

# Multiparadigmen- Programmiersprachen

Martin Grabmüller

`magr@cs.tu-berlin.de`

Fachgebiet Übersetzerbau und Programmiersprachen  
Fakultät IV – Elektrotechnik und Informatik  
Technische Universität Berlin

# Übersicht

- Einführung
- Paradigmen und Paradigmen-Kombinationen
- Beispiele
- Syntax und Semantik
- Ausblick

# Einführung

# Definitionen

**Definition 1** Ein Paradigma ist ein Musterbeispiel.

# Definitionen

**Definition 1** Ein Paradigma ist ein Musterbeispiel.

**Definition 2** Ein Programmierparadigma ist eine Sichtweise, die zur Lösung eines Problems mittels einer Programmiersprache eingenommen wird.

# Definitionen

**Definition 1** Ein Paradigma ist ein Musterbeispiel.

**Definition 2** Ein Programmierparadigma ist eine Sichtweise, die zur Lösung eines Problems mittels einer Programmiersprache eingenommen wird.

**Definition 3** Eine Multiparadigmen- Programmiersprache ist eine Programmiersprache, die zur Problemlösung mit verschiedenen Programmierparadigmen die geeigneten Sprachmittel besitzt.

# Programmierparadigmen

- Problemstellung  
(unabhängig von einem Paradigma)

# Programmierparadigmen

- Problemstellung  
(unabhängig von einem Paradigma)
- Sichtweise auf ein Problem  
(abhängig)

# Programmierparadigmen

- Problemstellung  
(unabhängig von einem Paradigma)
- Sichtweise auf ein Problem  
(abhängig)
- Damit verbundene Lösungsstrategie  
(abhängig)

# Programmierparadigmen

- Problemstellung  
(unabhängig von einem Paradigma)
- Sichtweise auf ein Problem  
(abhängig)
- Damit verbundene Lösungsstrategie  
(abhängig)
- Beispiel: Listentraversierung
  - imperativ: Schleife
  - funktional: Menge rekursiver Gleichungen
  - logisch: Menge rekursiver Prädikate

# Paradigmen und Paradigmen-Kombinationen

# Verschiedene Paradigmen

- Imperativ
- Funktional
- Logisch
- Constraint-basiert
- Objekt-orientiert
- Nebenläufig
- Verteilt
- Aspekt-orientiert
- Intentional
- Generativ
- Literate Programming
- ...

# Paradigmen-Kombinationen

- Funktional-logisch
- Constraint-logisch
- Constraint-imperativ
- Funktional-imperativ
- Funktional-logisch-imperativ-objekt-orientiert
- ...

# Beispiele

# Beispiele

- Funktional-logische Programmierung mit Curry
- Multiparadigmatische Programmierung mit GED

# Curry

- Programm besteht aus Regeln der Form  
 $f p_1 \dots p_n = t \Leftarrow C$

# Curry

- Programm besteht aus Regeln der Form  
 $f p_1 \dots p_n = t \Leftarrow C$
- Regelauswahl durch Unifikation

# Curry

- Programm besteht aus Regeln der Form  
 $f p_1 \dots p_n = t \Leftarrow C$
- Regelauswahl durch Unifikation
- $C$  kann freie Variable enthalten

# Curry

- Programm besteht aus Regeln der Form  
 $f p_1 \dots p_n = t \Leftarrow C$
- Regelauswahl durch Unifikation
- $C$  kann freie Variable enthalten
- Vorteile: Effizienz funktionaler Sprachen +  
Suchmöglichkeiten logischer Sprachen

# Curry – Auswertungsstrategien

`append [] L = L`

`append [E | R] L = [E | append R L]`

# Curry – Auswertungsstrategien

append [] L = L

append [E | R] L = [E | append R L]

append X [3,4] == [1,2,3,4]  $\Rightarrow$  X=[1,2]

# Curry – Auswertungsstrategien

`append [] L = L`

`append [E | R] L = [E | append R L]`

`append X [3,4] == [1,2,3,4] ⇒ X=[1,2]`

---

`eval append 1:rigid`

# Curry – Auswertungsstrategien

`append [] L = L`

`append [E | R] L = [E | append R L]`

`append X [3,4] == [1,2,3,4] ⇒ X=[1,2]`

---

`eval append 1:rigid`

`append X [3,4] == [1,2,3,4] ⇒ delayed`

# GED

- Entwurf als Multiparadigmen-Sprache

# GED

- Entwurf als Multiparadigmen-Sprache
- Imperativ, funktional, logisch und objekt-orientiert

# GED

- Entwurf als Multiparadigmen-Sprache
- Imperativ, funktional, logisch und objekt-orientiert
- Vor allem in der Lehre benutzt

# GED – imperativ

```
func delete(x, list) {  
    local [a:[]];  
    foreach i in list do  
        if (i != x) then  
            a := a | [i];  
        end  
    end;  
    a;  
}.
```

