Generelles

Meine Hoffnung war dass Fresh K. zur Abwechslung prüft, statt dessen
drehte bei mir wie in der Midterm DJ G (aka. der Verwirrer) die Turntables.
Macht euch nicht verrückt, bei Fehlern erlauben die beiden immer eine Selbstkorrektur
(Oft mit Tipps) und verbuchen den Fehler dann unter Nervosität.
Die Benotung danach ist super fair und lässt Raum für kleine Unfälle.

Ich habe versucht mich so gut es geht an alles zu erinnern, allerdings wählte Gerhard ab einem gewissen Punkt eine out-of-order Abfragereihenfolge und ich kann mich nicht mehr so ganz an alles
erinnern. Wo möglich habe ich nach bestem Wissen und gewissen sortiert damit man eher sieht aus welchen Stellen vom Stoff das ganze stammt.

TL;DR Für SE-Jahrgang 14

Sehr ähnlich zu bisherigen Klausuren. Schelli ist nur sehr lange auf dem Spezifikationskram und KIV rumgeritten, weswegen weniger aus den anderen Themengebieten dran kam. Den Rest der
Themen hat er zu meiner Freude Bogosortiert und dann ausgefragt. Von der Schwierigkeit/den Fragen generell kamen bis auf ein paar Geistesblitze von Gerhard keine grob unterschiedlichen Sachen dran.

Beweisen sollte ich die Korrektheit von MergeSort mit dem Twist, dass diesmal nicht nur Termination
nachgewiesen werden sollte, sondern auch noch das MergeSort tatsächlich richtig sortiert.
Die Situation begab sich wie folgt:

- "Für Heute habe ich mir einen ganz besonderen KIV Beweis ausgedacht [fiese Lache, vgl. was man immer ausm Lautsprecher klappern hört wenn die Prämissen sich einfach nicht schließen wollen in KIV]"~G
- "Da hat sich der Gerhard echt was feines überlegt [herzliches Lachen]"~K
- "Ohje"~Ich

Es stellte sich dann raus dass der Beweis doch wieder genau dem Schema F folgt bis zu der Stelle an der man Subst. Equation verwenden muss (vgl. andere Klausurprotokolle) damit man das
Lemma anwenden kann (Gerhard hat von sich aus darauf hingewiesen als diese Stelle kam, dass ich doch vielleicht mal in die IDE schauen soll was da noch im Kommentar über dem Lemma für Merge steht und überlegen soll wie ich das jetzt auf die Form bringen kann damits anwendbar ist).
Mehr als subst. equation. + apply eben jenes Lemma war nicht notwendig, i.e., der Beweis funktioniert ansonsten identisch zu dem Terminationsbeweis von MergeSort.

TL;DR Für SE-Jahrgang ≥ 15

Begriffe sind beiden an vielen Stellen wichtig. Mindestens die Intuition sollte bei allem da sein, als Erklärung ists auch oft hinreichend.
Schiebt keine Panik, beide sind absolut korrekt und die Benotung ist wirklich fair.

Act 1 - Spezifikation eines Datentypen

Ich sollte Matrizen spezifizieren. DJ G erwies sich wieder als verlässliche Quelle von Verwirrung. Wie gesagt alles kein Drama, lasst euch nicht zu sehr rausbringen. Verwirrung wurde - bei mir zumindest - nicht bestraft.

Beide wollten das generelle Vorgehen zur Spezifizierung von Datentypen hören:

1) Sorten festlegen: matrix, elem, nat; (Letzteres zur Indizierung & Gerhard wollte 'nat' explizit hören)
2) Konstruktoren festlegen:

Erzeugen einer Matrix mit Breite, Höhe und initialem Element für alle Felder (z.B. 0 wenn elem = nat):

make-matrix: nat × nat × elem → matrix;

(Hier hat G sein KIV Specials Mixtape gedropped und wollte noch wissen wie ich in KIV an die Konstante von Elem [vgl. 0 bei nat, ∅ bei set], damit ich immer einfach das reinschreiben kann [oder so ähnlich]. Ehrlich gesagt keine Ahnung was er da von mir wollte, war aber am Ende offensichtlich egal da er nach 2-3 Versuchen einfach davon absah mich weiter zu fragen)

