

Was können Computeralgebrasysteme von funktionaler Programmierung lernen?

Henning Thielemann

Martin-Luther-Universität Halle-Wittenberg

2006-10-06

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung
 - Funktionalanalysis
 - Theoretische Korrektheit
- 4 Bedarfsauswertung
 - Reelle Zahlen
 - Potenzreihen
- 5 Typisierung
- 6 Ende

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung
- 4 Bedarfsauswertung
- 5 Typisierung
- 6 Ende

Was Computeralgebrasysteme heute können

- Vereinfachen
- Faktorisieren
- Gleichungen lösen
- Summieren
- Grenzwerte bestimmen
- Integrieren
- Differentialgleichungen lösen

Was Computeralgebrasysteme nicht oder kaum können

- Stochastik:

$$\text{normal } v_0 \ m_0 + \text{normal } v_1 \ m_1 \longrightarrow \text{normal } (v_0+v_1) \ (m_0+m_1)$$

- explizite Formeln für rekursive Folgen
(außer Summenrekursionen):

$$\mathbf{iterate} \ (x^*) \ 1 \longrightarrow \mathbf{map} \ (x^{\wedge}) \ [0..]$$

- Funktionalanalysis

$$\text{integrate } \mathbf{cos} \longrightarrow \mathbf{sin}$$

- Nichtstandardanalysis

$$\text{standardPart} \ ((f(x+\text{eps})-f(x)) / \text{eps}) \longrightarrow \text{derive } f \ x$$

- unendliche Datenstrukturen (Bedarfsauswertung)
- Typisierung

- 1 Überblick
- 2 Grundlagen**
- 3 Funktionen höherer Ordnung
- 4 Bedarfsauswertung
- 5 Typisierung
- 6 Ende

Mathematische Objekte

Mathematisches Objekt:

(bei Grundlegung der Mathematik über die Mengenlehre)

Alles was sich als Menge darstellen lässt.

Beispiele

- natürliche Zahlen: PEANO-Zahlen

$$0 \hat{=} \{\}, 1 \hat{=} \{0\}, 2 \hat{=} \{0, 1\}$$

- gebrochene Zahlen:

Äquivalenzklassen von Paaren ganzer Zahlen

$$\frac{2}{3} \hat{=} \{(2, 3), (4, 6), (6, 9), \dots\}$$

- reelle Zahlen: DEDEKINDSche Schnitte

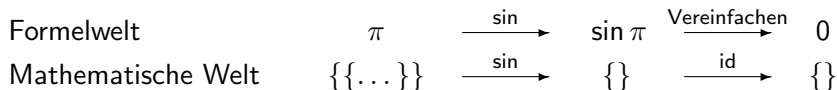
- Funktionen: rechtseindeutige Relationen

$$\sin \hat{=} \{(0, 0), (\frac{\pi}{6}, \frac{1}{2}), (\frac{\pi}{2}, 1), \dots\}$$

Bezeichner für mathematische Objekte

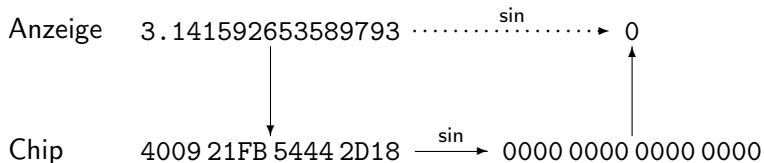
- Zahlenkonstanten: $2, 0.3, i, \pi$
- Funktionskonstanten: $+, \sin$
- Variablen: n, x, f

Parallelwelten



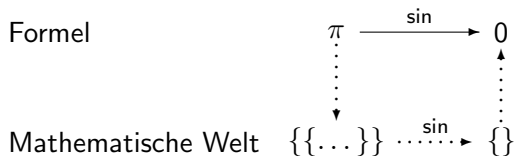
- Formeln - endliche Größe
- Mathematische Welt enthält Unendlichkeiten
- Ähnlichkeit zur Programmierung:
 Programmtext vs. ausführbares Programm
 ausführbares Programm enthält keine Variablennamen

Taschenrechner



- Taschenrechner durchläuft eigene mathematische Welt
- Nur eine Art Objekt: Gleitkommazahl
- „Taschenrechnerprinzip“ (idealisiert):
 - Eingabe Ausdruck
 - Ausgabe (einfacher) Ausdruck für gleiches Objekt

Computeralgebra



Computeralgebra

- arbeitet auf Formelebene
- kann viele Aufgaben exakt lösen
- kann noch mehr Aufgaben gar nicht lösen

Entspricht Computeralgebrafunktion
immer mathematischer Funktion?

