Prinzipien von Programmiersprachen

Michael Hanus

9. Juli 2015

In dieser Vorlesung werden grundlegende Prinzipien heutiger Programmiersprachen vorgestellt. Dabei steht die praktische Anwendung von Sprachkonzepten zur Lösung von Softwareproblemen im Vordergrund.
Bei der Programmierung kommt es weniger darauf an, irgendein Programm zu schreiben, das eine gegebene Aufgabe löst. Vielmehr muss das Programm so geschrieben sein, dass es verständlich und damit wartbar ist, und es muss auch an neue Anforderungen leicht anpassbar sein. Daher ist es wichtig, die für die Problemstellung geeigneten Programmiersprachen und Sprachkonstrukte zu verwenden. Leider gibt es nicht die für alle Probleme gleich gut geeignete universelle Programmiersprache. Daher ist es wichtig zu wissen, welche Sprachkonzepte für welche Problemstellungen geeignet sind. Diese Vorlesung soll hierzu einen Beitrag leisten, indem ein Überblick über wichtige Sprachkonzepte moderner Programmiersprachen gegeben wird. Dadurch werden die Studierenden in die Lage versetzt, sich einerseits schnell in unbekannte Programmiersprachen einzuarbeiten (da viele Konzepte in den verschiedenen Sprachen immer wieder vorkommen), andererseits sollen sie verschiedene Sprachen und Sprachkonzepte aufgrund ihrer Eignung für ein Softwareproblem kritisch beurteilen können.

Kurzübersicht

1. Einführung (Sprachklassifikation, Paradigmen, Sprachaspekte)
2. Grundlagen (Syntax- und Semantikbeschreibung, abstrakte Datentypen, Ausdrücke, Deklarationen, Bindungen, Blockstruktur)
3. Imperative Sprachen (Variablen, Datentypen, Kontrollabstraktion, Ausnahmebehandlung, Prozeduren)
4. Sprachmechanismen zur Programmierung im Großen (Module, Schnittstellen, Objekte, Vererbung, Generizität)
5. Funktionale Programmiersprachen (referentielle Transparenz, Funktionen höherer Ordnung, Auswertungsstrategien, Typsysteme)
6. Logische Programmiersprachen (Resolution, Unifikation, Constraints, Datenbanksprachen)
7. Sprachkonzepte zur nebenläufigen und parallelen Programmierung (Synchronisationskonzepte, Semaphore, Monitore, Message passing, Linda, CCP)

Die begleitenden praktischen Übungen werden in den Sprachen Java, Haskell und Prolog durchgeführt.

Detaillierter Vorlesungsverlauf

14.4.: Einführung: Programmiersprache, Klassifikation, Programmierparadigmen, Elemente und Aspekte von Programmiersprachen

16.4.: Grundlagen: Syntaxbeschreibung (BNF, EBNF, Mehrdeutigkeit, Syntaxdiagramme), Semantikbeschreibung (Kurzüberblick, strukturierte operationale Semantik für eine einfache imperative Sprache), Inferenzsysteme

21.4.: Natürliche Semantik für Ausdrücke, abstrakte Datentypen (Rechnen mit Gleichungen, Konstruktoren), Parametrisierte ADTs, ADTs in Programmiersprachen

23.4.: Bindungen (statisch, dynamisch), getypt, Seiteneffekt, Gültigkeitsbereich, Block, Sichtbarkeitsregeln, lexikalische/dynamische Bindung
Imperative Programmiersprachen: Variablen (Referenz, Typ, Werte), Operationales Modell für imperative Sprachen (Umgebung, Speicher)

28.4.: L-/R-Werte, Zuweisung, Konstanten, Typkompatibilität und -konvertierung; Standarddatentypen: Grundtypen (Aufzählungstypen, Ausschnittstypen), zusammengesetzte/strukturierte Typen, Felder (Deklaration, Erzeugung)

30.4.: Feldzugriff auf Elemente, Zeichenketten, Verbunde, Vereinigungstypen (variante Records), Mengen, Zeiger (Semantik von Dereferenzierung/Adressoperator), Zeigerarithmetik, Listen; Kontrollabstraktion: Sequenz, bedingte Anweisungen

5.5.: Semantik bedingter Anweisungen, Schleifen; Prozeduren: Deklaration und Aufruf

7.5.: Behandlung indirekter Rekursion, Parameterübergabe (call by value, call by result, call by value/result), Parameterübergabe (call by reference, call by name, Jensen-Trick), Typäquivalenz (Namensgleichheit $\leftrightarrow$ Strukturgleichheit), Prozeduren als Parameter, Einordnung in Programmiersprachen

