Fortgeschrittene Programmierung
WS 20/21

Michael Hanus

8. Februar 2021

Detaillierter Vorlesungsverlauf

2.11. Organsiatorisches;
Nebenläufige Programmierung in Java: grundlegende Begriffe, Synchronisationsproblem, Semaphore, Dining Philosophers

3.11. Klasse `Thread`, Interface `Runnable`, Eigenschaften von Thread-Objekten, Monitor-Konzept, Synchronisation von Threads in Java, `synchronized`-Methoden, `synchronized`-Anweisung, synchronisierte Methoden vs. synchronisierte Anweisungen, synchronisierte Collections, Kommunikation zwischen Threads, `wait`, `notify`, `notifyAll`

9.11. genaue Bedeutung und sinnvolle Benutzung von `wait`, `notify`, `notifyAll`, einelementiger Puffer, Benutzung von Synchronisationsobjekten, Beenden und Unterbrechen von Threads, `InterruptedException`

10.11. Serialisierung von Daten, Idee von RMI, Parameterübertragung, Serverseite, Clientseite, RMI-Registrierung, Probleme bei Client-side Synchronisation mit RMI

16.11. Einführung in die funktionale Programmierung: Variablenbegriff, Programm, Funktionsdefinitionen, Ausdrücke, Beispiel `square`, Beispiele `min`/`fac`, Auswertungsmöglichkeiten, Fibonacci-Zahlen (rekursiv und iterativ)

17.11.: Lokale Definitionen, Layout-Regel, Vorteile lokaler Definitionen; Basisdatentypen, Typannotationen, algebraische Datentypen (Aufzählungstypen, Verbundtypen, gemischte Typen, Listen), Ausgabe von Daten (`deriving Show`), Operatoren

23.11. polymorphe Funktionen (`length`, `(++)`, `last`), Definition polymorpher algebraischer Datentypen, `Maybe` und `maybeLast`, Binärbäume, `String`, `Either`, Tupel (`fst`, `snd`, `zip`, `unzip`)

24.11. Pattern Matching (Patternaufbau, case-Ausdrücke), Guards, Funktionen höherer Ordnung, anynome Funktionen, partielle Applikation, Currying, Sections, Funktion `flip`

30.11. generische Programmierung (`map`, `foldr`, `filter`, `foldl`), Kontrollstrukturen als Funktionen höherer Ordnung (`while`)

1.12. Funktionen als Datenstrukturen (Implementierung von Feldern), wichtige Funktionen höherer Ordnung (Komposition, `curry`/`uncurry`, `const`), Funktionen höherer Ordnung in imperativen Sprachen (Ruby, Java 8)

7.12. Funktionen höherer Ordnung in JavaScript;

Motivation und Struktur von Typklassen, Instanzen, vordefinierte Funktionen in Typklassen, Standardklassen, `deriving`, Klasse `Read` und Funktionen `read` und `reads`;

8.12. Unterschiede bei Auswertungsstrategien, Programmsignatur, Terme, Programm, Termersetzungssystem Substitution, Position, Teilterm, Reduktionsschritt, Normalform, Wertaufruf, Namensaufruf, Reduktionsstrategien (LI, RI, LO, RO, PI, PO), Berechnungsstärke von outermost und parallel outermost

14.12. Rechnen mit unendlichen Datenstrukturen (`from`, `primes`, `fibs`, `repeat`, `iterate`), arithmetische und andere Sequenzen, Typklassen `Enum` und `Bounded`, Lazy Evaluation, Sharing, Graphreduktion, Vorteile von Lazy Evaluation

15.12. List comprehensions, Idee der Ein-/Ausgabe, I/O-Aktionen, Aktionen zur Ein- und Ausgabe, do-Notation, Beispiel Ausgabe von Zwischenergebnissen

4.1. I/O-Aktionen zum Lesen und Schreiben von Dateien, Zeilen einer Datei numerieren, Module, Exportdeklarationen, Importdeklarationen; Transformationen auf Containerstrukturen: `Functor`, `fmap`, `Functor`-Gesetze

5.1. Transformationen mit beliebigen Funktionen: `Applicative`, `pure`, `<*>`, arithmetische Ausdrücke und deren Auswertung, `Applicative`-Gesetze, `Applicative`-Instanzen für Listen und `IO`, effektvolle Berechnungen, Verbesserung der Auswertung arithmetischer Ausdrücke durch monadische Struktur, Klasse `Monad`, `Monad`-Instanz für Listen

11.1. Testen mit QuickCheck: Eigenschaften, `==>`, Referenzimplementierung, Regressionstests, Eingabeklassifikation mit `classify` und `collect`, eigene Definitionen von Testdaten: Klasse `Arbitrary`, `elements`, `choose`, `oneof`, `sized`, `vector`, Fallstudien Peano-Arithmetik, `frequency`

12.1. Datenabstraktion, rationale Zahlen, Abstraktion rationaler Zahlen, abstrakter Datentyp, Rat-ADT, Mengen-ADT, Implementierung und Testen von Mengen, Implementierung von Mengen als ungeordnete Listen

18.1. Implementierung von Mengen als geordnete Listen

Einführung in die Logikprogrammierung: Motivation, Verwandtschaftsbeispiel, Prolog-Programme, Fakten, Regeln, Anfragen

19.1. Prolog-Syntax (Zahlen, Atome, Strukturen, Listen), Variablen, Rechnen mit Listenstrukturen Operatoren, Gleichheit von Termen; Programmiertechnik Aufzählung des Suchraumes (Färben einer Landkarte, Sortieren von Zahlenlisten)

25.1. Programmiertechnik Musterorientierte Wissensrepräsentation (`append`), Verwendung von Relationen, Peano-Zahlen (Definition, Addition, Subtraktion), Rechnen in der Logikprogrammierung: einfaches Resolutionsprinzip, Substitution, Unifikator, mgu

26.1. Disagreement sets, Unifikationsalgorithmus, occur check, Komplexität, allgemeines Resolutionsprinzip (SLD-Resolution), Auswertungsstrategie und SLD-Baum

1.2. Beweisstrategie von Prolog, Endlosschleifen, Negation als Fehlschlag, Probleme der Negation, Cut-Operator, Fallunterscheidung, Arithmetik in Prolog (`is`, Fakultätsfunktion), arithmetische Constraints

2.2. Beispiel Schaltkreisanalyse Beispiel Hypothekenberechnung, Constraint-Programmierung über endlichen Bereichen, allgemeines Vorgehen, send-more-money-Beispiel, 8-Damen-Problem, verbesserte Labeling-Strategien

8.2. Meta-Programmierung: Prädikate höherer Ordnung (`call`, `maplist`), Kapselung des Nichtdeterminismus (`findall`, `bagof`, `setof`), Veränderung der Wissensbasis (`assert`, `retract`), Meta-Interpretierer zur Beweislängenberechnung, Prädikate zur Ein- und Ausgabe von Daten

9.2. Tracing von Prolog-Programmen, Differenzlisten, Definite Clause Grammars, Kurzüberblick zu Multiparadigmen-Sprachen, Einführung in Curry (Verwandtschaftsbeispiel), funktionale Muster (`last`, `perm`)