Prüfungsprotokoll Wahlmodul II der Informatik 25610 (Bachelor Informatik)
logisches und funktionales Programmieren 01816
Datum: 05.03.2018
Prüfer: Prof. Dr. Beierle
Dauer: ca. 35min
Note: 1,0
Videokonferenzprüfung

Woraus besteht ein Prologprogramm?
Aus Fakten und Regeln. Fakten sind Tatsachen über Objekte und deren Eigenschaften. Regeln beschreiben wie aus vorhandenen Tatsachen neue Tatsachen gefolgert werden können.

Wie sieht eine Regel aus?
Eine Regel besteht aus wenn- und dann-Teil. Der dann-Teil ist auf der rechten Seite und stellt Voraussetzungen dar, unter welchen der wenn-Teil auf der linken Seite gefolgert werden kann. Fakten sind Regeln ohne Voraussetzungen.

Wie sieht eine Anfrage aus?
Ein oder mehrere Literale. Ein Literal gleicht der linken Seite einer Regel, muss jedoch erst bewiesen werden.

Wie wird diese nun bewiesen?
Von links nach rechts. Zu jedem Literal wird eine Klausel gesucht, deren Kopf mit dem Literal unifiziert werden kann. Kann dann die rechte Seite dieser Klausel bewiesen werden, ist das Literal beweisbar. Es wird mit dem nächsten Literal fortgefahren, wobei die Substitutionen die für Variablen angewandt wurden mitgenommen werden. Können alle Literale bewiesen werden, dann ist die Anfrage beweisbar.

Was ist dann die Ausgabe?
Yes. Für Variablen die nicht-instantiiert übergeben wurden, wird eine Substitution ausgegeben.
Wie funktioniert Backtracking?
Wenn es beim Beweis eines Literals Alternativen gibt, z.B. mehrere Klauseln, deren linke Seite mit dem Literal unifiziert werden kann, wird ein Backtrackingpunkt angelegt. Scheitert der Beweis des Literals, kann zu diesem Punkt zurückgekehrt werden, und alternativ eine weitere Klausel versucht werden.

Was ist das Ziel der Unifikation?
Bei der Unifikation werden Variablen durch Terme substituiert. Es wird ein allgemeinster Unifikator bestimmt. Da ich sah, dass Prof. Beierle etwas anderes wollte, habe ich begonnen den Unifikationsalgorithmus zu erklären, wurde aber gleich unterbrochen. Die prägnante Antwort, die der Professor wollte lautete, dass zwei unifizierte Terme dadurch gleich werden.

Was ist ein allgemeinster Unifikator?
σ ist ein allgemeinster Unifikator, wenn für jeden anderen Unifikator σ' gilt, es gibt eine weitere Substitution ϕ so, dass $\sigma' = \phi \circ \sigma$.

Sie kennen die Fakultätsfunktion aus der Mathematik?
Ja, Fakultät von n ist das Produkt der natürlichen Zahlen von 1 bis n.

Schreiben Sie ein Prädikat, das die Fakultät berechnet.
fac(1,1).
fac(N,E) :-

N1 is N – 1,
fac(N1, E1),
*E is E1 * N.*

Wie funktioniert das is?
is ist ein partielles Prädikat. Das bedeutet, dass alle Variablen auf der rechten Seite instantiiert sein müssen. is wertet dann die rechte Seite aus und unifiziert die Variable auf der linken Seite mit dem Wert.

Muss auf der linken Seite eine Variable stehen?
Nein. Beispielsweise 5 is 2 + 3 ist möglich. 2 + 3 wird berechnet und ergibt 5. 5 und 5 können unifiziert werden, da es sich um dieselbe Konstante handelt.

Können Sie ein Prädikat programmieren, das eine Liste umdreht?
reverse(L,L1) :- rev(L,[],L1).
rev([],A,A).
rev([K|R],A,L) :- rev(R,[K|A],L).

Können Sie dazu noch ein bisschen etwas sagen?
L *ist die Eingabeliste, L1 die Ausgabeliste. Es wird ein Hilfsprädikat mit 3 Argumenten verwendet. Die Eingabeliste, in der Mitte ein Akkumulator und die Lösungsliste. Es werden zunächst der Reihe nach einzelne Elemente von der Eingabeliste abgespalten und vorne an den Akkumulator angehängt. Wenn die Eingabeliste leer ist, enthält der Akkumulator die Lösung und wird als solche im 3. Argument wieder nach oben gereicht.*

Können Sie ein Prädikat aufschreiben, das beweisbar ist, wenn die 2. Liste nicht die umgekehrte 1. Liste ist?
notrev(L,L1) :- \+ reverse(L,L1).

Was ist, wenn Ihnen das not nicht zur Verfügung steht?
Dann kann ich es programmieren mit
\+ X :- call(X),!,fail.
\+.
Wenn X beweisbar ist, dann führt die erste Regel zum scheitern. Sonst ergibt die 2. Regel den Beweis.

Was macht das !?
Das ! ist der „Cut“-Operator. Er beschneidet den Suchraum. Wenn X im obigen Beispiel beweisbar ist, dann führt das fail zum Scheitern. Der Cut verhindert, dass eine Alternative für call(X) oder für \+ gesucht wird. Es wird also kein Backtracking links vom ! durchgeführt. Logisch hat er keine Auswirkung.

Was ist funktionale Programmierung?
Sie orientiert sich an mathematischen Funktionen. Programme sind kurz, abstrakt, gut versteh- und wartbar. Besondere Merkmale sind

- *Funktionen höherer Ordnung*
- *arbeitet mit Werten und Ausdrücken*
- *jeder Ausdruck hat einen Wert*
- *Konzept der Abschlussobjekte*
- *Symbole werden verarbeitet*
- *…*

Wie wird ein funktionales Programm durch den Scheme-Interpreter ausgewertet?
Der Interpreter liest einen Ausdruck, wertet ihn aus und gibt das Ergebnis aus. Er wertet die geschachtelten Ausdrücke dabei rekursiv aus.

Was macht ein define-Ausdruck?
Ein Ausdruck der Form (define x 3) erzeugt eine neue Bindung x:3 in der aktuellen Umgebung. Die Auswertung von 3 ergibt dabei 3 selbst, da es eine Konstante ist.

Was ist eine Umgebung?
Eine Umgebung ist eine verkettete Liste von Rahmen. Ein Rahmen wiederum ist eine Tabelle von Bindungen und eine Bindung ist ein Paar aus Symbol und Wert.

Wie wird ein lambda-Ausdruck ausgewertet?
Ein Lambda-Ausdruck ist eine Funktionsdefinition. Er wird zu einem Abschlussobjekt ausgewertet. Dieses besteht aus der Funktionsdefinition und einem Verweis auf die aktuelle Umgebung.

Sei nun U die aktuelle Umgebung. Wie wird ein Ausdruck der Form (e_0 e_1 … e_n) ausgewertet?
Zunächst werden alle e_i in der aktuellen Umgebung ausgewertet zu Werten v_i. Eine besondere Stellung nimmt dabei v_0 ein, da es ein Abschlussobjekt ist. Sei die Umgebung auf die es verweist U‘, e‘ der Rumpf der Funktionsdefinition und x_1...x_m seien die formalen Parameter. Wenn m nicht gleich n ist, dann wird ein Fehler ausgegeben. Sonst wird eine Umgebung U“ erzeugt. Diese besteht aus einem neuen Rahmen r und U‘, also U“ = r . U‘. In den neuen Rahmen r werden nun Bindungen für die Parameter eingetragen, also x_1:v_1 … x_n:v_n. e‘ wird dann in U“ ausgewertet.

Weshalb wird nicht in U ausgewertet?
Durch die Benennung U‘ wird nicht vorausgesetzt, dass U‘ = U ist, aber es könnte so sein. Die Auswertung erfolgt so, weil e‘ außer den Parametern noch weitere Symbole, z.B. f enthalten könnte. Wenn der Interpreter das Symbol f auswerten muss, dann sucht er ausgehend vom aktuellen Rahmen r entlang der verketteten Liste nach einer Bindung für f.

Prof. Beierle: Also wegen des statischen Bindens.
Ja, beim statischen Binden sind die Variablenbindungen anhand der Programmstruktur erkennbar.

Nochmal zurück zur logischen Programmierung. Was versteht man unter CLP?
Im Unterschied zur klassischen logischen Programmierung kommen bei der Constraint-logischen Programmierung noch Constraints hinzu. Das sind Bedingungen und Einschränkungen für die Bindungen die Variablen eingehen können, z.B. arithmetische Constraints (Gleichungen und Ungleichungen) oder Bereiche aus denen Werte für die Variablen gewählt werden dürfen. Beim klassischen logischen Programmieren müssen alle Bedingungen durch den Programmierer verwaltet werden. Das macht Programme oft schwer wart- und modifizierbar. Bei CLP kann der Programmierer Bedingungen festlegen, andere werden vom System daraus erzeugt. Verwaltet werden sie jedenfalls vom System, sodass sich der Programmierer nicht darum kümmern muss.

Betrachten Sie CLP über endlichen Bereichen. Was ändert sich?
Der Unifikationsalgorithmus muss um die Behandlung von Domainvariablen erweitert werden.

Teilweise könnten die Antworten ausführlicher sein, aber der Professor hat direkt eine weiterführende Frage gestellt.
Ich kann nicht garantieren, dass die Fragen vollständig sind, oder dass die Lösungsstichworte alle richtig sind. Daher unbedingt die entsprechenden Passagen im Kurstext konsultieren.
Prüfung fing etwas später an.
Prüfungsklima so angenehm wie es unter den Bedingungen einer Prüfung überhaupt möglich ist.

