Gröbner-Basen
Algorithmen in OCaml

Jonas Lippert

19. Juni 2020

Inhaltsverzeichnis

1 Abstrakt

Wir haben im vorherigen Vortrag gesehen, dass sich die Gültigkeit universeller komplexer Formeln in der Sprache der Ringe $\{+, \cdot, 0, 1, =\}$ auf Idealzugehörigkeitsprobleme im Polynomring über $\mathbb{Q}$ zurückführen lassen. In diesem Vortrag sollen die Lösung solcher IZP mit Hilfe von Gröbner-Basen nach Buchberger sowie die Implementierung der Lösungsalgorithmen in OCaml diskutiert werden.

Die das Ideal aufspannenden Polynome definieren eine Reduktionsrelation im Sinne einer verallgemeinerten Modulo-Rechnung. Wird ein gegebenes Polynom zu 0 reduziert, ist es im Ideal enthalten. Eine Reduktionsrelation heißt konfluent, wenn sie jedes Polynom zu einer eindeutigen Normalform reduziert. In diesem Fall gilt auch die Umkehrung und wir nennen die aufspannenden Polynome eine Gröbner-Basis. Wir werden sehen, dass die Reduktion immer terminiert und dass jedes Ideal eine Gröbner-Basis besitzt.

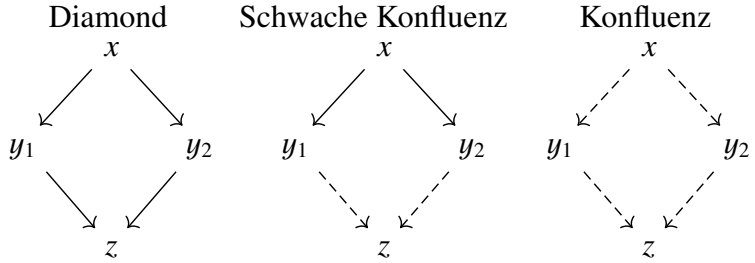
2 Polynomreduktion

2.1 Reduktionsrelationen im Allgemeinen

Eine **Reduktionsrelation** $\to$ ist eine binäre Relation auf einer Menge X. Wir bezeichnen mit $\to^*$ den transitiven Abschluss, das heißt, es gilt $x \to^* y$ genau dann, wenn eine (möglicherweise leere) Sequenz $x_i \in X$ existiert, sodass $x = x_1 \to ... \to x_n = y$. Ein $x \in X$ ist in **Normalform**, falls kein $y \in X$ existiert mit $x \to y$. Die Relation **terminiert**, falls keine unendliche Reduktionssequenz existiert. Die Relation besitzt die **Diamond Eigenschaft**, falls gilt:

$$x \to y_1 \wedge x \to y_2 \Rightarrow \exists z : \; y - 1 \to z \wedge y_2 \to z.$$

Sie ist **konfluent**, falls $\to^*$ die Diamond Eigenschaft besitzt. Weiter seien x und y **zusammenführbar**, $x \downarrow y$, falls ein z existiert, sodass $x \to^* z$ und $y \to^* z$. Wir sagen, $\to$ ist **schwach konfluent**, wenn aus $x \to y$ und $x \to y'$ folgt, dass $y \downarrow y'$.



Satz 2.1. *Es sei $\to$ eine terminierende Reduktionsrelation. Dann sind folgende Aussagen äquivalent:*

(i) *$\to$ ist konfluent.*

(ii) *$\downarrow$ ist transitiv.*

(iii) *Die Normalformen sind eindeutig.*

(iv) *$\to$ ist schwach konfluent. [Die Implikation $(iv) \Rightarrow (i)$ trägt den Namen „Newman's Lemma"]*

Beweis.

(i)⇒(ii) Es gelte $x \downarrow_a y \downarrow_b z$. Aus $y \to^* a$ und $y \to^* b$ folgt mit Konfluenz $a \downarrow_c b$ und damit $x \downarrow_c z$.

(ii)⇒(iii) Es seien y und z zwei Normalformen für x. Dann gilt $y \downarrow_y x$ und $x \downarrow_z z$, also auch $y \downarrow z$ und damit $y = z$, denn Zusammenführbarkeit von Normalformen ist nur bei Gleichheit gegeben.

(iii)⇒(i) Es sei x nach y und z reduzierbar. Da $\to$ terminiert, können wir o.E. annehmen, dass diese bereits in Normalform sind. Dann gilt nach Eindeutigkeit $y = z$ und damit $y \downarrow z$.

(i)⇒(iv) Klar.

(iv)⇒(iii) Terminierende Reduktionsrelationen sind wohlfundiert, sodass wir Induktion über die Reduktionsketten verwenden können. Es seien y und y' zwei Normalformen für x. Falls $x = y = y'$, sind wir fertig. Ansonsten betrachte $x \to r \to^* y$ und $x \to r' \to^* y'$. Die schwache Konvergenz liefert $r \downarrow_{z'} r'$. Betrachte eine Normalform z von z'. Nach Induktionsvoraussetzung ist z die eindeutige Normalformen von r und r'. Da y und y' schon in Normalform sind, erhalten wir $y = z = y'$.

