NTNU
Det skapende universitet

Gröbnerbasis: Algoritmer og kryptografi

Kristin Krogh Arnesen

Master i matematikk
Oppgaven levert: Mai 2010
Hovedveileder: Aslak Bakke Buan, MATH

Norges teknisk-naturvitenskapelige universitet
Institutt for matematiske fag

Sammendrag

Oppgaven er delt inn i tre deler, der de to siste kan leses uavhengige av hverandre, men begge bygger på den første. Felles for dem alle er temaet *Gröbnerbasiser*. De tre delene er:

I – En innføring i Gröbnerbasiser. Gitt en polynomring R over en kropp med et endelig antall variabler og et ideal I i denne kroppen, finnes det da en unik normalform av et element f i R i faktorringen R/I? Dette problemet løses ved hjelp av en Gröbnerbasis G, der G genererer I, som er slik at når vi bruker den generaliserte divisjonsalgoritmen og deler f med elementene i G, får vi en unik rest uansett hvilken rekkefølge vi har på elementene i G.

Dette kan brukes til å løse en rekke problemer, som for eksempel hvorvidt et element f i R er i idealet I, eller om to delmengder $F_1 \subseteq I$ og $F_2 \subseteq I$ genererer samme ideal – og det kan også være et nyttig redskap for å løse likningssett der likningene er på formen $f_i = 0$ og f_i'ene er polynomer i R.

R kan være både en kommutativ og en ikke-kommutativ ring. Både i det kommutative og det ikke-kommutative tilfellet har vi at det alltid finnes en Gröbnerbasis for et endeliggenerert ideal, men bare i det kommutative tilfellet kan vi garantere at det finnes en endelig sådan. I begge tilfellene viser vi at *Buchbergers algoritme* finner den ønskede Gröbnerbasisen.

II – Algoritmer for å finne Gröbnerbasiser. Buchbergers algoritme er, i sin mest banale form, ikke spesielt effektiv. Vi ser på flere forbedringer av algoritmen. I det kommutative tilfellet tar vi for oss Buchbergers kriterier, Gebauer-Möller-forbedringen og Faugères F_4- og F_5-algoritmer. Videre har vi «oversatt» disse algoritmene og forbedringene, med unntak av F_5, til ikke-kommutative polynomringer.

III – Polly Cracker-kryptografi. Kryptosystemer i polynomringer finnes i flere varianter, derav mange som benytter seg av Gröbnerbasiser. Her ser vi på *Polly Cracker*-systemer, først definert i det kommutative tilfellet av Fellows og Koblitz. Ideen bak disse systemene var å lage polynomer basert på $\mathcal{NP}$-komplette problemer. Ikke-kommutative Polly Cracker-systemer er lite utforsket, de foreløpige forsøkene bygger på idealer med uendelig Gröbnerbasis og er første gang beskrevet av Rai. Disse systemene og forutsetningene for dem er omtalt, og delen avsluttes med en kritisk gjennomgang av dem.

Selv om de siste to delene ikke har noen direkte sammenheng, er de ikke urelaterte, siden kryptosystemene i del III baserer seg på at det kan finnes idealer der det er (beregnbarhetsmessig) vanskelig å finne en Gröbnerbasis. Dette er ofte et spørsmål om algoritmer, som altså er tema i del II.

Takk

Denne masteroppgaven markerer avslutningen på min fem år lange matematikk-utdanning ved NTNU, men mitt liv med matematikk begynte naturligvis lenge før. I «Barnets første år»-boken min skriver mamma i januar 1991 (da var jeg enda ikke fylt fem år) at «Kristin forbauser også med stor dyktighet i regning (pluss/minus) på denne tida», eksemplifisert med $2+3=5$ og $3+3=6$. Jeg vil derfor starte med å takke foreldrene mine for å ha introdusert meg for tallenes verden, nesten før jeg hadde lært å lese, og for all støtte og oppmuntring gjennom hele min 17-årige skolegang. Videre skylder jeg samtlige av mine matematikklærere opp gjennom årene mye, da særlig lærerne jeg hadde på barne- og ungdomsskolen, men jeg retter også en takk til alle inspirerende forelesere jeg har hatt i løpet av årene her på Gløshaugen.

Dessuten vil jeg benytte anledningen til å prise en av mine yndlingsbøker, *Talldjevelen* av Hans Magnus Enzenberger, som jeg leste for første gang som elleveåring og som viste meg at matematikk er mye mer enn hoderegning og pugging av gangetabellen. Selv om *Talldjevelen* strengt tatt er en barnebok, måtte jeg gå i alle fall halvannet år på universitetet før jeg formelt hadde lært alt som stod i den!

Når det gjelder selve masteroppgaven må jeg først og fremst takke min veileder Aslak Bakke Buan, som hele tiden har vært behjelpelig med gode tips og råd, og samtidig latt meg få bestemme nokså mye. Temaet for oppgaven var også hans idé, og det viste seg fort å fange min interesse. Ellers takk til alle som har hjulpet meg underveis, særlig synes jeg det var morsomt å treffe Ed Green, som jo er en av de store innen dette fagområdet.

Selv om oppgaveskrivingen har medført en til tider ensom livsstil, har vi jo vært flere i samme båt, og det har vært fint å dele bekymringer, trøst og tips med Maria og Pernille (og ikke minst avvekslende prat om helt andre ting, og dessuten et bordtennisspill i ny og ne). Til slutt takk til Iselin for Arne og Bjarne!

Innhold

Forord

Som det framgår av sammendraget har denne oppgaven tre hovedtemaer: Gröbnerbasiser, algoritmer for å finne slike og Polly Cracker-kryptografi. I motsetning til svært mange Gröbnerbasis-tekster som er *enten* kommutative *eller* ikke-kommutative, er denne oppgaven begge deler, og jeg har lagt vekt på å samkjøre oppbyggingen av de kommutative og ikke-kommutative delene. Avhandlingen er derfor overordnet inndelt etter tema, deretter etter kommutativt/ikke-kommutativt. Oppgaven var i utgangspunktet ment å handle utelukkende om Polly Cracker-kryptosystemer, med unntak av de to obligatoriske «innføring i Gröbnerbasis»-kapitlene som utgjør del I. Disse to kapitlene er relativt avgrensede i innhold, siden det ikke blir gjennomgått mer enn det som trengs senere i oppgaven, men de er like fullt grundige, da alle resultater som er med er bevist.

Det ble etter hvert naturlig å skifte fokus fra kryptosystemer basert på Gröbnerbasiser til algoritmer for å finne Gröbnerbasiser, siden det ene av disse temaene er en motivasjon for å se på det andre, og omvendt. De opprinnelige (kommutative) Polly Cracker-systemene er konstruert rundt beregnbarhetsmessig vanskelige kombinatoriske problemer – det skal være beregnbarhetsmessig vanskelig å finne en Gröbnerbasis for idealet generert av polynomene i den offentlige nøkkelen. Det er klart at det da er av interesse å kjenne så gode Gröbnerbasis-algoritmer som mulig, fordi det å si at et problem er «beregnbarhetsmessig vanskelig» er det samme som å si at det ikke finnes noen (god) polynomisk tid-algoritme for å løse problemet. Omvendt er det slik at det å knekke kryptosystemer er en viktig motivasjon for algoritmeutviklere.

Polly Cracker-systemer har etter hvert blitt en ganske stor klasse av kryptosystemer, og det er ikke alle variantene som er beskrevet i denne oppgaven. Jeg har fokusert på de generelle oppsettene og de vanligste (og mest effektive) angrepene, særlig de ulike typene lineæralgebraangrep er grundig gjennomgått og illustrert med flere omfattende eksempler. I det kommutative tilfellet har jeg også laget et nytt eksempel på et Polly Cracker-system bygd på et konkret $\mathcal{NP}$-komplett problem.

Algoritmedelen av oppgaven er den lengste målt i antall sider, noe som delvis skyldes at pseudokoder tar stor plass. Kapittel 6 er en ikke-kommutativ «versjon» av kapittel 4. Jeg har imidlertid ikke gitt noen ikke-kommutativ variant av kapittel 5 om F_5-algoritmen. Dette kapitlet var det aller siste jeg skrev, og tiden satte en effektiv stopper for videre utforskning. Uten å ha sett mye på dette vil jeg likevel gjette at en ikke-kommutativ oversettelse av F_5 *ikke* er mulig, av to grunner. Den første er at beviset for kriteriet bak F_5 bruker såkalte *hovedsyzygyer*, som bygger på at for to vilkårlige polynomer f_i og f_j har vi $f_i f_j = f_j f_i$. Den andre grunnen er at F_5 kun fungerer for *homogene* polynomer, og mens det i det kommutative tilfellet er enkelt å homogenisere vilkårlige polynomer, har vi ingen ikke-kommutativ tolkning av dette (i alle fall ikke en geometrisk sådan).

Selv om jeg i utgangspunktet ikke hadde noen planer om å studere Gröbnerbasis-algoritmer

(utover den mest banale versjonen av Buchbergers algoritme), laget jeg av praktiske årsaker en enkel implementasjon av ikke-kommutativ Buchbergers algoritme allerede mens jeg holdt på med kryptografidelen. Samtidig som jeg satte meg mer inn i ulike effektiviseringer av Buchbergers algoritme, utvidet jeg også mitt eget program til å inkludere disse. Programmet Poly benytter dermed en versjon av Buchbergers algoritme som er så godt som identisk med den som er gjengitt i kapittel 6. To av vedleggene (B og C) består av henholdsvis dokumentasjon av og kildekode for programmet. Poly-programmet er også velegnet til andre enkle beregninger med ikke-kommutative polynomer, men programmets største styrke er muligheten til å beregne delvise Gröbnerbasiser.

Det finnes mange implementasjoner av både kommutative og ikke-kommutative Gröbnerbasisalgoritmer. En liste over mange av disse finnes på nettsiden *Gröbner Basis Implementations*,

http://www.risc.uni-linz.ac.at/Groebner-Bases-Implementations/

Dette er en detaljert database som gir en svært god oversikt over hvilke effektiviseringer og ekstra funksjonaliteter som er inkludert i hver enkelt implementasjon. Enkelte programvareutviklere er imidlertid nokså restriktive på hvor mye av denne informasjonen som de vil dele med omverdenen. Selv er jeg tilhenger av fri programvare og fri kildekode, så alle som har lyst kan bruke Poly-pakken og videreutvikle den etter behov (python-filen er inkludert som elektronisk vedlegg).

En annen nettside jeg har brukt mye er *Gröbner Bases Bibliography*,

http://www.ricam.oeaw.ac.at/Groebner-Bases-Bibliography/

– en skattkiste for Gröbnerbasisinteresserte (men dessverre ikke helt oppdatert). Alle artiklene her ligger andre steder også, men det er stor forskjell på å søke i en spesiallaget database og på å søke på Google. Så dersom du leser denne oppgaven og blir hektet på Gröbnerbasiser, er dette et godt sted å fortsette.

Trondheim, 25. mai 2010

Kristin Krogh Arnesen

kristin_ka@hotmail.com

1 Innledning

Dette kapitlet har to formål: Først skal det si litt om måten resten av oppgaven er skrevet på – dette omfatter blant annet språk, notasjon og symbolbruk. Deretter gir kapitlet en kort innføring i to temaer, nærmere bestemt *beregnbarhets- og kompleksitetsteori* og *offentlig nøkkel-kryptografi*. Dette stoffet er ikke en del av arbeidet med oppgaven, men kan snarere anses som forkunnskaper. Vi har kun gjengitt noen få relevante definisjoner og teoremer. For en grundigere innføring, se kildehenvisningene. Beregnbarhets- og kompleksitetsteorien brukes implisitt og eksplisitt gjennom store deler av oppgaven, mens kryptografien er avgrenset til del III.

1.1 Språk og notasjon

Symbolbruken i oppgaven, utover det matematiske, er svært sparsom. Som markering for at et bevis er avsluttet brukes symbolet ♣, men dersom bare en idé for beviset er gitt, brukes ♠. Så framt det har vært naturlig i den aktuelle sammenhengen, er de fleste resultater bevist.

Dersom man leser flere tekster om emnene som er behandlet i denne oppgaven (for eksempel de artiklene som er oppgitt i bibliografien), vil man raskt se at notasjonen forandrer seg, til tider drastisk, fra tekst til tekst. Spesielt er det forvirrende med begrepene *monom* og *ledd*, som i mange tekster brukes *omvendt* av det vi gjør her. Vi har forsøkt å være mest mulig konsekvente, og ikke minst lagt vekt på å få samsvar mellom de kommutative og ikke-kommutative tilfellene. Notasjonen er mest påvirket av Cox, Little og O'Shea [7] (kommutativ) og Green [16] (ikke-kommutativ), til fordel for notasjonen som brukes hos Mora [21] [22] eller Faugère [10] [9].

Flere av kapitlene gjengir *algoritmer*. Det er mange måter å skrive algoritmer i en tekst på, men mest naturlig – i alle fall i en setting som her, der algoritmene ikke skal assosieres med et bestemt programmeringsspråk – er det å bruke pseudokoder. Da forfatteren er vant med å skrive og lese programkode, kan det hende at pseudokodene i noen tilfeller blir mer detaljert enn strengt tatt nødvendig, men for de virkelig kompliserte algoritmene har vi også inkludert en mer overordnet, språklig forklaring på algoritmene. For de fleste algoritmene er dessuten de matematiske ideene og resultatene som ligger til grunn gjennomgått på forhånd.

Til slutt bør leseren advares mot merkelige *oversettelser*, idet denne teksten er skrevet på norsk, mens samtlige av kildetekstene er på engelsk, og mange av dem inneholder uttrykk som ikke har noen bestemt norsk oversettelse i denne sammenhengen. Ofte er de engelske «original-uttrykkene» oppgitt i tillegg til de norske der de blir definert. For eksempel er det engelske «obstruction» oversatt til «blokkering» snarere enn «obstruksjon» – rett og slett fordi det gir bedre flyt i en norsk tekst med norske ord. Som en hjelp til leseren har vi lagt ved en ordliste (vedlegg A) der de mest uvanlige oversettelsene er listet opp.

1.2 Beregnbarhets- og kompleksitetsteori

For å ikke gjøre dette mer komplisert enn nødvendig, har vi valgt å holde denne teksten fri for *Turingmaskiner* og *språk*. Det er klart at definisjonene av kompleksitetsklassene $\mathcal{P}$ og $\mathcal{NP}$ da blir noe mangelfulle, men like fullt tilstrekkelige for det vi trenger dem til. Se [24] for for hvordan Turingmaskiner gir et mer helhetlig bilde av de nedenstående begrepene.

Vi er interessert i *algoritmer*, og kjøretiden av disse. En algoritme er et sett med entydige instruksjoner som løser et *problem*, gitt en *instans* av problemet (denne instansen kalles *input*'en til algoritmen, mens løsningen av instansen er *output*'en). Vi sier at algoritmen *terminerer* dersom den stopper etter et endelig antall operasjoner.

Merk. En algoritme og et problem er *ikke* det samme. For et bestemt problem kan det finnes både «gode» og «dårlige» algoritmer (hva vi legger i dette skal vi snart se), og det finnes sågar problemer som ikke kan løses med *noen* terminerende algoritme.

Hvor effektiv en algoritme er, måles i antall operasjoner (steg) som blir utført, gitt som en funksjon av lengden på input'en. I en veldig formell setting er inputlengden avhengig av implementasjonen, som hvilken base vi bruker for å representere tall. I denne teksten er input til en algoritme typisk en liste, og dersom lista har n elementer, sier vi at lengden av input'en er n. Vi lar $t(n)$ notere antall steg en gitt algoritme gjør fra start til terminering på input med lengde n. Dette er altså en funksjon $t(n) : \mathbb{N} \longrightarrow \mathbb{N}$.

Når vi snakker om *kompleksiteten* til en algoritme, er vi typisk interessert i *kompleksitetsklasser*, ikke i det nøyaktige antall steg. Eksempelvis sier vi at en algoritme som terminerer etter $t(n) = 3n^2 + \frac{1}{2}n$ steg, er en $O(n^2)$-algoritme. Denne såkalte «stor O-notasjonen» defineres gjennom *vekst av funksjoner:*

Definisjon 1.2.1. La f og g være funksjoner fra $\mathbb{N}$ til $\mathbb{R}^+$. vi sier at $f(x)$ er $O(g(x))$ dersom det finnes konstanter C i $\mathbb{R}^+$ og k i $\mathbb{N}$ slik at

$$|f(x)| \leq C \cdot |g(x)| \text{ når } x > k$$

Eksempel 1.2.2. Funksjonen $t(n) = 3n^2 + \frac{1}{2}n$ er $O(n^2)$, siden

$$|t(n)| \leq 4n^2 \quad \text{når } x > \frac{1}{2}$$

Det finnes en tilsvarende nedre grense (snu ulikhetstegnet i definisjonen), den betegnes $\Omega(g(x))$. Dersom $f(x)$ er både $O(g(x))$ og $\Omega(g(x))$, sier vi at $f(x)$ er $\Theta(g(x))$. En algoritme kan ha store forskjeller mellom «best case», «average case» og «worst case». Et eksempelet på en slik algoritme er (kommutativ) Buchbergers algoritme: I flere av eksemplene vi skal se på (kapittel 2 og 4) løser den raskt et problem, mens andre ganger (kapittel 7) bruker den *årevis* på å terminere! (Den ikke-kommutative versjonen av algoritmen oppfører seg ganske likt, bortsett fra at den ikke er garantert å terminere, og dermed kan kjøre til evig tid på enkelte input'er.)

