Symbolisches Rechnen
Vorlesung
Wintersemester 2006, 2014
Sommersemester 2021

Johannes Waldmann, HTWK Leipzig

13. April 2021

Symbolisches Rechnen: Beispiele: Zahlen

- numerisches Rechnen mit Maschinenzahlen

```
sqrt 2 + sqrt 3  ==> 3.1462643699419726
(sqrt 2 + sqrt 3)*(sqrt 2 - sqrt 3) ==> ...
```

- exaktes Rechnen (mit algebraischen Ausdrücken)
  $(\sqrt{2}+\sqrt{3})\cdot(\sqrt{2}-\sqrt{3}) = \ldots$,
  maxima: `expand(%)`

Symbolisches Rechnen: Beisp.: Funktionen

- auf konkreten Daten:
  `let f x = (x+1)^2 in f 3.1 - f 3`
- auf symbolischen Daten: `diff((x+1)^2,x)`
- `subst([x=3],diff((x+1)^2,x))`
- eigentlich `diff(\x -> (x+1)^2)`
  mit `diff::(R -> R) -> (R -> R)`,
  aber da die Mathematiker Funktionen (höhere Ordnung) immer unzweckmäßig bezeichnen, um den Lambda-Kalkül zu vermeiden ...

Symbolisches Rechnen: Motivation

hat weitreichende Anwendungen:

- Lösen von (parametrisierten) Aufgaben*klassen*
  (für numerisches Rechnen muß Parameter fixiert werden)
- *exaktes* Lösen von Aufgaben
  (numer. R. mit Maschinenzahlen: nur Approximation)
- experimentelle, explorative, exakte Mathematik

ist nützlich im Studium, benutzt und vertieft:

- Mathematik (Analysis, Algebra)
- Algorithmen-Entwurf, -Analyse
- Prinzipien von Programmiersprachen

Überblick

- Zahlen (große, genaue)
- Vektoren (Gitterbasen)
- Polynome
- Terme, Term-Ersetzungs-Systeme

  (Anwendung: Differentiation, Vereinfachung)
- Gröbnerbasen (Termination, Vervollständigung)
- Geometrische Konfigurationen
- . . . und Beweise (Anwendung von Gröbnerbasen)
- Ausblick: $A = B$, Musik, Logik, Refactoring

Literatur

- Wolfram Koepf: *Computeralgebra*, Springer, 2006. http://www.mathematik.uni-kassel.de/~koepf/CA/
- Hans-Gert Gräbe: *Einführung in das Symbolische Rechnen, Gröbnerbasen und Anwendungen*, Skripte, Universität Leipzig http://www.informatik.uni-leipzig.de/~graebe/skripte/
- Franz Baader and Tobias Nipkow: *Term Rewriting and All That*, Cambridge, 1998. http://www21.in.tum.de/~nipkow/TRaAT/
- weitere Literatur siehe z.B. https://portal.risc.jku.at/Members/hemmecke/teaching/ppscs

Software

- wir benutzen
  - Maxima `http://maxima.sourceforge.net/`
  - FriCAS `https://github.com/fricas/fricas/`
  - Geonext `http://geonext.uni-bayreuth.de/`
  - GHC `http://www.haskell.org/ghc/`
- ist alles im Pool installiert (ssh, tmux, x2go)
- allgemeine Hinweise, auch zum Selbstbauen
  `https://imweb.imn.htwk-leipzig.de/~waldmann/etc/cas/`

Beispiel: S.R. und Term-Ersetzung

Regeln für symbolisches Differenzieren (nach `t`):

```
D(t) -> 1              D(constant) -> 0
D(+(x,y)) -> +(D(x),D(y))
D(*(x,y)) -> +(*(y,D(x)),*(x,D(y)))
D(-(x,y)) -> -(D(x),D(y))
```

Robert Floyd 1967, zitiert in: Nachum Dershowitz: *33 Examples of Termination*, `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.9447`

- Korrektheit? Termination? Komplexität?
- Strategie (Auswahl von Regel und Position)?
- ausreichend? angemessen?

