# Prüfungsprotokoll

- Automatisierte Programmverifikation (Giesl SS 2003)
- Funktionale Programmierung (Giesl WS 2002/03)
- Model Checking (Katoen WS 2005/06)

Prüfer: Prof. Dr. Jürgen Giesl
Beisitzer: Peter Schneider-Kamp
Datum: 30.01.2007
Dauer: 55 Minuten
Note: 1,0

**APV**

Frage: Was bedeutet, eine Formel ist wahr?

Frage: Definieren Sie die Funktion `append`.
Es war dabei nicht nötig die Definition der Listendatenstruktur anzugeben.

Frage: Wie würde man die Assoziativität von `append` beweisen?
Erstmal die Formel dazu aufgeschrieben:
$\forall$ `append(append(xs, ys), zs) = append(xs, append(ys, zs))`
Prinzipiell erst die Formel mit symbolischer Auswertung zur Normalform auswerten, dann Induktion anwenden. Hier lässt sich die Formel jedoch nicht weiter auswerten.
Frage: Angenommen wir könnten eine Aussage nur mit symbolischer Auswertung beweisen, in welchen Interpreteationen würde die Aussage noch gelten?
In allen, die Modell der definierenden Gleichung sind.
Frage: Welche Arten von Induktion gibt es?
Induktion über Datenstrukturen und Algorithmen.
Frage: Welche würde hier angewendet?
Induktionsheuristik und veränderbare Positionen erklärt. Also Induktion über `append(xs, ys)`.
Frage: Schreiben Sie mal die Induktionsformeln auf.
Dabei wurde auf die korrekte Allquantifizierung geachtet.
Frage: Wie würden sich die Induktionsformeln bei einer Induktion über `xs` ändern?
`ys` wäre dann auch in der Induktionshypothese allquantifiziert. Dies könnte es ermöglichen, dass die Induktionshypothese eher angewendet werden könnte. Trotzdem wird eine andere Induktionsheuristik verwendet.

Noethersche Induktionsformel aufgeschrieben und Beweis skizziert.

Frage: Wir untersuchen hier ja eine funktionale Programmiersprache. Wo liegt der Unterschied zu Haskell?
Hier wird eine innermost Auswertungsstrategie verwendet.

Frage: Lassen sich da auch Induktionsbeweise führen?
Frage: Versuchen Sie mal für das Programm `a(x) = 0 * a(x)` (mit einem nicht striktem $*$) die offensichtlich falsche Aussage `a(x) = 1` zu beweisen.
Der Beweis funktioniert, weil die Induktionshypothese gleich der Induktionshypothese ist.
Frage: Warum funktionieren also keine Induktionsbeweise?
Keine fundierte Induktionsrelation.

Frage: Wie können wir die Terminierung eines Programms zeigen?
Zum Beispiel mit einer LPO oder anderen Simplifikationsordnungen. Oder mit Dependency Pairs.
Frage: Untersuchen Sie damit mal das `append`-Beispiel.
Das Dependecy Pair aufgeschrieben. Dabei auf die nicht notwändige Einführung der Tupelsymbole hingewiesen. Das Constraint erklärt. Dabei auch darauf hingewiesen, wie bei bei anderen Problemen die Usable Rules untersucht werden müssen.

**FP**

Frage: Definieren Sie `concat`.
Frage: Definieren Sie nun `concat` mittels einer higher order Funktion.
```
concat = foldr (++) []
```

Frage: Wie definieren man die Semantik dieser Funktion?
Bei der denotationellen Semantik ist es eine Funktion im Domain. Bei der operationellen Semantik wird ins Lambda-Kalkül übersetzt und ausgewertet.
Frage: Wie ist die denotationelle Semantik dieser Funktion?
Kleinster Fixpunkt der zugehörigen higher order Funktion, die man erhält, wenn man `concat` durch eine Variable ersetzt. Ich habe diese auch aufgeschrieben.