Når vi kjenner $O(g(n))$ for en algoritme, kan vi bedømme hvor «god» den er. De beste algoritmene har såkalt *polynomisk kjøretid*:

Definisjon 1.2.3. En algoritme har **polynomisk kjøretid** dersom kjøretiden $t(n)$ til algoritmen er $O(n^k)$ for et heltall $k \geq 1$. Klassen av alle problemer som kan løses med en algoritme som har polynomisk kjøretid, kalles $\mathcal{P}$, der $\mathcal{P}$ står for «polynomial».

Noen eksempler på problemer som tilhører $\mathcal{P}$ er, foruten alle som har algoritmer med kjøretid $t(n) = n^k$, også alle problemer som kan løses med algoritmer hvis kjøretid er $O(\log n)$ eller $O(n \log n)$.

Merk at selv om polynomisk tid-algoritmer generelt regnes som gode, er det forskjell på $O(n)$ og $O(n^4)$, og har vi $O(n^{100})$ er det ikke veldig praktisk. Imidlertid er det ofte slik at når det først er funnet én polynomisk tid-algoritme for å løse et problem, er det snart noen som finner en bedre [24, s. 262].

«Dårlige» kjøretider er kjøretider som ikke er $O(n^k)$, som for eksempel $n!$, 2^n og n^n – da snakker vi om algoritmer som kanskje må kjøre i milliarder av år, selv på den beste datamaskin og med moderat input (som $n = 100$).

Vi flytter nå fokuset fra algoritmer til problemer. Vi har allerede definert klassen $\mathcal{P}$, men særlig er vi interessert i en klasse som kalles $\mathcal{NP}$, for «nondeterministically polynomial». Dette kryptiske begrepet forklares ofte med at vi «ikke vet» om kjøretiden er polynomisk, altså at $\mathcal{NP}$ består av problemer for hvilke det ennå ikke er oppdaget en polynomisk tid-algoritme. Dette er bare delvis korrekt (ikke-determinisme er ikke det samme som uvitenhet), og for å klare å gi en god definisjon av klassen $\mathcal{NP}$ uten bruk av Turingmaskiner, må vi innføre følgende definisjon:

Definisjon 1.2.4. En *verifiseringsalgoritme* for et problem P er en algoritme A som tar inn en instans ρ av P og en mulig løsning x og og returnerer 0 eller 1 slik at

$$A(\rho, x) = \begin{cases} 1 & \text{dersom } x \text{ er en løsning av } \rho \\ 0 & \text{ellers} \end{cases}$$

Merk at både ρ og x må være gitt i en forhåndsbestemt koding. For å få en bedre forståelse av dette, gir vi et eksempel.

Eksempel 1.2.5. La P være problemet *Hamiltonsti*: Gitt en rettet graf $\mathfrak{G} = (V, E)$, finnes det en sti i $\mathfrak{G}$ som besøker hvert hjørne nøyaktig én gang? Vi vil ikke finne en algoritme for å løse P, men en verifiseringsalgoritme. Først må vi se på kodingen av probleminstansen ρ for en graf $\mathfrak{G} = (V, E)$.

Vi representerer $\mathfrak{G}$ med en naboskapsmatrise (M_{ij}), som er en $|V| \times |V|$-matrise der $M_{ij} = 1$ hvis det finnes en kant fra hjørne i til hjørne j, og 0 ellers. En sti av lengde n i $\mathfrak{G}$ representeres som en vektor i $\mathbb{Z}_{|V|}^n$, der første koordinat er første hjørne i stien og så videre.

Inputen til en verifiseringsalgoritme A vil altså være en matrise (M_{ij}) og en vektor $\overline{v}$. Algoritmen vil utføre følgende steg:

1. sjekke at $|\overline{v}| = |V|$
2. sjekke at alle hjørnene i V opptrer én gang i $\overline{v}$

3. sjekke at dersom hjørne v_j etterfølger hjørne v_i i $\overline{v}$, er $M_{ij} = 1$

Dersom alle tre punkter er oppfylt, er $\overline{v}$ en løsning av ρ.

Vi kan nå gi en definisjon av klassen $\mathcal{NP}$.

Definisjon 1.2.6. Klassen $\mathcal{NP}$ består av alle problemer som har en verifiseringsalgoritme med polynomisk kjøretid.

Merk. Kjøretiden til en verifiseringsalgoritme er en funksjon av lengden av input'en ρ. Dersom en verifiseringsalgoritme har polynomisk kjøretid, vil også løsningen x av ρ ha polynomisk lengde (siden algoritmen leser hele x for å verifisere at det er en løsning).

Det er lett å se at verifisieringsalgoritmen for *Hamiltonsti* har polynomisk kjøretid, så *Hamiltonsti* tilhører $\mathcal{NP}$. Men finnes det en polynomisk tid-algoritme for å *løse* dette problemet? Generelt har vi at dersom et problem P er i $\mathcal{P}$, er også P i $\mathcal{NP}$ (en algoritme for å løse P kan lett omskrives til en verifiseringsalgoritme). Altså har vi

$$\mathcal{P} \subseteq \mathcal{NP}$$

Det er vanlig å *anta* at $\mathcal{P} \neq \mathcal{NP}$, men dette er fortsatt ikke bevist – noe som innebærer at man ikke har klart å finne et problem som er i $\mathcal{NP}$ og samtidig ikke i $\mathcal{P}$. Derimot har vi såkalte $\mathcal{NP}$*-komplette problemer*, men for å definere dem, må vi først vite hva en *reduksjon* er.

Definisjon 1.2.7. La P_1 og P_2 være to problemer, og ρ en instans av P_1. La τ være en algoritme som tar en instans av P_1 og en tilhørende mulig løsning som input, og returnerer en instans og en mulig løsning av P_2. Vi sier at τ er en **reduksjon** fra P_1 til P_2 dersom det følgende er ekvivalent:

- x er en løsning av ρ
- $\tau(x)$ er en løsning av $\tau(\rho)$

Vi skriver da $P_1 \overset{\tau}{\Longrightarrow} P_2$ (og leser P_1 **reduserer til** P_2).

Spesielt er vi interessert i reduksjoner med polynomisk kjøretid.

Definisjon 1.2.8. Problemet P i $\mathcal{NP}$ er $\mathcal{NP}$**-komplett** dersom det, for alle problemer P' i $\mathcal{NP}$, finnes en polynomisk tid-reduksjon τ fra P' til P.

Dette fører umiddelbart til følgende resultat:

Korollar 1.2.9. *La* P *være et* $\mathcal{NP}$*-komplett problem. Dersom* P *er i* $\mathcal{P}$*, er* $\mathcal{P} = \mathcal{NP}$*.*

Det finnes mange kjente $\mathcal{NP}$-komplette problemer. Man viser vanligvis at et problem P er $\mathcal{NP}$-komplett ved å redusere et kjent $\mathcal{NP}$-komplett problem til P. Det første problemet som ble bevist å være $\mathcal{NP}$-komplett heter *Satisfiability* (Cook-Levin-teoremet, se [24, s. 280–286] for bevis). *Hamiltonsti* er også $\mathcal{NP}$-komplett.

I denne oppgaven skal vi se på en type kryptosystemer hvis sikkerhet er tuftet på at det er beregnbarhetsmessig vanskelig å løse $\mathcal{NP}$-komplette problemer. Det er klart at denne sikkerheten er alt annet enn god dersom det skulle vise seg at $\mathcal{P} = \mathcal{NP}$, men foreløpig er det ingenting som indikerer at dette kan være tilfelle.

1.3 Offentlig nøkkel-kryptografi

Ordet *kryptografi* betyr, direkte oversatt fra gresk, «gjemt skrift».

Når to parter skal kommunisere over en åpen kanal, uten at en tredje part som overhører beskjedene skal forstå hva som blir sagt, må meldingene krypteres. Kryptografi har eksisert siden førkristen tid, men først på 1900-tallet har dette blitt et stort og nødvendig forskningsområde. I dag foregår svært mye kommunikasjon gjennom usikre kanaler som internett og telefonnettverk, og kryptografi har dermed blitt en del av vanlige folks hverdag, ofte uten at de er klare over det selv.

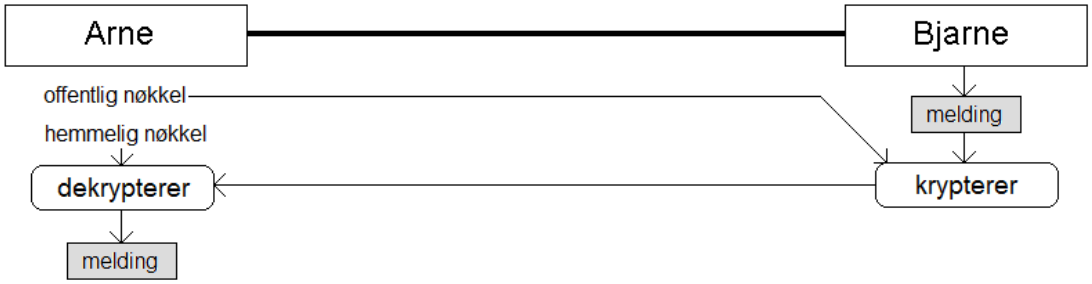
Et *kryptosystem* består av en mengde av mulige meldinger (*meldingsrommet*), en mengde av mulige krypterte meldinger (*chiffertekstrommet*), et *nøkkelrom* og *krypterings-* og *dekrypteringsalgoritmer*, der den siste må være entydig. Siden beregningene foretas av en datamaskin, må meldingsrommet og chiffertekstrommet være endelige mengder.

I *symmetrisk kryptografi* har hvert par av kommuniserende parter en nøkkel som bare brukes av disse, og begge parter kan både kryptere og dekryptere meldinger. Systemer som DES og AES, som brukes i dag, er symmetriske kryptosystemer, og så lenge ingen andre kjenner nøkkelen, er AES både sikkert og effektivt. Men den hemmelige nøkkelen må typisk også sendes over den usikre kommunikasjonskanalen, og den må også krypteres – med hvilket system? For å løse dette dilemmaet, ble *offentlig nøkkel-kryptografi* (eller *asymmetrisk kryptografi*) oppfunnet, nærmere bestemt en protokoll ved navn *Diffie-Hellman-nøkkelutveksling* (1976). I en mer teoretisk setting kan man også bruke offentlig nøkkel-kryptografi på *all* kommunikasjon mellom to parter, ikke bare som initialisering av en symmetrisk kryptografi-protokoll. Offentlig nøkkel-systemer brukes også i signeringsprotokoller. Et velbrukt og velkjent eksempel på et offentlig nøkkel-system er *RSA*. Kryptosystemet som er i fokus i del III av denne oppgaven, *Polly Cracker*, er også et offentlig-nøkkel-system.

Den generelle settingen for et offentlig nøkkel-system er som følger:

To personer – la oss kalle dem Arne og Bjarne – skal kommunisere over en usikker linje. Arne har to nøkler, en hemmelig, som han ikke har gitt fra seg til noen (heller ikke til Bjarne), og en offentlig, som er åpen for alle. Ved hjelp av den offentlige nøkkelen og en kjent krypteringsalgoritme, kan alle kryptere meldinger (forutsatt at meldingen ligger i meldingsrommet), men det er bare Arne som kan dekryptere meldingene, for dekrypteringsalgoritmen benytter den hemmelige nøkkelen. Dersom kommunikasjonen er toveis, altså at Arne også skal sende meldinger til Bjarne, må Bjarne også ha en hemmelig og en offentlig nøkkel. Vi ser hvorfor en slik protokoll kalles asymmetrisk. Prosessen der Bjarne sender en melding til Arne, kan illustreres som på figur 1.1.

Merk at hvordan dekrypteringsalgoritmen fungerer, også er kjent, men den er avhengig av den hemmelige nøkkelen som parameter.



Figur 1.1: Bjarne sender melding til Arne. All kommunikasjon går over en åpen linje.

Sikkerhet og angrep

Det er alltid en sammenheng mellom den offentlige og den hemmelige nøkkelen, siden dekrypteringen, ved hjelp av den hemmelige nøkkelen, skal «reversere» det krypteringsalgoritmen har gjort (merk at vi ikke snakker om funksjoner med tilhørende invers funksjon, men om algoritmer – Polly Cracker er for eksempel konstruert slik at krypteringen ikke er deterministisk, og den kan dermed ikke beskrives som en funksjon). Derfor er mye av sikkerheten i kryptosystemet basert på at den offentlige nøkkelen kan brukes til å avsløre den hemmelige. Den strukturelle sammenhengen mellom de to nøklene er offentlig kjent, så i praksis betyr dette at det skal være beregnbarhetsmessig vanskelig å finne den hemmelige nøkkelen ut fra den offentlige.

I tillegg har vi tre typer angrep vi må sikre systemet mot:

Chiffertekstangrep: Vi må gå ut fra at en uønsket avlytter, kryptanalysten som vi i denne teksten har døpt Katla, leser de krypterte meldingene som sendes. Derfor er det viktig at krypteringsalgoritmen må være laget slik at input'en, altså meldingen, ikke skal kunne beregnes ut fra chifferteksten.

Kjent meldingsangrep: Katla har et eller flere par bestående av en melding og en tilhørende chiffertekst. Kan hun sammenlikne dem og forstå hvordan dekrypteringen har foregått?

Valgt chiffertekstangrep: Katla har midlertidig tilgang til dekrypteringsprogrammet (tenk på dekrypteringen som en «black box» der chifferteksten går inn og meldingen kommer ut, uten at Katla får vite noe om selve prosessen).

Det er altså to måter å overvinne et slikt kryptosystem på. Den første er å skaffe seg tilgang til den hemmelige nøkkelen, slik at man kan dekryptere enhver melding. Den andre måten er å bruke et av angrepene over til å dekryptere enkeltbeskjeder – helt eller delvis – eller til å lage sin egen dekrypteringsalgoritme, som ikke bruker den hemmelige nøkkelen, men som like fullt gir riktig svar.

Del I
Innføring i Gröbnerbasiser

– Opp med deg, Robert, sa han. – I dag skal vi dividere!
– Er jeg nødt? spurte Robert. – Forresten kunne du godt ha ventet til jeg var sovnet. Og deling er det verste jeg vet.
– Hvorfor det?
– Jo, for når det er snakk om pluss eller minus eller ganging, så går alle regnestykker opp. Men ikke med deling. Da blir det ofte en rest til overs, og det syns jeg er et ork.
– Spørsmålet er bare når?
– Når hva? spurte Robert.
– Når det blir en rest til overs og når det ikke blir det, forklarte talldjevelen. – Det er nemlig det springende punkt.

fra *Talldjevelen* av Hans Magnus Enzenberger

2 Kommutative Gröbnerbasiser

2.1 Innledning

Dette kapitlet er en introduksjon til Gröbnerbasiser over en (kommutativ) polynomring på formen $R = k[x_1, x_2, \ldots, x_n]$, der k er en kropp. Vi skriver R for denne ringen gjennom hele kapitlet. Alle idealer vi ser på er i R, enten det er presisert eller ikke.

En Gröbnerbasis er en genererende mengde for et ideal $I \subseteq R$, som har noen spesielle egenskaper som gjør at den er svært nyttig i mange sammenhenger. Anvendelsene for Gröbnerbasiser er mange, og vi skal innom noen av dem i denne introduksjonen:

1. Ligger polynomet f i R også i idealet I?
2. Har f en «normalform» i R/I?
3. Gitt to idealer i R, er de det samme idealet?
4. Hvordan løse likningssystemer på formen $f_1(x_1, x_2, \ldots, x_n) = f_2(x_1, x_2, \ldots, x_n) = \ldots = f_s(x_1, x_2, \ldots, x_n) = 0$, der f_i er i R?

I første omgang konsentrerer vi oss om de to første av disse punktene. Det tredje ser vi på mot slutten av del 5, og det siste blir overflatisk behandlet i del 6.

Kapitlet er hovedsaklig basert på [7, kapittel 2], der man også kan finne de bevisene som er utelatt her, med mindre andre kilder er oppgitt.

2.2 Motivasjon: $R = k[x]$

La oss først se på en polynomring med bare én variabel, $R = k[x]$, der k er en kropp. Denne ringen har følgende egenskaper (bevis er ikke gitt her, se [2, Theorem 4.1 s. 220]):

(i) R er et hovedidealområde, det vil si at alle idealer $I \subseteq R$ er på formen $I = \langle f(x) \rangle$ for en $f(x)$ i I.