Protokoll mündl. Modulprüfung 01816 Logisches und funktionales Programmieren

Fach: Master Prakt. Informatik
Prüfer: Herr Professor Beierle (+ Beisitzer)
Dauer: ca. 45 Minuten
Note: 2,3

Zettel und Stift liegen bereit und man wird während der Prüfung aufgefordert, einige Prädikate/Funktionen aufzuschreiben. Zu Beginn wurde ich gefragt, ob ich mit logischem oder mit funktionalem Programmieren beginnen will. Die Fragen, die mir noch einfallen:

Logisches Programmieren:

- Aus was besteht ein Prolog-Programm? (→Klauseln = Fakten + Regeln, Fakten + Regeln definieren)
- Wie wird eine Anfrage Q1, Q2, Q3 bewiesen?(→Klauselkopfsuche, Unifikation, …)
- Was ist Backtracking?
- Cut sehr genau erläutern
- Prädikat, mit dem überprüft werden kann, ob die Zahl 8 in einer Liste enthalten ist (true wenn enthalten)
- Prädikat, mit dem überprüft werden kann, ob die Zahl 8 nicht in einer Liste enthalten ist (true wenn nicht enthalten) + detailliert erklären, was die genau passiert, wenn z.B. Liste [7,8] überprüft wird (→welche Klausel wird genommen, was wird mit was unifiziert,…)
- map-funktion - Was macht sie, wie kann sie in Prolog implementiert werden? (konnte ich leider nicht aufschreiben)

Funktionales Programmieren

- Was ist funktionales Programmieren (→basiert auf mathematischem Funktionenmodell, zentrales Element sind Symbole, Funktionen können Ein- und Ausgaben von Funktionen sein = Funktionen höherer Ordnung,…)
- Wie wird ein Symbol ausgewertet? (→Rahmen, Umgebung, Abschlussobjekt erklären)
- Wie wird eine Applikation ausgewertet (→ wichtig, dass der die neue Umgebung mit Rahmen auf die Definitionsumgebung des Operators verweist)
- Funktion, die Funktion zurückgibt, die zweimal beliebige Funktion f ausführt
  (define (zweimal f) (lambda (x) (f(f x)))))
- Funktion, die Funktion zurückgibt, die n-mal beliebige Funktion f ausführt (siehe Funktion repeated im Skript, das „id“ für den Fall, dass Fkt. 0 mal ausgeführt wird, ist wichtig!)

CLP

- Was ist CLP/was sind Constraints?
- WAs ist anders im Vgl. zu logischer Programmierung ohne Constraints? (Constraint Solver, Unifikation von Domainvariablen mit den drei beschriebenen Fällen, …)

Mit dem logischen Programmieren bin ich nie so richtig warm geworden und hatte daher in diesem Bereich durchaus Defizite, funktionales Programmieren und der Bereich CLP war für mich leichter nachvollziehbar und daher lief auch der Teil der Prüfung besser. Die Notengebung empfand ich als wohlwollend und Prüfer sowie Beisitzer waren sehr freundlich. Meiner Ansicht nach aber schon ein recht anspruchsvoller Kurs, bei dem man sich sehr intensiv insbesondere mit dem logischen Programmieren beschäftigen muss, insbesondere wenn man vorher noch nie damit Berührungspunkte hatte.

Modul: 01816 Logisches und funktionales Programmieren
Prüfer: Prof. Dr. Christoph Beierle
Datum: November 2016

Die meisten Fragen, waren so wie in den vorhergehenden Protokollen. Ein paar sind jedoch neu hinzugekommen. An den exakten Wortlaut der Fragen, kann ich mich nicht mehr erinnern. Unten sind also nur die sinngemäßen Fragen aufgeführt. Die Bewertung ist wie schon meine Vorredner sagten sehr wohlwollend. Obwohl ich zwei Schwächen (bei der alternativen Definition des Not und beim Looking ahead) hatte, wurde die Prüfung mit sehr gut bewertet.

Logisches Programmieren:

- Woraus besteht ein Prolog-Programm?
- Wie wird eine Anfrage L_1,...,L_n von Prolog bewiesen?
- Skizzieren Sie ein Prolog-Programm, welche als erstes Argument eine Liste von Zahlen und als zweites Argument das Produkt der Zahlen in der Liste enthält!
- Welche nicht-logischen Elemente gibt es in Prolog?
- Für was braucht man Datenbankprädikate?
- Erklären Sie den Cut ausführlich!
- die Implementierung des not ohne Zuhilfenahme des vordefinierten not (\+)!
  ich wollte dabei das Beispiel aus dem Kurs mit unverheiratet(S):- verheiratet(S),!,fail anführen. Prof. Beierle wollte aber eine Lösung für ein beliebiges Prädikat, worauf ich jedoch nicht gekommen bin. Wenn ich das richtig verstanden habe folgendermassen, wobei X Platzhalter für ein beliebiges Prädikat ist:
  not(X):- call(X),!,fail.
  not(X).
  (keine Gewähr für die Richtigkeit)
- Was versteht man unter relationaler Programmierung?

funktionales Programmieren:

- Was versteht man unter funktionaler Programmierung?
- Nennen Sie eine Funktion in der Funktionen als Parameter übergeben werden!
- Was macht das compose?
- Schreiben Sie eine Funktion für das append!

- Was sind Ströme?
- Wie implementiert man in Scheme unendliche Ströme?
- Was ist ein Homomorphismus?
- Können Sie die Funktion right-reduce implementieren?

Constraint-logisches Programmieren

- Was ist Constraint-Logisches Programmieren?
- Was ändert sich beim Constraint-logischen Programmieren?
- Was ändert sich am Unifikationsalgorithmus?
- Welche Inferenzregeln kommen hinzu?
- Was ist Forward-Checking? (genaue Definition)
- Was ist Looking-Ahead?
- Was ist bei der Inferenzregel beim Looking-Ahead anders als beim Forward-Checking?

Viel Glück!!!

Gedächtnisprotokoll

Modul: 01816 Logisches und funktionales Programmieren
Prüfer: Prof. Dr. Christoph Beierle
Datum: April 2015

Meine mündliche Prüfung verlief im Wesentlichen wie auch die anderen hier in den Protokollen aufgeführten Prüfungen: Prof. Beierle prüfte mein Wissen relativ kurstext-chronologisch in einem Mix aus Fragen zur Theorie und praktischen Programmieraufgaben.

Logische Programmierung:

- Woraus besteht ein Prolog-Programm? (Fakten und Klauseln)
- Was ist eine Anfrage in Prolog? Woraus besteht eine Anfrage? (Antwort: Aus einem oder mehreren Literalen.)
- Wie wird die Anfrage l1, l2, l3 bewiesen?
- Sehr (!) detaillierte Rückfragen zum Backtracking: Wenn nach erfolgreichem Beweis von l1 der Beweis bei l2 oder l3 scheitert, kommt Backtracking zum Einsatz. Kann es denn vorkommen, dass l1, obwohl es schon bewiesen wurde nochmals bewiesen wird? (Scheinbar nein, hat wohl etwas damit zu tun, dass der Unifikationsalgorithmus immer den allgemeinsten Unifikator bestimmt. Hier bin ich geschwommen.)
- Schreiben Sie ein Prädikat, das erfüllt ist, wenn die Zahl 7 in einer Liste vorkommt.
- Modifikation: Wie muss das Prädikat aussehen, wenn das Prädikat erfüllt sein soll, wenn die 7 ODER die 8 vorkommt in der Liste?
- Modifikation: Wie muss das Prädikat aussehen, wenn das Prädikat erfüllt sein soll, wenn sowohl die 7 UND die 8 vorkommen in der Liste?
- Welche Programmiertechniken gibt es in Prolog? (Etwas erzählt über das Programmieren mit Akkumulatoren und das relationale Programmieren.)
- Warum ist nicht jedes Prädikat der logischen Programmierung immer für alle Argumente relational anwendbar? (z.B. Warum macht es keinen Sinn, ein reverse in der klassischen Implementierung mit Akkumulator mit ?- reverse(X, Y). oder ?- reverse(X, [2, 3]). aufzurufen, anstatt X mit einem Grundterm zu belegen und das zweite Argument mit einer Variable zu belegen? Diese Frage konnte ich nicht gut beantworten, habe dann gesagt, dass es eben darauf ankommt, wie das Prädikat implementiert ist.)
- Welche nicht-logischen Komponenten gibt es in Prolog? (Cut, Negation)

Funktionale Programmierung:

- Was ist funktionale Programmierung?
- Was genau macht das define? (Ein Makro, das dafür sorgt, dass Bindungen zwischen Symbolen und ihren Werten im Rahmen der aktuellen Umgebung eingetragen werden…)
- Nennen Sie eine Funktion höherer Ordnung (compose erwähnt).
- Wie wird ein Symbol ausgewertet?
- Wie wird eine Applikation (e0 e1 … en) ausgewertet?

- Was ist eine Umgebung?
- Auf welche Umgebung verweist ein Rahmen, der erzeugt wurde durch den Aufruf einer Funktion? (Habe geantwortet, auf die Definitionsumgebung, also auf diejenige Umgebung, innerhalb derer die Funktion definiert wurde, damit schien Prof. Beierle aber nicht so richtig zufrieden zu sein. Er stellte mehrere Rückfragen dazu, was mich aber eher verwirrte.)
- Schreiben Sie eine zweistellige Funktion, die als Input eine Liste und eine Funktion bekommt und die Funktion anwendet auf jedes Element der Liste.
- Was sind Ströme?
- Was liefert die Funktion sum-stream (basierend auf right-reduce) zurück?
- Was sind Constraints?
- Wie kann man im Umfeld von CLP entscheiden, welche Variable als nächstes mit einem Wert belegt werden soll? (First-Fail-Prinzip)

Fazit:
Ich habe Prof. Beierle als sehr angenehmen Prüfer empfunden, wenn ich auch nicht jede seiner Fragen sofort richtig verstanden habe und die detaillierten Rückfragen teilweise durchaus anspruchsvoll fand. Recht schwierig fand ich auch das “spontane” Programmieren von Hand, da kaum Zeit blieb, über die eigentliche Implementierung nachzudenken und ich das Gefühl hatte, sofort nach Aufgabenstellung losschreiben zu müssen. Ich konnte nicht jede der Fragen sauber beantworten und habe auch die gewünschten Programme nicht alle auf Anhieb richtig programmiert, daher gab es ein wenig Abzug, dennoch fand ich die Benotung mit 1.7 durchaus fair.
Zur Vorbereitung auf die Prüfung empfehle ich, die hier in den Protokollen aufgeführten Fragen gut zu üben (insbesondere die Fragen zur Auswertung in Prolog und Scheme). Außerdem sollte man einige “Standard-Programme” schnell und sauber von Hand programmieren können. Hierfür habe ich mir eine Liste von Programmen zur Übung erstellt:

Prolog:

Prädikat member/2: Prüfe, ob ein Element in einer Liste enthalten ist.
Prädikat last/2: Prüfe, ob ein Element das letzte Element einer Liste ist.
Prädikat is_list/1: Prüfe, ob ein Element eine Liste ist.
Prädikat reverse/2: Invertiere eine gegebene Liste
Prädikat sortiert/1: Prüfe, ob eine Liste sortiert ist.
Prädikat append/3: soll erfüllt sein, wenn die drei Argumente Listen sind und das dritte Argument die Aneinanderreihung der ersten beiden Listenargumente ist.
Prädikat member/2 per append
Prädikat last/2 per append
Prädikat ith/3: Prüft, ob das erste Argument, an der Stelle von N in der Liste L steht.
if-then-else mit Cut-Operator

Prädikat length: soll die Länge einer Liste berechnen
Anwendung des fail-Prädikats (alternative Darstellung für \+)
Prädikat drittletztes: Gibt das drittletzte Element einer Liste zurück
Prädikat drittes: Gibt das dritte Element einer Liste zurück
Prädikat hatDrei: Ist erfüllt, wenn die Zahl 3 in einer Liste vorkommt
Prädikat anzahlDrei: Ermittelt das Vorkommen der Zahl 3 in einer Liste.