□ Schleife, Zuweisung

# GED – funktional

```
func delete(x, list)
  if list then
    if (x = head(list)) then
      delete(x, tail(list))
    else
      [head(list)] | delete(x, tail(list))
    end
  end
end.
```

□ Rekursion, Listen

# GED – logisch

```
func delete(x, list) {  
    local [temp];  
    list[temp], temp != x -> temp  
}
```

□ Nichtdeterminismus

# GED – objekt-orientiert

```
addtype (MyList, List, local[list], #, #).
```

```
func MyList init(me, x:[])
```

```
    list := x.
```

```
func MyList add(me, x)
```

```
    list := list | [x].
```

```
func MyList delete(me, x)
```

```
    list := list[<x] | list[>x].
```

- Klassen, Vererbung, Methoden, Kombination mit anderen Paradigmen

# Syntax und Semantik

# Syntax

Syntax muss Nutzung aller Paradigmen erlauben.

# Syntax

Syntax muss Nutzung aller Paradigmen erlauben.

- Einfache Syntax, die alle Konzepte einschließt

# Syntax

Syntax muss Nutzung aller Paradigmen erlauben.

- Einfache Syntax, die alle Konzepte einschließt
- Unterschiedliche Syntax für unterschiedliche Paradigmen

# Syntax

Syntax muss Nutzung aller Paradigmen erlauben.

- Einfache Syntax, die alle Konzepte einschließt
- Unterschiedliche Syntax für unterschiedliche Paradigmen

Vorteile bei beiden Ansätzen: Geschmacksache?

# Integration deklarativer Sprachen

- Constraint-logisch, funktional-logisch
- Erfolgreich, theoretisch sauber
- Implementiert (CLP, Curry, Escher, AKL, ...)

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz
  2. Deklarative Semantik

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz
  2. Deklarative Semantik
- Lösungen:

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz
  2. Deklarative Semantik
- Lösungen:
  1. Ignorieren

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz
  2. Deklarative Semantik
- Lösungen:
  1. Ignorieren
  2. Seiteneffekte in die Typisierung aufnehmen

# Integration deklarativer und imperativer Sprachen

- Zustandsfreie und zustandsbehaftete Berechnungen
- Zerstört nützliche Eigenschaften deklarativer Sprachen
  1. Referentielle Transparenz
  2. Deklarative Semantik
- Lösungen:
  1. Ignorieren
  2. Seiteneffekte in die Typisierung aufnehmen
  3. Getrennte Teilsprachen

# Seiteneffekte und Typisierung

- Monadische Seiteneffekte
  - Seiteneffektbehaftete Aktionen als Werte
  - Übergeordneter Mechanismus führt Aktionen aus

# Seiteneffekte und Typisierung

- Monadische Seiteneffekte
  - Seiteneffektbehaftete Aktionen als Werte
  - Übergeordneter Mechanismus führt Aktionen aus
- Effektsysteme
  - Seiteneffekte werden in Typisierung aufgenommen
  - Effekte werden inferiert
  - Lokal begrenzte Effekte können maskiert werden

# Getrennte Teilsprachen

- Berechnungssprache  
sequentiell, ausdrucksbasiert, nebeneffektfrei

# Getrennte Teilsprachen

- Berechnungssprache  
sequentiell, ausdrucksbasiert, nebeneffektfrei
- Koordinationssprache  
erzeugt Berechnungen, ermöglicht Kommunikation

# Getrennte Teilsprachen

- Berechnungssprache  
sequentiell, ausdrucksbasiert, nebeneffektfrei
- Koordinationssprache  
erzeugt Berechnungen, ermöglicht Kommunikation
- Goffin: funktionale Berechnungssprache,  
nebenläufige Constraints als Koordinationssprache

# Getrennte Teilsprachen

- Berechnungssprache  
sequentiell, ausdrucksbasiert, nebeneffektfrei
- Koordinationssprache  
erzeugt Berechnungen, ermöglicht Kommunikation
- Goffin: funktionale Berechnungssprache,  
nebenläufige Constraints als Koordinationssprache
- Linda: beliebige Berechnungssprache,  
Tuple Spaces zur Koordination

# Ausblick

# Weitere Arbeiten

- Untersuchung verschiedener Kalküle für imperative Sprachen

# Weitere Arbeiten

- Untersuchung verschiedener Kalküle für imperative Sprachen
- Untersuchung von Effektsystemen

# Ausblick

- Unterschiede zwischen deklarativen und imperativen Sprachen
- Ähnlichkeiten zwischen deklarativen und imperativen Sprachen
- Definition einer einfachen, integrierenden Semantik
- Definition einer unterstützenden Syntax

# Ausblick

- Unterschiede zwischen deklarativen und imperativen Sprachen
- Ähnlichkeiten zwischen deklarativen und imperativen Sprachen
- Definition einer einfachen, integrierenden Semantik
- Definition einer unterstützenden Syntax
- ... und natürlich eine Implementierung

**Vielen Dank**