ℂ

Sarah Lena Zengl

Mai 2020

Normalform

- Wir verwenden das **Horner-schema** als Normalform:
- D.h.: $a_n x^n + ... + a_0$ wird geschrieben als $a_0 + x(a_1 + x(...(a_{n-1} + x \cdot a_n)...))$.
- Wobei a_i Polynom in verbleibenden Variablen ist.

Normalform

- Wir benötigen also eine Liste von Variablen. Diese ist so von innen nach außen gelistet, das die innerste Variable im Listenkopf steht.
- Beispielsweise ist mit $Vars = [x, y, z]$
- Die Normalform von $3xy^2 + 2x^2yz + zx + 3yx$
- $[0 + y(0 + z \cdot 3)] + x[((0 + z \cdot 1) + y(0 + y \cdot 3)) + x(0 + y(0 + z \cdot 2))]$

Rechenoperationen in Normalform

Innerhalb dieser Normalform können wir die bekannten Rechenoperationen $+, -, \cdot, /$ sowie $()^n$ formalisieren:

Listing 1: Addition

```
let rec poly_add vars pol1 pol2 =
 match (pol1,pol2) with
  (Fn("+",[c; Fn("*",[Var x; p])]),Fn("+",[d; Fn("*",[Var y; q
      ])])) ->
    if earlier vars x y then poly_ladd vars pol2 pol1
    else if earlier vars y x then poly_ladd vars pol1 pol2 else
    let e = poly_add vars c d and r = poly_add vars p q in
   (if r = zero then e else Fn("+",[e; Fn("*",[Var x; r])]))
  | (_,Fn("+",_)) -> poly_ladd vars pol1 pol2
  | (Fn("+",_),pol2) -> poly_ladd vars pol2 pol1
  | _ -> numeral2 (+/) pol1 pol2
and poly_ladd vars =
 fun pol1 (Fn("+",[d; Fn("*",[Var y; q])])) ->
  Fn("+",[poly_add vars pol1 d; Fn("*",[Var y; q])]);;
```

Listing 2: Negation

```
let rec poly_neg =
 function (Fn("+",[c; Fn("*",[Var x; p])])) ->
   Fn("+",[poly_neg c; Fn("*",[Var x; poly_neg p])])
  | n -> numeral1 minus_num n;;
```

Listing 3: Subtraktion

```
let poly_sub varspq= poly_add vars p (poly_neg q);;
```

Listing 4: Multiplikation

```
let rec poly_mul vars pol1 pol2 =
 match (pol1,pol2) with
  (Fn("+",[c; Fn("*",[Var x; p])]),Fn("+",[d; Fn("*",[Var y; q
      ])])) ->
    if earlier vars x y then poly_lmul vars pol2 pol1
    else poly_lmul vars pol1 pol2
   | (Fn("0",[]),_) | (_,Fn("0",[])) -> zero
   | (_,Fn("+",_)) -> poly_lmul vars pol1 pol2
   | (Fn("+",_),_) -> poly_lmul vars pol2 pol1
   | _ -> numeral2 ( */ ) pol1 pol2
and poly_lmul vars =
 fun pol1 (Fn("+",[d; Fn("*",[Var y; q])])) ->
  poly_add vars (poly_mul vars pol1 d)
   (Fn("+",[zero;
    Fn("*",[Var y; poly_mul vars pol1 q])]));;
```

Listing 5: Potenzierung

```
let poly_pow varspn= funpow n (poly_mul vars p) (Fn("1",[]));;
```

Listing 6: Division

```
let poly_div varspq= poly_mul vars p (numeral1((//) (Int 1)) q)
    ;;
```

Des weiteren ist es möglich eine Variable in ein Polynom umzuwandeln:

Listing 7: Variable zu Polynom

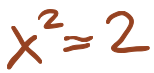
```
let poly_var x = Fn("+",[zero; Fn("*",[Var x; Fn("1",[])])]);;
```

Umformen in Normalform

Aus diesen Operationen zusammengesetzte Terme lassen sich nun einfach in Normalform bringen:

Listing 8: Normalisierer

```
let rec polynate vars tm =
 match tm with
   Var x -> poly_var x
  | Fn("-",[t]) -> poly_neg (polynate vars t)
  | Fn("+",[s;t]) -> poly_add vars (polynate vars s) (polynate
    vars t)
  | Fn("-",[s;t]) -> poly_sub vars (polynate vars s) (polynate
    vars t)
  | Fn("*",[s;t]) -> poly_mul vars (polynate vars s) (polynate
    vars t)
  | Fn("/",[s;t]) -> poly_div vars (polynate vars s) (polynate
    vars t)
  | Fn("^",[p;Fn(n,[])]) -> poly_pow vars (polynate vars p) (
    int_of_string n)
  | _ -> if is_numeral tm then tm else failwith "lint: unknown
    term";;
```



Äquivalenz

Damit können wir nun Äquivalenz prüfen, in dem wir Gleichungen $p = q$ in die Form $p - q = 0$ bringen, wobei wir $p - q$ normalisieren:

Listing 9: Äquivalenz Prüfer

```
1 let polyatom vars fm =
2  match fm with
3   Atom(R(a,[s;t])) -> Atom(R(a,[polynate vars (Fn("-",[s;t]));
        zero]))
4  | _ -> failwith "polyatom: not an atom";;
```

Äquivalenz - Beispiel

Das sieht dann so aus:

Listing 10: Äquivalenz Prüfung

```
# polyatom ["w"; "x"; "y"; "z"]
 <<((w + x)^4 + (w + y)^4 + (w + z)^4 +
    (x + y)^4 + (x + z)^4 + (y + z)^4 +
    (w - x)^4 + (w - y)^4 + (w - z)^4 +
    (x - y)^4 + (x - z)^4 + (y - z)^4) / 6 =
  (w^2 + x^2 + y^2 + z^2)^2>>;;
- : fol formula = <<0 = 0>>
```

Eigenschaften von Polynomen in einer Variable

Im folgenden möchten wir uns Polynome in einer Variable (aber mit Parametern) anschauen.

$p(x, y, z)$	$p_{y,z}(x)$
$x - 1 \nmid x^2y - xz$	$x - y \mid x^2y - xz$ mit $y = z$

Grad

Für Polynome in einer Variable ist der Grad von p, $\delta(p)$, das größte n, sd. $a_n x^n$ mit $a_n \neq 0$ in p enthalten ist.

Satz (1)

Für beliebiges Polynom $p(x)$ und Wert a ist $p(x) - p(a)$ durch $x - a$ teilbar.

Beweis.

Zeige die Aussage zunächst für alle $q(x) = x^n$ mit $n \in \mathbb{N}$:

$\underline{n = 0}$

$x^0 - a^0 = 1 - 1 = 0 = (x - a) \cdot 0$

$\underline{n \geq 1}$

$x^n - a^n = (x - a)(x^{n-1} + ax^{n-2} + ... + a^{n-2}x + a^{n-1})$

Da sich jedes Polynom schreiben lässt als $p(x) = b_n x^n + ... + b_1 x + b_0$ folgt die Aussage. □

Satz (2)

Für beliebiges Polynom $p(x)$ und Wert a ist der Grad von $(p(x) - p(a))/(x - a)$ um genau eins kleiner als der Grad von $p(x) - p(a)$.

Beweis.

Angenommen, das wäre nicht so. Dann gäbe es ein Polynom $p(x)$ und einen Wert a sd:
$\delta(p(x) - p(a)) \neq \delta((p(x) - p(a))/(x - a)) + 1 =$
$\delta((p(x) - p(a))/(x - a)) * (x + a)) = \delta(p(x) - p(a))$
Widerspruch. □

Eine Nullstelle ist ein a, sd. $p(a) = 0$.

Satz (3)

Falls $p(a) = 0$ *dann ist* $p(x)$ *teilbar durch* $(x - a)$

Beweis.

Aus Satz 1 folgt $(x - a) \mid (p(x) - p(a))$ aber $p(x) - p(a) = p(x) - 0 = p(x)$ □

Satz (4)

Ein Polynom in einer Variable $p(x)$ vom Grad n hat höchstens n Nullstellen.