Scheme:

Funktion average: Berechnung des Durchschnitts zweier Zahlen
Funktion abs: Berechnung des Absolutbetrags einer Zahl
Prädikat atom? mit cond: Testet, ob das ihr übergebene Argument ein Atom ist.
Prädikat atom? mit or: Testet, ob das ihr übergebene Argument ein Atom ist.
Funktion ith: Liefert das i-te Element einer Liste zurück.
Funktion length: Berechnet die Länge einer Liste
Funktion append: Listenkonkatenation (zweistellig)
Funktion produkt: Gibt das Produkt einer Liste von Zahlen zurück
Funktion sum-with: Eine Funktion, die eine andere Funktion zurückgibt.
Funktion dreimal: Gibt eine Funktion zurück, die die übergebene Funktion dreimal anwendet. Dabei sei ein Funktions-Argument vorausgesetzt.
compose: ein Programm, dass die Komposition zweier Funktionen darstellt
n-mal: eine Funktion, die eine Funktion f und eine natürliche Zahl n annimmt und eine Funktion ausgibt, die f n-mal ausführt
filter: eine Funktion, die eine Liste und ein Prädikat als Eingabe erwartet und eine Liste der Elemente aus der Eingabeliste zurückgibt, die das Prädikat nicht erfüllen
insert: Fügt ein Element sortiert in eine Liste ein
Prädikat gerade?: Prüft, ob eine übergebene Zahl gerade ist.
Funktion member: Überprüft, ob ein Element in einer Liste enthalten ist
Funktion last: Überprüft, ob ein Element das letzte Element einer Liste ist.
right-reduce-Funktion für Listen
sum-stream: Berechnet die Summe aller Elemente eines Stroms von Zahlen

Viel Erfolg (und auch ein bisschen Spaß) bei der Prüfung!

Gedächtnisprotokoll

Modul: 01816 Logisches und funktionales Programmieren
Prüfer: Herr Prof. Dr. Christoph Beierle
Beisitzer: Herr Nico Potyka
Datum: 07. Juli 2014

- Was ist bei der logischen Programmierung anders als bei der imperativen Programmierung?
  - Formulierung logischer Zusammenhänge eines Problems
  - Keine Formulierung eines Algorithmus' (Lösungssuche erfolgt durch Prolog)
  - Höhere Abstraktionsebene
  - ...

- Wie wird eine Anfrage (Beispiel-Anfrage: ? - l1, l2, l3) ausgewertet?
  - Resolutionsprinzip erklärt
  - Unifikation erklärt

- Was ist Backtracking?
  - Allgemein beschrieben
  - Im Kontext von Prolog beschrieben

- Prädikat definieren, das erfüllt ist, wenn die Zahl 3 in einer Liste vorkommt.

- Prädikat definieren, das das Vorkommen der Zahl 3 in einer Liste ermittelt.
  - Verwendung des is-Prädikats
  - Verwendung des Cut-Operators

- Was ist der Cut-Operator?

- Welche anderen nichtlogischen Komponenten gibt es noch in Prolog?
  - Beweisstrategie (keine parallele Auswertung von Klauseln)
  - Unifikationsalgorithmus (Verzicht auf Vorkommenstest)
  - is-Prädikat
  - ...

- Was versteht man unter funktionaler Programmierung?
  - Funktionen höherer Ordnung
  - Abschlussobjekte
  - ...

- Funktion höherer Ordnung definieren.
  - Hier habe ich die compose-Funktion definiert

- Funktion definieren, die eine einstellige Funktion und eine Zahl n entgegennimmt und eine Funktion liefert, die die übergebene Funktion n-mal ausführt.

- Das Konzept der Umgebung sollte ich noch erklären.

- Was ist ein Homomorphismus?
    - Strukturerhaltene Abbildung
    - Im Kontext von Strömen erklärt

- Was ist constraint-logische Programmierung?
    - Definieren von Constraints (genauer erklärt - Wertebereichseinschränkung auf Variablen)
    - Aufbau Constraint-System
    - Zusätzliche Komponente "Constraint-Solver" prüft Erfüllbarkeit des Constraint-Systems
    - Vorteile erklärt

- Was ist das First-Fail-Prinzip?

Gedächtnisprotokoll zur mündlichen Prüfung im Kurs 1816 (Masterprüfung praktische Informatik)

Kurs: Logisches und funktionales Programmieren (01816)
Prüfer: Prof. Beierle
Dauer: ca. 30 Minuten
Datum: Juli 2013

Allgemeines :
Die Atmosphäre der Prüfung war sehr angenehm. Ich habe sowohl Herrn Prof. Beierle als auch seinen Beisitzer als sehr freundlich empfunden. Herr Beierle ist als Prüfer wirklich zu empfehlen.
Es lohnt sich typische Programmierbeispiele im Kopf zu haben, damit man sich nicht alles in der Prüfung herleiten muss. Allerdings gibt Herr Beierle durchaus auch Hilfestellung, wenn er sieht, dass man Probleme bei einer Prädikats- bzw. Funktionsherleitung hat.
Anhand der Beispiele fragt Herr Beierle natürlich dann noch weiter in Tiefe.
Mir schien es, als ob es besonders wichtig ist, die Auswertungsstrategien sowohl für Prolog als auch Scheme gut erklären zu können, d.h. was sowohl Prolog als auch Scheme mit einer Anfrage macht.

Die folgenden Fragen sind in etwa in der Reihenfolge niedergeschrieben, in der sie auch in der Prüfung auftauchten. Es sind jedoch nur die, die mir noch in Erinnerung geblieben sind. Es waren auf jeden Fall noch mehr, aber die grobe Richtung sollte aus den Fragen erkennbar sein. Wenn Antworten angegeben sind, dann auch nur stichpunktartig, da ich mich nicht genau erinnere, was ich gesagt habe. Die Antworten muss man sich ohnehin auf jeden Fall selber erarbeiten, ein bloßes daherbeten ist nicht ratsam!

Gestellte Fragen:

- Was bedeutet logische Programmierung, wie funktioniert ein logisches Programm?

- Wie wertet Prolog Anfragen aus? z.B. ?- L1, L2, L3.
(Klauselkopfsuche, Beweis von rechter Seite, Backtracking, Variablenbindung)

- Schreiben Sie ein Prädikat, das das dritte Element einer Liste ausgibt
(Habe eine Lösung mit append geschrieben. Die einfache Lösung drittes([A,B,C|R],C). wäre sicher auch super gewesen)

- Prädikat, das das ite Element einer Liste ausgibt.
(Wieder Lösung mit append aufgeschrieben, “is” kam vor)

- Erläutern Sie das Prädikat is
(nicht-logische Komponente von Prolog; erwartet vollständig instantiierten Ausdruck auf der rechten Seite; beweisbar, wenn 1. Argument mit dem 2. unifiziert werden kann, 1. Argument typischerweise Variable)

- Welche weiteren nicht-logischen Komponenten gibt es in Prolog?
(Beweisstrategie, zyklische Terme wegen fehlendem Vorkommenstest, Cut-operator. Prädikat not hatte ich vergessen, daher wohl gleich folgende Frage)

- Wie ist denn die Negation in Prolog realisiert?
(\+ aussage; beweisbar, wenn aussage nicht beweisbar und umgekehrt)

- Können Sie “not” anders aufschreiben?
(mit cut und fail (siehe Kurstext), erklärt, was da genau passiert)

Übergang : Kommen wir zur funktionalen Programmierung!

- Was ist/bedeutet funktionale Programmierung?
(Basis : mathematisches Funktionenmodell, Programme bestehen aus Funktionen, die Ergebnisse aufnehmen und übergeben, so in etwa)

- Da wir es vorhin in Prolog hatten, wie schreibe ich eine Funktion, die das ite Element einer Liste ausgibt?

- Schreiben Sie eine Funktion auf, die eine Funktion f als Parameter hat und eine Funktion zurückgibt, die f zweimal ausführt.
(compose erwähnt, aber letztendlich einfach (define (zweimal f) (lambda(x) (f (f x)))) geschrieben)

- Schreiben Sie eine Funktion, die eine Funktion f und eine natürliche Zahl n annimmt und eine Funktion ausgibt, die f n-mal ausführt.
(rekursive Struktur zusammen mit Herrn Beierle erarbeitet)

- Wie wird (e0, e1, …. , en) ausgewertet?
(Viel von Auswertung der einzelnen e1 bis en erzählt, Suche nach Bindungen in der Umgebung etc. und Anwendung von e0 auf e1,..,en. Am Ende aber wohl wichtig, dass die ausgewerteten e1 bis en an die formalen Parameter von e0 gebunden werden, im neuen Rahmen wohl auch erwähnt)

- Warum macht man das mit den Rahmen, Umgebungen, Abschlussobjekten etc.?
(statisches Binden)

- Wie können logische und funktionale Programmierung zusammen geführt werden?

- Was versteht man unter CLP?
(Beschränkung von Wertebereichen für Variablen. Das ist natürlich nur ein kleiner Teil von CLP! Siehe Kurstext)

- Was muss bei der CLP(FD) im Vergleich zur herkömmlichen logischen Programmierung geändert werden?
(Ich habe vom geänderten Unifikationsalgorithmus erzählt)

- Was ändert sich im Unifikationsalgorithmus?
(Unifikation von Domainvariablen, haben wir dann noch etwas vertieft)

Ende.