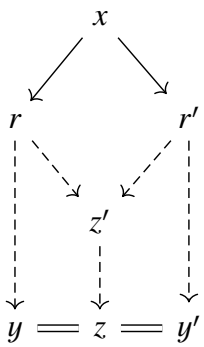


Abbildung 1: Induktionsschritt

□

2.2 Der Reduktions-Algorithmus

Es sei ein Ideal $\langle F \rangle$ gegeben, wobei $F = \{p_1 = m_1 + p_1', ..., p_n = m_n + p_n'\}$. Hier sei m_i dasjenige Monom in p_i, welches den höchsten Grad bzgl. `morder_lt` hat. Dies definiert eine Reduktionsrelation via

$$p \to_F q :\Leftrightarrow p \text{ lässt sich via } F \text{ (in einem Schritt) zu } q \text{ reduzieren.}$$

Ein Schritt bedeutet:

(i) Ein Monom m in p wird als Vielfaches eines der m_i identifiziert.

(ii) Ersetze m in p durch $\frac{m}{m_i} \cdot (-p_i')$.

Das nachfolgende Programm wiederholt diesen Schritt, bis keine Reduktion mehr möglich ist: In `reduce1` wird zunächst das Monom `cm` durch das Polynom `pol` wie oben beschrieben reduziert:

```
let reduce1 cm pol =
   match pol with
    [] -> failwith "reduce1"
   | hm::cms -> let c,m = mdiv cm hm in mpoly_mmul (minus_num c,m) cms;;
```

In `reduce2` wird aus einer Liste `pols` von Polynomen durch den Operator `tryfind` das erste ausgewählt, das von `reduce1` auf `cm` angewendet werden kann:

```
let reduce2 cm pols = tryfind (reduce1 cm) pols;;
```

Schließlich reduziert `reduce` das Polynom `pol` mittels `pols` zu einer Normalform:

```
let rec reduce pols pol =
   match pol with
    [] -> []
   | cm::ptl -> try reduce pols (mpoly_add (reduce2 cm pols) ptl)
      with Failure _ -> cm::(reduce pols ptl);;
```

Satz 2.2.

(i) Der Algorithmus liefert eine Normalform, deren Monome weiterhin durch `morder_lt` *geordnet sind.*

(ii) Der Algorithmus terminiert.

(iii) Falls `pol` *zu* `[]` *reduziert wird, ist es in dem von* `pols` *aufgespannten Ideal enthalten.*

Beweis.

(i) Jedes Monom im Zielpolynom wurde bereits auf Reduzierbarkeit getestet. Durch `mpoly_add` ist die Sortierung der Monome nach `morder_lt` gewährleistet, wie wir im vorherigen Vortrag gesehen haben.

(ii) Bei jedem Schritt wird der Faktor `hm` eines Monoms in `pol` durch `-cms` ersetzt. Die Monome eines Polynoms sind nach Konstruktion absteigend nach ihrem Grad sortiert, sodass `-cms` nur aus Monomen besteht, deren Grad kleiner ist als der von `hm`.

(iii) (a) Falls $p \to q$, so gilt $p - q \in Id_{\mathbb{Q}}\langle F\rangle$.

Beweis. Es sei $p_1 = m_1 - p_1' \in F$ die Rewriting-Regel, welche $p = m'm_1 + r$ zu $q = m'p_1' + r$ reduziert. Dann gilt

$$p - q = (m'm_1 + r) - (m'p_1' + r) = m'(m_1 - p_1') = m'p_1 \in Id_{\mathbb{Q}}\langle F\rangle.$$

(b) **Lemma 2.3.** *Falls $p \to^* q$, so gilt $p - q \in Id_{\mathbb{Q}}\langle F\rangle$.*

Beweis. Falls $p = q$ sind wir fertig, da jedes Ideal die 0 enthält. Betrachte die Kette $p = r_1 \to \ldots \to r_n = q$. Dann ist

$$p - q = \sum_{i=1}^{n-1} r_i - r_{i+1}.$$

Nach (a) ist $r_i - r_{i+1} \in Id_{\mathbb{Q}}\langle F\rangle$ für alle $1 \le i \le n$. Die Aussage folgt aus der Abgeschlossenheit von Idealen bzgl. Addition. □

Mit $q = 0$ folgt die Behauptung. □

Es ist noch nicht sichergestellt, dass jedes Polynom in $\langle F \rangle$ zu Null reduziert wird, wie das folgende Beispiel zeigt.