Beispiel: Termersetzung (cont.)

```
data E = Zero | One | T
       | Plus E E | Times E E  deriving Show

e :: E
e = let b = Plus T One in Times b b

d :: E ->  E
d e = case e of
    Zero -> Zero ; One -> Zero ; T -> One
    Plus x y -> Plus (d x) (d y)
    Times x y ->
        Plus (Times y (d x)) (Times x (d y))
```

Beispiel: Inverse Symbolic Calculator

- `http://wayback.cecm.sfu.ca/projects/ISC/ISCmain.html`
  zur Bestimmung ganzzahliger Relationen (z.B. zwischen Potenzen einer numerisch gegebenen Zahl)
- `sqrt(2+sqrt 3)  ==>  1.931851652578136 6`

  ```
  integer relations algorithm, run:
  K = 1.931851652578136 6
                                      4       2
  K satisfies the polynomial, X  - 4 X  + 1
  ```

  mit LLL-Algorithmus (Lenstra, Lenstra, and Lovasz, 1982), der kurzen Vektor in geeignetem Gitter bestimmt.

Hausaufgaben KW 14, Organisatorisches

1. zum Haskell-Programm zum Symb. Differenzieren:
   - füge Syntax und Regel für Quotienten hinzu
   - schlage Regeln zur Vereinfachung vor
2. ISC Simple Lookup and Browser sagt für $\sqrt{2+\sqrt{3}}$:

   ```
   Mixed constants with 5 operations
   1931851652578136 = 1/2/sin(Pi/12)
   ```

   begründen Sie das (geometrisch oder schriftlich)
3. ein Polynom mit Nullstelle $\sqrt[2]{2}+\sqrt[3]{3}$ bestimmen, nachrechnen.
4. Geonext: Satz von Napoleon illustrieren (gleichseitige Dreiecke über den Seiten eines beliebigen Dreiecks)
5. eigener Rechner: `rlwrap maxima` installieren,

   Rechner im Pool: `ssh` und `tmux` ausprobieren, auch Management von Sessions, Windows, Panes (split horizontal, vertikal), vgl. `https:`

Überblick

- exakte Zahlen: natürlich, ganz, rational

  Darstellungen: Positionssystem, Bruch

  Rechnungen: Addition, Multiplikation

- beliebig genau genäherte Zahlen

  (berechenbare reelle Zahlen)

  Darstellungen: Positions-System, Kettenbruch

  Rechnungen: arithmetische und irrationale Funktionen

- später:

  exaktes Rechnen mit algebraischen Zahlen

Darstellung natürlicher Zahlen

- die Zahl $n \in$ im Positionssystem zur Basis B:
  $n = \sum_{k \geq 0} x_k B^k$
  mit $\forall i : 0 \leq x_i < B$ und $\{i \mid x_i \neq 0\}$ endlich
  Bsp: $25 = 1 \cdot 3^0 + 2 \cdot 3^1 + 2 \cdot 3^2 = [1, 2, 2]_3$
- Darstellung ist eindeutig, kann durch fortgesetzte Division mit Rest bestimmt werden $25 = 1 + 3 \cdot 8, 8 = 2 + 3 \cdot 2$
- praktisch wählt man die Basis
  - 10 für schriftliches Rechnen
  - historisch auch 60 (Zeit- und Winkelteilung)
  - 2 (oder 2^w) binäre Hardware, maschinennahe Software

Natürliche Zahlen, Addition

- Darstellung

```
type Digit = Word64 ; data N = Z | C Digit N
```

- Semantik

```
value :: N -> Natural
value Z = 0 ; value (C x xs) = x + 2^64 * value xs
```

- Rechnung

```
instance Num N where (+) = plus_carry False
plus_carry :: Bool -> N -> N -> N
plus_carry cin (C x xs) (C y ys) =
  let z = bool 0 1 cin + x + y
      cout = z < x || z < y || ...
  in  C z (plus_carry cout xs ys)
plus_carry ...
```

