau.

CB:

Noll: OK, beginnen wir mal ganz hinten. Wie geht denn die Codegenerierung?
Ich: (Na super...) Äh, also man hat da Funktionen, die rekursiv den Code aufbauen.
Noll: Also mal konkreter bezogen auf das Beispiel von boolsche Formeln. Wie geht man da vor?
Ich: Also, da hat man dann z.B. die Funktion: b: BExp x Tab x Adr x Level −> AM−Code. Dabei ist BExp der Ausdruck, Tab ist die symbolische Tabelle, Adr die Code−Adresse und Level das aktuelle Level.
Noll: Was genau steht denn in der Tabelle drin?
Ich: Die Adressen von den ganzen Variablen und so...
Noll: Ja, und wie sieht das z.B. bei globalen Variablen aus?
Ich: (Keine Ahnung, mal ein bisschen raten.) Also, wenn man die auf dem Heap speichert, ist das dann die Heapadresse.
Noll: Einen Heap brauchen wir hierbei aber gar nicht, wir kommen schon mit dem Prozedurkeller aus. Wenn die globale Variable in einer Prozedur gespeichert ist, was müssen wir dann denn wissen?
Ich: Also, man speichert die Leveldifferenz dann zu der Prozedur und deren Adresse.
Noll: Nicht ganz, man speichert gerade das absolute Level. OK, wie sieht das nun konkret aus für den OR−Fall?
Ich: (Ah, das weiß ich zumindest noch so ungefähr von gestern) Dann hat man z.B. für OR: b(L1 OR L2, ... a) := b(L1,..), b(L2,..), OR..;
Noll: Ja genau. Das ist jetzt der nicht−strikte Fall. Wie sieht denn der strikte aus?
Ich: (Ah, das weiß ich sowieso) Ja genau, das hat man z.B. in C. Da sieht die Def ein bisschen allgemeiner aus, nämlich hat man da nbt(..., a, a_t, a_f) und a_t ist die Adresse, wo man hinspringt im Falle von true und a_f entspr.
Noll: Genau. Und wie sieht dann die Def. aus?
Ich: nbt(L1 OR L2, ..) := nbt(L1, .., a_t, a'), nbt(L2, .., a_t, a_f).

Noll: Genau. Kommen wir mal zu der Syntaxanalyse. Wie sieht das denn dort mit Prioritäten aus? Also z.B. das AND vor dem OR gewertet wird?
Ich: (Na toll, ich dachte ich wäre sicher in der Syntaxanalyse. Aber gerade das mit den Prioritäten weiß ich gar nicht. Ich frage mich ob das überhaupt im Skript stand.) Hm, also man könnte einfach die Grammatiken so bilden, dass es automatisch klar ist.
Noll: Ja gut, das könnte man tun. Aber es geht auch ohne Anpassung. Wie macht man das denn dann?
Ich: Hm, also man könnte in der aktions−Tabelle die Prioritäten mit einbeziehen und dann je nach Priorität Reducen.
Noll: Hm ja, wenn man z.B. A OR B AND C hat und von links parst, wie sieht das jetzt aus, wenn man gerade hinter B ist?
Ich: Also er liest dann das AND und wegen der höheren Priorität macht er dann kein Reduce, sondern shiftet erstmal.
Noll: Genau. Wir haben uns nun ja auch mit Generatoren wie yacc beschäftigt. Wie wird das denn da gelöst?
Ich: (Na toll, ich hatte gehofft, dass ich das gar nicht Lernen muss, weil bestimmt keine Fragen zu den Tools kommen...) Hm, also da wird vermutlich eine Liste von Prioritäten gespeichert, d.h. erst kommt das AND und dann das OR.
Noll: Ok, wie sieht das denn mit Assoziativitäten aus? Wie wird linksassoziativ gehandhabt?
Ich: (Wusste nicht wirklich was das ist, aber mal was hingeschrieben) Hm, also wenn man das Beispiel "a << b + c" betrachtet... und dann das "b"...
Noll: Also das Beispiel ist jetzt eher schlecht, da hat man Assoziativität ja gar nicht.
Ich: (Ah!) Ah, also wenn man das Beispiel "a + b + c" betrachtet, dann heißt linksass., dass erst a+b zusammengefasst wird und danach das c.
Noll: Also jetzt nach dem Lesen vom b, was wird dann gemacht? Shift oder Reduce?
Ich: Da wird dann der Reduce−Schritt gemacht.
Noll: Ja genau.

Dann musste ich wie immer raus. Grundsätzlich war ich nicht so sonderlich nervös während der Prüfung und beide Prüfer waren auch sehr nett. Thomas kannte ich ja schon vorher, Noll habe ich dort zum ersten Mal getroffen. Ich habe trotzdem be

i vielen Fragen sehr gestockt und war mir nicht sofort ganz sicher. Erst als ich im Gespräch dann langsam die Sachen wieder hergeleitet hatte, wurde mir selbst langsam alles wieder klar.

Letztendlich war das dann der Hauptkritikpunkt, dass ich mich häufig erstmal und eutlich ausgedrückt habe und nicht konkret sofort eine exakte deutliche Antwort geben konnte. Ansonsten meinte Thomas, ich hätte alles aber sehr gut durchschaut . Daher die Note 1.3.