(ii) Vi har en divisjonsalgoritme i ringen – vanlig polynomdivisjon.

(iii) For alle $g(x)$ i R, har vi unike $q(x)$ og $r(x)$ i R med $deg(r(x)) < deg(g(x))$ slik at $g(x) = q(x) \cdot f(x) + r(x)$.

Ved hjelp av disse egenskapene er det lett å finne metoder for å svare på de to første punktene i forrige avsnitt: Polynomet $f(x)$ er i I hvis og bare hvis resten $r(x)$ er null, og denne resten kan betraktes som normalformen av f i R/I (to polynomer har samme rest hvis og bare hvis differansen mellom dem er i I, og da representerer de det samme elementet i R/I).

Allerede når vi setter $n = 2$ og får ringen $R = k[x, y]$, oppstår første problem. Denne ringen er nemlig ikke et hovedidealområde. Idealet $\langle x, y \rangle$ kan for eksempel ikke genereres av ett enkelt element (strengt tatt har vi ikke en gang slått fast enda at alle idealer er endeliggenererte, dette kommer vi til senere). Vi trenger mer avanserte verktøy for å finne en fornuftig genererende mengde for denne ringen.

2.3 Polynomer: monomsortering og divisjonsalgoritme

La f være et element i ringen $R = k[x_1, x_2, \ldots, x_n]$. Da er f et polynom på formen

$$\sum_{\alpha} a_\alpha \cdot x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$$

der α'ene i summen er vektorer $(\alpha_1, \ldots, \alpha_n)$ i $\mathbb{Z}^n_{\geq 0}$. Summen er endelig, så de fleste $a_\alpha = 0$. Dette er atskillig mer komplisert enn i $R[x]$, der vi bare hadde én variabel! Det er klart at vi trenger en dose forenklende notasjon.

Definisjon 2.3.1. Et **monom** (eller *potensprodukt*) er et produkt på formen

$$x^\alpha = {x_1}^{\alpha_1} \cdot {x_2}^{\alpha_2} \cdot \ldots \cdot {x_n}^{\alpha_n}$$

der $\alpha = (\alpha_1, \ldots, \alpha_n)$ er en vektor i $\mathbb{Z}^n_{\geq 0}$.

Definisjon 2.3.2. $a_\alpha \cdot x^\alpha$ i polynomet $f = \sum_\alpha a_\alpha \cdot x^\alpha$ kalles et **ledd**, med a_α som **koeffisient**.

Med dette på plass, kan vi begynne å vende blikket mot en velfungerende divisjonsalgoritme i R. Imidlertid er det en ting til vi må ta i betraktning: La oss igjen ta for oss divisjonsalgoritmen i $R[x]$, altså vanlig polynomdivisjon. For at algoritmen skal fungere, også når den utføres av en datamaskin uten «intelligens», er vi avhengige av at leddene står i en bestemt rekkefølge. Faktisk ser vi kun på leddet som står lengst til venstre, både i dividenden og divisoren:

$$\begin{array}{rcrcrcl}
x^5 & + & 2x^2 & - & 3 & : & x^3 - x + 1 = x^2 + \ldots \\
x^5 & - & x^3 & + & x^2 & & \\
 & & x^3 & + & x^2 & - & 3 \\
 & & & & \vdots & &
\end{array}$$

Vi fant det første leddet i kvotienten, x^2, kun ved å se på det første leddet i dividenden og hvor mange ganger det første leddet i divisoren gikk opp i dette. Men hvordan vet vi hva som er det første leddet i for eksempel

$$f = 13x^4y + 4x^6y^3 + 7x^3 + y - 8 + x^3$$

i $\mathbb{Z}_{17}[x, y]$? Vi trenger å velge en *ordning* på monomene, og den neste definisjonen gir oss noen egenskaper en slik ordning må ha.

Definisjon 2.3.3. En **monomordning** på $k[x_1, \ldots, x_n]$, gitt $x_1 > x_2 > \ldots > x_n$, er en relasjon $\succ$ på $\mathbb{Z}^n_{\geq 0}$ slik at

(i) $\succ$ er en total ordning på $\mathbb{Z}^n_{\geq 0}$ (det vil si at to vilkårlige elementer kan sammenliknes).

(ii) Dersom $\alpha \succ \beta$, og vi har gitt en γ i $\mathbb{Z}^n_{\geq 0}$, har vi $\alpha + \gamma \succ \beta + \gamma$.

(iii) $\succ$ er en velordning på $\mathbb{Z}^n_{\geq 0}$ (altså at alle ikketomme delmengder av $\mathbb{Z}^n_{\geq 0}$ har et minste element).

Det finnes en rekke ordninger som oppfyller disse kravene. De kanskje mest intuitive, og de eneste vi skal ta for oss her, er *leksikografisk* og *gradert leksikografisk* ordning.

Definisjon 2.3.4. Leksikografisk ordning er gitt ved at to vektorer α og β ordnes som i et leksikon, der man sammenlikner fra venstre. Hvis $\alpha_0 > \beta_0$, er $\alpha \succ \beta$. Hvis første koordinat er lik, går man videre til neste koordinat, og så videre. Formelt kan dette beskrives ved at $\alpha \succ \beta$ hvis og bare hvis den første koordinaten ulik 0 i vektoren $\alpha - \beta$ er positiv.

Eksempel 2.3.5. $(3,4,8) \succ (2,5,3)$ siden $3 > 2$, og $(5,6,2) \succ (5,0,3)$ siden $5 = 5$ og $6 > 0$. For monomer i $k[x,y,z]$, gitt $x > y > z$, gir dette at $x^3y^4z^8 \succ x^2y^5z^3$ og $x^5y^6z^2 \succ x^5z^3$.

Definisjon 2.3.6. Gradert leksikografisk ordning er gitt ved at man først sorterer vektorene etter tverrsummen, altså $|\alpha| = \alpha_0 + \alpha_1 + \ldots + \alpha_n$, og deretter sorterer de som har lik tverrsum med leksikografisk ordning.

Eksempel 2.3.7. $(3,4,8) \succ (2,5,3)$ siden $15 > 10$, og $(5,6,2) \succ (5,5,3)$ siden $13 = 13$ og $6 > 5$. For monomer i $k[x,y,z]$, gitt $x > y > z$, gir dette at $x^3y^4z^8 \succ x^2y^5z^3$ og $x^5y^6z^2 \succ x^5y^5z^3$.

Når vi skal utføre beregninger i ringen R, holder vi oss til én valgt monomordning. Vi skriver nå alle polynomene med leddene sortert fra venstre mot høyre, slik at det leddet som, gitt den valgte monomordningen, er størst, havner lengst til venstre. Dette leddet kalles det **ledende leddet** av et polynom f, og vi skriver det som $LT(f)$. Monomet i dette leddet kalles det **ledende monomet**, og koeffisienten den **ledende koeffisienten**. Vi får $LT(f) = LC(f) \cdot LM(f)$.

Graden til f er definert som $deg(f) = \alpha = (\alpha_1, \ldots, \alpha_n)$ der $LM(f) = x^\alpha$. Den **totale graden** til f er tverrsummen av α, altså $\alpha_1 + \ldots + \alpha_n$. Vi skriver $deg_{\text{tot}}(f)$ for den totale graden til f.

Nå har vi det vi trenger for å beskrive en divisjonsalgoritme i R. En ting vi imidlertid må notere oss først, er at siden vi allerede har sett at idealene i R ofte har mer enn én generator, er vi interessert i å dele et polynom på *flere* polynomer (generatorer for idealet). Som input til divisjonsalgoritmen har vi dividenden f og divisorene $\{f_1, f_2, \ldots, f_t\}$. Output blir et t-tuppel med polynomer i R, $(a_1, a_2, \ldots, a_t)$, og en rest r i R, slik at $f = a_1f_1 + a_2f_2 + \ldots + a_tf_t + r$ der $LT(r)$ ikke er delelig med noen $LT(f_i)$. Divisjonsalgoritmen er gitt i algoritme 2.1.

Eksempel 2.3.8. Divisjonsalgoritme i $R = \mathbb{F}_2[x,y], x > y$, gradert leksikografisk ordning. La $f = x^2y + x, f_1 = x^2 + 1, f_2 = x + y$. Vi vil finne a_1, a_2 og r, og avgjøre om f er i idealet generert av $\{f_1, f_2\}$. Divisjonsalgoritmen gir: $f = y \cdot f_1 + 1 \cdot f_2 + 0$. Vi ser at vi får 0 i rest. Da har vi at f er i $\langle f_1, f_2 \rangle$. Men hvis vi bytter rekkefølge på f_2 og f_1 i algoritmen, får vi $f = 0 \cdot f_1 + (xy + y^2 + 1) \cdot f_2 + (y^3 + y)$ – altså en rest som ikke er 0. Hadde vi kun brukt denne rekkefølgen, kunne vi feilaktig antatt at $f \notin I$.

Med andre ord er $\{f_1, f_2\}$ en genererende mengde for idealet utspent av samme polynomer, men den har *ikke* de gode egenskapene vi er ute etter.

Algoritme 2.1 Divisjonsalgoritme i $k[x_1, \ldots, x_n]$

input: $f, \{f_1, \ldots, f_s\}$
output: $r, (a_1, \ldots, a_s)$
initialiserer
$h := f$
$a_i := 0, \ldots, a_s := 0, r := 0$
$i := 1$
# kjører selve divisjonsalgoritmen
while $h \neq 0$ **do**
  if $lt(f_i)$ deler $LT(h)$ **then**
    # hvis vi kan redusere f med f_i
    $a_i := a_i + LT(h)/LT(f_i)$
    $h := h - LT(h) \cdot f_i/LT(f_i)$ # reduserer med f_i
    $i := 1$ # start på f_1 igjen
  else if $i \leq s$ **then**
    # hvis vi ikke kan redusere med f_i, gå til f_{i+1}
    $i := i + 1$
  else
    # hvis vi ikke kan redusere med noen f_i, trekk ut $LT(h)$
    $r := r + LT(h)$
    $h := h - LT(h)$
    $i := 1$
  end if
end while
return $r, (a_1, \ldots, a_s)$

2.4 Gröbnerbasiser

Før vi introduserer Gröbnerbasiser, la oss oppfriske våre kunnskaper om ringen $R = k[x_1, \ldots, x_n]$. Dette er en **noethersk** ring, siden vi har

(i) Ringen $k[x]$ er noethersk, siden alle idealer i denne ringen er underrom av $k[x]$ betraktet som et vektorrom over k [2, Example 2.4c) s. 371].

(ii) R er noethersk fra følgende resultat: Dersom en ring Λ er noethersk, er også polynomringen $\Lambda[x]$ noethersk [2, Theorem 2.14 s. 375].

Dermed har vi, fra definisjonen av en noethersk ring, at for en uendelig stigende kjede av idealer i R, må vi nødvendigvis få likhet etter en stund. Med andre ord, dersom $I_i \subseteq R$ er idealer for $i \geq 0$, og disse danner en stigende kjede

$$I_0 \subseteq I_1 \subseteq \ldots \subseteq I_i \subseteq \ldots$$

så finnes det en j i $\mathbb{N}$ slik at $I_j = I_{j+1} = \ldots$

Vi har også en ekvivalent egenskap, nemlig at alle idealer i R er endeliggenererte (Hilbertbasisteoremet, [2, teorem 2.3 s. 371]). Det vil si at for et ideal $I \subseteq R$, finnes det $\{f_1, \ldots, f_s\} \subseteq I$

slik at $I = \langle f_1, \ldots, f_s \rangle$. Dersom vi har et ideal generert av monomer, finnes et enda strengere resultat kalt *Dicksons lemma*.

Teorem 2.4.1 (Dicksons lemma)**.** *Gitt en monomordning $\prec$, la I være et ideal generert av en (ikke nødvendigvis endelig) mengde monomer i R, $\{x^\alpha | \alpha \in A\}$. Da kan I skrives på formen $I = \langle x^{\alpha_1}, \ldots, x^{\alpha_s} \rangle$, med $\alpha_1, \ldots, \alpha_s$ i A.*

Bevis. La $X = \{x^\alpha \mid \alpha \in A\}$. Vi definerer $M \subseteq X$ ved

$$M = \{x^\alpha \in X \mid \text{ingen elementer ulik } x^\alpha \text{ i } X \text{ deler } x^\alpha\}$$

Det følger av at $\prec$ er en velordning at M ikke er tom. Vi skal vise at M genererer I og at M er endelig. For å vise at $\langle M \rangle = I$ holder det å vise at alle elementer i X er på formen $x^\alpha x^\beta$, der x^α er i M og x^β er et vilkårlig monom i R. Anta at det finnes en x^γ i X som *ikke* er delelig med noen x^α i M. Da har vi to muligheter:

(1) x^γ er heller ikke delelig med noen andre elementer i X, men da er x^γ per definisjon et element i M. Dette er en motsigelse.

(2) x^γ er delelig med x^δ i $X \setminus M$. Dersom x^δ er delelig med et element i M er også x^γ det, en motsigelse. Hvis ikke, gjenta argumentet for x^δ. Denne prosessen må stoppe siden $x^\gamma \succ x^\delta \succ \ldots$ fra definisjonen av monomordninger. Vi får også her en motsigelse, så det kan ikke finnes noen slik x^γ.

Altså genererer M idealet I. Vi må vise at M er en endelig mengde. Merk at, fra definisjonen av M, har vi at ingen elementer i M deler *andre* elementer i M. Gitt en vilkårlig nummerering på elementene i M, si $M = \{m_1, m_2, \ldots\}$ har vi derfor at

$$\langle m_1 \rangle \subset \langle m_1, m_2 \rangle \subset \langle m_1, m_2, m_3 \rangle \subset \ldots$$

er en strengt økende kjede av idealer. Siden disse er idealer i en noethersk ring, kan vi ikke ha en uendelig kjede på denne formen. Dermed må M være endelig. ♣

Vi definerer nå et meget sentralt ideal generert av monomer, nemlig *idealet generert av ledende ledd*.

Definisjon 2.4.2. La $I \subseteq R$ være et ideal, $I \neq (0)$. Da definerer vi følgende, fiksert en ordning:

(a) **Mengden av ledende ledd**, $LT(I) = \{cx^\alpha | \exists f \in I \text{ med } LT(f) = cx^\alpha\}$

(b) **Idealet generert av ledende ledd**, $\langle LT(I) \rangle$ er idealet generert av $LT(I)$

Merk. Selv om vi har $I = \langle f_1, \ldots, f_n \rangle$, er ikke nødvendigvis $\langle LT(I) \rangle = \langle LT(f_1), \ldots, LT(f_n) \rangle$. Det første av disse idealene kan inneholde elementer som ikke er i det siste.

Nå kan vi endelig definere en Gröbnerbasis.

Definisjon 2.4.3. En endelig delmengde $G = \{g_1, \ldots, g_t\} \subseteq I$, der I er et ideal i R, er en **Gröbnerbasis** for I hvis $\langle LT(g_1), \ldots, LT(g_t) \rangle = \langle LT(I) \rangle$.

En ekvivalent versjon av denne definisjonen, som kanskje er mer intuitiv, er at G er en Gröbnerbasis for I hvis og bare hvis det ledende leddet i alle polynomene i I er delelig med $LT(g_i)$ for en g_i i G. At G også er en generatormengde for I, framkommer av det følgende teoremet:

Teorem 2.4.4. *Fiksér en monomordning på $R = k[x_1, \ldots, x_n]$. Da har alle idealer $I \subseteq R$ en endelig Gröbnerbasis, G, slik at $\langle G \rangle = I$.*

Bevis. La I være et ideal i R. Det er klart at $\langle LT(I) \rangle$ er på formen $\langle x^\alpha | \alpha \in A \rangle$, med x^α monomer i R, siden $\langle LT(I) \rangle = \langle LM(I) \rangle$ (og $LM(I)$ består kun av monomer). Da kan vi bruke Dicksons lemma, som sier at det finnes $x^{\alpha_1}, \ldots, x^{\alpha_t}$ i $\langle LT(I) \rangle$ slik at $\langle x^{\alpha_1}, \ldots, x^{\alpha_t} \rangle = \langle LT(I) \rangle$.

La $g_1, \ldots, g_t$ i I være polynomer i I med $LM(g_i) = x^{\alpha_i}$. Disse vet vi finnes siden x^{α_i} kommer fra $LM(I)$. Med andre ord har vi $\langle LT(I) \rangle = \langle LT(g_1), \ldots, LT(g_t) \rangle$. Det er klart at $\langle g_1, \ldots, g_t \rangle \subseteq I$. Dersom vi i tillegg kan vise at I er generert av $\{g_1, \ldots, g_t\}$, er vi i mål.