Setzen eines Elements, i.e. m[i][j] = e:

set-elem: matrix × nat × nat × elem → matrix;

(Ich empfehle keine fancy syntax zu probieren wie m[i][j] = e. Zwar hatte das bei mir den lustigen Nebeneffekt dass wir kurz über diverse Programmiersprachen diskutiert haben und woher dies 'Hässliche'~G Notation kommt, aber das Endresultat war dass KIV das ja nicht kann und dafür wurde ich getadelt)

3) Überlegen ob der Datentyp frei ist oder nicht frei. Anhand der Konstruktoren ist ersichtlich, dass
man mit unterschiedlicher Reihenfolge von set-elem(...) Aufrufen identische Matrizen produzieren kann.
D.h. - Überraschung - Datentyp ist nicht frei.

4) Da Matrix nicht frei ist definiert man über ein Extensionalitätsaxiom Äquivalenzklassen,
i.e.,

// Gegeben Variablen m : matrix; x, y : nat; Nicht vergessen zu prüfen ob Zugriffe "legal", i.e., indices nicht größer als Dimensionen der Matrix

matrix-extensionality: m1 = m2 ↔ ∀ x, y .
width(m1) = width(m2)
∧ height(m1) = height(m2)
∧ x < width(m1)
∧ y < height(m1)
∧ getelement(m1, x, y) = getelement(m2, x, y);

5) In Extensionalitätsaxiom vorkommende Prädikate und Funktionen durch 'gutes' Definitionsprinzip definieren

[...] (bedeutet analog zu den anderen/you get the gist ;)

6) Wie hast du jetzt definiert?

[...]

(Da ich im Eifer des Gefechts fälschlicherweise 'Nichtrekursiv' statt 'Strukturell Rekursiv' genannt habe
sollte ich im Anschluss nochmal als walk-of-shame [vgl. https://www.youtube.com/watch?v=9S6P_Zz0Zb0] das
Nichtrekursive Definitionsprinzip detailiert erklären)

7) Sind wir jetzt fertig?

Nein, wir müssen noch zeigen, dass EQ(m1, m2) eine Äquivalenzrelation ist und 'Verträglich', i.e.: m1 = m2 → f(m1) = f(m2)

8) Was gilt jetzt für unsere Spezifikation von Matrix?

```
Konsistent weil nur "gute" Definitionsprinzipien verwendet und Monomorph modulo
unspezifizierter
sachen (was passiert wenn ich out of bounds auf die Matrix zugreife?) und module
des parameters 'elem'
```

9) So jetzt hast du ja schon was von guten Definitionsprinzipien erzählt, was kennst du den noch für welche?

```
Nichtrekursiv (grinsen von El Knappo), Strukturell Rekursiv (breiteres Grinsen
von El Knappo), Wohlfundiert und über Existenz und Eindeutigkeit
```

Act 2 - Noethersch & Wohlfundiert

Es folgte die unerwartete ;) Überleitung:

1) Wohlfundiert ist das interessanteste dieser Prinzipien, was liegt dem ganzen den zugrunde?

```
Noethersche Relationen, z.B. < auf nat
```

2) Was ist der Clou bei der ganzen Sache?

```
Es gibt keine unendlich absteigenden Ketten
```

3) Zeig uns mal eine **unendlich** absteigende Kette

```
... < a3 < a2 < a1 < a0
```

4) Was ist jetzt dieses Wohlfundiert?

```
Argumente der rekursiven Aufrufe nehmen immer noethersch ab, was uns damit
Terminierung garantiert
```

5) Wo bringen uns noethersche Relationen noch was?

```
DL-Regel für while, noethersche Induktion
```

