Funktionales Programmieren
WS 2014/15

Leitfaden zur Prüfungsvorbereitung

Mit den folgenden Fragen/Themen/Konzepten sollte man sich während der Prüfungsvorbereitung beschäftigen. Insbesondere sollte man die Programmiertechnik soweit beherrschen, dass man zu jedem Programmierkonzept kleine Programmstücke verstehen und auch selbst entwickeln kann.

- Wie unterscheidet sich die (rein) funktionale Programmierung von anderen Programmierparadigmen, insbesondere von der imperativen und objektorientierten Programmierung?
- Welches sind die typischen Merkmale der funktionalen Programmierung? Stichworte: Funktionen als *first class citizens*, höhere Funktionen, Curryfizierung, partielle Applikation, namenlose Funktionen, polymorphe Funktionen.
- Welche Arten der Parameterübergabe kennt man in funktionalen Programmiersprachen? Stichworte: call-by-value und call-by-name, eager und lazy evaluation, strikte und nicht strikte Funktionen.
- Rekursion und tail-Rekursion: Wie kann man eine rekursive in eine tailrekursive Funktion umwandeln? Klappt das immer? Ist es immer sinnvoll? Wie beweist man die Korrektheit bzw. das Terminieren einer rekursiven Funktion?
- Listen und Tupel: Rekursive Funktionen auf Listen, insbesondere höhere Funktionen, foldleft und foldright, pattern matching.
- Typen, Typurteile und Typregeln. Die wichtigsten Typregeln sollte man beherrschen und vor allem intuitiv verstanden haben.
- Was versteht man unter dem allgemeinsten Typ eines Ausdrucks? Den allgemeinsten Typ einer Funktion sollte man intuitiv bestimmen können.
- Was versteht man unter einem polymorphen Typ? Welche Funktionen können einen polymorphen Typ haben? (Stichwort: let-Polymorphie). Wie erkennt man an der Antwort des O'Caml-Interpreters, ob eine polymorphe Funktion vorliegt oder nicht?
- Varianten- und record-Typen. Wozu dienen sie? Was ist das besondere an ihnen? Wie funktioniert hier das pattern matching?
- Exceptions: Wozu dienen sie? Wie wird eine exception deklariert, geworfen, aufgefangen? Zusammenhang mit Variantentypen, pattern matching. Wie passen exceptions ins Typsystem?
- Streams: Was versteht man intuitiv unter einem stream? Wie implementiert man streams und typische Funktionen auf streams?
- Imperative Konzepte: Referenzen, arrays, veränderliche records. Typregeln und intuitive Semantik für die imperativen Konzepte. Wozu braucht man überhaupt imperative Konzepte in der funktionalen Programmierung?
- Modulsystem von O'Caml. Module, Funktoren und Signaturen. Abstrakte Datentypen und damit zusammenhängende Begriffe: Darstellungstyp, Darstellungsinvariante. Unterteilung der Funktionen in Konstruktoren, Operatoren, Observatoren.

- Objekte und Klassen. Typ eines Objekts. Klassentyp. Unterschiede zwischen Objekt- und Klassentyp. Vererbung und late binding. Mehrfach-Vererbung. Virtuelle Klassen. Rekursive Objekttypen. Binäre Methoden.
- Was versteht man unter Typinferenz (im Vergleich zur reinen Typüberprüfung)? Wie arbeitet der Typinferenz-Algorithmus?