Vi må vise at $I \subseteq \langle g_1, \ldots, g_t \rangle$. La f være i I. Fra divisjonsalgoritmen har vi $f = a_1 g_1 + \ldots + a_t g_t + r$, der r ikke er delelig med noen $LT(g_i)$. Dette gir $r = f - (a_1 g_1 + \ldots + a_t g_t)$. Vi ser at høyre side av denne likningen er element i I, siden I er et ideal. Da må også r være i I, og vi har $LT(r)$ i $LT(I)$, hvilket medfører at $LT(r)$ er i $\langle LT(g_1), \ldots, LT(g_t) \rangle$. Siden $LT(r)$ er et monom, må det være delelig med en $LT(g_i)$. Dette gir at eneste mulighet for r er $r = 0$. Da er f i $\langle g_1, \ldots, g_t \rangle$, og vi har at $\{g_1, \ldots, g_t\}$ er en Gröbnerbasis for I. ♣

Innledningsvis ønsket vi å finne en fin genererende mengde for I, slik at resten ved polynomdivisjon blir 0 hvis og bare hvis dividenden er i I, uavhengig av rekkefølgen på divisorene. Den følgende proposisjonen stadfester at Gröbnerbasisen oppfyller dette kriteriet.

Proposisjon 2.4.5. *La $G = \{g_1, \ldots, g_t\}$ være en Gröbnerbasis for idealet $I \subseteq R$, og la f være et element i R. Da finnes en unik r i R slik at*

(i) Ingen ledd i r er divisible med noen av $LT(g_i)$, $1 \leq i \leq t$.

(ii) Det finnes en g i I slik at $f = g + r$.

(iii) f er i I hvis og bare hvis $r = 0$.

Bevis. Vi ser at *(iii)* følger fra *(i)* og *(ii)*. Divisjonsalgoritmen fra avsnitt 2.3 gir $f = a_1 g_1 + \ldots + a_t g_t + r$, der ingen ledd i r er delelig med noen $LT(g_i)$. Vi må vise at r er unik, uavhengig av rekkefølgen på basiselementene når vi reduserer f.

Anta at vi har $f = g + r = g' + r'$, der r, r' oppfyller kravet i *(i)*, og at g og g' er i I. Da har vi

$$g - g' = r' - r$$

så $r' - r$ ligger i I. Siden G er en Gröbnerbasis, har vi minst en g_i i G slik at $LT(g_i)$ deler $r - r'$. Men ingen $LT(g_i)$ deler *noen* ledd i r eller r', så vi må ha $r - r' = 0$ og dermed $r = r'$. ♣

Merk. Anta at vi har $f = g + r$ som i beviset over, der $g = a_1 g_1 + \cdots + a_t g_t$. Vi har vist at resten r er uavhengig av rekkefølge på elementene i G, men det er ikke a_i'ene.

2.5 Hvordan finne en Gröbnerbasis?

Vi har vist at en Gröbnerbasis lar oss løse «normalformproblemet», og at den dermed er en «fin» genererende mengde som vi etterlyste innledningsvis. Nye spørsmål reiser seg imidlertid fort: Hvordan sjekker man at noe er en Gröbnerbasis (man kjenner vanligvis ikke alle elementene i et ideal), og hvordan kan man selv konstruere en Gröbnerbasis? Igjen må vi innføre noen nye begreper for å gjøre arbeidet lettere.

Definisjon 2.5.1. La f være et polynom i R.

(a) Når f divideres med det ordnede s-tuppelet $H = (h_1, \ldots, h_s)$, der h_i er i R, skriver vi $\overline{f}^H$ for resten ved polynomdivisjonen.

(b) La $F = \{f_1, \ldots, f_s\} \subseteq R$. Dersom det finnes polynomer $a_1, \ldots, a_s$ i R slik at

$$f = a_1 f_1 + \ldots + a_s f_s + f'$$

med $LT(a_i f_i) \preceq LT(f)$ for alle i, sier vi at f **reduserer til** f' modulo F. Vi skriver dette som $f \xrightarrow{F} f'$.

Merk. 1. Definisjon (a) forutsetter en bestemt rekkefølge på elementene i H, mens definisjon (b) sier at det *finnes* en rekkefølge på elementene i F, slik at vi får nevnte rest.

2. Dersom G er en Gröbnerbasis, vil $r = \overline{f}^G$ være uavhengig av rekkefølgen innad i G, så f modulo G vil alltid være r. Som tidligere påpekt, vil a_i'ene i del (b) i definisjonen variere.

3. Dersom F *ikke* er en Gröbnerbasis, har vi at dersom $\overline{f}^F = r$ for en rekkefølge på elementene i F, vil vi også ha $f \xrightarrow{F} r$ (følger fra definisjonen og divisjonsalgoritmen). Merk at det omvendte ikke gjelder i alminnelighet, siden resten ikke er unik.

Eksempel 2.5.2. $G = \{x^2 + 1, y^2 + 1, x + y\}$ er en Gröbnerbasis for idealet $I = \langle G \rangle \subseteq \mathbb{F}[x, y]$, gradert leksikografisk ordning med $x > y$. Vi vil redusere $f = x^3y + y^2 + x + 1$ modulo G. Vi får $f = xy(x^2+1) + 2(y^2+1) + (-y+1)(x+y) + (-y-1)$, det vil si at $\overline{f}^G = N(f) = -y - 1$.

Vi skal nå innføre *S-polynomene*, som er avgjørende for det videre arbeidet mot en Gröbnerbasisalgoritme. S-polynomene har nemlig en helt spesiell egenskap, som både gir oss et nytt kriterium for at en gitt mengde er en Gröbnerbasis, og muligheten til å *lage* en Gröbnerbasis.

Definisjon 2.5.3. La f og g være i $R \setminus \{0\}$.

(a) Hvis $deg(LT(f)) = \alpha = (\alpha_1, \ldots, \alpha_n)$ og $deg(LT(g)) = \beta = (\beta_1, \ldots, \beta_n)$, la $\gamma = (\gamma_1, \ldots, \gamma_n)$ der $\gamma_i = max(\alpha_i, \beta_i)$. Vi kaller $\mathbf{x}^\gamma$ det **minste felles multiplum** av $LM(f)$ og $LM(g)$, ofte betegnet $LCM(LM(f), LM(g))$.

(b) **S-polynomet** av f og g er

$$S(f, g) = \frac{x^\gamma}{LT(f)} \cdot f - \frac{x^\gamma}{LT(g)} \cdot g$$

S'en i S-polynom kommer fra ordet *syzygy*, som henviser til at de ledende leddene forsvinner. S-polynomet er nemlig laget slik at vi alltid får en kansellasjon. Dette illustreres best ved et eksempel. I eksempelet under er de to leddene som kansellerer hverandre understreket.

Eksempel 2.5.4. Vi ser på to polynomer i $\mathbb{R}[x,y]$: $f = 2x^4y - x^3 + 1$, $g = xy^2 + y^2$. Monomordingen er gradert leksikografisk ordning med $x > y$. Vi ser at $\gamma = (max(4,1), max(1,2)) = (4,2)$. Dette gir:

$$S(f,g) = \frac{x^4y^2}{2x^4y}f - \frac{x^4y^2}{xy^2}g = \underline{x^4y^2} - \frac{1}{2}x^3y + \frac{1}{2}y - \underline{x^4y^2} - x^3y^2 = -x^3y^2 - \frac{1}{2}x^3y + \frac{1}{2}y$$

Det følgende teoremet viser hvorfor S-polynomene er veldig nyttige. Beviset for teoremet er ikke gitt her, men det følger fra et mer generelt resultat gitt i kapittel 4 (se beviset for teorem 4.2.3 – se spesielt det andre punktet i merknaden som følger beviset).

Teorem 2.5.5. *La I være et ideal i R og $G = \{g_1, \ldots, g_t\}$ en genererende mengde for I. Da er G en Gröbnerbasis for I hvis og bare hvis $S(g_i, g_j) \xrightarrow{G} 0$ for alle par (i,j) der $1 \leq i, j \leq t$.*

2.5.1 Buchbergers algoritme

Buchbergers algoritme, oppkalt etter sin oppfinner, er den første og mest velkjente algoritmen for å finne Gröbnerbasiser. Den bygger direkte på teorien fra de foregående avsnittene. Algoritmen tar utgangspunkt i en genererende mengde for I, si $I = \langle f_1, \ldots, f_s \rangle$. Fra teorem 2.5.5 vet vi at vi har en Gröbnerbasis dersom alle mulige S-polynomer man kan konstruere fra basisen, reduserer til 0. Hovedstrategien i Buchbergers algoritme er som følger: Vi beregner S-polynomer, og hver gang vi finner et som *ikke* reduserer til 0, legger vi til det vi får som rest i den genererende mengden. Algoritmen er gitt på figur 2.2.

Teorem 2.5.6. *Buchbergers algoritme stopper etter et endelig antall steg, og returnerer en Gröbnerbasis for $I = \langle f_1, \ldots, f_s \rangle$.*

Bevis. Først viser vi at algoritmen stopper etter et endelig antall steg, deretter at den returnerte mengden er en Gröbnerbasis for I.

Det er to løkker i algoritmen, den ene kjører inne i den andre. Den innerste, `for`-løkken, kjører alltid et endelig antall ganger, siden F er endelig. Det er altså den ytterste, `while`-løkken, som er kritisk. Vi må vise at før eller siden får vi $F = H$, det vil si at det på et tidspunkt ikke vil legges til flere elementer i F.

Vi innfører følgende notasjon: Den opprinnelige input'en, F, kaller vi F_0. Den versjonen av F som lagres i variabelen H på starten av den i'te iterasjonen av `while`-løkken, kaller vi F_i. Vi har $F_0 \subseteq F_1 \subseteq \ldots \subseteq F_i \subseteq \ldots$. Når vi legger til et nytt element f_δ til F, har vi (siden f_δ er redusert modulo F_{i-1}) at $LT(f_\delta)$ ikke er delelig med noen av $LT(f_j)$, der f_j er i F_{i-1}. Dermed har vi en strengt økende kjede av idealer

$$\langle LT(F_0) \rangle \subset \langle LT(F_1) \rangle \subset \ldots \subset \langle LT(F_i) \rangle \subset \ldots$$

Algoritme 2.2 Buchbergers algoritme

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
$H := \{\}$
while $F \neq H$ **do**
  $H := F$
  $\delta := s + 1$
  for alle par (f_i, f_j) i H der $i \neq j$ **do**
    $S_{i,j} = \frac{x^\gamma}{LT(f_i)} \cdot f_i - \frac{x^\gamma}{LT(f_j)} \cdot f_j$
    $r := \overline{S_{i,j}}^H$
    if $r \neq 0$ **then**
      $f_\delta := LC(r)^{-1} \cdot r$ # vil ha f_δ monisk
      $F := F \cup \{f_\delta\}$
      $\delta := \delta + 1$
    end if
  end for
end while
$G := F$
return G

Men siden vi befinner oss i en noethersk ring, kan vi ikke ha en uendelig slik rekke. Altså må vi før eller siden slutte å fylle på nye elementer i F, og da er betingelsen $F = H$ oppfylt.

At den returnerte mengden G er en Gröbnerbasis, følger av konstruksjonen med S-polynomene (teorem 2.5.5). Det er klart at siden $G \subseteq I$ og $\{f_1, \ldots, f_s\} \subseteq G$, må G og $\{f_1, \ldots, f_s\}$ generere det samme idealet. Altså gir Buchbergers algoritme en Gröbnerbasis for det ønskede idealet etter et endelig antall steg.♣

Den forenklede versjonen av Buchbergers algoritme som er gjengitt her, er nokså upraktisk, blant annet fordi den beregner *samtlige* S-polynomer hver gang den kjører while-løkken. Dette er det imidlertid lett å fikse på. I eksempelet under beregnes hvert S-polynom kun én gang. Dessuten har vi tatt oss den frihet å gange de nye basiselementene med en passelig konstant slik at de får ledende koeffisient 1 (for enkelhets skyld, dette endrer ikke idealet de genererer). Vi har også oppdatert F fortløpende, i motsetning til algoritmen, som kjører en hel runde med S-polynomer før den legger til alle nye funn til slutt.

Eksempel 2.5.7. Vi ser på $R = \mathbb{F}_5[x, y]$, gradert leksikografisk ordning med $x > y$. $F = \{f_1, f_2\}, f_1 = x^3y + 1, f_2 = xy^2 + y$. Finn en Gröbnerbasis for $\langle f_1, f_2 \rangle$. Vi går i gang med å beregne S-polynomer, og legger til de reduserte S-polynomene dersom de ikke er 0. Slik fortsetter vi helt til alle nye S-polynomer reduserer til 0 modulo F.

$S(f_1, f_2) = \frac{x^3y^2}{x^3y}(x^3y + 1) - \frac{x^3y^2}{xy^2}(xy^2 + y) = -x^2y + y \xrightarrow{F} -x^2y + y \implies f_3 = x^2y - y$, $F := F \cup \{f_3\}$
$S(f_1, f_3) = xy + 1 \xrightarrow{F} xy + 1 \implies f_4 = xy + 1,\ F := F \cup \{f_4\}$
$S(f_2, f_3) = xy - y^2 \xrightarrow{F} -y^2 + 1 \implies f_5 = y^2 - 1,\ F := F \cup \{f_5\}$

$S(f_1, f_4) = -x^2 + 1 \xrightarrow{F} -x^2 + 1 \implies f_6 = x^2 - 1,\ F := F \cup \{f_6\}$
$S(f_2, f_4) \xrightarrow{F} 0$
$S(f_3, f_4) = -x - y \xrightarrow{F} -x - y \implies f_7 = x + y,\ F := F \cup \{f_7\}$
$S(f_1, f_5) \xrightarrow{F} 0,\ S(f_2, f_5) \xrightarrow{F} 0$ ...
Faktisk får vi $S(f_i, f_j) \xrightarrow{F} 0$ for $1 \leq i \leq 6$ og $5 \leq j \leq 7$, så vi setter $G = F$ og konkluderer med at G er en Gröbnerbasis for $\langle f_1, f_2 \rangle$.

Buchbergers algoritme i versjonen brukt over kan fort gi unødvendig store Gröbnerbasiser. Vi vet at dersom G er en Gröbnerbasis for I, og vi har gitt et element f i I, så finnes en $g_i \in G$ slik at $LT(g_i)$ deler $LT(f)$. I eksempelet over ser vi at mange av de ledende leddene i G deler *hverandre*. Det er nærliggende å tenke at dette er unødvendig, og at noen av elementene i G kanskje kan være overflødige. Dette leder oss til spørsmålet: Finnes det en «beste» Gröbnerbasis? Vi tenker på Gröbnerbasiser som en «fin» genererende mengde, men noen fine ting er jo finere enn andre ...

Definisjon 2.5.8. En **minimal Gröbnerbasis** for I er en Gröbnerbasis G for I der følgende er oppfylt:

(i) $LC(g) = 1$ for alle g i G.

(ii) For alle g i G kan ikke $LT(g)$ deles med $LT(g')$, der $g' \neq g$ er i G.

Lemma 2.5.9. *Gitt en Gröbnerbasis G der alle elementene har ledende koeffisient lik 1. Dersom vi har g_1 og g_2 i G slik at $LT(g_1)$ deler $LT(g_2)$, kan vi ta ut g_2 fra G og sitte igjen med en ny Gröbnerbasis $G' = G \setminus \{g_2\}$, der $\langle G' \rangle = \langle G \rangle$.*

Beviset for lemmaet følger rett av definisjonen på Gröbnerbasis. Med dette lemmaet er det lett å gjøre en Gröbnerbasis minimal.

Eksempel 2.5.10. I forrige eksempel fant vi Gröbnerbasisen $G = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$. Vi ser at

$$\begin{array}{lcl} LT(f_3) | LT(f_1) & \implies & G := G \setminus \{f_1\} \\ LT(f_4) | LT(f_2) & \implies & G := G \setminus \{f_2\} \\ LT(f_4) | LT(f_3) & \implies & G := G \setminus \{f_3\} \\ LT(f_7) | LT(f_4) & \implies & G := G \setminus \{f_4\} \\ LT(f_7) | LT(f_6) & \implies & G := G \setminus \{f_6\} \end{array}$$

Basisen vi sitter igjen med, er liten og håndterlig: $G = \{f_7, f_5\} = \{x + y, y^2 - 1\}$.

Men selv om vi kan lage minimale Gröbnerbasiser, vil de ikke være *entydige* for idealene de genererer. For å få til dette, må vi innføre *reduserte* Gröbnerbasiser.

Definisjon 2.5.11. En **redusert Gröbnerbasis** for I er en Gröbnerbasis G for I der følgende er oppfylt:

(i) $LC(g) = 1$ for alle g i G.

(ii) For alle g i G har vi at ingen monomer som er i g ligger i $\langle LT(G \setminus \{g\}) \rangle$.

Det siste resultatet vi har med, forteller oss at alle idealer har en redusert Gröbnerbasis, og i beviset ser vi hvordan vi konstruerer en, gitt en minimal Gröbnerbasis.

