

Prinzipien von Programmiersprachen

Michael Hanus

23. Juni 2020

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

- 6.4.: Einführung:** Programmiersprache, Klassifikation, Programmierparadigmen, Elemente und Aspekte von Programmiersprachen
- 9.4.: Grundlagen:** Syntaxbeschreibung (BNF, EBNF, Mehrdeutigkeit, Syntaxdiagramme), Semantikbeschreibung (Kurzüberblick, strukturierte operationale Semantik für eine einfache imperative Sprache), Inferenzsysteme
- 16.4.:** Natürliche Semantik für Ausdrücke, Implementierung in Prolog, denotationelle Semantik für Ausdrücke und Anweisungen, Implementierung in Haskell
- 20.4.:** Abstrakte Datentypen (Rechnen mit Gleichungen, Konstruktoren), Parametrisierte ADTs, ADTs in Programmiersprachen; Bindungen (statisch, dynamisch), getypt, Seiteneffekt, Gültigkeitsbereich, Block, Sichtbarkeitsregeln, lexikalische/dynamische Bindung
- 23.4.: Imperative Programmiersprachen:** Variablen (mutierbare, nicht mutierbare), Operationales Modell für imperative Sprachen (Umgebung, Speicher), L-/R-Werte, Zuweisung, Konstanten, Typkompatibilität und -konvertierung; Kontrollabstraktion: Sequenz, bedingte Anweisungen, dangling else, Semantik bedingter Anweisungen, Schleifen
- 27.4.:** Standarddatentypen: Grundtypen (Aufzählungstypen, Ausschnittstypen), zusammengesetzte/strukturierte Typen, Felder (Deklaration, Erzeugung), Feldzugriff auf Elemente, Zeichenketten, Verbunde, Vereinigungstypen (variante Records), Mengen, Listen
- 30.4.:** Zeiger (Semantik von Dereferenzierung/Adressoperator), Zeigerarithmetik, Listen; Prozeduren: Deklaration und Aufruf, 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
- 4.5.:** Speicherverwaltung (Stack, Heap, Garbage Collection), Besitzerprinzip von Rust (bei Variablen, Funktionen, Referenzen), Vermeidung von dangling pointers in Rust; Ausnahmebehandlung: Auslösen von Ausnahmen, resumption vs. termination model, Ausnahmebehandlung in Ada und Java
- 7.5.: Sprachmechanismen zur Programmierung im Großen:** Module, Schnittstellen, modulare Programmierung; Klassen und Objekte: Attribute, Methoden, Sichtbarkeit, operationale Semantik (Klassendeklaration, Objektdeklaration, Objekterzeugung), Konstruktoren, statische Attribute und Methoden, Konstanten (final), Aufrufmethode `main`, rein/gemischt OO Sprachen
- 11.5.:** Sichtbarkeit von Attributen, Vererbung, überschreiben vs. überladen, Modul- vs. Typsichtweise der Vererbung, Konformität, Polymorphie, dynamische Bindung, Konstruktoren (Überladung) Abstrakte Klassen (Benchmark-Beispiel), Interfaces (Table-Beispiel), Generizität, Pakete
- 14.5.:** Motivation der **funktionalen Programmierung:** Probleme der imperativen Programmierung, Seiteneffekt, referentielle Transparenz, flexible Auswertungsreihenfolge, Quicksort-Beispiel Einfache Funktionsdefinitionen in Haskell Syntax von Haskell, Funktionsdefinition,

Fallunterscheidung, Muster, elementare und algebraische Datentypen, Funktionsdefinition mit Mustern, case-Ausdrücke

- 18.5.:** Operationale Semantik: Grundbegriffe (Ausdruck, Position, Teilausdruck, Ersetzung, Substitution, Ersetzungsschritt), Berechnung, Auswertungsstrategien (innermost \leftrightarrow outermost, nicht-strikt \leftrightarrow strikt), Konfluenz, Kopfnormalform, Modularität durch lazy Auswertung, let-Ausdrücke und flache Form von Ausdrücken, Launchbury-Semantik für lazy evaluation
- 25.5.:** Erkennung unendlicher Schleifen mit Launchbury-Semantik, Typkorrektheit, ad-hoc und parametrischer Polymorphismus, polymorphe Typsysteme, Typausdrücke, Typprüfung a la Hindley/Milner
- 28.5.:** Grenzen der Typinferenz (Beispiele); Funktionale Konstrukte in imperativen Sprachen am Beispiel von Scala (`val/val`, generische Klassen, Funktionen höherer Ordnung, Parameterübergabe call-by-name, Pattern Matching, case-Klassen)
Logische Programmierung: Einführung, Syntax, Verwandtschaftsbeispiel, Zusammenhang funktionale/logische Programmierung Flexibilität der logischen Programmierung (`append`, `last`)
- 4.6.:** Einfaches Resolutionsprinzip, Unifikation, mgu, mgu-Berechnung nach Martelli/Montanari SLD-Resolution (Korrektheit und Vollständigkeit), SLD-Ableitungen, Formalisierung der Backtracking-Strategie von Prolog Beispiel hierzu
- 8.6.:** Erweiterungen: Negation, SLDNF-Resolution, flexible Berechnungsregeln (Und/Oder-Parallelismus), Constraints, CLP(R), Beispiel: Hypothekenberechnung CLP(FD) (Beispiel: `BASE+BALL=GAMES`)
- 11.6.:** Fakultät mit CLP(FD), 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), Verklemmung (Deadlock), Blockade (Livelock), Fairness, Busy waiting
- 15.6.:** Sprachkonzepte: Semaphore, Monitore (Concurrent Pascal), Rendezvous (Ada), Nachrichtenaustausch (symmetrisch/asymmetrisch, synchron/asynchron), Prozedurfernaufruf, Gruppenkommunikation, Prozesse und Nachrichtenaustausch in der Programmiersprache Go, Beispiel: pythagoreische Tripel in Go
- 18.6.:** Zusammenfassung Synchronisationsmethoden, nebenläufige Programmierung in Java: Thread, synchronized, wait/notify, Pufferbeispiel; Grundidee der Tupelräume, Grundoperationen im Linda-Modell, Beispiele („Ping-Pong“, parallele Worker) Linda-Beispiele (Dining Philosophers, Client-Server)
- 22.6.:** Konzept der nebenläufigen logische Programmierung (CCP), Committed Choice, CCP-Konzept, Maximum-Agent CCP-Beispiel faires Mischen, Client/Server-Kommunikation über Datenströme
Nebenläufige funktionale Programmierung: Concurrent Haskell (`forkIO` und `MVar`), Erlang (Prozesserzeugung, Kommunikation, Codeaustausch)
- 25.6.:** Aktor-Modell, Erlang als Aktor-Modell (Datenbankserver), Futures (in Java), Software Transactional Memory, STM-Bibliothek von Haskell, Ausblick