Ringe, Ideale und Wortprobleme

Der Algorithmus von letzten Vortrag ist häufig ineffizient, da die Formel immer größer werden kann. Wenn wir uns darauf einschränken nur universelle Formeln über $\mathbb{C}$ zu benutzen, können wir einen völlig anderen Ansatz wählen, der sich mit allen Variablen auf einmal befasst.

Wortproblem

Definition Sei K eine Klasse von algebraischen Strukturen z.B. alle Gruppen. Das *Wortproblem* für K fragt, ob eine Menge E von Gleichungen in einer gegebenen Sprache eine andere Gleichung $s = t$ in allen Strukturen der Klasse K impliziert. Es gibt verschiedene Arten von Wortproblemen:

- Das uniforme Wortproblem für K: entscheidet gegeben ein beliebiges E und $s = t$, ob $E \models_M s = t$ für alle Modelle M in K;
- Das Wortproblem für K,E: mit E fixiert, entscheidet gegeben ein beliebiges $s = t$, ob $E \models_M s = t$ für alle Modelle M in K;
- Das freie Wortproblem für K: entscheidet gegeben ein beliebiges $s = t$, ob $\models_M s = t$ für alle Modelle M in K;

Anmerkung Es stellt sich heraus, das es endliche E gibt, so dass das Wortproblem für Gruppen und E nicht entscheidbar ist. Interessant ist auch, daß es Klassen K gibt, für welche es keinen uniformen Entscheidungsalgorithmus gibt mit E und $s = t$ als Input, obwohl es für jedes spezifische, endliche E einen gibt, mit $s = t$ als Input.

Angenommen K wird von Σ axiomatisiert, dann lässt sich das Wortproblem als $\Sigma \cup E \models s = t$ schreiben. Angenommen E ist endlich, und wir ersetzen Konstanten, die nicht in den Axiomen vorkommen mit Variablen, dann können wir das Wortproblem folgendermaßen ausdrücken, wobei alle Terme nur Konstanten und Funktionssymoble enthalten, die auch in Σ vorkommen:
$\Sigma \models \forall x_1 ... x_n . \bigwedge_i s_i = t_i \Rightarrow s = t$

Ringe

Sei $S_R = \{+, \cdot, -, 1, 0, =\}$ die Sprache der Ringe.
Definition (kommutative) Ringe sind algebraische Strukturen, die eine Addition und Multiplikation, mit 0 und 1, haben und folgende Axiome erfüllen:

- $x + y = y + x$
- $x + (y + z) = (x + y) + z$
- $x + 0 = x$
- $x + (-x) = 0$
- $x \cdot y = y \cdot x$
- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- $x \cdot 1 = x$
- $x \cdot (y + z) = x \cdot y + x \cdot z$

Um alle Ringe zu axiomatisieren, müssen wir die oben genannten Axiome mit einigen Gleichheits -und Kongruenzeigenschaften ergänzen, da wir das = als Relationssymbol auffassen.

- $x = x$
- $x = y \Rightarrow y = x$
- $x = y \wedge y = z \Rightarrow x = z$
- $x = x' \Rightarrow -x' = -x$
- $x = x' \wedge y = y' \Rightarrow x + y = x' + y'$
- $x = x' \wedge y = y' \Rightarrow x \cdot y = x' \cdot y'$

Sei *Ring* die Menge dieser Axiome. Dann gilt p in allen Ringen g.d.w. $Ring \models p$.
Es gibt viele verschiedene Arten von Ringen, z.B. $\mathbb{R}$ oder $\mathbb{Z}/n\mathbb{Z}$ mit $n \in \mathbb{N}$ und $n > 0$.
Ich möchte an dieser Stelle noch einen weiteren interessanten Ring betrachten, der über $\wp(A)$ für eine Menge A definiert ist mit $0 = \emptyset$, $1 = A$, $S + T = (S - T) \cup (T - S)$ und $S \cdot T = S \cap T$.
Es folgen noch viele weitere Axiome aus *Ring* z.B. $0 \cdot x = x \cdot 0 = 0$ oder $(-1) \cdot x = -x$ und insbesondere $s = t \Leftrightarrow s - t = 0$.