Proposisjon 2.5.12. *La $I \neq (0)$ være et ideal i R. Da, gitt en monomordning, har I en unik redusert Gröbnerbasis.*

Bevis. Vi viser først at dersom vi har to reduserte Gröbnerbasiser for idealet I, la oss kalle dem G og G', må de være like. For g_i i G har vi, siden g_i er i I, en g'_j i G' slik at $LT(g_i) = h \cdot LT(g'_j)$. Tilsvarende, siden g'_j er i I, har vi en g_k i G slik at $LT(g'_j) = f \cdot LT(g_k)$. Da er $LT(g_i) = h \cdot f \cdot LT(g_k)$. Siden G, G' er minimale, får vi dermed $LT(G) = LT(G')$, og for hver g_i i G finnes g'_j i G' med $LT(g_i) = LT(g'_j)$.

Vi har at $g_i - g'_j$ er i I, og dermed har vi $g_i - g'_j \xrightarrow{G} 0$. Siden $LT(g_i) = LT(g'_j)$, kanselleres disse leddene når vi trekker dem fra hverandre. Det vil si at $(g_i - LT(g_i)) - (g'_j - LT(g'_j))$ er i I, og siden G og G' er reduserte, er ingen av leddene som er igjen divisible med noen av elementene i $LT(G)$. Dermed lar ikke $g_i - g'_j$ seg redusere, men siden vi allerede har vist at $g_i - g'_j \xrightarrow{G} 0$, må vi ha $g_i - g'_j = 0$, som gir $g_i = g'_j$, og dermed har vi $G = G'$.

Det gjenstår å vise at en redusert Gröbnerbasis for I faktisk eksisterer. Vi vet at vi kan finne en minimal Gröbnerbasis, $G = \{g_1, \ldots, g_t\}$, for I. Ved å gå gjennom elementene i denne og bytte ut hver enkel g_i med $\overline{g_i}^{G\setminus\{g_i\}}$ (først beregne $\overline{g_1}^{G\setminus\{g_1\}}$, deretter erstatte den gamle g_1 med denne før vi går videre osv.), får vi en mengde der alle elementene er reduserte. Siden de ledende leddene er i behold (G var minimal), er dette fortsatt en Gröbnerbasis for I. ♣

Merk. Beviset over er konstruktivt, men det forutsetter en minimal Gröbnerbasis. Den eneste måten vi har sett på for å finne en minimal Gröbnerbasis, tar utgangspunkt i en Gröbnerbasis, altså må vi fortsatt kjøre Buchbergers algoritme først. Å modifisere denne slik at den finner en minimal/redusert Gröbnerbasis direkte er noe mer komplisert enn prosessen vi brukte i beviset over.

Eksempel 2.5.13. $G = \{x + y, y^2 - 1\}$ er en redusert Gröbnerbasis for $\langle x^3y + 1, xy^2 + y\rangle$ under betingelsene i eksempel 2.5.7.

Ved hjelp av reduserte Gröbnerbasiser kan vi løse problem nr. 2 fra innledningen. For å finne ut om to idealer er de samme, er det «bare» å beregne reduserte Gröbnerbasiser for begge, og se om de er like. (Grunnen til at «bare» står i hermetegn, er at selv med de beste algoritmer for å finne Gröbnerbasiser, kan det ta svært lang tid. Basisene kan bli store. Mer om algoritmer og effektivitet kommer i kapittel 4 og kapittel 5.)

2.6 Løsing av likningssystemer

En av de praktiske anvendelsene av Gröbnerbasiser er, som nevnt innledningsvis, at de kan brukes til å løse likningssystemer på formen

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

der f_i er i R for alle $1 \leq i \leq s$. Siden vi trenger denne anvendelsen i neste kapittel, tar vi med en kort introduksjon til emnet her. For en grundigere innføring i elimininasjonsteori, se [7, kapittel 3].

Anta at vi har et liknigssystem som over. La $I = \langle f_1, \ldots, f_s \rangle$, et ideal i R. La $G = \{g_1, \ldots, g_t\}$ være en Gröbnerbasis for I. Siden I og $\langle G \rangle$ er like, har vi at alle de opprinnelige likningene er 0 hvis og bare hvis vi har $g_1(x_1, \ldots, x_n) = 0, \ldots, g_t(x_1, \ldots, x_t) = 0$. Av grunner vi snart skal komme til, endrer vi fokus fra vårt opprinnelige likningssystem til det siste, der venstresidene i systemet utgjør en Gröbnerbasis for I.

Når vi løser *lineære* likningssystemer med n ukjente ved hjelp av Gauss-eliminasjon, er målet vårt å få en matrise på trappeform, der vi typisk kan lese av verdien for x_n, bruke denne til å finne verdien for x_{n-1} og så videre. Men systemet over er ikke nødvendigvis lineært, så vanlig Gauss-eliminasjon fungerer ikke. Istedenfor skal vi bruke følgende definisjon for å eliminere variabler:

Definisjon 2.6.1. Fikser en monomordning på R. La $I = \langle f_1, \ldots, f_s \rangle \subseteq k[x_1, \ldots, x_n] = R$ være et ideal. Da er det **l'te eliminasjonsidealet** av I, kalt I_l, gitt ved

$$I_l = I \cap k[x_{l+1}, \ldots, x_n]$$

Vi ser lett at I_l er et ideal.

Vi ønsker å eliminere variabler fra likningene våre, for å stå igjen med en likning i bare én variabel. I idealet I_{n-1} finnes bare variabelen x_n. Hvordan vi kan utnytte I_{n-1} og Gröbnerbasisen, ser vi av det følgende resultatet:

Teorem 2.6.2 (Eliminasjonsteoremet)**.** *La $I \subseteq R$ være et ideal med Gröbnerbasis G. Vi bruker leksikografisk ordning med $x_1 > \ldots > x_n$. Da er, for alle $0 \leq l \leq n$, mengden*

$$G_l = G \cap k[x_{l+1}, \ldots, x_n]$$

en Gröbnerbasis for I_l.

Før vi beviser eliminasjonsteoremet, trenger vi dette lemmaet:

Lemma 2.6.3. *Fikser en l mellom 0 og n, og se på underringen $k[x_l, \ldots, x_n]$ av R under leksikografisk ordning med $x_1 > \ldots > x_n$. Da har vi at et polynom f i R er i $k[x_l, \ldots, x_n]$ hvis og bare hvis $LT(f)$ er i $k[x_l, \ldots, x_n]$.*

Bevis. Det er klart at dersom f er i $k[x_l, \ldots, x_n]$ er også $LT(f)$ det. Vi må vise den motsatte implikasjonen. Anta at vi har et element f i R der $LT(f)$ er i $k[x_l, \ldots, x_n]$. Dersom det finnes et ledd i f som *ikke* er i $k[x_l, \ldots, x_n]$, inneholder dette leddet minst en x_j der $1 \leq j \leq l-1$. Med da er dette leddet større enn $LT(f)$, fra definisjonen av leksikografisk ordning. Dermed finnes ingen slike ledd i f, og f er da i $k[x_l, \ldots, x_n]$. ♣

Nå kan vi bevise Eliminasjonsteoremet.

Bevis. Vi fikserer en l mellom 0 og n. Det er klart at $G_l \subseteq I_l$. Vi må vise at $\langle LT(G_l)\rangle = \langle LT(I_l)\rangle$. Siden $G_l \subseteq I_l$, har vi $\langle LT(G_l)\rangle \subseteq \langle LT(I_l)\rangle$, så vi trenger bare å vise at den omvendte inklusjonen holder.

La f være et polynom i I_l. Siden f også ligger i I, finnes en g i G slik at $LT(g)$ deler $LT(f)$. Siden $LT(g)$ deler $LT(f)$, ligger $LT(g)$ i $k[x_l, \ldots, x_n]$, og fra lemmaet over vet vi at da gjør også g det. Det følger at g er i G_l. For enhver f i I_l finnes altså en g i G_l slik at $LT(g)|LT(f)$, så dermed er $\langle LT(I_l)\rangle \subseteq \langle LT(G_l)\rangle$, og teoremet følger fra dette. ♣

Eksempel 2.6.4. Vi skal bruke eliminasjonsteoremet til å løse følgende likningssystem over $\mathbb{F}_2[x_1, \ldots, x_6]$. Ordningen er leksikografisk med $x_1 > \ldots > x_6$.

$$\begin{cases} f_1 &= x_1 + x_3 + x_5 + 1 = 0 \\ f_2 &= x_2 + x_4 + x_6 + 1 = 0 \\ f_3 &= x_1x_2 = 0 \\ f_4 &= x_3x_4 = 0 \\ f_5 &= x_5x_6 = 0 \\ f_6 &= x_3x_6 = 0 \\ f_7 &= x_4x_5 = 0 \end{cases}$$

Vi finner følgende reduserte Gröbnerbasis G for $I = \langle f_1, \ldots, f_7\rangle$:

$$\begin{aligned} G = \{&x_1 + x_4 + x_6, \\ &x_2 + x_4 + x_6 + 1, \\ &x_5x_6, \\ &x_4x_5, \\ &x_3 + x_4 + x_5 + x_6 + 1, \\ &x_4x_6\} \end{aligned}$$

Nå bruker vi eliminasjonsteoremet til å finne mulige verdier for variablene $x_1, \ldots, x_6$. Vi starter med x_6, ved å se på idealet I_5:

$G_5 = G \cap \mathbb{F}_2[x_6] = \emptyset$, så idealet $I_5 = (0)$. Altså kan x_6 være både 1 og 0 (tenk på frie variabler i et lineært likningssystem). Vi går videre til I_4:

$G_4 = G \cap \mathbb{F}_2[x_5, x_6] = \{x_5x_6\}$. Vi har altså at $I_4 = \langle x_5x_6\rangle$. Vi vil nå finne x_5. Vi hadde to muligheter for x_6. Dersom x_6 er 1, må x_5 være 0. Dersom x_6 er 0, kan x_5 være både 0 og 1. Vi har nå tre ufullstendige løsninger: Løsning 1 er $(-,-,-,-,0,1)$, løsning 2 er $(-,-,-,-,0,0)$ og løsning 3 er $(-,-,-,-,1,0)$.

$G_3 = G \cap \mathbb{F}_2[x_4, x_5, x_6] = G_4 \cup \{x_4x_5, x_4x_6\}$. Vi ser at løsning 1 og 3 gir bare en mulighet for x_4, mens løsning 2 gir to. Altså har vi nå fire mulige løsninger: $(-,-,-,0,0,1)$, $(-,-,-,0,0,0)$, $(-,-,-,1,0,0)$ og $(-,-,-,0,1,0)$.

$G_2 = G \cap \mathbb{F}_2[x_3, \ldots, x_6] = G_3 \cup \{x_3 + x_4 + x_5 + x_6 + 1\}$ gir oss bare en mulighet for x_3 for hvert av de fire alternativene over, og vi har nå de foreløpige løsningene $(-,-,0,0,0,1)$, $(-,-,1,0,0,0)$, $(-,-,0,1,0,0)$ og $(-,-,0,0,1,0)$.

$G_1 = G \cap \mathbb{F}_2[x_2 \ldots, x_6] = G_2 \cup \{x_2 + x_4 + x_5 + x_6 + 1\}$ som igjen gir en x_2 for hvert alternativ: $(-,0,0,0,0,1)$, $(-,1,1,0,0,0)$, $(-,0,0,1,0,0)$ og $(-,1,0,0,1,0)$.

$G_0 = G \cap \mathbb{F}_2[x_1 \ldots, x_6] = G_1 \cup \{x_1 + x_4 + x_6\}$, og vi kan fylle ut verdiene for x_1 i våre fire løsninger: $(1,0,0,0,0,1)$, $(0,1,1,0,0,0)$, $(1,0,0,1,0,0)$ og $(0,1,0,0,1,0)$. Dette er alle mulige løsninger på det opprinnelige systemet.

Det finnes også andre versjoner av Eliminasjonsteoremet, basert på andre monomordninger. Merk at prosessen beskrevet over ikke alltid fører fram, da den forutsetter at vi klarer å løse likninger med én variabel, noe som ikke alltid er tilfelle. Det kan også oppstå problemer av kompleksitetsmessig art dersom vi får for mange frie variabler.

3 Ikke-kommutative Gröbnerbasiser

3.1 Innledning

Dette kapitlet gir en innføring i ikke-kommutative Gröbnerbasiser, dog ikke på det mest generelle nivået. Ikke-kommutative Gröbnerbasiser kan defineres for mange typer algebraer, men siden vi kun skal bruke teorien på ikke-kommutative polynomringer, begrenser vi oss til disse også i denne introduksjonen.

Som vi snart skal se, er mesteparten av stoffet analogt med de kommutative Gröbnerbasisene. Den mest iøynefallende forskjellen fra disse er at i det ikke-kommutative tilfellet er ting mer *komplisert*. Først og fremst ser elementene vi jobber med ofte mye mer hårete ut – ikke-kommuteringen gjør at vi må forholde oss til dobbeltsummer. Den ikke-kommutative «versjonen» av S-polynomet, overlappsrelasjonen, er også mer uhåndterlig enn sin kommutative lillebror. Men den største forskjellen mellom kommutative og ikke-kommutative Gröbnerbasiser er at gitt et ideal I, har vi nå ingen garanti for at I har en endelig Gröbnerbasis.

Kapitlet bruker stort sett (men ikke alltid) Greens notasjon [16], og definisjoner og resultater er hentet derfra med mindre annet er oppgitt. Imidlertid er oppbyggingen av denne teksten noe annerledes enn hos Green – den følger kapittel 1 om kommutative Gröbnerbasiser så nært som mulig.

3.2 Ringen $R = k\langle x_1, \ldots, x_n \rangle$

Notasjonen som introduseres i dette avsnittet vil bli brukt gjennom hele kapitlet.

Vi skal se på idealer i $R = k\langle x_1, \ldots, x_n \rangle$, der k er en endelig kropp. R er en fri k-algebra, der variablene $x_1, \ldots, x_n$ *ikke* kommuterer. Det vil si at vi for eksempel har $x_1x_2x_1 \neq x_1x_1x_2$, mens vi i det kommutative tilfellet ville skrevet $x_1^2x_2$ for begge uttrykkene. Merk at siden R er en k-algebra, kommuterer konstantene i k: For elementer a og b i k, har vi $(ax_1)(bx_2x_1) = (ab)x_1x_2x_1$.

R har en multiplikativ basis $\mathcal{B}$, som er den ikke-kommutative analogien av mengden monomer (elementene i $\mathcal{B}$ kalles også monomer), og den enkleste måten å tenke på elementene i $\mathcal{B}$ på, er som strenger over alfabetet $\mathcal{V} = \{x_1, \ldots, x_n\}$. *Alle* mulige strenger over $\mathcal{V}$ er med i $\mathcal{B}$, så vi har $\mathcal{B} = \mathcal{V}^*$ (Kleene-tillukningen av $\mathcal{V}$).

Vi noterer oss følgende om $\mathcal{B}$:

(i) Multiplikasjon av to elementer i $\mathcal{B}$ er definert som konkatenasjon av strenger.

(ii) For x og y i $\mathcal{B}$ har vi $x \cdot y$ i $\mathcal{B}$, siden $\mathcal{V}^*$ er lukket under konkatenasjon.

(iii) Siden λ er i $\mathcal{V}^*$, der λ er den tomme strengen, har vi at λ også er i $\mathcal{B}$. Vi har $\lambda x = x\lambda = x$ for alle x i $\mathcal{B}$, og $a\lambda = \lambda a = a$ for alle a i k. Vi skriver 1 istedenfor λ. Merk at 1 er den eneste idempotenten i $\mathcal{B}$.

(iv) $\mathcal{B}$ har ingen nulldivisorer.

Vi skriver $x = x_{\alpha_1} x_{\alpha_2} \cdots x_{\alpha_t}$ for x i $\mathcal{B}$, der x_{α_i} er i $\mathcal{V}$ for $1 \leq i \leq t$. **Lengden** av x er gitt ved $l(x) = |x| = t$ (for $x = 1$ har vi definert $l(1) = 0$). Elementene i R blir da på formen

$$\sum_{\alpha \in A} a_\alpha x_\alpha$$

der a_α er i k og x_α er i $\mathcal{B}$ (summen over er endelig, så $a_\alpha = 0$ for de fleste α). Dersom x er i $\mathcal{B}$, skriver vi $x(i)$ for å få det i'te symbolet i x.

Eksempel 3.2.1. Som nevnt innledningsvis kan Gröbnerbasisteorien for ikke-kommutative algebraer generaliseres til generelle veialgebraer. Merk at vi har $R \simeq k\Gamma$, der Γ er et quiver med $\Gamma_0 = \{v_1\}$ og $\Gamma_1 = \{\beta_1, \ldots, \beta_n\}$. Isomorfien lages ved å sende elementene i $\mathcal{V}$ på elementene i Γ_1 i et 1-1-forhold, og $\lambda \longmapsto e_1$. Avbildningen utvides til å sende $\mathcal{B}$ på basisen for $k\Gamma$.

Når det gjelder *idealer* i R, må vi først understreke at vi ser på *tosidige* idealer (i det kommutative tilfellet er alle idealer tosidige). Anta at vi har et ideal $I \subseteq R$ og en genererende mengde $\{f_1, \ldots, f_s\}$ for I. Fra definisjonen av et ideal har vi:

(i) Dersom f_1 og f_2 er i I, er også $f_1 + f_2$ i I.