Rekursive Multiplikation

- anstatt „Schulmethode“: rekursive Multiplikation
  $(p + qB)(r + sB) = pr + (ps + qr)B + qsB^2$
  Bsp. $B = 100$, $(12 + 34 \cdot 100)(56 + 78 \cdot 100) = 12 \cdot 56 + \ldots$
  $(1 + 2 \cdot 10)(5 + 6 \cdot 10) = 1 \cdot 2 + \ldots$
- $M(w) =$ Anzahl elementarer Operationen (Plus, Mal) für Multiplikation w-stelliger Zahlen nach diesem Verfahren
  $M(1) = 1, M(w) = 4 \cdot M(w/2) + 2 \cdot w$
  $M(2^k) = 3 \cdot 4^k - 2^{k+1}$, also $M(w) \in \Theta(w^2)$
- also wie bei Schulmethode, aber das läßt sich verbessern, und darauf muß man erstmal kommen

Karatsuba-Multiplikation

- A. Karatsuba, Yu. Ofman 1962
- $(p + qB)(r + sB) = pr + (ps + qr)B + qsB^2$
  $= pr + ((p + q)(r + s) - pq - rs)B + qsB^2$
- $K(w) =$ Anzahl elementarer Operationen (Plus, Mal) für Multiplikation w-stelliger Zahlen nach diesem Verfahren

  $K(1) = 1, K(w) = 3 \cdot K(w/2) + 5 \cdot w$

  (eine Multiplikation weniger, drei Additionen mehr)
- asymptotisch mit Ansatz $K(w) \approx C \cdot w^e$

  $C \cdot w^e = 3Cw^e/2^e + 5w \Rightarrow 1 \approx 3/2^e$, also
  $e = \log_2 3 \approx 1.58$
- explizite Werte für $w = 2^k$,

  ab wann besser als Schulmethode?

Ganze Zahlen

- Darstellung: Bsp GMP (GNU Multi Precision Library)

  Natürliche Zahl (magnitude) mit Vorzeichen (sign)

  nicht Zweierkomplement, denn das ist nur für Rechnungen modulo 2^w sinnvoll

- GHC (`integer-gmp`)

```
data Integer = S Int | Jp BigNat | Jn BigNat
data BigNat = BN ByteArray
```

- NB: ganzzahlige Zahlbereiche in Haskell/GHC:
  - `Int` (ist immer die falsche Wahl)
  - Maschinenzahlen (mod 2^{64}): `Int64`, `Word64`,
  - beliebig große: `Integer`, `Numeric.Natural`

Rationale Zahlen

- rationale Zahl q ist Paar von ganzen Zahlen (z, n)

  ```
  data Q = Q Integer Integer
  ```

- normalisiert: $n \geq 1$ und $\gcd(|z|, |n|) = 1$
- normalisierte Darstellung ist eindeutig
- Vorteil:

  semantische Gleichheit (dieselbe Zahl wird bezeichnet) *ist* syntaktische Gleichheit (der Komponenten)

- Nachteil (?)

  nach jeder arithmetischen Operation normalisieren

- nicht normale Zahlen verhindern: Konstruktor `Q` nicht exportieren (sonst `let q = Q 8 12`)

Explosion der Stellenanzahl (Beispiel)

- Funktion $f :\to: x \mapsto x/2 + 1/x$
- $x_k = f^k(1)$, $x_0 = 1, x_1 = 3/2, x_2 = 17/12, x_3 = 577/408$
- Zähler (und Nenner) in jedem Schritt (wenigstens) quadriert, d.h., Stellenzahl verdoppelt (Bsp: x_{10})
  $p/q \mapsto p/(2q) + q/p = (p^2 + 2q^2)/(2pq)$
- das ist typisch für Folgen arithmetischer Op., Normalisierung wirkt nur selten verkleinernd.
- die Folge x_k nähert $\sqrt{2}$ an: $(p_k/q_k)^2 - 2 = 1/q_k^2$
  Konvergenz ist quadratisch (jeder Schritt quadriert den Fehler, verdoppelt Anzahl gültiger Stellen)

Endliche und unendliche „Dezimal“brüche

- Darstellug von Zahlen in $[0 \ldots 1)$
  mit Basis $1/B$ und Ziffern $\in \{0 \ldots B-1\}$
- Bsp: $B = 10$, $0.314 = 0 \cdot B^0 + 3 \cdot B^{-1} + 1 \cdot B^{-2} + 4 \cdot B^{-3}$
- hier sind auch unendliche Ziffernfolgen sinnvoll,
  bezeichnet Limes der Werte der endlichen Partialfolgen
- Konversion

```
decimal :: Rational -> [ Nat ]
decimal x = let q = truncate (fromRational x)
            in  q : decimal (10 * (x-q))
```