Gedächtnisprotokoll zur mündlichen Prüfung (nicht Klausurersatzgespräch) im Kurs 1816

Kurs: Logisches und funktionales Programmieren (01816)
Prüfer: Prof. Beierle
Dauer: ca. 45 Minuten
Datum: Mai 2012
Note: 1,3

Allgemeines: Zur Prüfungsvorbereitung habe ich mich ausschließlich auf das Skript bezogen. Weitere Literatur habe ich nicht benutzt. Unklarheiten (insbesondere auch größere Verständnisprobleme) zum Stoff konnten unter Rückgriff auf verschiedene Foren im Netz meistens schnell gelöst werden.
Meine Prüfung begann ca. 40 Minuten früher als geplant, da ich etwas früher am Lehrgebiet eingetroffen bin (diese Tatsache dürfte auch die etwas längere Prüfungsdauer erklären). Die längere Prüfungsdauer hat sich nicht nachteilig auf das Ergebnis ausgewirkt, insgesamt ist die Zeit auch schnell vergangen, da die Prüfung in einer sehr angenehmen und umgänglichen Atmosphäre stattgefunden hat. Wenn ich an einer Stelle stecken geblieben bin, hat Herr Beierle kleinere Hinweise zu Hilfestellung gegeben.
Mit den beiden verwendeten Programmiersprachen (Prolog und Scheme) so wie den Konzepten der logischen und funktionalen Programmierung bin ich im Rahmen dieses Kurses zu ersten Mal in Berührung gekommen. Dass ich während der Prüfung gelegentlich die Notationen verwechselt habe (z.B. in einem Scheme-Programm für eine Liste die Notation von Prolog [] verwendet) wurde mir auch nicht weiter angekreidet.
Meine Prüfung bestand aus einer ausgewogenen Kombination von Abfragen der theoretischen Grundlagen und dem aufschreiben von Codebeispielen (bei letzterem hatte ich einige Schwächen).

Prüfunsverlauf: Anbei versuche ich den Prüfungsverlauf so zu skizzieren, wie ich ihn noch im Gedächtnis habe. Bei den Antworten sind nur die wichtigsten Stichpunkte angegeben, die ich jeweils noch etwas ausformuliert habe.

F: Mit welchem Thema möchten Sie beginnen?
A: Mit der logischen Programmierung

F: Woraus besteht ein Prolog-Programm?
A: Aus Klauseln, also Fakten und Regeln. Regeln als umgekehrte Wenn-Dann-Beziehung. Fakten als Regeln ohne Voraussetzung. In der logischen Programmierung wir eher das WAS als das WIE beschrieben.

F: Angenommen es liegt ein Programm mit der Klausel `h(x) :- l1, l2, … , ln` vor. Wie wird die Anfrage `?- h(x)` ausgewertet?
A: Diese Frage scheint eine Standardfrage zu sein. Hier legte Herr Beierle großen Wert auf die Beschreibung des allgemeinen Resolutionsprinzips, der Unifikation und der Erwähnung der Tatsache, dass die Anfrage an die rechte Seite der Klausel gebunden wird.

F: Können Sie eine Klausel definieren, die die Länge einer Liste zurückgibt?
A: Die Definition wie in KE2 S. 71 aufgeschrieben

```
ue length([ ],0).
ue length([E|R],N) :-
     ue length(R,NR),
     N is NR+1.
```

F: Können Sie etwas zu dem Prädikat `is` sagen?
A: Es handelt sich um eine partielles Prädikat, dies bedeutet, dass die rechte Seit ein vollständig instantiierter arithmetischer Ausdruck sein muss.

F: Welche typischen Programmiertechniken gibt es in Prolog?
A: Musterorientierte Programmierung. Hierbei werden die Klauselköpfe so definiert, das immer nur eine Klausel ausgewertet werden kann → Effizienzgewinn.
Programmieren mit Akkumulatoren. Diese bestehen immer aus (mindestens) drei Argumenten. Ein Argument für die Eingabe, ein weiteres für das Ergebnis und eine drittes in dem das Ergebnis schrittweise aufgebaut wird.

F: Können Sie ein Klausel die einen Akkumulator nutzt definieren?
A. Ein Beispiel hierzu wäre ein Prädikat, dass eine Liste annimmt und die gespiegelte Liste zurückgibt. Entspricht dem Prädikat `reverse` in KE1 S.41. Diese Prädikat habe ich dann auch aufgeschrieben.

```
reverse(L,A) :- rev(L,[ ], A).
rev([ ],A,A).
rev([H|L],R,A) :- rev(L,[H|R],A).
```

F: Eine weiter typische Programmiertechnik ist die relationale Programmierung. Was versteht man darunter?
A: Bei einer Anfrage ist nicht festgelegt welche Argumente mit einem Grundterm belegt sein müssen. Dadurch steht immer auch die Umkehrfunktion zur Verfügung. Beispiel `append`, mit dem eine Liste auch in zwei Teillisten aufgespalten werden kann.

F: Können Sie das Prädikat `Not` erläutern.
A: Für den Beweis des Literals \+ X versucht Prolog erst einen Beweis für das Literal X zu finden. Wenn es hierfür einen Beweis findet, kann \+ X nicht beweisen werden.

F: Können Sie eine alternative Darstellung für `Not` angeben?
A: Aufgeschrieben und erläutert: s. KE2 S. 66

```
unverheiratet(S) :- verheiratet(S), !, fail.
unverheiratet(S).
```

F: Nun wechseln wir zum Thema funktionale Programmierung. Was können Sie dazu sagen?
A: Die Grundlage der funktionalen Programmierung sind Symbole und mathematische Funktionen. Es stehen Funktionen höhere Ordnung zur Verfügung, das sind Funktion, die Funktionen als Ein- und Ausgabe erwarten.

F: Wie wird ein Symbol ausgewertet?
A: Der Wert eines Symbols wird in der Umgebung gesucht. Noch kurz erläutert was eine Umgebung ist (mit den Begriffen Rahmen, Umgebung und Umgebungsspeicher)

F: Können Sie einen Term definieren
A: Mittels define. Aufgeschrieben `(define symbol wert).`

F: Können Sie eine Funktion definieren?
A: Aufgeschrieben: `(define (square x) (* x x))`

F: Wie wird eine Anfrage der Form `(e0, e1, … en)` ausgewertet?
A: Bei der Auswertung von `(e0 e1 … en)` werden erst alle Elemente in der Liste ausgewertet, erst dann wird das erste Listenobjekt, `e0` (muss ein Funktionsobjekt sein), als Funktion interpretiert und aufgerufen. Die Stelligkeit der Funktion muss mit den n Attributen `e1` bis `en` übereinstimmen.

F: Was liefert, eine Anfrage `?- (define e square)` zurück?
A: Ein Abschlussobjekt. Gerade als ich ansetzen wollte etwas mehr zum Abschlussobjekt zu erläutern kam auch schon die offensichtliche Frage

F: Was ist ein Abschlussobjekt?
A: Neben dem Symbol und der eigentlichen Funktionsbeschreibung trägt ein Funktionsobjekt immer die Information mit sich in welcher Umgeben sie definiert wurde.

F: Können Sie eine Funktion definieren, die sowohl eine Funktion als Argument erwartet und eine Funktion als Ergebnis liefert.
A: Da ich bereits die Funktion `square` definiert hatte, eine Funktion, die `square` zweimal hintereinander ausführt. Aufgeschrieben:

```
(define (power4 f)
     (lambda (x) (f (f x))))
```

F: Können Sie eine Funktion definieren die eine Liste und ein Prädikat als Eingabe erwartet und eine Liste der Elemente aus der Eingabeliste zurückgibt, die das Prädikat nicht erfüllen.
A: Dies ist eine Filterfunktion. Leider hatte ich Schwierigkeiten den Code vgl. KE4 S.188 zu notieren, da ich mir diese Funktion nicht eingeprägt hatte.

Die Themen Constraints und Integration von logischem und funktionalem Programmieren waren nicht Gegenstand meiner Prüfung.
Insgesamt kann ich sowohl den Kurs als auch den Prüfer empfehlen. Teilnehmer die sich bisher hauptsächlich mit der imperativen Programmierung beschäftigt haben, finden in diesem Kurs einige interessante Ansätze. Jedoch kann die Masse der Konzepte und der neuen Sichtweise unter Umständen im ersten Augenblick erdrückend wirken.

Gedächtnisprotokoll
Kurs: Logisches und funktionales Programmieren (01816)
Prüfer: Prof. Beierle
Dauer: ca. 25 Minuten
Datum: Februar 2012
Note: 1,0 (mündliche Fachprüfung)

Herr Beierle ist ein freundlicher und ruhiger Prüfer, der zu beiden Themen der funktionalen und der logischen Programmierung sowohl Konzepte abfragt als auch konkrete Code-Beispiele mit Pen-and-Paper-Implementierung verlangt. Es lohnt sich deshalb auch sehr die Einsendeaufgaben zu bearbeiten, die ich auch immer wieder im Verlauf der Prüfung als Referenzbeispiele zur Sprache brachte. Wer die Aufgaben selbstständig bearbeitet, kann die im Vergleich eher leichteren Beispiele in der Prüfung sicherlich problemlos lösen. Beide Themengebiete nehmen dabei in etwa gleich viel Raum ein. Wichtig ist bei der Theorie insbesondere, dass das Fachvokabular sitzt. Die Fragen selbst sind nach meinem Empfinden klar formuliert und aus den Prüfungsprotokollen lässt sich ein allgemeiner Ablauf der Prüfung gut erahnen. Gerade die Einstiegsfragen unterscheiden sich kaum von Prüfung zu Prüfung. Offensichtlich ist diese in der Reihenfolge der Fragen auch stark entlang der Themenfolge im Kurstext orientiert. Wer zu einem Thema übrigens viel zu erzählen hat, darf dies auch tun – somit kann man sich selbst einen kleinen Schwerpunkt setzen. Ich kann übrigens den Kurs als solchen auch sehr empfehlen, falls jemand sich noch nicht zu einer Prüfung entschieden hat. Sympathisch finde ich auch, dass im Anschluss an die Prüfung ein kurzes Feedbackgespräch stattfindet und ich noch Fragen zu Inhalten stellen konnte, was ich auch kurz tat.

Die folgenden Codebeispiele werden in der Prüfung wohl regelmäßig abgefragt. In meiner Prüfung musste ich allerdings zusätzlich davon abweichende Beispiele herleiten. Man muss also auf jeden Fall die Konzepte verstanden haben. Auswendiglernen nutzt wenig, da keine reine Reproduktion verlangt wird. Der Vollständigkeit wegen seien trotzdem einige Beispiele aufgeführt.
Prolog: x-tes Element einer Liste, ist sortierte Liste?, append, member, reverse
Scheme: append, Funktionskonkatenation

F: Was zeichnet denn logisches Programmieren aus?
A: Deklarative Programmiersprache, die mehr das Was als das Wie beschreibt, insbesondere im Vergleich zur imperativen Programmierung. (Habe ich ausführlicher als hier dargestellt erklärt.)