(ii) Dersom f er i I og h og g er i R, har vi at $h \cdot f \cdot g$ er i I.

Ut fra dette får vi at et generelt element i I er på formen

$$f = \sum_{i=1}^{s} \sum_{j=1}^{d_i} h_{ij} f_i g_{ij}$$

Det holder altså ikke med en enkeltsum, slik vi hadde før. Vi illustrerer dette med et eksempel.

Eksempel 3.2.2. La $R = \mathbb{F}_2\langle x, y \rangle$, $I = \langle xy, xxx \rangle = \langle f_1, f_2 \rangle$. Gitt to polynomer g og h i I, der $g = x^2 \cdot xy \cdot y + y \cdot xxx \cdot y^2$ og $h = y \cdot xy \cdot x^3 + yx \cdot xxx \cdot yx$. Da har vi

$$\begin{aligned} g + h &= x^2 \cdot xy \cdot y + y \cdot xy \cdot x^3 + y \cdot xxx \cdot y^2 + yx \cdot xxx \cdot yx \\ &= \sum_{j=1}^{2} a_{1j} f_1 b_{1j} + \sum_{j=1}^{2} a_{j2} f_2 b_{j2} \\ &= \sum_{i=1}^{2} \sum_{j=1}^{d_i} a_{ij} f_i b_{ij} \end{aligned}$$

Vi klarer ikke å trekke sammen denne dobbeltsummen til en enkeltsum.

Nå har vi «blitt kjent med» ringen R, og vi går videre til neste punkt, som er monomordninger, eller *tillatelige ordninger* som de heter i det ikke-kommutative tilfellet.

3.3 Tillatelige ordninger og divisjonsalgoritme

Vi ser på ordninger på $\mathcal{B}$ for å kunne sortere leddene i polynomene i R.

Definisjon 3.3.1. Gitt en ordning på $\mathcal{V}$, si $x_1 > \ldots > x_n$ (vi har dessuten $1 < x_n$). En velordning (og dermed en total ordning) $\prec$ på $\mathcal{B}$ kalles **tillatelig** dersom vi har, for p, q, r, s i $\mathcal{B}$, det følgende:

(i) Dersom $p \prec q$, er $pr \prec qr$ og $sp \prec sq$.

(ii) Dersom $p = qr$, er $q \preceq p$ og $r \preceq p$.

Merk. Hvis vi har $q, r \neq 1$ i (ii), har vi at dersom $p = qr$, må $q \prec p$ og $r \prec p$.

Merk. Leksikografisk ordning er *ikke* en tillatelig ordning. Faktisk er den ikke en gang en velordning på $\mathcal{B}$! I det kommutative tilfellet var det eksponentvektorene til monomene vi ordnet leksikografisk. Nå må vi istedenfor se på strengene og bokstavelig talt ordne dem som i et leksikon. Vi ser at mengden $\{x \in \mathcal{B} | x \text{ består kun av } x_1\} = \{x_1, x_1x_1, x_1x_1x_1, \ldots\}$ ikke har et minste element, siden $x_1 > x_1x_1 > x_1x_1x_1 > \ldots$ Altså er ikke leksikografisk ordning en velordning på $\mathcal{B}$.

Her følger noen av de vanligste tillatelige monomordningene.

Definisjon 3.3.2. (Venstre) lengde-leksikografisk ordning, $<_{lex}$. La $p, q \in \mathcal{B}$. Vi har $p <_{lex} q$ hvis $l(p) < l(q)$, eller hvis $l(q) = l(p)$ og det finnes en $i \leq l(p)$ slik at $p(j) = q(j)$ for alle $j < i$ og $p(i) < q(i)$.

Eksempel 3.3.3. $\mathcal{V} = \{a, b\}$, gitt ordningen $a > b$. La $p = aba, q = aaab$ og $r = abb$. Da er $p <_{lex} q$ siden $l(p) = 3 < 4 = l(q)$, og $r <_{lex} p$ siden $l(p) = l(r)$, $a = a, b = b$ og $b < a$.

Definisjon 3.3.4. (Venstre) vekt-leksikografisk ordning, $<_{wlex}$. Definer en funksjon $w : \mathcal{V} \to \mathbb{N}$, som tilordner alle symboler i $\mathcal{V}$ en vekt $w(x_i)$. Vi utvider w til $W : \mathcal{B} \to \mathbb{N}$, slik at for $p = x_{\alpha_1} \cdots x_{\alpha_t}$ har vi $W(p) = w(x_{\alpha_1}) + \ldots + w(x_{\alpha_t})$. Vi har $p <_{wlex} q$ hvis $W(p) < W(q)$, eller $W(p) = W(q)$ og $p <_{lex} q$.

Merk.

1. Lengde-leksikografisk ordning kan defineres som vekt-leksikografisk ordning med $w(x_i) = 1$ for alle x_i i $\mathcal{V}$.
2. Vi må være påpasselige med elementet 1 i $\mathcal{B}$. I forrige eksempel hadde vi $r <_{lex} p$. Dette holder selv om vi skriver om $r = a1b1b$, siden $l(r) = l(a) + l(1) + l(b) + l(1) + l(b) = 1 + 0 + 1 + 0 + 1 = 3 = l(abb)$. Tilsvarende må vi ha $W(1) = 0$ for at vekt-ordningen skal gi mening.
3. (Høyre) lengde-leksikografisk ordning er definert likt som den venstre, bortsett fra at vi sammenlikner fra høyre dersom lengden er lik.
4. (Venstre) vekt-omvendt-leksikografisk ordning er definert likt som den vanlige, bortsett fra at vi bruker (høyre) lengde-leksikografisk ordning dersom vekten er lik.

Definisjon 3.3.5. Kommutative ordninger. Velg en monomordning $<_c$ på de kommutative

monomene over $\{x_1, \ldots, x_n\}$. For p i $\mathcal{B}$, definer $\overline{p}$ som p der variablene kommuterer. Da er $p \prec q$ hvis $\overline{p} <_c \overline{q}$, eller hvis $\overline{p} =_c \overline{q}$ og $p <_{lex} q$.

Eksempel 3.3.6. Hvis vi setter $<_c$ til gradert leksikografisk ordning, får vi såkalt *total leksikografisk ordning*. La $p = abaac$ og $q = ccaba$ i $k\langle a, b, c\rangle$ med $a > b > c$. Vi får $q <_{tot} p$, siden $\overline{q} = a^2bc^2 <_c a^3bc = \overline{p}$.

Definisjon 3.3.7. Gitt en tillatelig ordning $\prec$ og en f i R, der $f = c_1p_1 + \ldots + c_tp_t$, med p_i i $\mathcal{B}$ og c_i i k. Anta at leddene i f er sortert med hensyn til $\prec$ slik at $p_1 > p_2 \ldots > p_t$. Da definerer vi følgende:

(a) **Tuppen** av f, skrevet $\text{tip}(f)$, er monomet i f lengst til venstre – altså p_1.

(b) **Tuppkoeffisienten** til f, skrevet $\text{Ctip}(f)$, er koeffisienten til $\text{tip}(f)$ – altså c_1.

(c) **Halen** av f er gitt ved $\text{tail}(f) = f - \text{Ctip}(f) \cdot \text{tip}(f)$.

(d) **Tuppmengden** til en mengde F er gitt ved $\text{Tip}(F) = \{\text{tip}(f) | f \in F\}$.

(e) Komplementet av $\text{Tip}(F)$ er gitt ved $\text{NonTip}(F) = \mathcal{B} \setminus \text{Tip}(F)$.

Merk. Vi ser at tip(f) tilsvarer ledende monom for det kommutative tilfellet, og det brukes stort sett analogt med ledende ledd. NonTip(F) kommer vi til tilbake til når vi skal se på normalformen av f i R/I.

Polynomdivisjon i R er nesten helt likt som for det kommutative tilfellet, bortsett fra at vi får dobbeltsummer og tosidig divisjon. Divisjonsalgoritmen for R er gitt i algoritme 3.1, der f deles med $F = \{f_1, \ldots, f_s\}$ slik at

$$f = \sum_{i=1}^{s} \sum_{j=1}^{k_i} a_{ij} f_i b_{ij} + r$$

der resten r er slik at ingen elementer i $\text{Tip}(F)$ deler noen ledd i r. Output'en er gitt som s lister med k_i tupler (a_{ij}, b_{ij}) og r.

Eksempel 3.3.8. $\mathbb{F}_2\langle x, y, z\rangle$, lengde-leksikografisk ordning med $x > y > z$. Vi har gitt $f = yxzx + xzy + yxz + x$ og $f_1 = xz - y$ og $f_2 = yx - z$. Vi ønsker å finne (a_{ij}, b_{ij}), $1 < j < 2$, $1 < i < k_j$, og r, ved hjelp av divisjonsalgoritmen. Utregningene er utelatt herfra.

Dersom vi beholder rekkefølgen f_1, f_2, får vi:

$$f = yf_1x + f_1y + yf_1 + yf_2 + (2yy + yz + x)$$

Bytter vi på rekkefølgen til f_2, f_1, får vi derimot

$$f = f_2zx + f_2z + f_1y + zf_1 + (zzx + yy + zz + x)$$

Vi ser at resten ikke ble den samme i de to tilfellene.

Definisjon 3.3.9. Dersom F er en uordnet mengde med polynomer, sier vi at f **reduserer til** f' modulo F dersom det finnes polynomer a_{ij} og b_{ij} i R slik at

$$f = \sum_{i,j} a_{ij} f_i b_{ij} + f'$$

Algoritme 3.1 Divisjonsalgoritme i $k\langle x_1, \ldots, x_n\rangle$

input: $f, \{f_1, \ldots, f_s\}$
output: $r, (a_{ij}, b_{ij})$
initialiserer
$h := f$
$a_{ij} := 0, b_{ij} := 0, r := 0$
$i := 1$
$t_i := 0, \ldots, t_s := 0$
# kjører selve divisjonsalgoritmen
while $h \neq 0$ **do**
  if $\text{tip}(h) = a \cdot \text{tip}\ (f_i) \cdot b$ for $a, b \in \mathcal{B}$ **then**
    # hvis vi kan redusere f med f_i
    $t_i := t_i + 1$
    $a_{it_i} := [\text{Ctip}(h)/\text{Ctip}(f_i)] \cdot a$
    $b_{it_i} := b$
    $h := h - a_{it_i} f_i b_{it_i}$ # reduserer med f_i
    $i := 1$ # start på f_1 igjen
  else if $i < s$ **then**
    # hvis vi ikke kan redusere med f_i, gå til f_{i+1}
    $i := i + 1$
  else
    # hvis vi ikke kan redusere med noen f_i, trekk ut $\text{Ctip}(h) \cdot \text{tip}(h)$
    $r := r + \text{Ctip}(h) \cdot \text{tip}(h)$
    $h := h - \text{Ctip}(h) \cdot \text{tip}(h)$
    $i := 1$
  end if
end while
return $r, (a_{ij}, b_{ij})$

der ingen ledd i f' er delelige med $\text{tip}(f_i)$ for noen f_i i F, og vi har, for alle i og j, at $\text{tip}(f) \succeq \text{tip}(a_{ij} f_i b_{ij})$. Dette kan også skrives som

$$f \xrightarrow{F} f'$$

Som i det kommutative tilfellet skriver vi $\overline{f}^F$ for resten ved polynomdivisjonen der f deles på det ordnede s-tuppelet F.

Vi har samme mål som i kapittel 2: Gitt et ideal $I \subseteq R$, finnes en genererende mengde G for I slik at $\overline{f}^G$ er uavhengig av rekkefølgen på elementene i G under divisjonen? Dette vil i så fall gjøre at vi kan finne *normalformen* $N(f)$ av f i R/I.

3.4 Gröbnerbasiser

Den følgende proposisjonen ser tilsynelatende ut som den ikke-kommutative versjonen av Dicksons lemma, skjønt like anvendelig er den ikke. Der Dicksons lemma ga oss en *endelig* generatormengde for et ideal generert av monomer, må vi her klare oss med en *minimal* sådan. Beviset for denne proposisjonen er viktig fordi det gir en konstruksjon som vi kommer til å bruke i beviset for proposisjon 3.4.11.

Proposisjon 3.4.1. *La I være et ideal generert av en (ikke nødvendigvis endelig) mengde elementer i $\mathcal{B}$. Da har I en unik minimal genererende (ikke nøvendigvis endelig) mengde av elementer i $\mathcal{B}$ med hensyn til en fiksert tillatelig ordning $\prec$.*

Bevis. La $A = \{p_1, p_2, \ldots\}$ være mengden av alle monomer i I. Vi vil plukke ut alle monomer p i A som er slik at dersom et annet monom p' i A deler p, så må $p' = p$. Vi kaller mengden av slike elementer for M. Vi vil vise at M er den unike minimale genererende mengden for I.

Først må vi vise at M ikke er en tom mengde. Anta at vi har en kjede av elementer $q_1, q_2, \ldots$ der q_i er i A for alle i, og slik at q_i deler q_{i-1} for alle $i > 1$ og minst en q_i er ulik 0. Siden q_i deler q_{i-1}, er $q_i \preceq q_{i-1}$ fra definisjonen av tillatelige ordninger. Siden $\prec$ også er en velordning, må $\{q_1, q_2, \ldots\}$ ha et minste element, si q_k, slik at $q_k = q_{k+j}$ for $j \geq 0$.

La oss nå se på alle kjeder $(q_k, q_1', q_2' \ldots)$ i A der hvert element deler det foregående. Hver av disse kjedene har et «stabiliseringspunkt» som er mindre eller lik q_k. Vi bruker velordnetheten igjen og finner det minste av disse «stabiliseringspunktene», og kaller det q_s. Nå har vi at dersom et element p i A deler q_s, har vi $p = q_s$, og $q_s \neq 0$ er dermed et element i M.

Vi må videre vise at $\langle M \rangle = I$, og at M er minimal – det vil si at for alle genererende mengder M' for I, har vi $M \subseteq M'$.

$\langle M \rangle = I$: Vi har $\langle M \rangle \subseteq I$. La p være et monom i I. Anta at $p \notin M$. Da finnes et monom $q_1 \neq p$ i I slik at q_1 deler p. Dersom q_1 er i M, har vi $p = u \cdot q_1 \cdot v$ i M for monomer u og v i $\mathcal{B}$. Hvis ikke, finnes et monom $q_2 \neq q_1$ i I slik at q_2 deler q_1. Slik kan det fortsette, men prosessen er nødt til å stoppe siden $\prec$ er en tillatelig ordning. La q være minste slike stoppunkt for alle mulige rekker $p \succeq q_1 \succeq q_2 \succeq \ldots$ der $q_1 | p$ og $q_{i+1} | q_i$. Da er q i M, og q deler p. Altså er p i $\langle M \rangle$, som gir $\langle M \rangle = I$.

$M \subseteq M'$ når $\langle M' \rangle = I$: La m være i M. Da finnes en m' i M' slik at m' deler m. Men da må $m' = m$, og vi har $M \subseteq M'$. ♣

Definisjon 3.4.2. Gitt et ideal $I \subseteq R$ og en mengde $G \subset I$, og en tillatelig ordning $\prec$ på R. Da er $G \subseteq I$ en **Gröbnerbasis** for I dersom $\langle \text{Tip}(G) \rangle = \langle \text{Tip}(I) \rangle$. Ekvivalent har vi at for alle f i I, finnes g i G og p, q i $\mathcal{B}$ slik at $\text{tip}(f) = p \cdot \text{tip}(g) \cdot q$.

Senere skal vi se at dersom G er Gröbnerbasis for I, har vi at G generer I. Vi har allerede vist at $\langle \text{Tip}(I) \rangle$ har en unik genererende mengde av monomer. Dette gir oss:

Korollar 3.4.3. *La I være et ideal, og la $\mathcal{T}$ være den unike minimale genererende mengden av $\langle \text{Tip}(I) \rangle$ med hensyn til $\prec$. Da er $\{g \in I | \text{tip}(g) \in \mathcal{T}\}$ en Gröbnerbasis for I.*

Altså har vi at alle idealer har en Gröbnerbasis (som ikke nødvendigvis er unik). En Gröbnerbasis for I konstruert som over trenger ikke være endelig, og den kan være unødvendig stor – for eksempel kan det finnes *mange* elementer g i I med $\text{tip}(g) = t$ for en t i $\mathcal{T}$. Vi kan bøte litt på dette ved å innføre en definisjon som er analog med den kommutative minimale Gröbnerbasisen.

Definisjon 3.4.4. La G være en Gröbnerbasis for I ideal. G er **tuppredusert** med hensyn til $\prec$ dersom ingen $\text{tip}(g)$ deler noen $\text{tip}(g')$ for $g \neq g'$ i G.