6) Male doch mal die DL-Regel für while auf. "Bitte die interessanteste!"~K

```
(R⟨while⟩) [...]
```

```
(Hier passierte mir ein weiteres Missgeschick, weil ich unterm Erzählen nicht
bemerkt habe,
 dass ich aus Gewohnheit in der mittleren Prämisse im Anthezedens noch ein Γ
 dazugeschrieben habe. Zu allem Übel hab ich auch noch verpeilt dass der
Buchstabe 'Gamma'
 heißt und habe 'Rho' dazu gesagt. Gerhard hat geschaut als ob sein Simplifier
ausversehen
 P≡NP gezeigt hat, Fresh K war ebenfalls sichtlich irritiert.

  * "Zuerstmal ist das kein Rho [verwirrt]"~G
  * "Wie kommst du darauf? [verwirrt]"~K
  * "Öh ich kann kein Altgriechisch"~Ich
  * "Sieht man [grinsend]"~K
  * "Weiterhin mit dem Gamma..."~G
  *  "Oh ups ja das ist jetzt ausversehen passiert, ich wollte die Invariante
eigentlich statt dessen dort haben"~Ich
)
```

7) Jetzt hat sich das Γ da eingeschlichen und du hast es entfernt. Dürfte es dort bleiben?
Und was ist mit Δ?

```
Nein, da beide im Allgemeinen über den Zustand vor der Schleifenausführung reden
darf man die
nicht einfach mitnehmen, da das währenddessen und danach falsch ist.
Ich habe mir jedoch sagen lassen (props an meine Kommilitonen vor mir :D), dass
KIV
nur das wegwirft, was über Variablen redet, die im Schleifenrumpf angetouched
werden.

(An dieser Stelle folgte DJ G's zweites Mixtape, diesmal der KIV-Optimiert-Fast-
Alles Remix
 in der extended Fassung. Mit strahlenden Augen verprasste Schelli bei seinem
Kurzvortrag zu
 meinem Vorteil weitere Prüfungszeit, bis Alex ihn einbremste)
```

Act 3 - Semantik

1) Jetzt haben wir schon ein bisschen vorgegriffen, wie haben wir den in dynamischer Logik die Semantik von unseren Programmen zu definieren versucht?

```
Relationale Semantik
```

2) was ist dort die Idee?

```
Beischreibung der Semantik als einfache Relation
```

3) Vorteil?

```
Benötigt nur sehr einfache Mathematik
```

4) Nachteil?

```
While und Prozeduraufrufe sind - laut Vorlesung - hässlich. Ich persönlich finde
jedoch
es kommt nicht auf die äußeren Werte an sondern das die Semantik korrekt
funktioniert. Im
übrigen haben wir mit natürlicher Semantik exakt wieder das selbe erreicht.

(Letzteres war zugegebenermaßen ein Knieschuss. Alex hat zwar gegrinnst, aber
dann gings los mit Natural Semantics. Erwähne nie was du nicht ausführlich
erklären
willst ...)
```

5) Was ist den die natürliche Semantik?

```
Eine andere Semantikdefinition.
```

6) Weiter?

```
Wir haben definiert: Ein Zustandsübergang ist in der natürlichen Semantik
eines Programms enthalten, wenn es einen **geschlossenen** Ableitungsbaum mit
den Regeln
der natürlichen Semantik gibt.
```

7) wie finden wir so eine Ableitung jetzt?

```
Wir haben uns einen Regeloperator gebaut (An dieser Stelle Rns-Hut Geste machen
wenn ihr Alex lachen sehen wollt), welcher je für eine Menge von Prämissen alle
daraus in einem Schritt mit
Regeln der natürlichen Semantik ableitbaren Konklusionen liefert
```

8) Und was suchen wir dann immer für Rns?

```
Den **kleeeeeinsten** Fixpunkt
```

9) Was ist so ein Fixpunkt?

```
Nochmaliges Anwenden von Rns liefert wieder identische Menge, i.e., 'nix neues
mehr'
```

10) Wie finden wir jetzt solche Fixpunkte?

```
Knaster-Tarski oder Kleene, ersteres ist hübsch aber nicht gut um damit zu
beweisen,
zweiteres leider nicht immer verwendbar aber dafür besser zum Beweisen geeignet
```

11) Erklär (aus Zeitmangel) mal schnell Knaster-Tarski

```
Wenn f (=Rns) monoton ist der durchschnitt von allen Fixpunkten = kleinster
Fixpunkt
```