Beispiel. *Im Allgemeinen gilt nicht* $p_i \to^* 0 \Rightarrow \sum p_i \to^* 0$.
Beweis: Betrachte das Ideal $\langle p_1 = x + 1, p_2 = x - 1 \rangle$. *Dann ist* $\sum p_i = 2x$, *was zu* -2 *und* 2 *reduzierbar ist, jedoch nicht zu* 0; $x \pm 1$ *via* $x = \mp 1$ *hingegen schon.*

Dass dies genau dann der Fall, wenn die durch F induzierte Relation konfluent ist, ist die Aussage des folgenden Theorems:

Theorem 2.4. *Folgende Aussagen sind äquivalent:*

(i) $F = \{p_1, ...p_n\}$ *ist eine* ***Gröbner-Basis*** *für* $Id_{\mathbb{Q}}\langle F \rangle$, *das heißt* $\to_F$ *ist konfluent.*

(ii) Für alle Polynome p *gilt* $p \to_F^* 0$ *genau dann, wenn* $p \in Id_{\mathbb{Q}}\langle F \rangle$.

(iii) Für alle Polynome p *und* q *gilt* $p \downarrow_F q$ *genau dann, wenn* $p - q \in Id_{\mathbb{Q}}\langle F \rangle$.

Anmerkung. *Da Konfluenz äquivalent zur Eindeutigkeit der Normalform ist, liefert* $p \to^* 0$ *in (ii) eine hinreichende Bedingung dafür, dass* `pol` *tatsächlich durch den obigen Algorithmus zu* `[]` *reduziert werden würde.*

Beweis. Die Hinrichtung in (ii) wurde schon in Satz 2.2 gezeigt. Hieraus folgt auch die Hinrichtung in (iii): Nach Voraussetzung existiert ein r, sodass

$$p \to^* r \text{ und } q \to^* r.$$

Das heißt, es gilt mit Lemma 2.3

$$p - r \in Id_{\mathbb{Q}}\langle F \rangle \text{ und } q - r \in Id_{\mathbb{Q}}\langle F \rangle.$$

Die Abgeschlossenheit von $Id_{\mathbb{Q}}\langle F \rangle$ bzgl. Addition liefert

$$p - q = (p - r) - (q - r) \in Id_{\mathbb{Q}}\langle F \rangle.$$

(i)⇒(ii) Sei $p \in Id_{\mathbb{Q}}\langle F \rangle$, dh. $p = \sum a_i p_i = \sum a_i (m_i' - p_i')$ für entsprechende Polynomfaktoren a_i und $p_i = m_i' - p_i' \in F$. Mittels $m_i' \to p_i'$ sind die Summanden zu 0 reduzierbar. Um zu sehen, dass dann auch die Summe zu 0 reduzierbar ist, benötigen wir vier Hilfsresultate:

Lemma 2.5. *Falls* $p \to q$, *gilt* $p + r \downarrow q + r$.

Beweis. Es sei die Reduktion $p \to q$ durch die Reduktion von m in $p = m + p'$ zu s nach $q = s + p'$ gegeben. Sei nun a der Koeffizient von m in r, sodass $r = am + r'$. Dann haben wir

$$p + r = m + p' + am + r' = (a + 1)m + p' + r' \to^* (a + 1)s + p' + r'$$

und

$$q + r = s + p' + am + r' \to^* s + p' + as + r' = (a + 1)s + p' + r'.$$

Lemma 2.6. *Es sei $\to$ konfluent und es gelte $p \to^* q$. Dann gilt $p + r \downarrow q + r$.*

Beweis. Betrachte die (möglicherweise leere) Reduktionssequenz $p = s_1 \to ... \to q = s_n$. Wir wissen nach Lemma 2.5, dass $s_i + r \downarrow s_{i+1} + r$ gilt für alle $1 \leq i \leq n-1$. Da Konfluenz äquivalent zur Transitivität von $\downarrow$ ist, sind wir fertig. □

Lemma 2.7. *Es sei $\to$ konfluent und es gelte $p \downarrow_s q$. Dann gilt $p + r \downarrow q + r$.*

Beweis. Wir wollen per Induktion über $p \to^* s$ zeigen, dass für alle $q \to^* s$ gilt $p + r \downarrow q + r$. Falls $p = s$, gilt $q \to^* p$ und wir können Lemma 2.6 verwenden. Ansonsten sei $p \to p' \to^* s$ gegeben. Nach Lemma 2.5 gilt $p + r \downarrow p' + r$. Nach Induktionsvoraussetzung gilt $p' + r \downarrow q + r$. Wir können die Transitivität von $\downarrow$ benutzen und sind fertig. □

Lemma 2.8. *Es sei $\to$ eine konfluente Polynomreduktion. Dann folgt aus $p \downarrow q$ und $r \downarrow s$, dass $p + r \downarrow q + s$.*

Beweis. Wir wenden zweimal Lemma 2.7 an und erhalten

a) $p + r \downarrow q + r$

b) $q + r \downarrow q + s$.

