Prüfungsprotokoll Dominik

Gerhard hat fragen gestellt; Generell ist die Prüfung auch bei ihm kein Stress, z.B. hatte ich bei den KIV Fragen manchmal einen Blackout-Moment/Habe Mist erzählt über KIV aber am Ende war das nicht weiter schlimm.

Wenn man kurz davor ist einen Fehler zu begehen merkt man das Alex entweder an (siehe rechts) und kann sich noch fix selbst korrigieren oder Gerhard grätscht rein und verhindert einen Unfall. Trotz das Gerhard sichtlich unerfreut war, dass ich seinen Simpilfier nicht wie die den Alphabetsong auswendig kenne (Es fielen u.a. Sätze wie “Nein falsch das ist nicht der Grund warum der Simplifier das kann”) hat sich das nicht negativ auf die Benotung ausgewirkt. Einziger Kritikpunkt nach Notenbekanntgabe war, dass ich Simplifierregeln vergessen habe einzutragen bei dem KIV Praxisteil und dann zu lange überlegt habe wo ich von Hand klicken muss , da mich verwirrt hat dass das Ding nicht wie bei dem gestrigen KIV Teil einfach zuging nach einem Klick (“Das haben wir doch schon so viel geübt da musst du schneller klicken/Die Simplifierregeln halt eintragen”).

Aussagenlogik:

1. Papierbeweis : $A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$
2. Was hätte der Simplifier gemacht (Simplifier macht keine case distinction!) => automatisch alles zu wegen **kontextabhängiger Simplifikation**

Prädikatenlogik:

1. Regel $(R\exists)$ aufschreiben und Intuition erklären (insbesondere warum man $\exists x \,.\, \varphi$ nicht wegwirft):

$$\frac{\Gamma \implies \Delta, \varphi\{x \mapsto t\}, \exists x \,.\, \varphi}{\Gamma \implies \Delta, \exists x \,.\, \varphi}$$

2. Was können wir mit PL 1. Stufe nicht? Peano hat *Nat* spezifiziert mit PL 2. Stufe, es geht aber nicht *Nat* **monomorph** mit PL 1. Stufe zu spezifizieren (**Skolem; Junkelemente**)
3. Was machen wir also? **Generiertheitsklauseln**
4. Was bedeutet das? **Spezialaxiom:** Elemente meiner Trägermenge werden nur genau durch die möglichen Konstruktorterme gebildet
5. Was machen wir idealerweise noch? **Freie Generiertheit**
6. Was kann das mehr? **Injektivität + Disjunktion** (Semantik/Intuition erklären)
7. Was gibts für freie Typen? **Nat, Liste, Binärbäume, Aufzählungstypen** (Da Stack ≈ Liste nicht als eigenes akzeptiert)

KIV Theorie

Kam überall immer wieder vor und hat den Hauptteil meiner Prüfung gestellt. Hier in einem Gruppiert zur Lesbarkeit

1. Was kann der Simplifier alles (Termersetzung, Äquivalenzregel, …)
2. Termersetzungregel aufschreiben und erklären $\Gamma' \implies \varphi \to t = u$
   2.1. Suche nach Instanz von t
   2.2. Suche ob φ in Antezedens/$\neg\varphi$ in Sukzedenz
   2.3. Versuche $\Gamma \implies \left(\bigwedge \Gamma'\right)\theta, \Delta$ durch **rekursive Simplifieraufrufe** zu beweisen
3. KIV setzt beliebige **Instanzen** ein
4. Was macht KIV mit einer Äquivalenzregel?
5. Apply rewrite lemma?
6. Wie spezifiziert man frei generierte Datentypen in KIV? **Data Specification**
7. Was mache ich noch? **Enrichments**
8. Wie implementiert man spezifizierte Datentypen in Java? Jeder Konstruktor **(z.B. nilp [])** als eine **eigene Klasse** definieren die von der **abstrakten Klasse Liste** erbt
9. Leider habe ich mir lange nicht alle KIV fragen gemerkt, es kamen definitiv noch viel mehr dran. Einfach die KIV Folien gut ansehen am besten, Gerhard kommt zwar hart rüber (“Nein das ist falsch!”) aber führt einen dann doch an der Hand auch zur Lösung hin wenns sein muss

KIV Praxis

1. *intersection(x,y)* auf Listen strukturell rekursiv auf Papier definieren (liefert alle Elemente die sowohl in x als auch in y sind):
   1.1. *intersection-base*: $[] \cap y = []$ (wichtig: rekursive Position muss überall gleich sein)
   1.2. intersection-rec1: $a \in y \to (a + x) \cap y = a + (x \cap y)$
   1.3. intersection-rec2: $a \notin y \to (a + x) \cap y = x \cap y$
2. Beweis in KIV führen für Lemma: $append(x, y) \cap z = append(x \cap z, y \cap z)$
   2.1. Eintragen von Simplifierregeln (nicht wie ich Dödel *inj* vergessen, damit schießt man sich ins Knie)
   2.2. Heuristik anmachen (Ich habe PL Heuristik + Struct. Ind gewählt, hat denen anscheinend gefallen)
   2.3. Strukt. Induktion über **die Variable die links und rechts an rekursiver Position ist** x ist links und rechts an rek. Pos; y nur rechts an rek. Pos => x wählen
   2.4. Fallunterscheidung anhand $a \in y$ oder $a \notin y$ (Einfach intersection-rec1 und intersection-rec2 anwenden mit Rechtsklick)
   2.5. Je nach eingetragenen Simplifierregeln ist jetzt definitiv Schluss, sonst muss man halt den Rest nochmal von Hand anwenden)