F: Woraus besteht denn ein Prolog-Programm?
A: Aus einer Menge aus Fakten und Regeln (Klauseln). Dabei beschreibt eine Regel eine „umgekehrte“ Wenn-Dann-Beziehung.

F: Schreiben Sie mal eine Beispiel-Anfrage auf. (Ich schrieb: „?- a(X), b(X), c(X).“ auf.) Wie wertet ein Prolog-Programm denn so eine Anfrage aus?
A: Allgemeines Resolutionsprinzip und damit verbundene Unifikation ausführlich erklärt. Insbesondere war wichtig zu erläutern, dass eine Unifikation von bspw. a(X) mit a(y) auch Konsequenzen für den Rest der Anfrage hat, wenn gleiche Variablen – hier X – benutzt werden. Dazu habe ich auch das Backtracking erläutert. Ich kam dann kurz ins Stocken, als ich erläutert habe, wie die Substitution der rechten Seite einer Regel erfolgt. Ich sprach dabei immer davon, dass bspw. a(X) mit einer zugehörigen Regel unifiziert würde. Herr Beierle war damit nicht ganz zufrieden, da ja nur die linke Seite einer Regel mit einem Anfrage-Literal unifiziert wird, was ich aber nach kurzer Nachfrage korrigiert habe.

F: Was ist der allgemeinste Unifikator?
A: Alle anderen Unifikatoren können durch Konkatenation des allgemeinsten Unifikators mit einem weiteren Unifikator berechnet werden.

F: Definieren Sie ein Prädikat, das das Vorkommen der Zahl 3 in einer Liste wiedergibt.
A: Ich wählte die folgende Definition:

```
drei([],0).
drei([3|R],X) :- !, drei(R,XR), X is XR + 1.
drei([_|R],X) :- drei(R,X).
```

Den Cut-Operator habe ich dabei anschließend eingefügt, und erklärt, wie das Beispiel ohne den Operator funktioniert und wie mit dem Operator das Backtracking auf die falschen „Sekundärlösungen“, bei denen sonst nach einmalig korrekter Ausgabe noch falsche Ergebnisse als alternative Lösungen geliefert würde, beschnitten wird.

F: Ist das denn noch streng logisches Programmieren?
A: Nein, da hier auch unsinnige Lösungen erzwungen werden. Unterschied zwischen deklarativer und operationaler Semantik erläutert. Dazu habe ich das folgende Beispiel gewählt.

```
a :- !, fail.
a.
```

Trivialerweise wäre die Anfrage „?- a.“ in der deklarativen Semantik richtig, operational würde dies aber nie als Lösung rückgegeben. Ich habe das Backtrackingverfahren übrigens immer mit einer Tiefensuche verglichen, da es der Implementierung von Prolog schon sehr nahe kommt und sehr anschaulich ist.

F: Was sind denn andere unlogische Elemente in Prolog?
A: Arithmetik, Prolog-Lösungsverfahren an sich; da nicht vollständig, Programmieren auf beliebigen Termstrukturen, Negation; schon allein, da ja auch mittels Cut realisiert, zyklische Terme.

F: Wie entstehen denn zyklische Terme?
A: Auf Grund des fehlenden Vorkommenstests. „X = f(X)“ sollte beispielsweise nicht unifiziert werden, wird zur Effizienzgewinnung aber in Kauf genommen.“

F: Was ist funktionale Programmierung?
A: Funktionen werden als gleichberechtigter Datentyp akzeptiert. Funktionen können also Gegenstand und Resultat von Berechnungen sein.

F: Schreiben Sie ein Programm, das eine Funktion zurückgibt, die die übergebene Funktion dreimal anwendet. Dabei sei ein Funktions-Argument vorausgesetzt.
A: Ich wählte die folgende Form

```
(define (dreimal f)
        (lambda (x) (f (f (f x)))))
```

Dazu sollte ich dann gleich das Prinzip des Abschlussobjekts erläutern. Wichtig war Herrn Beierle dabei wohl auch, dass ich darauf hingewiesen habe, dass bei der obigen Notation zwei geschachtelte Lambda-Ausdrücke definiert werden, wobei die innere, hier sichtbare, erst dann ein Abschlussobjekt erzeugt, wenn obiges Abschlussobjekt mittels z.B. „(dreimal (lambda (x) x + 1))“ angewandt würde.

F: Was sind denn Rahmen?
A: Eine Tabelle mit Symbolbindungen und einer Folgeumgebung (Ausnahme: Globale Umgebung).

F: Schreiben Sie eine Funktion, die das Produkt einer Liste von Zahlen zurückgibt.
A: Ich schrieb folgendes Beispiel auf.

```
(define (produkt liste)
        (if (null? liste)
                1
                (* (car liste) (produkt (cdr liste)))))
```

F: Was ist denn ein Homomorphismus?
A: Eine stukturerhaltende Abbildung, was ich kurz erklärt habe. Dabei Rechtsreduktion erläutert und erklärt inwiefern eine Standardfunktion mittels Übergabe von „combine“ und „initial“ oben stehende Funktion generieren könnte. Dazu habe ich auch den Nutzen des Konzepts für die funktionale Programmierung erläutert.

F: Was sind denn endliche bzw. unendliche Ströme?
A: Sequentielle Datenspeicher, Prinzip an UNIX-System erläutert, mit Standardein- und -ausgabe. In dem Zusammenhang auch Lazy Evaluation / verzögerten Auswertung erklärt und dazu erläutert, dass bei unendlichen Strömen ein right-reduce mittels Homomorphismen nicht möglich ist, weshalb das force / delay-Konzept hier unabdingbar wird. Dazu den Nutzen für die Gewinnung von Speicherplatz und zur Effizienzsteigerung dargestellt.

F: Was ist logische Programmierung mit Constraints?
A: Constraints sind ein Sprachkonstrukt zur Wertbereichsbeschränkung von Variablen. Prinzip erläutert und als Beispiel der Sinnhaftigkeit das BIER*SEKT-Beispiel aus den Einsendeaufgaben kurz mit generate-and-test und constraint-and-generate verglichen. Prinzip des Constraint-Solvers erläutert und Unifikation als mögliches spezialisiertes Constraint angeführt.

Masterprüfung logische und funktionale Programmierung (1816)

Prüfung: Oktober 2005
Prüfer: Prof. Beierle
Zeit: ca. 30 min

Allgemein:
==========
Prof. Beierle war freundlich und fair, wirkte aber irgendwie nervös. Während der Prüfung wusste ich z.T. nicht so ganz, worauf die Frage abziehlte, und leider konnte ich der Mimik und Gestik von Prof. Beierle nicht ablesen, ob er mit meiner Antwort zufrieden war. Mit dem Ergebnis war ich aber sehr zufrieden :-).

Prolog
======

Was zeichnet logische Programmierung gegenüber der imperativen Programmierung aus? (Basiert auf der Prädikatenlogik 1. Ordnung, deklarativ, nicht deterministisch etc.)

Wie ist ein Prolog-Programm aufgebaut?
(Wissensrepräsentation in Klauseln, d.h. Fakten und Regeln, dann Auswertung des "Wissens" durch Abfragen).

Wenn ich jetzt eine Abfrage (der Form ?- Z1, Z2, Z3) an dieses Prologprogramm stelle, was passiert dann? (Zunächst wird versucht, Z1 zu beweisen. Dazu wird versucht das Literal Z1 mit den Klauselköpfen zu unifizieren. Handelt es sich um ein Faktum, ist der Beweis für Z1 zu Ende, ansonsten wird die rechte Seite der Regel als neue Anfrage bearbeitet. Ist Z1 bewiesen, wird mit Z2 fortgefahren. Die Antwort war Prof. Baierle offensichtlich zu ungenau, deshalb fragte er nach)

Wenn beim Beweis von Z1 eine Variable gebunden wurde, wird dann Z2 einfach so bewiesen? (Nein, die Bindungen, die beim Beweis von Z1 entstanden sind, bleiben auch beim Beweis von Z2 und Z3 gültig.)

Wenn der Beweis von Z3 (nach erfolgreichem Beweis von Z1 und Z2) scheitert, was passiert dann? (Backtracking: Zunächst wird versucht, alternative Beweise für Z2 zu suchen, und mit dieser neuen Variablenbelegung Z3 zu beweisen. Scheitert dies, werden alternative Beweise für Z1 gesucht usw.)

Definition eines Prädikats, das das drittletzte Element einer Liste zurückgibt.

(Meine Lösung:

```
drittletztes([E, _, _], E).
drittletztes([_, F, G | R] , E) :- drittletztes([F, G | R], E).
```

Prof. Beierlein meinte, statt der 2. Klausel wäre

```
drittletztes([_ | R] , E) :- drittletztes( R, E).
```

einfacher, aber meine Lösung war auch o.k. Hatte mich auch bei der 1. Klausel verschrieben, hat er mir aber nicht übel genommen).

Was sind die nicht-logischen Elemente von Prolog?
(Die Sprach-Implementierung, Rechenoperationen, Not, Cut).

Cut genau erklären

(Ich hab ein Beispiel hingeschrieben
```
a(T) :- p(T), !, q(T).
a(T).
```
und dann erklärt, dass Cut das Backtracking einschränkt.)
Wenn jetzt Z2 das Literal a(test) ist, was passiert, wenn p(test) wahr ist, aber q(test) nicht, wie geht der Beweis dann weiter? (Backtracking zu Z1).
Einige Prologschnipsel, man sollte Sagen, was rauskommt, wenn sonst nichts im System definiert ist:

?- 7 is 2 * 5 - 3.
(yes)

?- 5 - 2 is X.
(Fehlermeldung, X ist nicht gebunden)

?- asserta(beispiel(hallo)), beispiel(I).
(I = hallo)

?- write(3+5).
(Ausgabe 3+5, Abfrage liefert yes)

?- L=[a,b,c,d], L=[A|[B|C]], write(C).
(Ausgabe von [c,d], L=[a,b,c,d], A=a, B=b, C=[c,d])

Scheme
======
Was macht eine funktionale Programmiersprache aus?
(mathematische Funktionen, man kann gut geschachtelte Funktionen beschreiben. Hätte noch Funktionen höherer Ordnung und verzögerte Ausführung nennen können, hab ich aber nicht getan, weil gleich die nächste Frage kam)

Wie ist ein funktionales Programm aufgebaut?
(Symbole, Zahlen, Applikationen, Zuweisungen mit define, etc.)