Definition n sei das Ringelement: $\overbrace{1 + ... + 1}^{n-mal}$

Definition Die Charakteristik eines Rings R ist $char(R) = \min\{n > 0 | n = 0 \ in \ R\}$ falls diese Menge nicht leer ist, ansonsten 0.
Beispiel $\text{char}(\mathbb{Z}/n\mathbb{Z}) = n$ oder $\text{char}(\mathbb{R}) = 0$.

Definition Wir können die Charakteristik $n > 1$ eines Rings axiomatisieren mit der Menge $C_n :=$

$$\begin{gathered}\neg(1 = 0) \\ \neg(2 = 0) \\ ... \\ \neg(n - 1 = 0) \\ n = 0\end{gathered}$$

und die Charakteristik 0 durch die unendliche Menge:

$$C_0 := \{\neg(n = 0) | n \in \mathbb{N} \wedge n > 0\}$$

Sei $C_1 := \{\neg(1 = 0)\}$ das Axiom, das den trivialen Ring ausschließt.

Satz $Ring \cup \Gamma \models \forall x_1, ..., x_n. \bigwedge_i s_i = t_i \Rightarrow s = t \iff$
$Ring \cup \Gamma \cup C_1 \models \forall x_1, ..., x_n. \bigwedge_i s_i = t_i \Rightarrow s = t$
Beweis $\Rightarrow$: Gilt trivialerweise wegen Monotonie.
$\Leftarrow$: Man stellt fest, dass jede Gleichung $s = t$ aus $1 = 0$ folgt. $\square$

Polynomringe

Sei ein Ring R gegeben. Wir möchten die Menge $R[x_1, ..., x_n]$ der Polynome in n Variablen mit Koeffizienten in R formalisieren.
Definition Ein Polynom ist eine Abbildung $p : \mathbb{N}^n \to R$, so dass $\{i \in \mathbb{N}^n | p(i) \neq 0\}$ endlich ist. Intuitiv beschreibt ein $(i_1, ..., i_n) \in \mathbb{N}^n$ das Monom $x_1^{i_1}...x_n^{i_n}$, wobei $p(i_1, ..., i_n)$ den Koeffizienten des jeweiligen Monoms angibt. Also repräsentiert die Abbildung p das Polynom: $\sum_{i=(i_1,...,i_n)\in\mathbb{N}^n} p(i) \cdot x_1^{i_1} x_2^{i_2}...x_n^{i_n}$
Beispiel $2x_1^3x_2 - 4x_2^3$: $(3, 1) \to 2, (0, 3) \to -4$ und alle anderen Paar $(i, j) \to 0$, oder $x_1^2 - x_2^2$: $(2, 0) \to 1, (0, 2) \to -1$ und alle anderen Paar $(i, j) \to 0$ in $\mathbb{Z}[x_1, x_2]$.
Wir können $R[x_1...x_n]$ wieder als Ring auffassen mit:

- 0 als Funktion, die alles auf 0 schickt
- 1 als Funktion mit $(0, ..., 0) \to 1$ und alles andere auf 0
- $-p$ definiert durch $(-p)(m) = -p(m)$
- $p + q$ definiert durch $(p + q)(m) = p(m) + q(m)$
- $p \cdot q$ definiert durch $(p \cdot q)(m) = \sum_{\{(m_1,m_2)|m_1 \cdot m_2 = m\}} p(m_1) \cdot q(m_2)$,
  wobei $(i_1, ..., i_n) \cdot (j_1, ..., j_n) = (i_1 + j_1, ..., i_n + j_n)$

Beispiel(Multiplikation) Betrachten $p = x + y$ und $q = x - y$ über $\mathbb{Z}[x, y]$. Was ist der Vorfaktor von $x \cdot y$ in $p \cdot q$? Intuitiv: $p \cdot q = x^2 + x \cdot y - x \cdot y - y^2 = x^2 - y^2$. Praktisch: $(p \cdot q)((1, 1)) = \sum_{\{(m_1,m_2)|m_1 \cdot m_2 = (1,1)\}} p(m_1) \cdot q(m_2) = p((1, 0)) \cdot q((0, 1)) + p((0, 1)) \cdot q((1, 0)) = 1 \cdot (-1) + 1 \cdot 1 = -1 + 1 = 0$