Beweis.

Per Induktion über den Grad:
Falls $p(x)$ keine Nullstellen hat, ist die Aussage Trivial.
Andernfalls sei a eine Nullstelle. Wir wissen (Nach Satz 3) dass $p(x) = (x - a)q(x)$ für ein Polynom $q(x)$, wobei $q(x)$ nach Satz 2 Grad $n - 1$ hat. Die Nullstellen von $p(x)$ sind daher die Nullstellen von $q(x)$ sowie $x = a$ falls a nicht bereits Nullstelle von $q(x)$. Da nach der Induktionshypothese $q(x)$ höchstens n-1 Nullstellen hat, gilt die Aussage. □

Satz (5)

Ein Polynom $p(x)$ in einer Variable vom Grad n über $\mathbb{C}$ zerfällt in Linearfaktoren. Dh: Es gibt $a_1, a_2, ..., a_n, k$ (nicht notwendigerweise verschieden) mit $p(x) = k \cdot (x - a_1)...(x - a_n)$.

Beweis.

Per Induktion über den Grad von $p(x)$.
Wenn $p(x)$ konstant ist, dann ist die Aussage offensichtlich. Ansonsten liefert uns algebraische Abgeschlossenheit, das es eine Nullstelle a gibt. Dann wissen wir, das es ein $q(x)$ mit $\delta(q) + 1 = \delta(p)$ und $p(x) = (x - a) \cdot q(x)$ gibt. Mit der Induktionshypothese folgt die Aussage. □

Quantorenelimination

- Wir wollen nun einen Quantoreneliminationsalgorithmus für komplexe Zahlen (Tarski/ Seidenberg) vorstellen.
- Aufgrund der Normalisierung, können wir davon ausgehen, das alle atomaren Formeln von der Form $p(x) = 0$ sind.
- Wir wissen, das es ausreicht, einen einzigen Existenzquantor aus einer Konjugation von Literalen
  $\exists x. p_1(x) = 0 \wedge ... \wedge p_n(x) = 0 \wedge q_1(x) \neq 0 \wedge ... \wedge q_m(x) \neq 0$
  zu eliminieren.

Reduktion zu $m, n \leq 1$

Satz (6)

Eine Formel der Form $\bigwedge p_i(x) = 0$, mit p_i Polynomen, ist äquivalent zu einer Formel $\bigwedge p_i'(x) = 0$ mit allen, bis auf höchstens eine Ausnahme, p_i' konstant in x.

Dafür folgendes Lemma:

Lemma (7)

Für zwei Polynome p und s mit $deg(p) \leq deg(s)$ gibt es ein Polynom s' mit $deg(s') < deg(p)$, s.d. $p(x) = 0 \wedge s(x) = 0$ äquivalent zu $p(x) = 0 \wedge s'(x) = 0$ ist.

Der Satz folgt dann aus der wiederholten Anwendung des Lemmas, jeweils mit dem Polynom mit geringstem Grad gepaart mit allen anderen.

Reduktion zu $m, n \leq 1$

Lemma (7)

Für zwei Polynome p und s mit $deg(p) \leq deg(s)$ *gibt es ein Polynom* s' *mit* $deg(s') < deg(p)$*, s.d.* $p(x) = 0 \wedge s(x) = 0$ *äquivalent zu* $p(x) = 0 \wedge s'(x) = 0$ *ist.*

Beweis.

Sei $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, mit a_i konstant in x für alle i. Sei l maximal, s.d. $a_l \neq 0$ (das hängt von dem Kontext ab, da es sich bei den a_i um Polynome in anderen Variablen handelt, und erfordert deswegen eine Fallunterscheidung im Algorithmus). Sei p' p nach entfernen aller Terme mit Exponenten größer als l. Sei $a_l^k s(x) = p'(x)q(x) + s'(x) = p(x)q(x) + s'(x)$ Das Ergbnis der Polynomdivision von s durch p'.
Beweis der Äquivalenz: ($\Rightarrow$) Sei $x \in \mathbb{C}$ s.d.
$p(x) = s(x) = 0 \Rightarrow 0 = a_l^k s(x) - p(x)q(x) = s'(x)$.
($\Leftarrow$) Sei $x \in \mathbb{C}$ s.d. $p(x) = s'(x) = 0 \Rightarrow a_l^k s(x) = p(x)q(x) + s'(x) = 0$. □