Tilsvarende som for ikke-kommutative Gröbnerbasiser har vi:

Lemma 3.4.5. *La G være en Gröbnerbasis for I. Dersom $\text{tip}(g)$ deler $\text{tip}(g')$ for $g \neq g'$ i G, er $G \setminus \{g'\}$ også en Gröbnerbasis for I.*

Beviset følger rett fra definisjonen av Gröbnerbasis. De følgende resultatene viser at den ikke-kommutative Gröbnerbasisen har de egenskapene vi er ute etter.

Proposisjon 3.4.6. *Gitt et ideal $I \subseteq R$ og en tillatelig ordning $\prec$. Da er $R = I \oplus \text{Span}(\text{NonTip}(I))$ som k-vektorrom. Altså har vi, for f i R, at $f = i_f + N(f)$ (med i_f i I og $N(f)$ i $\text{Span}(\text{NonTip}(I))$) er en entydig representasjon av f.*

Bevis. Vi viser først at $I \cap \text{Span}(\text{NonTip}(I)) = (0)$. La f være i R, og anta at f er i $I \cap \text{Span}(\text{NonTip}(I))$. Siden f er i I, må $\text{tip}(f)$ være i $\text{Tip}(I)$, og siden $\text{tip}(f)$ er i $\text{Span}(\text{NonTip}(I))$ må $\text{tip}(f)$ også være i $\text{NonTip}(I)$. Da er $\text{tip}(f)$ i $\text{Tip}(I) \cap \text{NonTip}(I) = (0)$, så $\text{tip}(f) = 0$ og dermed er $f = 0$.

Det gjenstår å vise at $R = I + \text{Span}(\text{NonTip}(I))$. La $F = R \setminus (I + \text{Span}(\text{NonTip}(I)))$, altså mengden av elementer i R som *ikke* er i $I + \text{Span}(\text{NonTip}(I))$. Vi vil vise at $F = \emptyset$.

Anta at $F \neq \emptyset$. Siden $\prec$ er en velordning, har $\text{Tip}(F)$ et minste element t, og vi lar f være et element i F med $\text{tip}(f) = t$. La $\text{Ctip}(f) = \alpha$. Vi har to muligheter:

(1) t er i $\text{NonTip}(I)$

Definer $f' = f - \alpha t$. Siden $\text{tip}(f') \prec \text{tip}(f)$, er ikke f i F, så vi har at

$$f' = i_f + s_f \text{ der } i_f \text{ er i } I \text{ og } s_f \text{ er i } \text{Span}(\text{NonTip}(I))$$

Da er $f = f' + \alpha t = i_f + (s_f + \alpha t)$, et element i $I + \text{Span}(\text{NonTip}(I))$, så vi har en motsigelse.

(2) t er i $\text{Tip}(I)$

Da finnes en h i I slik at $\text{tip}(h) = t$. La $\text{Ctip}(h) = \beta$. Definer $f' = f - \frac{\alpha}{\beta}h$. Siden $\text{tip}(f') \prec \text{tip}(f)$, er ikke f' i F, så vi får at

$$f' = i_f + s_f \text{ der } i_f \text{ er i } I \text{ og } s_f \text{ er i } \text{Span}(\text{NonTip}(I))$$

Vi skriver om på f og får $f = f' + \frac{\alpha}{\beta}h = (i_f + \frac{\alpha}{\beta}h) + s_f \in I + \text{Span}(\text{NonTip}(I))$, en motsigelse.

Altså er $F = \emptyset$, og vi har $R = I + \text{Span}(\text{NonTip}(I))$ og dermed $R = I \oplus \text{Span}(\text{NonTip}(I))$. ♣

Definisjon 3.4.7. Vi kaller $N(f)$ for **normalformen** av f i R/I.

La A være en mengde polynomer i R, og fikser et polynom f i R. Gitt en tillatelig ordning $\prec$. Det følger fra definisjonen av tillatelige ordninger at mengden $\{a \in A | \text{tip}(a) \prec \text{tip}(f)\}$ er endelig.

Korollar 3.4.8. *Gitt G, en Gröbnerbasis for idealet $I \subseteq R$ med hensyn til $\prec$, og en f i R. La $G' = \{g_1, \ldots, g_t\} = \{g \in G | \text{tip}(g) \prec \text{tip}(f)\}$. Da har vi $f \xrightarrow{G'} N(f)$, uavhengig av rekkefølgen på $g_1, \ldots, g_t$ i G'. Som en konsekvens er f et element i I hvis og bare hvis $f \xrightarrow{G'} 0$, og vi får $I = \langle G \rangle$.*

Bevis. Anta $f \xrightarrow{G'} r$. Elementene i Span(NonTip(I)) er på formen $\sum_{\alpha \in A} a_\alpha y_\alpha$, der a_α er i k og y_α er i NonTip$(I) = \mathcal{B} \setminus \text{Tip}(I)$. Det er klart at monomene i NonTip(I) ikke lar seg redusere modulo $\langle \text{Tip}(I) \rangle$. Da kan heller ikke elementene i Span(NonTip(I)) reduseres modulo $\langle \text{Tip}(I) \rangle$. Samtidig må alle elementene i R som ikke lar seg redusere modulo $\langle \text{Tip}(I) \rangle$, være i Span(NonTip($I$)). Altså har vi h er i Span(NonTip(I)) hvis og bare hvis $N(h) = h$.

Siden G er en Gröbnerbasis for I, har vi $\langle \text{Tip}(G) \rangle = \langle \text{Tip}(I) \rangle$. Kun de elementene g i G som har $\text{tip}(g) \prec \text{tip}(f)$, kan redusere f, så å redusere f modulo G' gir samme resultat som å redusere f modulo G. Dermed er r i Span(NonTip(I)), og $r = N(f)$ på grunn av entydigheten av $f = i_f + N(f)$.

Det følger at hvis $f \xrightarrow{G'} 0 = N(f)$, er f i I, men hvis $N(f) \neq 0$, er ikke f i I. Altså har vi $I = \langle G \rangle$. ♣

Merk. Vi har en isomorfi mellom k-vektorrommene Span(NonTip(I)) og R/I, gitt av $f \longmapsto N(f) + I$. Vi får

$$\begin{aligned} & f + I = g + I \\ \iff & f - g \in I \\ \iff & N(f) - N(g) = 0 \\ \iff & N(f) = N(g) \end{aligned}$$

Dette er samme egenskaper som for normalform i det kommutative tilfellet.

Eksempel 3.4.9. I det kommutative tilfellet hadde vi at dersom et ideal er generert av et element g, er $\{g\}$ en Gröbnerbasis for $\langle g \rangle$. Slik er det ikke nå. La $g = xx - yz$, og la $I = \langle g \rangle$ være et ideal i $k\langle x, y, z \rangle$, lengde-leksikografisk ordning med $x > y > z$. Vi vil finne normalformen av $f = xxxx$ i R/I. Vi har $f = g \cdot xx + yz \cdot g + yzyz$, men vi kan også skrive $f = x \cdot g \cdot x + xyzx$. Vi fikk ikke samme rest, og dermed er ikke $\{g\}$ en Gröbnerbasis. Det er fort gjort å glemme at reduksjon med et enkelt element ikke alltid er entydig.

Også i det ikke-kommutative tilfellet har vi reduserte Gröbnerbasiser.

Definisjon 3.4.10. La G være en Gröbnerbasis for I ideal. G er **redusert** med hensyn til $\prec$ dersom det følgende holder:

(i) G er en tuppredusert Gröbnerbasis for I.

(ii) $\text{Ctip}(g) = 1$ for alle g i G.

(iii) For alle $g \in G$ har vi at ingen monomer i tail(g) er delelige med noen tip(g') der g' er i $G \setminus \{g\}$.

Merk. Green bruker den nedenstående proposisjonen som definisjon av redusert Gröbnerbasis [16, Definition 2.5]. I denne teksten følger vi, så langt det er mulig, samme struktur som i kapitlet om kommutative Gröbnerbasiser.

Proposisjon 3.4.11. *La $I \neq (0)$ være et ideal i R. Da har I en unik redusert Gröbnerbasis G med hensyn til $\prec$. Videre er $G = \{t - N(t) | t \in \mathcal{T}\}$, der $\mathcal{T}$ er den minimale genererende mengden for $\langle \text{Tip}(I) \rangle$.*

Bevis. Beviset for proposisjon 3.4.1 forteller oss hvordan $\mathcal{T}$ er konstruert. Vi må vise at G er en Gröbnerbasis, at den oppfyller alle kriteriene for en redusert sådan og at den er unik.

G er en Gröbnerbasis for I: Vi har $R = I \oplus \text{Span}(\text{NonTip}(I))$. La t være i $\mathcal{T}$. Da er $t = t_i + N(t)$, som gir $t_i = t - N(t)$. Vi har t_i i I, så tip(t_i) er i Tip(I). Det følger fra divisjonsalgoritmen at tip$(N(t)) \prec$ tip(t), så derfor er tip$(t_i) =$ tip$(t) = t$. Dette betyr at tip(t_i) må være i Tip(G), og vi har $\mathcal{T} \subseteq \text{Tip}(G)$. Siden G ikke har flere elementer enn $\mathcal{T}$, har vi $\mathcal{T} = \text{Tip}(G)$ og dermed $\langle \text{Tip}(G) \rangle = \langle \text{Tip}(I) \rangle$. Det er klart at $G \subseteq I$. Altså er G en Gröbnerbasis for I.

Alle elementer i G har tuppkoeffisient 1: La $g = t - N(t)$ være i G. Vi har Ctip$(g) =$ Ctip$(t - N(t)) =$ Ctip$(t) = 1$ siden $t \in \mathcal{B}$, så alle elementer i G har tuppkoeffisient 1.

G er tuppredusert: La $g, g' \in G$, så $g = t - N(t), g' = t' - N(t')$ for noen $t, t' \in \mathcal{T}$. Anta at g og g' er slik at tip$(g)|$tip(g'). Vi har

$$\text{tip}(g)|\text{tip}(g') \iff \text{tip}(t)|\text{tip}(t') \iff t|t'$$

Da må $t = t'$ fra konstruksjonen av $\mathcal{T}$, så $g = g'$. Dermed er G tuppredusert.

G er redusert: Vi må vise at for en tilfeldig valgt $g \in G$, finnes ingen $g' \in G$ slik at tip(g') deler noen ledd i tail(g). Vi har

$$\text{tail}(g) = g - \text{tip}(g) = (t - N(t)) - t = -N(t) \in \text{Span}(\text{NonTip}(I))$$

og denne lar seg ikke redusere med noen g' i G. Altså er G redusert.

G er unik: Vi antar at G' er en redusert Gröbnerbasis for I. Vi har $\langle \text{Tip}(G') \rangle = \langle \mathcal{T} \rangle$, så $\mathcal{T} \subseteq$ Tip(G') (siden $\mathcal{T}$ er unik minimal genererende mengde). Dersom $\mathcal{T} \subsetneq \text{Tip}(G')$, finnes $g \in G'$ med tip$(g) \notin \mathcal{T}$. Siden $\langle \mathcal{T} \rangle = \langle \text{Tip}(G') \rangle$, må tip$(g) = f \cdot t$ med $f \in R \setminus \{1\}, t \in \mathcal{T}$. Men siden $\mathcal{T} \subseteq \text{Tip}(G')$, finnes $g' \in G'$ med tip$(g') = t$, det vil si at tip$(g')|$tip(g), en motsigelse til at G' er en tuppredusert Gröbnerbasis.

Altså er $\mathcal{T} =$ Tip(G'). For en vilkårlig g i G' har vi da en t i $\mathcal{T}$ slik at tip$(g) = t$. Vi har antatt at G' er en redusert Gröbnerbasis, så tail(g) er ikke reduserbar med noen g' i G', ergo er $N(\text{tail}(g)) = \text{tail}(g)$. Men $N(g) = 0$, siden g er i G'.

Vi har $N(g) = N(\text{tip}(g) + \text{tail}(g)) = N(t) + \text{tail}(g) = 0$. Dette medfører tail$(g) = -N(t)$, som gir $g = t - N(t)$ og endelig at $G' = G$. Så G er den unike reduserte Gröbnerbasisen for I. ♣

3.5 Hvordan finne en Gröbnerbasis?

I forrige avsnitt så vi hvordan vi finner den reduserte Gröbnerbasisen for et ideal, med den forutsetning at man allerede kjenner den minimale genererende mengden for $\langle \mathrm{Tip}(I) \rangle$, og denne vet vi ikke hvordan vi skal finne. Imidlertid finnes en ikke-kommutativ variant av Buchbergers algoritme, som i beste fall gir oss en endelig Gröbnerbasis (ikke nødvendigvis redusert, men det var heller ikke tilfellet for den kommutative utgaven). I verste fall stopper den ikke, noe som skjer dersom idealet vi vil finne Gröbnerbasis for, ikke *har* en endelig Gröbnerbasis.

La oss først definere den ikke-kommutative varianten av S-polynom: overlappsrelasjonen. Denne definisjonen er en blanding av Greens definisjon og Moras definisjon [21].

Definisjon 3.5.1. La f og g være i R, og anta at vi har l, r, λ og ρ i $\mathcal{B}$ slik at

(i) $l \cdot \mathrm{tip}(f) \cdot r = \lambda \cdot \mathrm{tip}(g) \cdot \rho$

(ii) Minst en av $\{l, \lambda\}$ og minst en av $\{r, \rho\}$ er 1.

(iii) $\mathrm{tip}(f)$ deler verken λ eller ρ, og $\mathrm{tip}(g)$ deler verken l eller r.

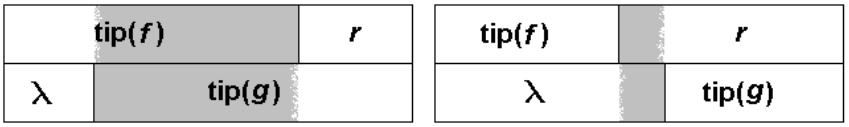
Da er **overlappsrelasjonen** mellom f og g med l, r, λ og ρ gitt ved

$$O(l \cdot f \cdot r, \lambda \cdot g \cdot \rho) = \frac{1}{\mathrm{Ctip}(f)} \cdot lfr - \frac{1}{\mathrm{Ctip}(g)} \cdot \lambda g \rho$$

Vi ser at vi får, som forventet, en kansellering i overlappsrelasjonen.

Merk.

1. To polynomer f og g kan ha flere overlapper. Et polynom kan også overlappe seg selv.
2. Fra punkt (ii) i definisjonen ser vi at vi har tre typer mulige overlapper mellom f og g:
   (a) $l = 1$ og $\rho = 1$. Da er $\mathrm{tip}(f)r = \lambda \mathrm{tip}(g)$.
   (b) $\lambda = 1$ og $r = 1$. Da er $l\mathrm{tip}(f) = \mathrm{tip}(g)\rho$.
   (c) Dersom $\mathrm{tip}(f) \prec \mathrm{tip}(g)$ kan vi ha $\lambda = \rho = 1$ og $l \neq 1 \neq r$ slik at $l\mathrm{tip}(f)r = \mathrm{tip}(g)$. Dersom $\mathrm{tip}(g) \prec \mathrm{tip}(f)$ kan vi tilsvarende ha $\mathrm{tip}(f) = \lambda \mathrm{tip}(g)\rho$. Disse to mulighetene er utelukket fra Greens definisjon.
3. Formålet med punkt (iii) i definisjonen er at $\mathrm{tip}(f)$ og $\mathrm{tip}(g)$ skal ha en «fysisk» overlapp. Se på figur 3.1: På begge bildene har vi $\mathrm{tip}(f)r = \lambda \mathrm{tip}(g)$, men det er bare på den første figuren at vi har en gyldig overlappsrelasjon i samsvar med definisjonen.



Figur 3.1: Overlappsrelasjoner?

Eksempel 3.5.2. Vi ser på polynomene $f = zxx + xy$, og $g = xyz - y$ i $k\langle x, y, z\rangle$, med lengdeleksikografisk ordning med $x > y > z$. Vi har en overlapp, siden $\text{tip}(f) \cdot yz = zx \cdot \text{tip}(g)$. Dette gir $O(f \cdot yz, zx \cdot g) = \underline{zxxyz} + xyyz - \underline{zxxyz} + zxy = xyyz + zxy$.

Det neste resultatet er den ikke-kommutative versjonen av teorem 2.5.5.

Teorem 3.5.3. *Gitt en tillatelig ordning $\prec$ og la $I = \langle G\rangle$ være et ideal i R, der $G = \{g_1, \ldots, g_t\}$. Anta at alle elementene i G er moniske, det vil si at vi har $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$. Da har vi at dersom $O(lg_ir, \lambda g_j\rho)$ reduserer til 0 modulo G for alle mulige overlapper mellom g_i og g_j i G, så er G en Gröbnerbasis for I.*

Bevis. Vi har gitt G, en genererende mengde for idealet $\langle G\rangle \subseteq R$, og lar f være et element i $\langle G\rangle$ slik at ingen g i G er slik at $\text{tip}(g)$ deler $\text{tip}(f)$. Med andre ord antar vi at G ikke er en Gröbnerbasis for $\langle G\rangle$. Vi antar at vi likevel har $O(lg_ir, \lambda g_j\rho) \xrightarrow{G} 0$ for alle overlapper mellom g_i og g_j i G, og viser at dette fører til en motsigelse.