Computeralgebrafunktion `diff`

Papier $\frac{d}{dx} \ln x$

MuPAD `diff(ln(x), x)`

- `diff(ln(x), 2)` hat keinen Sinn.
- `diff` benötigt von zweitem Argument nicht Variablenwert, sondern Variable selbst
- `diff` ist keine mathematische Funktion!
- Es sei denn, Variablen sind selbst mathematische Objekte.
- Was bedeutet `subs(diff(ln(x), x), x=2)` ?
⇒ Schachtelung von CA-Funktionen nicht assoziativ

Integrationsverweigerer

Wenn geschlossen integrierbar \rightarrow alles in Butter

$$\begin{aligned} & \text{subs}(\text{int}(\cos(x), x), x=2) \\ & \quad \rightarrow \quad \sin(2) \end{aligned}$$

Wenn nicht geschlossen integrierbar \rightarrow Unsinn

$$\begin{aligned} & \text{subs}(\text{int}(\sin(\sin(x)), x), x=2) \\ & \quad \rightarrow \quad \text{int}(\sin(\sin(2)), 2) \end{aligned}$$

Auswege

Axiom: neue Integrationsvariable einführen

Arbeit mit Funktionen

Prüfsteine für mathematische Funktion

- Angabe einer Wertetabelle
(Graph der Funktion)
- Angabe eines Typen
- Implementation in (nicht-reflexiver) Programmiersprache
(Compilersprache ohne selbstmodifizierende Programme,
Variablenabfrage zur Laufzeit unmöglich)

Taschenrechnerprinzip bei Computeralgebra

- Wünschenswert?
- Machbar?

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung**
- 4 Bedarfsauswertung
- 5 Typisierung
- 6 Ende

Operatoren und Funktionale

- Funktionen sind mathematische Objekte
- Funktionen höherer Ordnung:
Argument oder Wert ist Funktion
- Beispiele: Operatoren, Funktionale (Funktionalanalysis)
- konkret: Integration, Ableitung, Grenzwertbildung, Norm

Viele Nichtfunktionen in CAS ...

Viele wichtige CA-Funktionen
sind nicht mathematische Funktionen.

- `simplify` (aber nahe an der Identitätsfunktion)
- `solve`
- `sum`
- `limit`
- `diff` (im Gegensatz zu `D`)
- `series`
- `0`
- `integrate`
- `pdesolve`

... könnten mathematische Funktionen sein

```
simplify      :: a -> a
solve         :: (a -> Bool) -> Set a
sum         :: Sequence a -> a
limit        :: Sequence a -> a
diff         :: (a -> a) -> (a -> a)
series       :: (a -> a) -> Series a
asympBound  :: (a -> a) -> Set (a -> a)
integrate   :: (a -> a) -> (a -> a)
```

Beispiele

- algebraische Gleichung: `solve (\x -> x == 1 + 1/x)`
- Differentialgleichung: `solve (\y -> y == 2*id * diff y)`

Vorteile mathematischer Funktionen

Mathematische Funktionen

- haben keine Seiteneffekte,
- können Argument oder Ergebnis anderer Funktionen sein,
- können nach Typen systematisiert werden,
- reduzieren Notwendigkeit von `hold` und `evaluate`.
- Funktionsanwendung ist assoziativ.

$$\begin{aligned} f(g(h(x))) &= (f \circ g)(h(x)) \\ &= f((g \circ h)(x)) \end{aligned}$$

Taschenrechnerprinzip bei Differentiation

Eine Funktion, drei Schreibweisen

- $x \mapsto x \cdot x$
- $x \mapsto x^2$
- $y \mapsto y^2$

Drei Wege, ein Ergebnis

$$\text{Produktregel: } D(x \mapsto x \cdot x) = x \mapsto 1 \cdot x + x \cdot 1$$

$$\text{Potenzregel: } D(x \mapsto x^2) = x \mapsto 2 \cdot x$$

$$D(y \mapsto y^2) = y \mapsto 2 \cdot y$$

Gleiche Eingabe \Rightarrow gleiches Ergebnis

Formelumformung konsistent zur mathematischen Welt!