Die Aussage folgt wieder aus der Transitivität von $\downarrow$. □

Insbesondere gilt $p \to^* 0$, $q \to^* 0 \Rightarrow p + q \to^* 0$ und die Aussage folgt per Induktion.

(ii)⇒(iii) Folgt direkt aus dem folgenden Resultat:

Lemma 2.9. *Falls $p - q \to^* 0$, gilt auch $p \downarrow q$.*

Beweis. Per Induktion über die Reduktionssequenz: Falls $p - q = 0$, sind wir fertig. Sei also die Sequenz $p - q \to r \to^* 0$ gegeben. Der Schritt nach r sei induziert durch die Regel $m = s$. Seien a und b die Koeffizienten von m in p und q. Damit erhalten wir

$$\begin{aligned} p &= am + p_1, \\ q &= bm + q_1, \\ p - q &= (a-b)m + (p_1 - q_1), \\ r &= (a-b)s + (p_1 - q_1). \end{aligned}$$

Außerdem können wir p und q reduzieren und erhalten

$$\begin{aligned} p &\to^* p' = as + p_1, \\ q &\to^* q' = bs + q_1, \\ p' - q' &= (a-b)s + (p_1 - q_1) = r \to^* 0. \end{aligned}$$

Nach Induktionsvoraussetzung folgt $p' \downarrow q'$ und damit auch $p \downarrow q$. □

(iii)⇒(i) In (iii) folgt aus der Transitivität der durch die rechte Seite induzierten Relation die Transitivität von $\downarrow_F$ und daraus die Konfluenz von $\to_F$. □

3 Konstruktion von Gröbner-Basen

Da Konfluenz notwendig ist, um den obigen Algorithmus zur Lösung von IZP verwenden zu können, sind die Fragen zu beantworten, wie Konfluenz zu erkennen und idealerweise auch zu erzeugen ist. Um das herauszufinden, ist zunächst zu klären, wie Nicht-Konfluenz entstehen kann.

3.1 Konfluenz und S-Polynome

Lemma 3.1. *Nicht-Konfluenz kann auf zwei Arten entstehen:*

(i) Mehrere Monome in p lassen sich reduzieren.

(ii) Für ein Monom in p stehen mehrere Reduktionen zur Verfügung.

Im ersten Fall ist die Zusammenführbarkeit jedoch stets gegeben.

Beweis. Der Algorithmus wählt für einen Reduktionsschritt ein Monom in `pol` sowie ein Leitmonom in `pols` aus. Dementsprechend gibt es keine weitere Möglichkeit.

Im ersten Fall sei nun o.E. die Situation $p = m_1 + m_2 + r$ zusammen mit den Rewriting-Regeln $m_1 = p_1$ und $m_2 = p_2$ gegeben. Wir erhalten $p \to q_1 = p_1 + m_2 + r$ und $p \to q_2 = m_1 + p_2 + r$. Für die offensichtliche Lösung, nun q_1 und q_2 zu $p_1 + p_2 + r$ zusammenzuführen, muss der Fall genauer untersucht werden, dass o.B.d.A m_2 in $p_1 = am_2 + s$ vorkommt. Wir rechnen nach:

$$\begin{aligned} q_1 &= p_1 + m_2 + r \\ &= (a+1)m_2 + s + r \\ &\to^* (a+1)p_2 + s + r. \end{aligned}$$

$$\begin{aligned} q_2 &= m_1 + p_2 + r \\ &\to p_1 + p_2 + r \\ &= am_2 + s + p_2 + r \\ &\to^* ap_2 + s + p_2 + r \\ &= (a+1)p_2 + s + r. \end{aligned}$$

An zwei Stellen wurde $\to^*$ verwendet für den Fall, dass beispielsweise $(a+1)m_2$ schon 0 ist, da $\to$ nicht für leere Sequenzen definiert ist. □

Im zweiten Fall geht es im Kern um die Frage, ob für gegebene Rewriting-Regeln $m_1 = p_1'$ und $m_2 = p_2'$ die Differenz der Reduktionen des kleinsten gemeinsamen Vielfachen $LCM(m_1, m_2)$ der beiden Leitmonome zu 0 reduzierbar ist.