12) Was heißt Monoton?

```
Regeln haben nur positive Prämisen
```

13) Schreib mal Kleene hin

```
μ(Rns) = ∪(n ∈ ℕ) Rns^n(∅)
```

14) Heißt?

```
Aufsammeln von allen Bäumen der Höhe exakt n
```

Act 4 - Abstract Data Types (ADT)

Alex vermerkte hier nochmal explizit, dass das Gerhard's Lieblingsthema ist. Dieser wiederum grinste nur breit

1) Wie ist ein ADT definiert?

```
Tripel von Zuständen, Initialzuständen und Familie von Operationen
```

2) Wie sind Operationen definiert?

```
Kontrakte
```

3) Jaein, wie sind Operationen definiert?

```
Input, Output, Operationsname?
```

4) Jaein, wie sind Operationen definiert?

```
Precondition, Postcondition?

(Keine Ahnung worauf er hinaus wollte aber anscheinend war diese Antwort
richtig)
```

5) Ja eine Möglichkeit, was gibt es noch?

```
Relational und Operational
```

6) Was ist den das gängige Programmiersprachenäquivalent von ADT?

```
Interface
```

7) Was ist die Idee bei einem Refinement?

```
Substitution mit konkreter/anderer Implementierung
```

8) Warum darf ich des machen?

```
Information Hiding erlaubt mir eben genau dass ein äußerer Beobachter bei einem
**korrekten** Refinement nicht merkt dass die Implementierung gewechselt hat,
i.e.,
neue erfüllt die gleichen Kontrakte und produziert die gleichen Beobachtungen
```

9) Was ist eine Beobachtung?

```
**Array** von Trippeln (input, operationsname, output)
```

10) Was ist die Semantik eines ADT?

```
Sem(ADT) = Menge aller Beobachtungen des ADT
```

11) Wann ist ein Refinement korrekt?

```
Sem(CDT) ⊆ Sem(ADT)
```

12) Wie weiße ich das nach?

```
1. Abstraktionsrelation definieren
2. Die drei Kriterien überprüfen

(Vorwärtssimulation und intuitives Verständnis hier wichtig, insbesondere dass
man durch
die Vorwärtssimulation eine schöne [sic!] Stückelung in einzelne st -> st' Paare
kriegt
die man einzeln beweisen/verifizieren/durchkiffen/weiß-der-geier-was kann)
```

Level 5 - Endboss KIV Beweis

Hier sollte ich nicht nur die Termination von MergeSort beweisen, sondern auch die Korrektheit, i.e., ⟨#MergeSort(x,y)⟩(sorted(y) ∧ y = perm(x)). Prinz G hat das ganze erst als Herkules Aufgabe heraufbeschworen, allerdings war das ganze dann nicht schwerer als Blatt 10 2019/2020. I.e.:

1. DL-Heuristiken einstellen, hier wählte ich DL+Case Split.

   (Rückblickend hätte man auch
   waghalsig sein können und DL+Induction probieren, in vielen Fällen geht dann nämlich alles von
   alleine direkt zu [vgl. erwähntes Blatt 10])
2. Noethersche Induction über #x
3. "Abwickeln" des Programms bis zum rekursiven Aufruf, i.e. immer mit rechtsklick schön durch
   ratschen (vgl. cookie clicker) bis wieder⟨#MergeSort(x,y)⟩ unter dem mouse cursor erscheint
4. Apply Induction "so das es passt", i.e., genau die Argumente von x,y wählen die im Sukzedens
   stehen (normalerweise genau der Vorschlag von KIV)
5. G's Ratschläge befolgen und danach apply lemma
6. Skurille Schelli Fragen mit einem gekonnten Roundhouse Kick abschmettern

(Was hat KIV hier automatisch gemacht? -> execute call; folgt eig. immer nach apply induction
und reicht oft damit die Prämisse zu geht)

7. nächsten Ast mit gleichem Verfahren (apply induction -> execute call schließt Prämisse)
8. 50 Situps
9. Zauberspruch sprechen
10. Fertig

Viel Spaß und Erfolg in der Klausur :)