Frage: Wie viele Fixpunkte gibt es?
1
Frage: Geben Sie mal eine Funktion an, die mehrere Fixpunkte besitzt.
$f$ mit $f(\bot) = \bot$ und $f(x) = 1$

Diskussion darüber, dass $f$ mit $f(\bot) = 0$ nicht berechnbar wäre. Sonst wäre auch das

Halteproblem lösbar.
Frage: Wie beweist man, dass die Funktion nicht berechnbar ist?
Indem man zeigt, dass die Funktion nicht stetig ist. Man könnte überprüfen, ob die Definition der Stetigkeit erfüllt ist.
Frage: Gibt es noch einen anderen Weg?
Man zeigt, dass die Funktion nicht monoton ist.
Frage: Zeigen Sie das mal.
Definition von Monotonie aufgeschrieben und anhand einer Kette widerlegt.
Frage: Zeigen Sie nun doch mal, dass die Definition der Stetigkeit verletzt ist.

Frage: Wie wird der Fixpunkt im Lambda-Kalkül realisiert?
`fix = f -> f (fix f)`
Frage: Warum verwendet man nicht `fix f = f (fix f)`?
Die Argumente müssen geschlossene Lamba-Terme sein.
Realisierung des Fixpunktes im puren Lambda-Kalkül: `( x y -> y (x x y)) ( x y -> y (x x y))`
Frage: Warum wird das nicht verwendet?
Es ist nicht korrekt getypt.
Frage: Warum ist es nicht korrekt getypt.
Typinferenzalgorithmus darauf angewendet: Fehler beim Berechnen des mgu.

## MC

Frage: Welche Logiken haben Sie in MC kennengelernt?
Ich habe skizziert, wie $LTL$, $CTL$ und $CTL^*$ zueinander liegen. Ich habe auch erwähnt, dass wir einige Fragmente dieser Logiken kurz kennengelernt haben.

Frage: Wie geht das MC von LTL?
$TS \models \varphi \Leftrightarrow Traces(TS) \subseteq Words(\varphi) \Leftrightarrow Traces(TS) \cap \overline{Words(\varphi)} = \emptyset$
Da es sehr aufwändig ist, das Komplement eines $NBA$ zu konstruieren, nutze: $\overline{Words(\varphi)} = Words(\neg\varphi)$
Also $Traces(TS) \cap \overline{Words(\varphi)} = \emptyset \Leftrightarrow Traces(TS) \cap Words(\neg\varphi) = \emptyset$
Frage: Wie funktioniert die Konstruktion für den Automaten für $Words(\neg\varphi)$?"
Erst $GNBA$ konsturieren, dann daraus $NBA$ konstruieren.
Idee der $GNBA$-Konstruktion erklärt: Expansion des Wortes um Teilformeln, Konstruktion eines $GNBA$ der die Expansionsgesetze erfüllt. $closure(\varphi)$ erklärt. Elementary Sets erwähnt.
Frage: Welche Eigenschaften haben die Elementary Sets?
Logisch konsistent, lokal konsistent, maximal. Es war nicht nötig, die genaue Definition aufzuschreiben.

Dann Transitionsfunktion für eine Until-Teilformel aufgeschrieben. Dabei auch das Expansionsgesetz für Untilaufgeschrieben.
Frage: Wie ist die Komplexität von LTL-MC?
$O(|TS| \cdot 2^{|\varphi|})$ und PSPACE

Frage: Wie überprüft man Fairness bei $LTL$?
$TS \models_{fair} \varphi \Leftrightarrow TS \models fair \rightarrow \varphi$

Frage: Wie funktioniert MC bei $CTL$?
Induktive Überprüfung der Teilformeln, Einführung neuer Labels

Frage: Wie überprüft man Fairness bei $CTL$?
$CTL$-Fairness-Constraint ist einen $LTL$-Formel mit $CTL$-Zustandsformeln
Einführung neuer Labels: $\Box \diamond \phi_i \Rightarrow \Box \diamond \Psi_i \quad \rightsquigarrow \quad \Box \diamond a_i \Rightarrow \Box \diamond b_i$
$\exists \Box true$ zur Berechnung der FairPaths
Untersuchung der SCCs