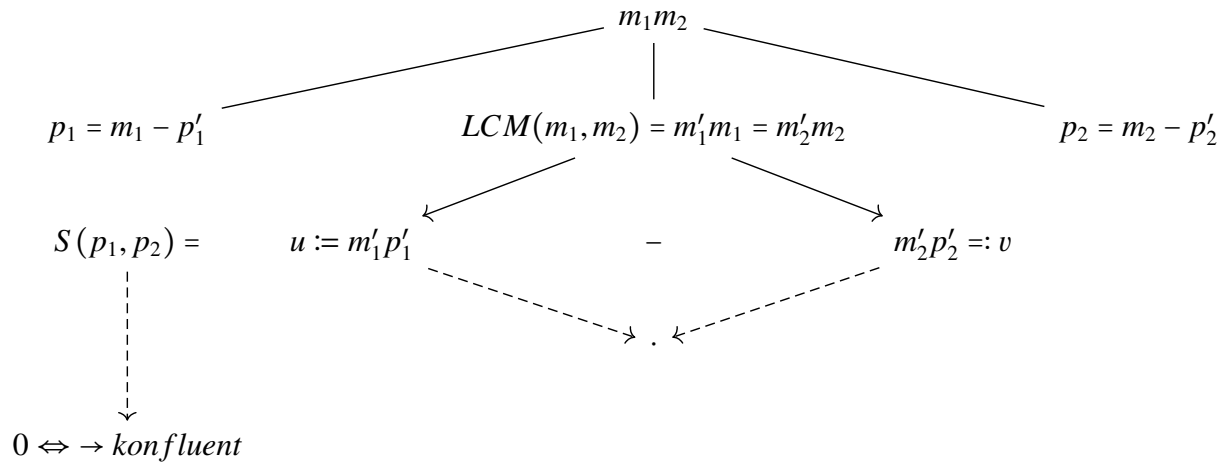


Abbildung 2: Theorem 3.2

Entsprechend sei für zwei Polynome $p_1 = m_1 + p_1'$ und $p_2 = m_2 + p_2'$ das **S-Polynom** wie folgt definiert:

$$S(p_1, p_2) := m_1'p_1' - m_2'p_2',$$

wobei $LCM(m_1, m_2) = m_1'm_1 = m_2'm_2$.

Anmerkung. Sind die $p_i = m_i + p_i'$ im Ideal enthalten, so ist es auch deren S-Polynom:

$$\begin{aligned} S(p_1, p_2) &:= m_1'p_1' - m_2'p_2' \\ &= m_1'(p_1 - m_1) - m_2'(p_2 - m_2) \\ &= (m_1'p_1 - m_1'm_1) - (m_2'p_2 - m_2'm_2) \\ &= (m_1'p_1 - LCM(m_1, m_2)) - (m_2'p_2 - LCM(m_1, m_2)) \\ &= m_1'p_1 - m_2'p_2. \end{aligned}$$

Die S-Polynome liefern ein Entscheidungskriterium für Konfluenz in folgendem Sinne:

Theorem 3.2. *Die durch eine Menge F von Polynomen definierte Reduktionsrelation $\rightarrow_F$ ist genau dann konfluent, wenn für je zwei Polynome $p_1, p_2 \in F$ gilt $S(p_1, p_2) \rightarrow_F^* 0$.*

Beweis.

„⇒“ Wir brauchen ein Hilfsresultat:

Korollar 3.3. *Es sei $\rightarrow$ konfluent und $p \downarrow q$ gegeben. Dann gilt $p - q \rightarrow^* 0$.*

Beweis. Verwende Lemma 2.7 mit r gleich $-q$. □

Betrachte $S(p_1, p_2) = p_1'm_1' - p_2'm_2'$. Wir wollen Korollar 3.3 benutzen. Das heißt, es ist die Zusammenführbarkeit von p_1m_1' und q_1m_2' zu zeigen. Dies folgt jedoch unmittelbar aus der Konfluenz, denn beide Polynome sind Reduktionen von $LCM(m_1, m_2)$.

„⇐“ Nach Lemma 3.1 kann Nicht-Konfluenz nur auftreten, wenn für ein Monom m in $p = m + p'$ zwei Rewriting-Regeln $m_1 = p_1'$ und $m_2 = p_2'$ existieren. Als Vielfaches von m_1 und m_2 ist m auch Vielfaches von $LCM(m_1, m_2)$. Dann wird

$$\begin{aligned} p &= m'\mathrm{LCM}(m_1, m_2) + p' \\ &= m'm_1'm_1 + p' \\ &= m'm_2'm_2 + p' \end{aligned}$$

reduziert zu

$$m'm_1'p_1' + p' \text{ und } m'm_2'p_2' + p'.$$

Behauptung: Falls $p \to^* q$, dann gilt für jedes Monom m auch $mp \to^* mq$.

Konsequenz: Nach Voraussetzung gilt

$$\begin{aligned} & m_1'p_1' - m_2'p_2' = S(m_1 - p_1', m_2 - p_2') \to^* 0 \\ \Rightarrow\ & m'm_1'p_1' - m'm_2'p_2' \to^* 0 \\ \Rightarrow\ & (m'm_1'p_1' + p') - (m'm_2'p_2' + p') \to^* 0 \\ \Rightarrow\ & m'm_1'p_1' + p' \downarrow m'm_2'p_2' + p', \end{aligned}$$

was zu zeigen war. Im letzten Schritt wurde Lemma 2.9 benutzt. Wir haben also schwache Konfluenz gezeigt und können Newman's Lemma verwenden.