Reduktion zu $m, n \leq 1$

Durch Anwendung von Satz 6 auf

$$\exists x.p_1(x) = 0 \wedge ... \wedge p_n(x) = 0 \wedge q_1(x) \neq 0 \wedge ... \wedge q_m(x) \neq 0 \tag{1}$$

erhalten wir die äquivalente Formel

$$\exists x.p_1'(x) = 0 \wedge ... \wedge p_n'(x) = 0 \wedge q_1(x) \neq 0 \wedge ... \wedge q_m(x) \neq 0 \tag{2}$$

wobei $\exists i \in n+1 \forall k \in n+1 (k \neq i) \rightarrow (p_k'(x) konstant)$
Seien i dieses i.
Unter diesen $p_k'(x), k \neq i$ gibt es nun entweder ein k mit $p_k'(x) \neq 0$ (Fall 1). Dann ist $p_k'(x) = 0 \leftrightarrow \perp$, und die ganze Formel (2) $\leftrightarrow \perp$, und somit Quantorenfrei.

Reduktion zu $m, n \leq 1$

Oder $\forall k \in n+1(k \neq i) \rightarrow (p'_k(x) = 0)$ (Fall 2). Dann sind aber all diese $p_k(x) = 0$ in (2) $\leftrightarrow \top$, und somit (2) $\leftrightarrow$

$$\exists x.p'_i(x) = 0 \wedge q_1(x) \neq 0 \wedge ... \wedge q_m(x) \neq 0 \tag{3}$$

Um m zu reduzieren können wir alle q_j, mit $j \in m+1$ multiplizieren, da

$$q_1(x) \neq 0 \wedge ... \wedge q_m(x) \neq 0 \Leftrightarrow q(x) := q_1(x) \cdot q_2(x) \cdot ... \cdot q_m(x) \neq 0 \tag{4}$$

Jetzt haben wir

$$\exists x.p(x) = 0 \wedge q(x) \neq 0 \tag{5}$$

Ausschluss von Nullpolynomen

$$\exists x.p(x) = 0 \wedge q(x) \neq 0 \tag{5}$$

4 Fälle:	p Nullpolynom	sonst
q Nullpolynom	$q(x) = 0 \Rightarrow$ $(5) \leftrightarrow \bot$	$q(x) = 0 \Rightarrow$ $(5) \leftrightarrow \bot$
sonst	$p(x) = 0 \Rightarrow$ $(5) \leftrightarrow \exists x.q(x) \neq 0$	???

$\exists x.q(x) \neq 0$ ist nach dem Fundamentalsatz der Algebra entweder zu $\top$ oder zu $\bot$ äquivalent, je nach dem ob q ein von Null verschiedenes, konstantes Polynom ist.

Im folgenden $p, q \neq 0$

Umwandlung mittels algebraischer Abgeschlossenheit

- Haben:
$$\exists x.p(x) = 0 \wedge q(x) \neq 0 \tag{5}$$
- Oder äquivalent
$$\neg\forall x.(p(x) = 0 \rightarrow q(x) = 0) \tag{6}$$
- Betrachten Negation
$$\forall x.(p(x) = 0 \rightarrow q(x) = 0) \tag{7}$$

Diese ist quantorenfrei gdw. (6) quantorenfrei.

Umwandlung mittels algebraischer Abgeschlossenheit

Mit Satz 5 lassen sich p und q schreiben als:

$$p(x) = c(x - a_1)(x - a_2)...(x - a_r) \tag{8}$$

$$q(x) = d(x - b_1)(x - b_2)...(x - b_s) \tag{9}$$

Wobei $c, d \neq 0$, da p, q keine Nullpolynome sind. Damit ist $p(x) = 0 \leftrightarrow \bigvee_{k \in r+1} x = a_k$ und $q(x) = 0 \leftrightarrow \bigvee_{l \in s+1} x = b_l$. Und somit

$$\forall x.(p(x) = 0 \rightarrow q(x) = 0) \tag{7}$$

äquivalent zu

$$\forall x. \bigvee_{k \in r+1} x = a_k \rightarrow \bigvee_{l \in s+1} x = b_l \tag{10}$$

In Worten: Alle a_k kommen unter den b_l vor.

