Theorie der Programmierung I
SS 2012

Leitfaden zur Prüfungsvorbereitung

Die folgenden Fragen stellen einen "Leitfaden" zur Prüfungsvorbereitung dar. Man sollte sich diese Fragen selbst beim Lernen stellen, und sie folglich in der Prüfung beantworten können. Das bedeutet nicht, dass in der Prüfung keine anderen Fragen gestellt werden. Es handelt sich also hier *nicht* um einen vollständigen Fragenkatalog.

Syntax und big step Semantik

- Wie ist die (abstrakte) Syntax unserer Programmiersprache definiert?
- Wie sieht ein big step aus?
- Was versteht man unter einem *gültigen* big step?
- Wie sehen die einzelnen big step Regeln aus und wie sind sie intuitiv zu verstehen? Man sollte die Regeln auf (kleine) Beipiele anwenden können.
- Welche Ausdrücke v kommen als Resultate von big steps in Frage? Wie beweist man das?
- Welcher Parameterübergabe-Mechanismus wird durch unsere big step Regeln beschrieben? Welche alternative Parameterübergabe wurde in der Vorlesung besprochen? Wie muss man die Regeln abändern, um diese alternative Parameterübergabe zu erhalten?
- Was ist der Unterschied zwischen statischer und dynamischer Bindung? Welche von beiden ist in unserer Programmiersprache realisiert?
- Was versteht man unter syntaktischem Zucker und unter abgeleiteten Regeln? Beispiele dazu!

Spracherweiterungen

- Wie führt man Rekursion in die Programmiersprache ein? Das ein oder andere Beispiel für eine rekursive Funktion sollte man parat haben.
- Wie führt man zusätzliche Datenstrukturen (Listen, Tupel) ein?
- Wie führt man exceptions ein und wozu braucht man sie?
- Für all diese Spracherweiterungen sollte man sich die Fragen aus dem letzten Absatz nochmals durch den Kopf gehen lassen: Gelten die alten Antworten noch, gelten sie in abgewandelter Form?

Typsystem

- Welchen Sinn hat ein Typsystem? Warum arbeitet man nicht einfach mit einer ungetypten Sprache?
- Wie sieht ein Typurteil aus? Was versteht man unter einer Typumgebung?
- Wie sehen die einzelnen Typregeln aus? Bei welchen Typregeln wird die Typumgebung verändert?
- Was versteht man unter Typsicherheit und wie beweist man sie? Welche Sätze haben wir in diesem Zusammenhang in der Vorlesung bewiesen?

- Wie beweist man die Typerhaltung beim big step? Welches Beweisprinzip wird dazu verwendet? Bedeutet Typerhaltung schon Typsicherheit? Was muss zum Nachweis der Typsicherheit noch gezeigt werden?
- Was versteht man unter einer Berechnung? Dazu gehört insbesondere die formale Definiton von PCTs und von Übergangsschritten zwischen PCTs.
- Welche 4 Möglichkeiten gibt es für eine Berechnung?
- Kann man unser Typsystem verbessern, d.h. kann man es flexibler gestalten, ohne dass dabei die Typsicherheit verloren geht?
- Gibt es das "optimale" Typsystem, d.h. ein Typsystem, das maximale Flexibilität mit Typsicherheit verbindet?

Algorithmen zur Überprüfung der Wohlgetyptheit

- Liefern uns die Typregeln schon einen Algorithmus zur Überprüfung der Wohlgetyptheit?
- Worin besteht der Unterschied zwischen einfachem "Type Checking" und Typinferenz?
- Was ist die Aufgabe des Unifikations-Algorithmus, d.h. was erhält er als Eingabe und was soll er als Ausgabe liefern?
- Was versteht man unter einer Lösung bzw. einer allgemeinsten Lösung einer Typgleichung oder Typgleichungsmenge?
- Wie sieht der Unifikations-Algorithmus im Einzelnen aus? Wie beweist man seine Korrektheit, insbesondere das Terminieren? In welchen Fällen liefert er die Ausgabe "nicht lösbar"? (Beispiele!)
- Was ist die Aufgabe des Typinferenz-Algorithmus?
- Was versteht man unter einem allgemeinsten Typ eines Ausdrucks? Für einfache Beispiel-Ausdrücke sollte man intuitiv einen allgemeinsten Typ angeben können.
- Wie sieht der Typinferenz-Algorithmus im Einzelnen aus? (Es genügen die interessanten Fälle!)
- Mit welcher Eingabe wird der Typinferenz-Algorithmus gestartet, wenn der Programmierer einen abgeschlossenen Ausdruck e eingibt?

Polymorphie

- Was versteht man unter einer polymorphen Funktion? Erhalten wir allein durch die Einführung von Typvariablen bereits polymorphe Funktionen?
- Wie muss das Typsystem verändert werden um polymorphe Funktionen zu erhalten? Dazu gehört die Definition von polymorphen Typen, von neuen Typumgebungen und Typurteilen und natürlich die Formulierung der neuen Typregeln. Für kleine Beispiele sollte man intuitiv den polymorphen Typ angeben können, der für einen Namen in die Typumgebung eingetragen wird.
- Wie muss der Typinferenz-Algorithmus verändert werden, damit er den neuen Typregeln entspricht? Muss der Unifikations-Algorithmus ebenfalls verändert werden?
- Wie zeigt man, dass die polymorphe Sprache nach wie vor typsicher ist?