Beweis der Behauptung. Man beachte, dass die Aussage im Falle $m = 0$ trivialerweise gilt. Wir zeigen

$$p \to q \Rightarrow mp \to mq.$$

Wiederholtes Anwenden dieser Regel liefert die Behauptung.
Es sei $p_1 = m_1 - p_1' \in F$ die Rewriting-Regel, welche $p = m'm_1 + r$ zu $q = m'p_1' + r$ reduziert. Dann ist

$$mp = m(m'm_1 + r) = mm'm_1 + mr.$$

sodass wir wie gewünscht zu

$$mm'p_1' + mr = m(m'p_1' + r) = mq$$

reduzieren können. □

Beispiel. *Betrachte wieder das Ideal $\langle p_1 = x + 1, p_2 = x - 1\rangle$. Wir haben gesehen, dass dies keine Gröbner-Basis ist. Das S-Polynom ist mit $LCM(x, x) = x$ entsprechend*

$$S(p_1, p_2) = \frac{x}{x}(x + 1) - \frac{x}{x}(x - 1) = 2.$$

3.2 Buchberger's Algorithmus

Mit Hilfe der S-Polynome ist es darüber hinaus auch möglich, Konfluenz zu erzeugen. Sie sind folgendermaßen implementiert:

```
let spoly pol1 pol2 =
   match (pol1,pol2) with
      ([],p) -> []
      | (p,[]) -> []
      | (m1::ptl1,m2::ptl2) ->
         let m = mlcm m1 m2 in
         mpoly_sub (mpoly_mmul (mdiv m m1) ptl1)
               (mpoly_mmul (mdiv m m2) ptl2);;
```

Buchberger's Algorithmus `groebner` erzeugt damit aus `basis` eine Gröbner-Basis. Die Idee ist recht einfach: Die Reduktion der S-Polynome je zweier Basispolynome wird, falls ungleich `[]`, zur Basis hinzugefügt. Dadurch werden diese S-Polynome per Konstruktion zu `[]` reduzierbar. Dass dieses Vorgehen terminiert und das Ideal unverändert lässt, wird nachfolgend in Satz 3.5 ausgeführt. Die Zeile

```
else if forall (forall ((=) 0) ** snd) sp then [sp] else
```

stoppt den Algorithmus, falls ein S-Polynom zu einer Konstanten reduziert wird. Das eigentliche Programm `grobner` wird durch `groebner` mit allen Paaren aus `pairs` aufgerufen, wobei aus Symmetriegründen jeweils nur $S(p,q)$ oder $S(q,p)$ benötigt wird.

```
let groebner basis = grobner basis (distinctpairs basis);;

let rec grobner basis pairs =
   print_string(string_of_int(length basis)^" basis elements and "^
            string_of_int(length pairs)^" pairs");
   print_newline();
   match pairs with
      [] -> basis
      | (p1,p2)::opairs ->
         let sp = reduce basis (spoly p1 p2) in
         if sp = [] then grobner basis opairs
         else if forall (forall ((=) 0) ** snd) sp then [sp] else
         let newcps = map (fun p -> p,sp) basis in
         grobner (sp::basis) (opairs @ newcps);;
```

Satz 3.4. *Der Algorithmus terminiert (i) und liefert eine Gröbner-Basis (ii) des ursprünglichen Ideals (iii).*

Beweis.

(i) Angenommen der Algorithmus terminiert nicht. Dann wird die letzte Zeile von `grobner` unendlich oft aufgerufen. Da `sp` bereits durch die Polynome in `basis` reduziert wurde,

ist kein Monom in `sp` durch ein Leitmonom der Polynome in `basis` teilbar. Es ließe sich also eine Folge von Monomen (m_i) konstruieren, sodass m_i nicht durch m_j teilbar ist für alle $j < i$. Wähle hierzu die Folge der Leitmonome der neu auftretenden `sp`. Da $x_1^{n_1}...x_l^{n_l}$ durch $y_1^{m_1}...y_1^{m_1}$ teilbar ist genau dann, wenn $m_i \leq n_i$ für alle $1 \leq i \leq l$, stünde dies im Widerspruch zum folgenden Lemma.

Lemma 3.5 (Dickson's Lemma). *Betrachte die Ordnung $\leq_n$ auf $\mathbb{N}^n$ definiert durch*

$$(x_1, ..., x_n) \leq_n (y_1, ..., y_n) :\Leftrightarrow x_i \leq_n y_i \text{ für alle } 1 \leq i \leq n.$$

Dann existiert keine unendliche Kette (t_i) mit Elementen aus $\mathbb{N}^n$, sodass $t_i \not\leq_n t_j$ für alle $i < j$.