Quantorenelimination mit Teilbarkeit

$$\forall x. \bigvee_{k \in r+1} x = a_k \rightarrow \bigvee_{l \in s+1} x = b_l \tag{10}$$

Nur r Linearfaktoren im Antezedens $\Rightarrow p(x) \mid q(x)^r$.
Außerdem: $p(x) \mid q(x)^r$, mit $r > 0 \Rightarrow \forall x. p(x) = 0 \rightarrow q(x) = 0$
Also:

$$(10) \leftrightarrow p(x) \mid q(x)^r \tag{11}$$

□

$p(x) \nmid q(x)^r$

Nützliche Hilfsfunktionen

Zunächst eine, die die Koeffizienten des Polynoms $c_0 + c_1x + ... + c_nx^n = c_0 + x(c_1 + x(...(c_{n-1} + x \cdot c_n)...))$ als Liste $[c_0, c_1, ..., c_n]$ speichert:

Listing 11: Koeffizienten

```
1 let rec coefficients vars =
2  function Fn("+",[c; Fn("*",[Var x; q])]) when x = hd vars ->
3    c::(coefficients vars q)
4   | p -> [p];;
```

Die erste Variable von Vars ist die eine Variable des Polynoms.

Listing 12: Grad

```
let degree vars p = length(coefficients vars p) - 1;;
```

Listing 13: Prüfung ob Konstant

```
let is_constant vars p = degree vars p = 0;;
```

Listing 14: Leitkoeffizient (head)

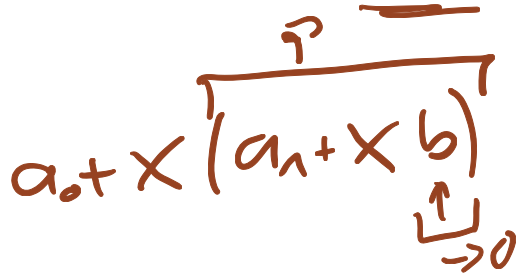
```
let head vars p = last(coefficients vars p);;
```

$x = hd\ vars$

Der Leitkoeffizient kann dennoch Null sein. Daher definieren wir

Listing 15: Behead

```
let rec behead vars =
 function Fn("+",[c; Fn("*",[Var x; p])]) when x = hd vars ->
   let p = behead vars p in
   if p = zero then c else Fn("+",[c; Fn("*",[Var x; p])])
 | _ -> zero;;
```



monische Form

- Wollen vielfache Eliminieren, um überflüssige Rechnungen zu verhindern.
- Nutzen monische Form, dh: der Leitkoeffizient in allen Variablen soll 1 sein.

monische Form

Listing 16: Konstantenmultiplikation

```
let rec poly_cmul k p =
 match p with
  Fn("+",[c; Fn("*",[Var x; q])]) ->
   Fn("+",[poly_cmul k c; Fn("*",[Var x; poly_cmul k q])])
| _ -> numeral1 (fun m -> k */ m) p;;
```

Listing 17: Konstanter Leitkoeffizient

```
let rec headconst p =
 match p with
  Fn("+",[c; Fn("*",[Var x; q])]) -> headconst q
| Fn(n,[]) -> dest_numeral p;;
```

monische Form

$-1x \rightsquigarrow 1x$, true

Listing 18: Monische Form

```
1 let monic p =
2  let h = headconst p in
3  if h =/ Int 0 then p,false else poly_cmul (Int 1 // h) p,h </
       Int 0;;
```

Beim umformen in monische Form geben wir außerdem zurück, ob sich das Vorzeichen des konstanten Leitkoeffizienten geändert hat.

Fragen