MuPAD: $D(x \mapsto x^2) \rightarrow 2 \text{ id}$

Funktionalanalysis in Computeralgebra

```

diff :: (a -> a) -> (a -> a)
diff ln = recip
diff ln 2 = 1/2

integrate :: (a -> a) -> (a -> a)
integrate cos = sin
integrate cos pi = 0
integrate (sin . sin) 2 =
                                integrate (sin . sin) 2

cont :: (a -> a) -> (a -> a)

```

`cont f x` $\hat{=}$ $\lim_{y \rightarrow x} f(y)$

stetige Fortsetzung

Vereinfachung: Kürzen

$$\frac{x^2 - 1}{x - 1} = \begin{cases} \text{undefiniert} & : x = 1 \\ x + 1 & : \text{sonst} \end{cases}$$

MuPAD:

```
simplify((x^2-1)/(x-1))
```

$x + 1$

```
limit((y^2-1)/(y-1),y=x)
```

$\frac{x^2 - 1}{x - 1}$

- Beide Antworten strenggenommen falsch
- Ergebnisse müssten vertauscht werden

Kürzen: Auswege

Frage: Warum will man kürzen?

⇒ Entscheidung für

- Fallunterscheidung

```
if x==1 then undefined else x+1
```

- Variable x irrelevant ⇒ Polynome, Polynomdivision
`poly [-1,0,1] / poly [-1,1] == poly [1,1]`

- stetige Fortsetzung

```
cont (\x -> (x^2-1)/(x-1)) == (\x -> x+1)
```


Rekursiv definierte Folgen

Rekursiv definierte Folgen ohne explizite Berechnungsvorschrift:
Kann Computeralgebrasystem Grenzwert berechnen?

Beispiel:

$$a_{n+1} = 2 \cdot a_n$$

Lösungsmöglichkeit: `solve (x=2*x, x)`

Probleme:

- Grenzwert hängt von Anfangswert ab
- `solve` kennt Anfangswert nicht
- Grenzwert eindeutig, aber `solve` liefert Lösungsmenge

Funktionale Schreibweise

```
limit :: Sequence a -> a
limit (iterate (2*) 1)
```

enthält alle Informationen.

Papagei

Einfachste Umsetzung des Taschenrechnerprinzips:
Gib Eingabe unverändert aus.

```
solve (\x -> exp x == cos x)
```

$$\{x : \exp x = \cos x\}$$

⇒ Einfachstes korrektes Computeralgebrasystem der Welt!

Nützlichkeit

Nützliches korrektes Computeralgebrasystem

- gibt korrekte Antwort aus,
- gibt Tipps, falls CAS Reserven sieht,
- falls gar nichts geht: Ausgabe der Eingabe ist immer richtig

Unvollständige Lösungen

```
solve (\x -> exp x == cos x)
```

$$\{x : \exp x = \cos x\} \cup \{0\}$$

Antwort brauchbar für weitere Umformungen

```
fmap exp (solve (\x -> exp x == cos x))
```

$$\{\exp x : \exp x = \cos x\} \cup \{1\}$$

Scheinlösungen

```
solve (\x -> sqrt(x+1) == x)
```

$$\{x : \sqrt{x+1} = x\} \cap \left\{ \frac{1 + \sqrt{5}}{2}, \frac{1 - \sqrt{5}}{2} \right\}$$

Lösungsvorschläge

eindeutiges Ergebnis, aber CAS kennt nur Lösungskandidaten

```
suggest :: a -> Set a
suggest x = solve (\y -> x==y)
```

```
let goldenRatio = limit (iterate (\x -> 1 + recip x) 1)
goldenRatio
```

goldenRatio

try suggest goldenRatio

```
suggest goldenRatio
```

$$\text{suggest goldenRatio} \cap \left\{ \frac{1 + \sqrt{5}}{2}, \frac{1 - \sqrt{5}}{2} \right\}$$

Lösungsvorschläge

Lösungsvorschlag nützlich?

```
isRational goldenRatio
```

```
isRational goldenRatio
```

```
any isRational (suggest goldenRatio)
```

```
false
```

Computeralgebra in Haskell

- Symbolische Differentiation mit Optimierer des Glasgow-Haskell-Übersetzers: SymbolicDifferentiation
- DoCon: Domain Constructor
- LambdaBot: pointless

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung
- 4 Bedarfsauswertung**
- 5 Typisierung
- 6 Ende

Bedarfsauswertung

Bedarfsauswertung (Lazy evaluation):

Berechne Werte erst, wenn sie gebraucht werden.