Beweis. Falls $n = 1$, ist die Aussage äquivalent zur Wohlfundiertheit der natürlichen Zahlen. Angenommen es gäbe eine Folge $(t_i = (a_i, s_i))_i$ mit $a_i \in \mathbb{N}$ und $s_i \in \mathbb{N}^n$, sodass $t_i \not\leq_{n+1} t_j$ für alle $i < j$. Wir können annehmen, dass die Folge in (a_i) minimal ist. Nach Induktionsvoraussetzung gibt es $k < l$, sodass $s_k \leq_n s_l$, das heißt es gilt $a_k > a_l$. Wir können jetzt eine neue Folge konstruieren, indem wir den Abschnitt $t_k, ..., t_{l-1}$ entfernen. Dies steht aber im Widerspruch zur Minimalität von a_k. □

(ii) Der Algorithmus stoppt erst, wenn die S-Polynome aller Paare aus `pairs` zu `[]` reduziert werden konnten, bzw. wenn ein `sp` konstant ist. In diesem Fall ist die Idealzugehörigkeit immer gegeben. Außerdem gilt trivialerweise, dass `spoly p1 p2` in `sp::basis` zu `[]` reduzierbar ist. Dies rechtfertigt, dass im Rekursionsschritt nur `opairs @ newcps` anstatt `basis @ newcps` übergeben wird. Das Symmetrieargument bzgl. `distinctpairs` und `newcps` ist dadurch gerechtfertigt, dass Multiplikation mit Konstanten die Reduzierbarkeit nicht beeinflusst. Da also schließlich eine Basis vorliegt, die alle S-Polynome zu `[]` reduziert, folgt die Aussage aus Theorem 3.2.

(iii) Es wurde schon gezeigt, dass $S(p, q)$ in dem von `basis` aufgespannten Ideal enthalten ist für alle $p, q \in$ `basis`. Reduktion ändert nicht die Idealzugehörigkeit: Dazu sei ein Polynom p in einem Ideal $\langle p_i = m_i + p_i' \rangle$ von der Form $p = \sum q_i p_i = \sum a_j m_j$, wobei die m_j die in $p_1, ..., p_n$ auftretenden Monome seien. Ein Rewrite-Schritt reduziert dann ein m_k zu $-p_k' = m_k - p_k$, sodass $p \to [\sum a_j m_j] - a_k p_k = [\sum q_i p_i] - a_k p_k \in I$. Der Algorithmus fügt nur solche reduzierten S-Polynome zur Basis hinzu. □

Beispiel. *Für unser Ideal $I = \langle p_1 = x + 1, p_2 = x - 1 \rangle$ würde also das konstante Polynom 2 ausgegeben werden, sodass $p_1 + p_2 \to^* 0$ trivialerweise gilt.*

4 Anwendung

4.1 Lösungsalgorithmus für universelle komplexe Formeln

Eine universelle Formel fm in der Sprache der Ringe gilt genau dann in $\mathbb{C}$, wenn aus deren Negation $1 = 0$ gefolgert werden kann. Im vorherigen Vortrag wurde gezeigt, dass die (mit Hilfe des Rabinowitsch-Tricks normierten) Literale $p_i(\bar{x}) = 0$ innerhalb einer Klausel der DNF von fm den Sachverhalt $q(\bar{x}) = 0$ genau dann implizieren, wenn ein r existiert, sodass $q^r \in Id_{\mathbb{Q}}(p_1, ..., p_n)$. In unserem Fall ist demnach für die gegebene Klausel zu prüfen, ob $1 \in Id_{\mathbb{Q}}(p_1, ..., p_n)$. Das Entscheidungsproblem lässt sich also mit Hilfe unserer Algorithmen `reduce` und `groebner` lösen.

Zunächst wird `fm` in Pränexnormalform gebracht, und mit `specialize` werden die (All-)Quantoren gelöscht. Anschließend wird die Negation in disjunktiver Normalform klauselweise an das eigentliche Programm `grobner_trivial` übergeben und gefragt, ob jede Klausel zum Widerspruch führt.

```
let grobner_decide fm =
   let fm1 = specialize(prenex(nnf(simplify fm))) in
   forall grobner_trivial (simpdnf(nnf(Not fm1)));;

let grobner_trivial fms =
   let vars0 = itlist (union ** fv) fms []
   and eqs,neqs = partition positive fms in
   let rvs = map (fun n -> variant ("_"^string_of_int n) vars0)
                 (1--length neqs) in
   let vars = vars0 @ rvs in
   let poleqs = map (mpolyatom vars) eqs
   and polneqs = map (mpolyatom vars ** negate) neqs in
   let pols = poleqs @ map2 (rabinowitsch vars) rvs polneqs in
   reduce (groebner pols) (mpoly_const vars (Int 1)) = [];;
```