Implementierung

Wir möchten nun $\mathbb{Q}[x_1, ..., x_n]$ in OCaml implementieren. Dafür repräsentieren wir die Polynome p durch Listen mit Elementen $(c, [i_1; ...; i_n])$, s.d. $p((i_1, ..., i_n)) = c$, falls $c \neq 0$. Damit ist das Nullpolynom die leere Liste. Erstmal definieren wir verschiedene Operationen auf den Monomen. In den Beispielen arbeite ich ab sofort immer über $\mathbb{Q}[x, y]$.

Monom Multiplikation

```
let mmul (c1,m1) (c2,m2) = (c1*/c2,map2 (+) m1 m2);;
```

Beispiel $3x \cdot (-4y^2) = -12xy^2$.
`c1` $= 3$, `m1` $= (1, 0)$, `c2` $= -4$, `m2` $= (0, 2) \rightarrow$
`c1*/c2` $= -12$, `map2 (+) m1 m2` $= (1, 2)$.

Monom Division

```
let mdiv =
 let index_sub n1 n2 = if n1 < n2 then failwith "mdiv" else n1-n2 in
 fun (c1,m1) (c2,m2) -> (c1//c2,map2 index_sub m1 m2);;
```

Beispiel $3x^3y/4xy = \frac{3}{4}x^2$.
`c1` $= 3$, `m1` $= (3, 1)$, `c2` $= 4$, `m2` $= (1, 1) \rightarrow$
`c1//c2` $= 3/4$, `map2 index_sub m1 m2` $= (3 - 1, 1 - 1) = (2, 0)$ und bricht nicht ab, da $3 \geq 1,\ 1 \geq 1$.

Monom ggT

```
let mlcm (c1,m1) (c2,m2) = (Int 1,map2 max m1 m2);;
```

Beispiel $3x^3y$ und $4xy$ wie oben hat ggT xy und `map2 max m1 m2` $= (1, 1)$.

Damit die Listendarstellung eindeutig ist benötigen wir noch eine Ordnung der Monome.
Definition: Sei $\ll$ ein Ordnung der Monome, s.d. $m_1 \ll m_2$, falls $grad(m_1) < grad(m_2)$ oder, falls diese gleich sind, falls $lexord(m_1) > lexord(m_2)$.

Ordnung $\ll$

```
let morder_lt m1 m2 =
 let n1 = itlist (+) m1 0 and n2 = itlist (+) m2 0 in
 n1 < n2 or n1 = n2 & lexord(>) m1 m2;;
```

Beispiel $xy \ll x^3y$, da die Grade unterschiedlich sind und $x^3y \ll xy^3$, da der Exponent von x in x^3y größer ist und damit die lexikographische Ordnung größer ist.
Diese Ordnung ist praktisch, da für Monome m, m_1 und m_2: $m_1 \ll m_2 \Rightarrow m_1 \cdot m \ll m_2 \cdot m$ gilt, d.h. wir müssen Polynome bei Multiplikation nicht neu ordnen.

Monom/Polynom Multiplikation

```
let mpoly_mmul cm pol = map (mmul cm) pol;;
```

Hierbei wird die Monom Multiplikation auf jedes Monom eines Polynoms angewandt.

Polynom Negation

```
let mpoly_neg = map (fun (c,m) -> (minus_num c,m));;
```

Hierbei wird jedes c in den Monomen (c, m) negiert.

Variable

```
let mpoly_var vars x =
  [Int 1,map (fun y -> if y = x then 1 else 0) vars];;
```

Polynom, das nur aus der gewünschten Variable besteht.

Konstantenpolynom

```
let mpoly_const vars c =
  if c =/ Int 0 then [] else [c,map (fun k -> 0) vars];;
```

Polynom, das nur aus dem Monom $(c, (0, ..., 0)$ also c besteht.