12.5.: Ausnahmebehandlung: Auslösen von Ausnahmen, resumption vs. termination model, Ausnahmebehandlung in Ada und Java
Sprachmechanismen zur Programmierung im Großen: Module, Schnittstellen, modulare Programmierung

19.5.: Klassen und Objekte: Attribute, Methoden, Sichtbarkeit, operationale Semantik (Klassendeklaration, Objektdeklaration, Objekterzeugung), Konstruktoren, statische Attribute und Methoden, Konstanten (final), Aufrufmethode `main`

21.5.: rein/gemischt OO Sprachen, Sichtbarkeit von Attributen, Vererbung, überschreiben vs. überladen, Modul- vs. Typsichtweise der Vererbung, Konformität, Polymorphie, dynamische Bindung

26.5.: Konstruktoren (Überladung), Abstrakte Klassen (Benchmark-Beispiel), Interfaces (Table-Beispiel), Generizität, Pakete; Motivation der funktionalen Programmierung

28.5.: Funktionale Programmierung: referentielle Transparenz $\leftrightarrow$ Seiteneffekte, flexible Auswertungsreihenfolge, einfache Funktionsdefinitionen in Haskell Syntax von Haskell, Funktionsdefinition, Fallunterscheidung, Muster, elementare und algebraische Datentypen, Funktionsdefinition mit Mustern, case-Ausdrücke

Operationale Semantik: Grundbegriffe (Ausdruck, Position, Teilausdruck, Ersetzung, Substitution, Ersetzungsschritt), Berechnung

2.6.: Auswertungsstrategien (innermost $\leftrightarrow$ outermost, lazy $\leftrightarrow$ strikt), Konfluenz, Kopfnormalform, Modularität durch lazy Auswertung, let-Ausdrücke und flache Form von Ausdrücken, Launchbury-Semantik für lazy evalution

4.6.: Erkennung unendlicher Schleifen mit Launchbury-Semantik, Funktionen höherer Ordnung am Beispiel von insertion sort, Typkorrektheit, ad-hoc und parametrischer Polymorphismus, polymorphe Typsysteme, Typausdrücke

9.6.: Typprüfung a la Hindley/Milner Grenzen der Typinferenz (Beispiel); Funktionale Konstrukte in imperativen Sprachen am Beispiel von Scala

11.6.: Logische Programmierung: Einführung, Syntax, Verwandtschaftsbeispiel Zusammenhang funktionale/logische Programmierung, Flexibilität der logischen Programmierung, Resolutionsprinzip, Unifikation, mgu

18.6.: mgu-Berechnung nach Martelli/Montanari SLD-Resolution (Korrektheit und Vollständigkeit), SLD-Ableitungen, Formalisierung der Backtracking-Strategie von Prolog, Beispiel hierzu

23.6.: Erweiterungen: Negation, SLDNF-Resolution, flexible Berechnungsregeln (Und/Oder-Parallelismus), Constraints, CLP(R) (Beispiel: Hypothekenberechnung), CLP(FD) (Beispiel: send+more=money)

25.6.: CLP(X), Constraint-Struktur CLP(X)-Resolution, Datenbanksprachen, Sprachkonzepte zur nebenläufigen und verteilten Programmierung: Grundbegriffe (Prozess, Thread, nebenläufiges, verteiltes und paralleles System), Probleme der Nebenläufigkeit, gegenseitiger Ausschluss, kritischer Bereich, Sperren (locks)

30.6.: Verklemmung (Deadlock), Blockade (Livelock), Fairness, Busy waiting
Sprachkonzepte: Semaphore, Monitore (Concurrent Pascal), Rendezvous (Ada)

2.7.: Nachrichtenaustausch (symmetrisch/asymmetrisch, synchron/asynchron), Prozedurfernaufruf, Gruppenkommunikation, Zusammenfassung Synchronisationsmethoden, nebenläufige Programmierung in Java: Thread, synchronized, wait/notify, Pufferbeispiel
Grundidee der Tupelräume

7.7.: Grundoperationen im Linda-Modell, Beispiele („Ping-Pong“, parallele Worker) Linda-Beispiele (Dining Philosophers, Client-Server), Konzept der nebenläufigen logische Programmierung (CCP), Committed Choice, CCP-Konzept, Maximum-Agent CCP-Beispiel faires Mischen, Client/Server-Kommunikation über Datenströme

9.7.: Nebenläufige funktionale Programmierung: Concurrent Haskell (`forkIO` und `MVar`), Erlang (Kommunikation, Codeaustausch)
Ausblick