Die Liste `vars0` der freien Variablen wird benötigt, um anschließend für jede negierte Gleichung eine neue Variable zu generieren. Diese neuen Variablen werden benötigt, um den Rabinowitsch-Trick anzuwenden. Mit Hilfe dessen lassen sich durch Einführung einer neuen Variablen Ungleichungen zu Gleichungen umformen, wie im vorherigen Vortrag gezeigt wurde. Anschließend werden die positiven und die negierten negativen Formeln zu Polynomen transformiert. Diese Listen werden in `pol` vereinigt, wobei die negativen Formeln nach Rabinowitsch zu $1 - zq(\bar{x})$ umgeformt werden:

```
let rabinowitsch vars v p =
   mpoly_sub (mpoly_const vars (Int 1))
   (mpoly_mul (mpoly_var vars v) p);;
```

In der letzten Zeile wird dann geprüft, ob die 1 bzgl. der durch `pols` induzierten Gröbner-Basis zu 0 reduziert werden kann.

4.2 Beispiele

Beispiel. Wir wollen den Algorithmus mit folgendem Beispiel ausprobieren:

$$a^2 = 2 \wedge x^2 + ax + 1 = 0 \Rightarrow x^4 + 1 = 0.$$

Wir erhalten die Zwischenresultate

(i)
```
simpdnf(nnf(Not(specialize(prenex(nnf(simplify
<<a^2 = 2 /\ x^2 + a*x + 1 = 0 ==> x^4 + 1 = 0>>
))))))) =
[[<<x^2 + a * x + 1 = 0>>; <<a^2 = 2>>; <<~x^4 + 1 = 0>>]]
```

(ii)
```
fms = [<<x^2 + a * x + 1 = 0>>; <<a^2 = 2>>; <<~x^4 + 1 = 0>>].
```

(iii)
```
vars = ["a"; "x"; "_1"].
```

(iv)
```
pols =[[(1, [0; 2; 0]); (1, [1; 1; 0]); (1, [0; 0; 0])];
       [(1, [2; 0; 0]); (-2, [0; 0; 0])];
       [(-1, [0; 4; 1]); (-1, [0; 0; 1]); (1, [0; 0; 0])]]
```

$\cong [x^2 + ax + 1; a^2 - 2; -x^4z - z + 1]$.

(v)
```
groebner pols = [[(1, [0; 0; 0])]].
```

Das Ergebnis ist

```
# grobner_decide
<<a^2 = 2 /\ x^2 + a*x + 1 = 0 ==> x^4 + 1 = 0>>;;
3 basis elements and 3 pairs
3 basis elements and 2 pairs
- : bool = true.
```

Beispiel. Aussagen der analytischen Geometrie sind ebenfalls mit Hilfe von Gröbner-Basen entscheidbar, wenn diese durch universelle Formeln über Punkte in einem Koordinatensystem ausgedrückt werden können. Mittelpunkte, Rechtwinkligkeit und Längengleichheit sind wie folgt formalisierbar:

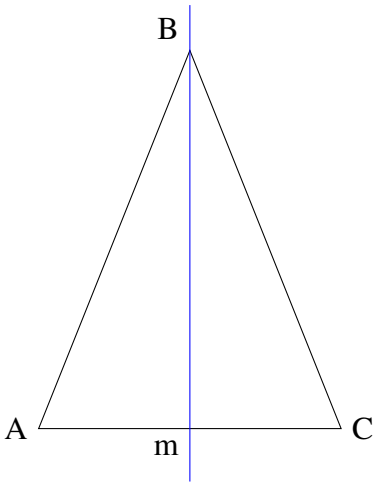
```
"is_midpoint", (** Point 1 is the midpoint of line (2,3) **)
<<2 * 1_x = 2_x + 3_x /\ 2 * 1_y = 2_y + 3_y>>;

"perpendicular", (** Lines (1,2) and (3,4) are perpendicular **)
<<(1_x - 2_x) * (3_x - 4_x) + (1_y - 2_y) * (3_y - 4_y) = 0>>;

"lengths_eq", (** Lines (1,2) and (3,4) have the same length **)
<<(1_x - 2_x)^2 + (1_y - 2_y)^2 = (3_x - 4_x)^2 + (3_y - 4_y)^2>>.
```

Damit lässt sich die folgende Aussage über Dreiecke formulieren: Verläuft die Höhe eines Punktes durch den Mittelpunkt der gegenüberliegenden Seite, so ist das Dreieck gleichschenklig:

```
# (grobner_decide ** originate)
   <<is_midpoint(m,A,C) /\ perpendicular(A,C,m,B)
      ==> lengths_eq(A,B,B,C)>>;;
...
- : bool = true
```



Man beachte, dass es sich hier eigentlich um Aussagen über $\mathbb{R}$ handelt, die aber natürlich gelten, wenn sie auch in $\mathbb{C}$ wahr sind. Durch `originate` werden die Polynome normiert und etwas umgeformt, damit sie als Formel von `grobner_decide` gelesen werden können. Auf die technischen Details wird hier verzichtet.