  Anwendung: vorige Näherung von $\sqrt{2}$

Berechenbare reelle Zahlen

- beschrieben wird hier nicht die exakte (symbolische) Rechnung, sondern eine konvergente Näherung
- Def. reelle Zahl r mit $0 \leq r < 1$ heißt *berechenbar* (zur Basis B), wenn die Funktion $d :\to \{0, \ldots, B-1\}$, welche die Darstellung von x zur Basis $1/B$ bestimmt, berechenbar ist (z.B. durch eine Turingmaschine)
- jede rationale Zahl ist berechenbar

  (Beweis: Dezimalbruch ist endlich oder periodisch)
- $\sqrt{2}$ ist berechenbar (Beweis: vorige Folge x_k)
- Menge der berechenbaren Zahlen ist abgeschlossen unter arithmetischen Operationen (und weiteren)

Potenzreihen, Exponentialfunktion

- Satz von Taylor: wenn f oft genug diff-bar,
  dann $f(x_0 + d) = \sum_{k=0}^{n-1} f^{(k)}(x_0) d^k / k! + \Delta_n$
  mit $\exists 0 \leq d' \leq d : \Delta_n = f^{(n)}(x_0 + d') d^n / n!$
- Bsp: $f(x) = \exp(x)$, dann $f = f' = f'' = f^{(3)} = \ldots$ und
  $\exp(0 + d) = 1 + d + d^2/2 + d^3/6 + \ldots$
- ```
  take 100 $ decimal
    $ sum $ take 100 $ scanl (/) (1::Rational) [1..]
      ==>   [2,7,1,8,2,8,1,8,2,8,4,5,9,0,4 ... ]
  ```
  Fehlerabschätzung? (reicht die zweite 100 für die erste?)
- $\exp(1) = \exp(1/2)^2$, diese Reihe konvergiert schneller
  ```

Potenzreihe für Wurzelfunktion

- Taylor-Reihe von $\sqrt{x}$ an der Stelle 1
  Ableitungen mit maxima:
  `diff(sqrt(x),x,5);` $105/32 \cdot x^{-9/2}$
  `diff(sqrt(x),x,6);` $-945/64 \cdot x^{-11/2}$
  Vermutung $f^{(n)}(1) = (-1)^{n+1} \cdot (2n-1)!! \cdot 2^{-n}$
- $\sqrt{1+d} = 1 + \sum_{k>0} (-1)^{k+1} \frac{(2k-1)!!}{2k!!} d^k$
  $= 1 + \frac{1}{2} \cdot d - \frac{1 \cdot 3}{2 \cdot 4} \cdot d^2 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot d^3 - \ldots$
- für $\sqrt{2}$: Berechnung als $\sqrt{1+1}$ konvergiert langsam
  besser: $\sqrt{2} = 7/5\sqrt{1+1/49}$

Hausaufgaben

1. Multiplikation in GMP (GNU Multiprecision Library)
   `https://gmplib.org/manual/Multiplication-Algorithms`
   - 1.1 Karatsuba-Rechnung ist dort etwas anders als hier auf der Folie, warum?
   - 1.2 GHC verwendet GMP für den Typ `Integer`.
     Bestimmen Sie experimentell den Anstieg der Rechenzeit bei Verdopplung der Stellenzahl, z.B.

     ```
     :set +s
     x = 10^10^8 :: Integer
     odd x -- damit x ausgewertet wird
     odd ((x-1)*(x+1)) -- die eigentliche Messung
       True
       (3.73 secs, 166,159,792 bytes)
     y = x*x -- hat doppelte Stellenzahl
     ```

     Ist die Anzahl der Bytes plausibel?
     Diskutieren Sie mögliche verkürzte Auswertungen für `odd ...`. Kann GMP/GHC das?
   - 1.3 Zusatz: warum ist (oder erscheint) `(x+1)^2` schneller als `(x+1)*(x+1)`?