Wie wird so ein Programm ausgewertet?
(Zahlen werden direkt zurückgegeben, Symbole werden der Umgebung nachgeschlagen)
Bei der Auswertung von Symbolen: Wo genau kommt der Wert des Symbols her? (Die Umgebung besteht aus einer Liste verketteter Rahmen. Die Rahmen werden dann nacheinander durchsucht, bis das Symbol gefunden wird, oder der letzte Rahmen durchsucht worden ist (dann wird ein Fehler gemeldet)).

Wie wird eine Applikation ausgewertet?
(Zunächst werden alle Elemente der Liste ausgewertet. Dann wird das erste Objekt in der Liste als Funktion interpretiert und aufgerufen. Dabei wird ein neuer Rahmen erzeugt.)

Was ist lazy Evaluation? Wozu wird die benutzt?
(ich hab delay und force erklärt, Benutzung z.B. für Streams)

Vereinigung von logischen und funktionalen Sprachen

Wie kann man logische und funktionale Elemente zusammen benutzen?
(Nebeneinanderbenutzen in einer Programmiersprache, Mode-Deklaration für Prädikate, Residuation / Narrowing. Residuation und Narrowing hab ich noch kurz erklärt).

Kurs 1816 Logisches und funktionales Programmieren

Prüfungszeit: circa 30min
Prüfer: Herr Prof. Beierle
Beisitzer: Manfred Widera

Wie ist ein logisches Programm aufgebaut?
- Regeln, Fakten,...

Erläutern Sie das an einem Beispiel. Schreiben Sie ein Programm, dass zwischen einer sortierten und einer nicht sortierten Liste unterscheidet.
- Das hatte ich nicht im Kopf, konnte es aber relativ schnell entwickeln. Vergaß allerdings den Spezialfall “leere Liste“.

Wie funktioniert Backtracking?
- Backtracking am Beispiel erklärt. Besonders die Variablenbindungen schienen ihm am Herzen zu liegen.

Welche nichtlogischen Bestandteile gibt es in Prolog?
- Sprachimplementierung, Not, Cut, ...

Erläutern Sie den Cut?
- Cut definiert und an einem Beispiel erklärt.

Tiefergehend wurde nur das Programm besprochen. Die grundlegenden Begriffe (Literal, Klausel,...) sollten klar sein und exakt angewendet werden.

Welche Grundlagen hat die funktionale Programmierung?

- mathematische Fnktionen,...

Wie ist ein funktionales Programm aufgebaut?
- ein Beispiel erläutert

Dann kam eine Frage über Rahmen, Umgebungen und Abschlußobjekte ...

Schreiben Sie ein Programm, dass die Komposition zweier Funktionen darstellt.
- Mit etwas Hilfestellung hab ich das auch hinbekommen.

Was sind Ströme? Wie kann ich sie erzeugen?
- kein Programmbeispiel, nur die Konzepte erläutert

Dann kam eine Frage zu Möglichkeiten der Zusammenführung von funktionalen und logischen Programmiersprachen. Da viel mir nur Narrowing ein, die kompletten Konzepte hatte ich leider nicht im Kopf.

- Was sind Constraints (inkl. FCIR und Unifikation)?
Da hat er mich einfach erzählen lassen. FCIR und Unifikation der Domainvariablen wollte er genau wissen.

Allgemeines:
Herr Prof. Beierle ist ein sehr freundlicher und fairer Prüfer. Die Prüfung bei ihm kann ich uneingeschränkt empfehlen. Er bohrt schon bei einigen Fragen nach, aber wenn er merkt, dass da nicht mehr kommt wechselt er das Thema.
Ich war ziemlich aufgeregt, am Anfang sowieso und bei funktionaler Programmierung, weil mir das Thema nicht so liegt. Einige Male hatte ich Schwierigkeiten zu erkennen, worauf er hinaus wollte, da hab ich einfach erzählt, was ich glaubte, dass er es meinen könnte. Wenn es falsch war, ist er ziemlich schnell eingeschritten und hat die Frage neu formuliert oder präzisiert.
Er hat mich auch ein wenig drumherum erzählen lassen, z.B. warum wurden Constraints eingeführt
oder wo kann ich den Cut einsetzen, obwohl das nicht explizit gefragt war.

Als Lehrmaterialien habe ich die Skripte und zwei Bücher (Programmieren in Prolog & Strukturierung und Interpretation von Computerprogrammen) verwendet. Hausarbeiten habe ich auch fast alle erledigt.

Das Wichtigste ist (denke ich), dass man die Konzepte und Begriffe kennt und erläutern kann. In welcher Form sie abgefragt werden, scheint unterschiedlich zu sein (siehe die anderen Prüfungsprotkolle). Desweiteren sollte man einfache Programme schreiben können. Und zu den anderen Themen (Dateiverarbeitung, Datenbankänderung, zustandsoriente Befehle, Narrowing, ...) sollte man zumindest einige Stichworte im Kopf haben.

Viel Glück bei der Prüfung.

Gedächtnisprotokoll zur Masterprüfung im Fach

1816 - Logische und Funktionale Programmieren.
==========

1. Masterprüfung (Master of Science Fach Informatik), Freiversuch
Prüfungsdatum: 10.03.2004
Note: 3,3
Prüfungsdauer: ca. 30 min.

Prüfungsort:
Im Büro von Herrn Prof. Beierle
Wir saßen an einem kleinen Tisch.
Ich am Kopfende, Prof. Beierle links und der Beisitzer rechts,
der auch das Prüfungsprotokoll schrieb.
Vor mir lag ein Papierblock DIN A4 und zwei Stifte.

Meine Vorbereitung:
==========
Während des Semesters habe ich hauptsächlich die Kurseinheiten (KE) 1 und 2
und teilweise 3 bearbeitet.
Die KE 4 und 5 wurden von mir nicht mehr bearbeitet.
Ca. 7-10 Tage vor der Prüfung habe ich begonnen mich intensiv vorzubereiten
und den restlichen Stoff aufzuholen, was allerdings nicht ganz funktionierte
wie man an der Note sieht.
Ich habe zwar bestanden, aber mit einer schlechteren Note als erhofft.

Meine Schwächen:
==========
- Da dies meine erste mündliche Prüfung war, war ich sehr aufgeregt.
  Ich habe die Nacht zuvor nur ca. 4 Stunden geschlafen und mußte am Tag
  der Prüfung mit dem Auto 300 km nach Hagen gefahren.
  Meine Freundin hat mich zwar chauffiert insgesamt war dies aber zu stressig.
  Ich werde in Zukunft einen Tag vorher anreisen und in Hagen übernachten

- Programmbeispiel habe ich mir während der Prüfung erarbeitet, was sehr viel
  Zeit gekostet hat und beim Akkumulatorbeispiel hängen geblieben bin und
  Prof. Beierle mir einen Tipp geben musste.

Falsch erklärt
- Cut-Operator
- Die Variablenbindung

Verbesserungsvorschläge:
==========
- Beispiele auswendig lernen.
  Man sollte beim Aufschreiben der Programmbeispiel keine Zeit verschwenden,
  da es eher wichtig ist die Konzepte oder Vorgehensweisen der
  Programmiersprachen zu erklären.
  Von daher ist es notwendig die Beipiele aus dem FF zu beherrschen.

- Man sollte vorher 2-3 mal die Prüfungssituation als Rollenspiel durchspielen.
  Hierbei fallen Defizite im Wissen auf, die beim "normalen" gar nicht zu Tage
  treten.

==========
= Prüfungsverlauf =
==========

Fragen während der Prüfung:
(Achtung, die Antworten auf die Fragen müssen nicht stimmen. Bei der Note ;-).

Prolog
======

1) Was ist das besondere an Prolog.
Prolog unterscheidet sich zu imperativen Programmiersprachen, die auf der Von-Neumann Rechnerarchitektur basieren und dort Speicherstellen manipuliert werden durch folgendes:
- Ein Prologprogramm besteht aus Regeln und Fakten
- Fakten sind Tatsachen
- Regeln bilden Beziehung zwichen Objekten ab um daraus neue Tatsachen zu gewinnen.
- Basiert auf der Prädikatenlogik 1. Stufe

2) Beispiel zu einem Prolog Programm. Zum Beispiel last
Ich habe begonnen es aufzuschreiben

```
last([A],A).
last([_|R],A) :- last([R],A).
```

Prof. Beierle zeigte in der Regel auf die rechte Seite, den Teil [R] und bat mich das nochmal genau anzuschauen.
Mir fiel der Flüchtigkeitsfehler auf, der mir unterlaufen war.
[R] muß durch R ersetzt werden.

Ich habe es entsprechend geändert.

```
last([A],A).
last([_|R],A) :- last(R,A).
```

3) Es gibt verschiedene Programmiertechniken in Prolog. Was ist ein Akkumulator?
Zeigen Sie dies am Beispiel von Reverse
- Ich habe begonnen reverse aufzuschreiben

```
reverse( L,  E) :- rev( L, [], E).
rev( [], A, A).
rev( [H|T], A , E) :- rev( ...
```

Hier bekam ich dann einen Hänger und wußte nicht mehr weiter.
Ich sagte noch, dass die Liste beim rekursiven Abstieg abgebaut wird und umgekehrt wieder aufgebaut wird. Prof. Beierle gab mir dann den entsprechenden Hinweis das Programmbeispiel fertig stellen zu können.

```
reverse( L,  E) :- rev( L, [], E).
rev( [], A, A).
rev( [H|T], A , E) :- rev( T, [H|A], E).
```

4a) Wie wertet Prolog eine Anfrage aus?
- Die Anfrage besteht aus Literalen.
- Prolog hält Regeln und Fakten. (--> Klauselmenge)
- Fakten könnenen als Regeln mit leerer rechter Seite aufgefaßt werden.
- In Prolog werden die Literale von links nach rechts durchgearbeitet.
- Das Literal wird mit dem Regelkopf verglichen und Fakten vergl

4b)Was macht der Cut-Operator?
Er grenzt den Suchraum ein.

Scheme
======
5) Was ist das besondere an Lisp/Scheme?
Scheme basiert auf Funktionen.
Scheme ist eine symbolorientierte Sprache.
Werte und Funktionen werden an Symbole gebunden.