Das erlaubt endlose Datenstrukturen.

- Folgen
- Potenzreihen
- Zeitreihen
- reelle Zahlen
- Kettenbrüche

Rechnen mit reellen Zahlen

- Darstellung reelle Zahl als Ziffernfolge
- beliebige Basis b , $b \in \mathbb{N} \wedge b \geq 2$
- Ziffern aus $[-b, b]$
- $R(b, e, (m_0, m_1, m_2, \dots)) = \sum_{j \in \mathbb{N}_0} b^{e-j} \cdot m_j$
- Ziffernfolge entspricht Intervallschachtelung
 $R(b, e, (m_0, \dots)) \in R(b, e, (m_0, \dots, m_k)) + [-b^{e-k}, b^{e-k}]$
- Redundanz vermeidet Überträge

Wurzel einer Potenzreihe

Gegeben: Potenzreihe für Funktion y

Gesucht: Potenzreihe für Funktion x mit $x(t)^2 = y(t)$

Beispiele:

$$y(t) = 1 + 2 \cdot t + t^2$$

$$x(t) = 1 + t$$

$$y(t) = 1 + t$$

$$x(t) = 1 + \frac{1}{2} \cdot t - \frac{1}{8} \cdot t^2 + \frac{1}{16} \cdot t^3 - \frac{5}{128} \cdot t^4$$

Wurzelpotenzreihe: Ansätze I

■ direkte Lösung

$$y(t) = y_h + t \cdot y_t(t)$$

$$x(t) = x_h + t \cdot x_t(t)$$

$$x(t)^2 = x_h^2 + 2 \cdot x_h \cdot t \cdot x_t(t) + t^2 \cdot x_t(t)^2$$

$$y_h = x_h^2$$

$$x_h = \sqrt{y_h}$$

$$y_t(t) = 2 \cdot x_h \cdot x_t(t) + t \cdot x_t(t)^2$$

$$x_t(t) = \frac{y_t(t) - t \cdot x_t(t)^2}{2 \cdot x_h}$$

Wurzelpotenzreihe: Ansätze II

- Lösung durch Differentialgleichung

$$\begin{aligned}x(t) &= \sqrt{y(t)} \\x'(t) &= \frac{y'(t)}{2 \cdot \sqrt{y(t)}} \\&= \frac{y'(t)}{2 \cdot x(t)}\end{aligned}$$

- iterative Lösung mit NEWTON

$$x \mapsto \frac{1}{2} \cdot \left(x + \frac{y}{x} \right)$$

Quadratische Konvergenz

⇒ Falls erstes Glied korrekt,

Anzahl korrekte Glieder verdoppelt sich in jedem Schritt

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung
- 4 Bedarfsauswertung
- 5 Typisierung**
- 6 Ende

Typen in Axiom

Keine mathematischen Typen wie

- natürliche Zahl,
- reelle Zahl,
- Wahrheitswert,
- Polynom,

sondern computeralgebrarelevante wie

- Ausdruck,
- Polynomausdruck,
- Gleichung.

Typen in Haskell

Vorteile

- statische Typzuweisung: Effizienz
- Fehler zur Übersetzungszeit aufdecken
- maschinengeprüfte Dokumentation
- Typabgleich (type inference):
Typangabe nur wenn nötig
- Typklassen: Wiederverwendbare Programmteile

Werkzeuge

- Typbestimmung eines Ausdruckes
- Hoogle: Suche Bibliotheksfunktion passend zu Typ
- Djinn: Erzeuge Funktion passend zu Typ

Klare Ansagen

Beispiele für komplexe Typen

- Polynom über ganzen Zahlen: `Polynomial Integer`
- Polynom über Restklassenring:
`Polynomial (ResidueClass Integer)`
- Restklassenring von Polynomen:
`ResidueClass (Polynomial Rational)`
- Element aus Polynomkörper:
`Ratio (Polynomial Rational)`

⇒ keine Missverständnisse beim Faktorisieren

- 1 Überblick
- 2 Grundlagen
- 3 Funktionen höherer Ordnung
- 4 Bedarfsauswertung
- 5 Typisierung
- 6 Ende**

Endlich isser fertig