Polynom Addition

```
let rec mpoly_add l1 l2 =
 match (l1,l2) with
   ([],l2) -> l2
 | (l1,[]) -> l1
 | ((c1,m1)::o1,(c2,m2)::o2) ->
    if m1 = m2 then
     let c = c1+/c2 and rest = mpoly_add o1 o2 in
     if c =/ Int 0 then rest else (c,m1)::rest
    else if morder_lt m2 m1 then (c1,m1)::(mpoly_add o1 l2)
    else (c2,m2)::(mpoly_add l1 o2);;
```

Beispiel $l1 = 2x^2 + y$, $l2 = x - y \rightarrow \ll$: $l1 = (\ (2,(2,0)),\ (1,(0,1))\ )$, $l2 = (\ (-1,(0,1)),\ (1,(1,0))\ )$.
Vergleich $(c_1, m_1) = (2,(2,0))$ mit $(c_2, m_2) = (-1,(0,1)) \rightarrow m_1 \neq m_2$ und $m_2 \ll m_1 \rightarrow (2,2,0))$ wird vorderstes Monom.
Vergleich $(c_1, m_1) = (1,(0,1))$ mit $(c_2, m_2) = (-1,(0,1)) \rightarrow m_1 = m_2 \ \rightarrow c = c_1 + c_2 =$

$1 - 1 = 0$, also dieses Monom fällt weg.
$l1 = 0$, d.h. es wird $l2 = x$ zurückgegeben. $\rightarrow$ Erhalten ($(2,(2,0)),\ (1,(1,0)) = 2x^2 + x$.

Polynom Substraktion

```
let mpoly_sub l1 l2 = mpoly_add l1 (mpoly_neg l2);;
```

Addition zusammen mit Negation eines der Polynome.

Polynom Multiplikation

```
let rec mpoly_mul l1 l2 =
 match l1 with
   [] -> []
 | (h1::t1) -> mpoly_add (mpoly_mmul h1 l2) (mpoly_mul t1 l2);;
```

Einzelne Multiplikation der Monome des ersten Polynoms mit dem zweiten Polynom.

Potenzieren eines Polynoms

```
let mpoly_pow vars l n =
 funpow n (mpoly_mul l) (mpoly_const vars (Int 1));;
```

Wiederholtes aufrufen der Multiplikation.

Invertieren von Konstantenpolynomen

```
let mpoly_inv p =
 match p with
   [(c,m)] when forall (fun i -> i = 0) m -> [(Int 1 // c),m]
 | _ -> failwith "mpoly_inv:  non-constant polynomial";;
```

Überprüft erst, ob Konstantenpolynom, was ungleich 0 ist, und invertiert dann einfach diese Konstante.

Polynom Division

```
let mpoly_div p q = mpoly_mul p (mpoly_inv q);;
```

Multiplikation mit dem Inversen.

Term $\rightarrow$ Polynom

```
let rec mpolynate vars tm =
 match tm with
   Var x -> mpoly_var vars x
 | Fn("-",[t]) -> mpoly_neg (mpolynate vars t)
 | Fn("+",[s;t]) -> mpoly_add (mpolynate vars s) (mpolynate vars t)
 | Fn("-",[s;t]) -> mpoly_sub (mpolynate vars s) (mpolynate vars t)
 | Fn("*",[s;t]) -> mpoly_mul (mpolynate vars s) (mpolynate vars t)
 | Fn("/",[s;t]) -> mpoly_div (mpolynate vars s) (mpolynate vars t)
 | Fn("^",[t;Fn(n,[])]) ->
       mpoly_pow vars (mpolynate vars t) (int_of_string n)
 | _ -> mpoly_const vars (dest_numeral tm);;
```

Überführt die entsprechenden Funktionssymbole rekursiv in die korrespondierenden Operationen auf Polynomen.

$\mathbf{s = t \ \rightarrow \ s - t}$

```
let mpolyatom vars fm =
 match fm with
   Atom(R("=",[s;t])) -> mpolynate vars (Fn("-",[s;t]))
 |_ -> failwith "mpolyatom:  not an equation";;
```

Überführt die Frage nach Gleichheit eines Terms, in die Frage, ob ein gegebenes Polynom 0 ist.
Anmerkung Im weiteren schreiben wir $norm(s)$ für `mpolynate vars s` und $s \approx t$ für $norm(s) = norm(t)$, also falls s und t das selbe Polynom definieren.