6) Bitte schreiben sie als Beispiel die Funktion append auf.

```
(define (append l r e)
```

```
        (if (null? l) r (cons (car l) append((cdr l) r))))
```

Hier habe ich einen Fehler gemacht. Ich war noch auf PROLOG eingestellt und habe ein drittes Argument e als Rückgabewert angegeben. Was natürlich falsch war?
Bei funktionalen Sprachen werden die Ergebnisse natürlich per Funktionsrückgabewert zurückgeliefert.
Prof. Beierle wies mich darauf hin und ich habe das Beispiel entsprechend geändert.

```
(define (append l r)
        (if (null? l) r (cons (car l) append((cdr l) r))))
```

7) Wie wird ein Schemeprogramm ausgewertet?
- Es gibt das Substitutionsmodell bei dem die Symbole aufgelöst werden.
- Umgebungsmodell erklärt.

Bemerkung: hier fehlte mir der Zusammenhang zwischen Substitutionsmodell und Umgebungsmodell.

Herr Professor Beierle wies mich darauf hin, dass wir jetzt schon einige Zeit verbraucht haben und er noch etwas über die Constraint-logische Programmierung wissen möchte.

Constraint logische Programmierung

8) Was ist ein Constraint mit endlichem Bereich?
- Ein endlicher Bereich auch Finite Domäne genannt ist eine nichtleere Menge von Konstanten.
  Diese Domäne kann einer Variablen zugewiesen werden.
  Die Variable nennt sich dann Domänenvariable
  Diese Variable, darf dann nur die Konstanten der Domäne zugeordnet bekommen.

9) Was ändert sich an der Prolog-Beweisstrategie durch die Constraint-logische Programmierung (CLP)?
- Zusätzlich zur Resolution und Unifikation wird ein Constraint-Solver benötigt.

10) Wie unterscheidet sich die Unifikation?
- Die Unifikation wird um den Fall der Domainvariablen erweitert.
  Die Unifikation erwartet als Eingabe zwei Terme.
- Bei der unifikation mit Domainvariablen gibt es drei zusätzliche Fälle

Nachgespräch:

- Prof. Beierle ist es wichtig, dass man erklären kann wie Prolog und Scheme funktioniert.

--> Prolog: Resolution, Unifikation, Substitution, Backtracking
--> Scheme: Wie wertet Scheme Programme aus.

Gedächtnisprotokoll zur Masterprüfung im Fach

1816 - Logische und Funktionale Programmieren.
==========

1. Masterprüfung (Master of Science Fach Informatik),
Wiederholungsprüfung nach bestandenem Freiversuch
Prüfungsdatum: 13.09.2004
Note: 2,3 (im Freiversuch vom 3,3)
Prüfungsdauer: ca. 30 min.

Prüfungsort:
==========
Im Büro von Herrn Professor Beierle
Wir saßen an einem kleinen Tisch.
Ich am Kopfende, Prof. Beierle links und der Beisitzer Herr Dr. Widera rechts, der auch das Prüfungsprotokoll schrieb.
Vor mir lag ein Papierblock DIN A4 und zwei Stifte.

Meine Vorbereitung:
==========
5 Tage Wiederholung von Prolog, Lisp und Constraint Logische Programmierung hauptsächlich:
- Auswendig lernen der Prolog-Beispiele last, member, append, reverse und Scheme Beispiel append.
- Funktionsweisen von Prolog und Scheme
  Bei Prolog: Resolutionsprinzip, Unifikation, Backtracking, ..
  Bei Lisp: Substitutionsprinzip, Umgebungsmodell

==========
= Prüfungsverlauf =
==========

Prolog
======
1) Wie wertet Prolog eine Programm aus?

Prolog besteht aus Regeln und Fakten, der sogenannten Klauselmenge.
Regeln sind Tatsachen. Aus Fakten können anhand der bestehenden Tatsachen neue abgeleitet werden. Mittels einer Anfrage an die Klauselmenge kann eine Aussage auf Gültigkeit überprüft werden.

Zum Beweis der Anfrage wendet Prolog das Allgemeine Resolutionsprinzip an.
Das Allgemeine Resolutionsprinzip ist eine Kombination aus Resolutionsprinzip und Variablenersetzung.

usw.

2) Bitte schreiben sie das Beispiel append auf und erklären daran den Beweis.

```
append([], R, R).
append((H|T), R, [H|Out] ):- append(T, R, Out).
```

3)Welches Ergebnis erzeugt der Aufruf append([], 3, X) ?
Ich versuchte den Aufruf zu beweisen.
Das erste Prädikat passt versuchte die Unifikatiion und sagte das das Ganze nicht unifizierbar ist, da ja an zweiter stelle keine Liste übergeben wurde.
Ich habe die Unifikation aber nochmal durchgespielt und meinte das Prolog als Ergebnis X=3 liefert, was auch richtig war.

Professor Beierle erklärte, dass auf den ersten Blick das Ergebnis etwas merkwürdig erscheint, da aber Prolog nicht typisiert ist, dies die korrekte Lösung sei.

4)Es gibt auch nichtlogische Komponenten in Prolog. Welche kennen Sie?
(Hier wusste ich Prof. Beierle wollte auf den Cut hinaus)

Cut !

5) Was bewirkt der Cut-Operator?
Er grenzt den Suchraum ein.
Anhand des Allgemeinen Beispiels aus den Unterlagen die Wirkungsweise erklärt:

```
p:- q, !, r.
```

q und r ist eine Menge von Klauseln.
1. Ist q nicht beweisbar, dann wird die Klausel vor Aufruf des Cut(!) verlassen. Der Cut hat in diesem Fall keine Wirkung.
2. Ist q beweisbar, dann wird versucht, mit der bestehenden Variablenbindung von q, die erste existierende Lösung zu finden. Wird keine Lösung gefunden, dann wird für p und q keine weitere Klausel ausprobiert.

6) Was ist ein Akkumulator?
Eine Hilfsdatenstruktur in dem Daten zwischengespeichert werden.

7) Schreiben Sie bitte das Beispiel reverse auf und erklären Sie die funktionsweise. Ich habe dann das folgende Beispiel aufgeschrieben und die funktionsweise erklärt.

```
reverse(In, Out):- rev(In, [], Out).
rev([], A, A).
rev([H|T], A, Out):- rev(T, [H|A], Out).
```

8) Warum können in Prolog zyklische Terme entstehen?
Ich habe versucht dies mit dem folgenden Beispiel (siehe Unterlagen S.33) zu erklären:

```
p(a):-p(a).
p(a).

?:- p(a).
```

.. und sagte dazu das Prolog nicht vollständig beweisbar wäre:

Es ging allerdings um das folgende Thema/Beispiel (siehe Unterlagen S.64):

Unifizierung von X und f(X) -> X= f(f(f(f(f(....

Die richtige Antwort wäre somit gewesen:

Bei der Unifikation zweier zyklischer Terme gerät Prolog ohne Vorkommenstest (occurr check) in eine Endlosschleife.

Scheme

9) Wie wird eine Scheme Programm ausgewertet?
Hier habe ich versucht das Umgebungsmodell zu erklären inklusive Rahmen, globale Umgebung, Symbolbindung, Abschlußobjekt.

Es gibt das Umgebungsmodell, ... Rahmen, globale Umgebung, Abschlußobjekt.
Allerdings war ich hier etwas unsicher und habe

10) Wie wird in Scheme eine Symbol aufgelöst?
Ich habe erklärt, dass Scheme versucht im Rahmen der aktuellen Umgebung das Symbol zu finden, falls das Symbol nicht vorhanden ist wird zum nächsten Rahmen gegangen solange bis das Symbol gefunden wurde. Ist das Symbol nicht vorhanden wird eine Fehlermeldung ausgegeben.
Falls das Symbol in verschiedenen Rahemen definiert ist wird immer das zuerst

gefundene verwendet.

11) Schreiben sie bitte folgende Beispielfunktion auf:
Eine Funktion, die zwei Funktionen als Übergabewert erwartet und eine Funktion zurückliefert, in der die übergebenen Funktionen nacheinander ausgeführt werden.

Ich habe versucht die Funktion hinzuschreiben.
Allerdings war ich etwas überfordert und habe dann nach 1-2 Minuten, gesagt dass ich die Lösung nicht weiss und es Zeitverschwendung wäre, wenn ich weiter versuchen würde die Lösung zu finden.

Constraint logische Programmierung

12) Was ist Constraint Logische Programmierung?
    Erweiterung von Prolog um Constraints
    Constraint erklärt.
    Beweisstrategie wird erweitert um Constraint Solver.
    Domänenvariable erklärt.
    Die Unifikation wird erweitert, Wie unterscheidet sich die Unifikation?
    - Die Unifikation wird um den Fall der Domainvariablen erweitert.
    - Bei der Unifikation mit Domainvariablen gibt es drei zusätzliche Fälle, die ich auch erklärt habe.

Kurs 01816 - Logisches und funktionales Programmieren
Mündliche Prüfung für einen Leistungsnachweis
Prüfer: Prof. Dr. Beierle
Prüfungsdauer: ca. 1 Stunde
Datum: Februar 2004

Fragen zu "logisches Programmieren"
---

1. Was ist logisches Programmieren? Erläutern Sie, wie Prolog funktioniert. Allgemeines Resolotionsprinzip (am Besten etwas formal). Einzelne Fragen bezüglich der Unifikation. Backtracking erläutern.

2. Beispiele für Programme in Prolog:
reverse, append, not

3. Nicht logische Komponenten von Prolog nennen.

4. Cut genau erklären. Zunächst in z.B.
p:- q,!,r.
p:- s.
Und dann sehr genau die Backtracking-Schritte erläutern.

5. Not erläutern.

6. Änderung der Datenbank: die Datenbankprädikate nennen und kurz erläutern.

7. Constraints: Was sind Constraints? Welche Inferenzregeln gibt es? Forward Checking erläutern (am Besten etwas formal), die Resolution und die Bindung in diesem Fall erläutern.

Fragen zu "funktionales Programmieren"
---

1. Was ist funktionales Programmieren? Wie funktioniert es?

2. Beispiele für Programme in Scheme:
append,

3. Umgebungen/Bindungen. Ich weiß nicht mehr, wie die Fragen waren, aber man sollte sich über die Konzepte im klaren sein.

4. Funktionen höherer Ordnung. Erläutern, wie dies möglich ist.

5. Ströme. Was sind Ströme? Unendliche Ströme, Memo-Funktion.

Gesamteindruck:

Sehr gut. Prof. Beierle ist sehr nett und ruhig. Besonderen Wert legt Prof. Beierle bei den Grunlagen. Er übertreibt nicht mit der mathematischen Formalität. Man kann alles auch mit einfachen Worten erklären. Trotzdem ist man auf der sicheren Seite, wenn man auch alles formal darstellen kann.

Bei mir ging es nur um einen Leistungsnachweis und da übertreibt man es nicht unbedingt mit dem Lernen. Für die, die eine benotete mündliche Prüfung machen, habe ich einen guten Tipp: Für die ersten Fragen der zwei Themenbereiche ('Was ist logisches Programmieren? Erläutern Sie, wie Prolog funktioniert' und 'Was ist funktionales Programmieren? Wie funktioniert es?') einen kleinen Vortrag bzw. eine lange, ins Detail gehende Antwort vorbereiten.

Außerdem sollte man schon die typische Programmierbeispiele ein wenig üben. Ich kannte sie nicht auswendig, aber ich hatte viele Beispiele geübt. Ich konnte sie also nicht schnell aufschreiben, aber ich durfte laut nachdenken, korrigieren und sogar überprüfen, ob es funktioniert, bevor ich die Lösung als beendet abgegeben habe.

Prof. Beierle ist sehr nett und ruhig. Mich hat sehr beeindruckt, dass er

mich nach der Prüfung gefragt hat, ob ich irgend welche Fragen zum Thema hätte. In der Tat hätte ich einige gehabt, doch nach einer Stunde Prüfung war ich so extrem erschöpft, dass mir keine mehr einfiel. Trotzdem haben wir uns über interessante Themen noch ein wenig unterhalten.

Kurs 01816 - Logisches und funktionales Programmieren
Mündliche Prüfung für einen Leistungsnachweis
Prüfer: Prof. Dr. Beierle
Prüfungsdauer: ca. 30 Minuten
Sommersemester 2002

Zum Prüfer: angenehme Atmosphäre, häufiges Nachfragen, das manchmal zu sehr ins Detail ging. Im Endeffekt konnte deshalb nicht das gesamte Kurs-Spektrum abgefragt werden.

1) Besonderheiten der Sprachen im Kurs?
meine Antwort: nicht zustandsorientiert; möglicherweise waren aber auch noch andere Kriterien gefragt, die dann aber spezifisch für funktionale oder logische Programmiersprachen gewesen wären

2) Beispiel in PROLOG?
„append“ aufschreiben und erklären. Hier hatte ich etwas Schwierigkeiten warm zu werden, weil ich nicht mit konkreten Programm-Beispielen gerechnet hatte.

3) Beweisstrategie?
Resolutionsprinzip, Unifikation, Backtracking; am Beispiel erläutern

4) Programmiertechniken?
musterorientiert, generate-and-test, ...

5) Nichtlogische Komponenten?
„cut“, Arithmetik,... benennen, ein Beispiel aufschreiben

6) Constraints?
forward checking,...

7) Beispiel in LISP?
„append“ aufschreiben und erläutern

8) Ströme?
definieren, verzögerte Auswertung erklären

Gedächtnisprotokoll zur mündlichen Diplomprüfung im Kurs 1816 "logische und funktionale Programmierung"

Prüfungsdatum: 28.1.2002
Prüfer: Prof. Beierle
Note: 1,0
Beisitzer: Gab's auch, Name ist an mir vorbei gegangen.

Fragen:

- Sie kennen ja imperative Programmierung, was ist bei der logischen Programmierung anders?
  Ich habe erst mal spontan "Alles" gesagt und dann ungeordnet aufgezählt: Anderes Berechenbarkeitsmodell, beruht auf der Prädikatenlogik 1. Stufe, Programm besteht aus Klauseln, die nur deklarativ die Bedeutung festlegen, keine globalen Variablen, bei rein logischer Programmierung keine Seiteneffekte
- Wie sieht das konkret aus, z. B. bei einem Programm für ein append?
  Ich habe dann angefangen aufzuschreiben: erst mal als Faktum append(A,[],A). Dann als Regel append([K|R],A,[K|R2]) :- append(R,A,R2). Dabei festgestellt, daß das Faktum nicht paßt, und append([],A,A) noch dadrüber geschrieben und dazu gesagt, daß durch die Rekursion das erste Element abgebaut wird, bis schließlich das Faktum paßt. Er fragte dann noch nach, ob ich die Fallunterscheidung nicht doch wie ursprünglich geplant im zweiten Argument machen könnte. Ich guckte etwas verwirrt, worauf er ir dagte, daß er mich nicht verwirren wolle. Ich fing an zu zeichen, erkläre, daß es mit dem zweiten Argument nicht paßt, weil ich nur ans erste und nicht ans letzte Listenelement käme, worauf er dann das Thema wechselte:
- Es gibt ja auch die nichtlogischen Komponenten in Prolog, welche? Ich sagte "Cut", holte Luft und er fragte weiter: Ja, erklären sie den mal. Daraufhin habe ich dann schnell etwas übers Backtracking erzählt und dann erklärte, daß das Backtracking genau das unterbinde. Er fragte dann nach einem konkreten Beispiel. Mir fiel nichts passendes ein und ich habe dann sinnfreie Prädikate genommen. Ich drückte mich zunächst nicht ganz korrekt aus, weil ich sagte, der Cut verhindere, daß ein Backtracking-Punkt (er sagte wohl "Wahlpunkt") angelegt wird. Er fragte dann nach, wann der Wahlpunkt angelegt wird (bei Betrachtung der Klauselköpfe) und wann der Cut verarbeitet wird (auf der linken Klauselseite), weshalb ich mich dann korrigierte zu "der Cut löscht den Wahlpunkt vom Stack". Dann fragte er noch, was bei mehr alternativen Klauseln passieren würde. Ich sagte dann, daß diese alle nicht mehr ausprobiert werden, wenn der Cut erreicht wird.
- Was ist ein Akkumulator? Eine Programmiertechnik, die eine Hilfsdatenstruktur für Zwischenergebnisse verwendet. Zeigen Sie das doch mal am Beispiel vom reverse. Ich habe dann die reverse-Klauseln wie im Kurstext aufgeschrieben und dazu erklärt, wie wann welche Parameter verwendet werden.
- Was ist funktionale Programmierung? Programmierung mit Funktionen, wobei diese Ein- und Ausgabeparameter von Funktionen sein können, in den Funktionen gibt es dann Funktionsaufrufe, ggf. auch rekursiv, und Fallunterscheidungen
- Wie läuft das ab? Im Interpreter in einer Lesen-Auswerten-Schreiben-Schleife (das

war glaube ich nicht so wichtig), ein Programm ist ein Ausdruck, der vom Interpreter ausgewertet wird. Nachfrage: Wie wertet er aus? Bindung, Rahmen, Umgebung erklärt (das war wohl der wichtige Punkt)

- Was ist ein Abschlußobjekt? Eine Bindung für ein Funktionssymbol zusammen mit einem Verweis auf die Definitionsumgebung. Dadurch wird statische Bindung möglich, die Variablenwerte ergeben sich also aus dem nackten Programmtext und sind von der Aufrufreihenfolge unabhängig.
- Geben Sie doch mal ein Beispiel an - (einen Moment fürchtete ich hier, er wolle ein Beispiel zu statischer und dynamischer Bindung) - z. B. wieder für das append. Ich habe dann das append in Lisp aufgeschrieben. Ich hatte zunächst einen Fehler im null?-Teil des if - Rückgabe war bei mir '(), und ich hatte sogar laut überlegt, ob ich das Quote setzen muß. Er hat mich erst zuende schreiben lassen und dann gesagt, "schauen sie sich das if noch mal an, was soll die Funktion machen?", worauf ich dann noch mal guckte und korrigierte, daß die zweite Eingabeliste Ausgabe sein muß.
- Was sind Streams, und wozu braucht man sie? Verallgemeinerte endliche oder unendliche Listen, Mittel, um komplexe Aufgaben auf mehrere kleinere Programme mit einer einheitlichen Schnittstelle zu verteilen. Diese akzeptieren jeweils einen Stream als Eingabe und geben meist auch einen raus. Wie implementiert man unendliche Streams? Als Paar aus Startwert und einem Funktionsobjekt, daß angibt, wie daraus die weiteren Werte berechnet werden können.
- Im Kurs geht es ja noch um constraintlogische Programmierung, was sind Constraints? Bedingungen, die an die Instantiierung der ansonsten universalen logischen Variablen gestellt werden. Müssen für vollständig instantiierte Eingaben effektiv überprüfbar sein.
- Was ändert sich im Vergleich zu normalen Prolog? Constraint-Solver kommt hinzu, schränkt Suchraum im Vorweg ein. Konkretisierung der Frage: Was ändert sich bei der Unifikation? Unifikation mit Domainvariablen erläutert.
- Er hat dann irgendwie nach den erweiterten Inferenzregeln gefragt, ich weiß nicht mehr, wie. Ich habe dann FCIR erklärt. Anschließend befand er, daß die Zeit um sei.

Gesamteindruck: Sehr gut. Ich habe mich eine knappe halbe Stunde vor dem eigentlichen Prüfungstermin bei seiner Sekretärin gemeldet. Er kam dann gleich, begrüßte mich und fragte, ob wir gleich anfangen können. Er hat dann den Beisitzer geholt, mir die Finanzamtsbescheinigung gegeben, kurz gefragt, ob es um den Kurs logische und funktionale Programmierung gehen soll, und angefangen. Das war zwar nicht großer Smalltalk wie bei einigen Kollegen, war aber ausreichend, damit ich chronisches Nervenbündel mich einigermaßen sammeln konnte.
Für seine Fragen hatte er einen Stapel Karteikarten, auf denen anscheinend sortiert Stichpunkte zum Kurs standen, und hat zu vielleicht etwa jeder dritten eine Frage gestellt. Wie oben hoffentlich herauskommt, fing er zu den drei Bereichen sehr allgemein an und hat dann allmählich konkretere Fragen gestellt. Dabei hat er mich immer erst mal ausreden oder -schreiben lassen und wies anschließend bei Bedarf auf Stellen hin, die falsch oder noch nicht präzise genug waren. Ich habe die Programme natürlich genau aufgeschrieben, ansonsten entweder nur geredet oder bei Bedarf eher halbformal die Definitionen mitgeschrieben, z. B. bei der Unifikation oder beim FCIR, und dazu erzählt,

was die Sachen im einzelnen bedeuten. War offensichtlich so in Ordnung.
Ich hatte vorher befürchtet, daß er mehr Programmbeispiele abfragt, und insbesondere Lisp-Programme für die Standardfunktionen (map, filter, right-reduce,..) geübt. So detailliert war das zwar anscheinend nicht nötig, aber es hat mir in dem Moment massiv geholfen, nicht groß nachzudenken, was und wie ich klammern muß. Stures Definitionslernen scheint überflüssig zu sein, wichtiger ist, für alles mögliche auf die Frage "Wie funktioniert das?" antworten zu können.