Siden f er i $\langle G\rangle$, finnes monomer $p_{i,j}$ og $q_{i,j}$ i $\mathcal{B}$ og koeffisienter $c_{i,j}$ i k slik at

$$f = \sum_{i,j} c_{i,j}p_{i,j}g_iq_{i,j} \tag{3.1}$$

der g_i er i G. Vi skriver om uttrykket til

$$f = \text{Ctip}(f)\text{tip}(f) + \text{tail}(f) = \sum_{i,j} c_{i,j}p_{i,j}\text{tip}(g_i)q_{i,j} + \sum_{i,j} c_{i,j}p_{i,j}\text{tail}(g_i)q_{i,j} \tag{3.2}$$

Siden $\text{tip}(f)$ finnes på venstre side, er den også på høyre side, men ikke blant leddene i den første summen, siden det ville gitt at $\text{tip}(f)$ er delelig med en $\text{tip}(g_i)$. Dermed finnes tuppen av f i den andre summen på høyre side i (3.2).

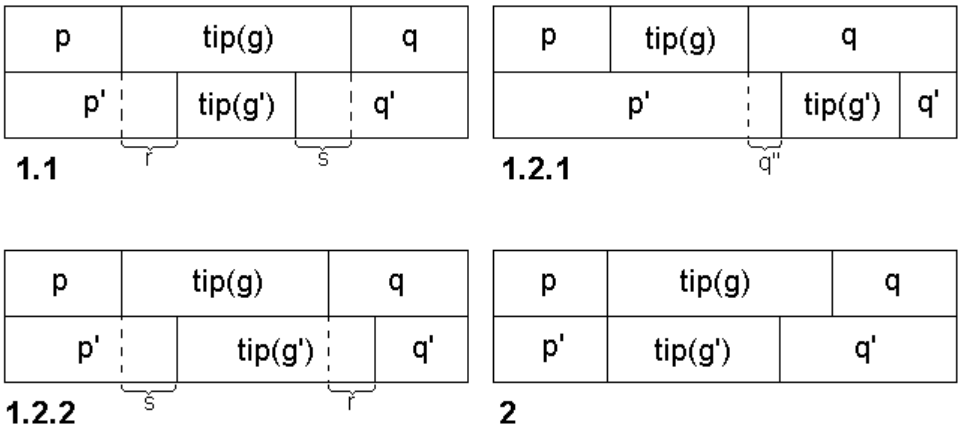
La $p^* = \max_{i,j}\{\text{tip}(p_{i,j}g_iq_{i,j})\} = \max_{i,j}\{p_{i,j}\text{tip}(g_i)q_{i,j}\}$, altså det største monomet på høyre side i (3.1). Merk at er p^* avhengig av hvilken representasjon for f vi valgte i (3.1) (en slik representasjon er ikke unik). Siden $\prec$ er en velordning, kan vi velge en representasjon av f som gir *minst mulig* p^*, og deretter en representasjon der denne p^* opptrer *færrest mulig* antall ganger.

Fra (3.2) har vi at $p^* \succeq \text{tip}(f)$, og siden $p^* = \text{tip}(f)$ går imot antakelsen for f, har vi $p^* \succ \text{tip}(f)$. Da må vi ha et annet ledd i den første summen på høyre siden i (3.2) som kansellerer med p^*. Altså finnes monomer p, q, p' og q' blant $\{p_{i,j}, q_{i,j}\}$ som inngår i representasjonen for f og to tilhørende elementer g og g' i G slik at

$$p^* = p\text{tip}(g)q = p'\text{tip}(g')q' \tag{3.3}$$

Dette gir oss et visst antall muligheter for hvordan p^* er bygd opp. Vi ønsker å vise at samtlige motsier antakelsene våre, og at det dermed ikke kan eksistere noen slik p^*. Figur 3.2 viser mulighetene vi har, dersom $\text{tip}(g) \succeq \text{tip}(g')$.

(1) $p' \succ p$ gir oss videre flere muligheter:



Figur 3.2: Ulike representasjoner av p^*.

(1.1) $q' \succ q$. Da finnes monomer r og s slik at $\text{tip}(g) = r\text{tip}(g')s$. Det betyr at vi har en overlapp med tilhørende overlappsrelasjon

$$O(g, rg's) = g - rg's$$

Vi bruker dette til å skrive om på uttrykket pgq:

$$\begin{aligned} pgq &= pgq - p'g'q' + p'g'q' \\ &= pgq - prg'sq + p'g'q' \\ &= p(g - rg's)q + p'g'q' \\ &= pO(g, rg's)q + p'g'q' \end{aligned} \tag{3.4}$$

Fra antakelsen om at overlappsrelasjonene reduserer til 0 modulo G, vet vi at vi kan uttrykke $O(g, rg's)$ som en sum der alle leddene er mindre enn $p\text{tip}(g)q$, som er det samme som $p'\text{tip}(g')q'$. Altså er tuppen til polynomet vi ender opp med i (3.4) i det *siste* av de to leddene. La nå c være koeffisienten til pgq i (3.2) og c' koeffisienten til $p'g'q'$ samme sted. Da kan vi skrive om to ledd i uttrykket (3.1) slik:

$$cpgq + c'p'g'q' = c(p'g'q' + pO(g, rg's)q) + c'p'g'q' = (c + c')p'g'q' + cpO(g, rg's)$$

som vil redusere antall forekomster av p^* i den valgte representasjonen av f, en motsigelse til antakelsen om minimalitet.

(1.2) $q \succeq q'$ gir oss igjen to muligheter:

(1.2.1) $p' \succeq p\text{tip}(g)$. Da finnes et monom q'' slik at $q = q''\text{tip}(g')q'$. Vi har $g' = \text{tip}(g') + \text{tail}(g')$, så vi kan uttrykke $\text{tip}(g')$ som $g' - \text{tail}(g')$. Vi bruker dette til å omskrive pgq:

$$\begin{aligned} pgq &= pgq''\text{tip}(g')q' \\ &= pgq''(g' - \text{tail}(g'))q' \\ &= pgq''g'q' - pgq''\text{tail}(g')q' \\ &= p(\text{tip}(g) + \text{tail}(g))q''g'q' - pgq''\text{tail}(g')q' \\ &= p\text{tip}(g)q''g'q' + p\text{tail}(g)q''g'q' - pgq''\text{tail}(g')q' \\ &= p'g'q' + p\text{tail}(g)q''g'q' - pgq''\text{tail}(g')q' \end{aligned} \tag{3.5}$$

Vi skriver om to ledd i summen (3.1) akkurat som i tilfelle 1.1.

(1.2.2) $p' \prec p\text{tip}(g)$. Da har vi en overlapp, si $\text{tip}(g) \cdot r = s \cdot \text{tip}(g')$. Fra definisjonen er

$$O(gr, sg') = gr - sg'$$

Igjen finner vi et uttrykk for pgq.

$$\begin{aligned} pgq &= pgq - p'g'q' + p'g'q' \\ &= pgrq' - psg'q' + p'g'q' \\ &= p(gr - sg')q' + p'g'q' \\ &= pO(gr, sg')q + p'g'q' \end{aligned} \tag{3.6}$$

Igjen befinner tuppen seg i siste del av polynomet vi endte opp med i (3.6), og vi kan skrive om som før og få en motsigelse til valgt representasjon av f.

(2) $p = p'$ Vi får en overlapp som i tilfelle 1.2.2, og kan bruke samme framgangsmåte for å komme til en motsigelse.

(3) $p \succ p'$ Dette blir akkurat det samme som tilfelle (1) (bytt rekkefølgen på de to skrivemåtene for p^*). Tilsvarende dersom $\text{tip}(g') \succeq \text{tip}(g)$.

Dette var alle mulighetene vi hadde, og vi har vist at ingen av dem kan være tilfelle under våre antakelser. Da finnes ingen slik f, en motsigelse til antakelsen om at G ikke er en Gröbnerbasis. Altså er G en Gröbnerbasis for I. ♣

Dette gir grunnlag for en modifisert utgave av Buchbergers algoritme, som altså ikke er garantert å stoppe. Algoritmen (gitt på figur 3.2) tar inn en generatormengde $F = \{f_1, \cdots, f_s\}$ og lager en Gröbnerbasis $G = \{g_1, g_2, \cdots\}$ der $\text{tip}(g_i) \notin \langle \text{tip}(g_1), \cdots, \text{tip}(g_{i-1}) \rangle$.

Algoritme 3.2 Ikke-kommutativ Buchbergers algoritme

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
$H := \{\}$
while $F \neq H$ **do**
  $H := F$
  $\delta := s + 1$
  for alle overlappsrelasjoner $O(lf_ir, \lambda f_j\rho)$ **do**
    $r := \overline{O(lf_ir, \lambda f_j\rho)}^H$
    if $r \neq 0$ **then**
      $f_\delta := \text{Ctip}(r)^{-1} \cdot r$ # vil ha f_δ monisk
      $F := F \cup \{f_\delta\}$
      $\delta := \delta + 1$
    end if
  end for
end while
$G := F$
return G

Eksempel 3.5.4. $F = \{f_1, f_2\} = \{xyz + xy, yyx + y\}$ i $k\langle x, y, z\rangle$, lengde-leksikografisk ordning med $x > y > z$. Finn en Gröbnerbasis G slik at $\langle F\rangle = \langle G\rangle$.

Vi ser at f_1 og f_2 bare har en overlapp: $O(f_2 \cdot yz, yy \cdot f_1) = -yyxy + yyz \xrightarrow{f_2} yyz + yy = f_3$. Mengden $G = \{f_1, f_2, f_3\}$ har ingen overlapper, og er dermed en Gröbnerbasis for $\langle F\rangle$.

Vi avslutter dette kapitlet med et noe større eksempel, der vi også får en endelig Gröbnerbasis som resultat.

Eksempel 3.5.5. La $F = \{f_1, f_2, f_3\}$, der

$$\begin{aligned} f_1 &= xyx - x \\ f_2 &= yx - x \\ f_3 &= xx - y \end{aligned}$$

Vi utfører beregningene i ringen $\mathbb{F}_7\langle x, y\rangle$ under lengde-leksikografisk ordning med $y > x$. Vi starter med å beregne overlappsrelasjoner der bare f_1 inngår. Vi ser at vi har to selv-overlapper:

$$\begin{aligned} O(f_1yx, xyf_1) &= -xyx + xyx = 0 \\ O(xyf_1, f_1yx) &= -xyx + xyx = 0 \end{aligned}$$

Vi ser at disse to i praksis er «samme» overlapp – slik vil det alltid være med selv-overlapper. Uansett ble begge 0 direkte, så vi går videre til overlapper mellom f_1 og f_2. Vi finner to stykker:

$$\begin{aligned} O(xf_2, f_1) &= -xx + x \xrightarrow{F} y - x \\ O(f_2yx, yf_1) &= -xyx + yx \xrightarrow{F} 0 \end{aligned}$$

Vi setter $f_4 = y - x$ og inkluderer denne i F. Videre finnes det 5 overlapper der f_3 overlapper seg selv, f_1 eller f_2, men alle reduserer til 0 modulo F. Til slutt får vi to nye overlapper mellom f_4 og de foregående elementene:

$$\begin{aligned} O(xf_4x, f_1) &= -xxx + x \xrightarrow{F} 0 \\ O(f_4x, f_2) &= -xx + x \xrightarrow{F} 0 \end{aligned}$$

Nå reduserer alle overlappsrelasjonene til 0, så $\{f_1, f_2, f_3, f_4\}$ er en Gröbnerbasis for $\langle f_1, f_2, f_3\rangle$. I kapittel 5 skal vi se på hvordan vi kan redusere antall overlappsrelasjoner vi må redusere modulo F.

Del II
Algoritmer for å finne Gröbnerbasiser

«Alt jeg ville si,» brølte datamaskinen, «er at mine kretser nå er ugjenkallelig viet beregningen av det endelige svar på spørsmålet om meningen med livet, universet og alt sammen –» Han stanset, fornøyd med å ha alles oppmerksomhet, før han fortsatte litt lavere. «Men det vil ta en stund før programmet har resultatet tilgjengelig.»

Pool så utålmodig på klokken.

«Hvor lenge?» sa han.

«Syv og en halv millioner år,» sa Dype tanke.

Galkwill og Pool blunket til hverandre.

fra *På tommeltotten til Melkeveien* av Douglas Adams

4 Forbedringer av kommutativ Buchbergers algoritme

4.1 Innledning

I kapittel 2 viste vi at, for et ideal $I = \langle F \rangle = \langle f_1, \ldots, f_s \rangle$ i ringen $R = k[x_1, \ldots, x_n]$, der k er en kropp, holder følgende:

- I har en endelig Gröbnerbasis, $G = \{g_1, \ldots, g_t\}$, slik at $I = \langle G \rangle$.
- Buchbergers algoritme, som gitt i algoritme 2.2, terminerer og returnerer en Gröbnerbasis G for idealet I, gitt $F = \{f_1, \ldots, f_s\}$ der $\langle F \rangle = I$.

Fra et rent teoretisk ståsted er dette nøyaktig det vi trenger – vi har vist at Gröbnerbasiser for at ideal alltid finnes, og vi kjenner til og med en metode for å finne den. Men Buchbergers algoritme, slik vi har sett den foreløpig, er ikke særlig effektiv. Som vi så i avsnitt 2.6, benyttes Gröbnerbasiser til å løse likningssystemer av polynomer, og når systemene blir store, er det en fordel å ha raske og effektive algoritmer for hånden. En anvendelse av Gröbnerbasiser som er spesielt relevant for denne oppgaven er innen *kryptografi*. I del III ser vi på kryptosystemer der mye av sikkerheten ligger i at det skal være beregnbarhetsmessig vanskelig å finne en Gröbnerbasis, og dette kan knyttes til hvor gode algoritmer som finnes. Gröbnerbasiser benyttes også innen kryptanalyse, men det er ikke tema i denne oppgaven.

Allerede i eksempel 2.5.7 modifiserte vi den opprinnelige pseudokoden litt, slik at vi slapp å beregne *samtlige* S-polynomer for hver iterasjon i `while`-løkken – dersom vi har vist at $S(f, g)$ reduserer til 0 modulo F, vil det også redusere til 0 når vi legger til flere elementer i F. (I tillegg gjorde vi elementene i F moniske, men det er en enkel modifikasjon med få beregnbarhetsmessige følger.)

Den enkleste måten å sørge for at hvert S-polynom blir beregnet, redusert og vurdert kun én gang, er å implementere Buchbergers algoritme ved hjelp av en simpel datastruktur ved navn *kø*, kalt Q i algoritmen vi snart gir. Kort fortalt starter vi med å beregne alle S-polynomer $S(f_i, f_j)$ der f_i og f_j er i input'en F, og plasserer dem i køen. Deretter kjører vi en `while`-løkke, som for hver iterasjon tar ut et S-polynom fra køen, reduserer det modulo F, og dersom resultatet er $f_\delta \neq 0$, gjøres dette:

1. Legg f_δ til i F.
2. Beregn $S(f_k, f_\delta)$ der $1 \leq k < \delta$ og plasser disse i køen.

Pseudokoden for denne utgaven av Buchbergers algoritme er vist i algoritme 4.1. Et nytt bevis for terminering og korrekthet er ikke nødvendig, for den eneste forandringen vi har gjort,

er å fjerne allerede utførte beregninger. Resten av dette kapitlet er viet mer betydelige effektiviseringer av Buchbergers algoritme, da særlig Faugères F_4-algoritme, som kanskje fortjener å bli kalt en «egen» algoritme, og ikke bare en modifisert utgave av Buchbergers.

Algoritme 4.1 Buchbergers algoritme, versjon 2

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
$\delta := s + 1$
$Q := \emptyset$
for $i := 1$ **to** $s - 1$ **do**
  for $j := i + 1$ **to** s **do**
    $Q := Q \cup \{S(f_i, f_j)\}$
  end for
end for
while $Q \neq \emptyset$ **do**
  velg $h \in Q$
  $Q := Q \setminus \{h\}$
  $r := h$ modulo F
  if $r \neq 0$ **then**
    $f_\delta := LC(r)^{-1} r$ # vil ha f_δ monisk
    $F := F \cup \{f_\delta\}$
    $\delta := \delta + 1$
    for $k := 1$ **to** $\delta - 1$ **do**
      $Q := Q \cup \{S(f_i, f_\delta)\}$
    end for
  end if
end while
$G := F$
return G

For å finne punkter der Buchbergers algoritme kan forbedres, er det tre spørsmål som er naturlige å stille:

1. Der vi gjør *valg*, som når vi tar et S-polynom ut av køen Q, kan vi gjøre dette på en bedre måte enn å velge tilfeldig?
2. Finnes det overflødige beregninger som kan unngås?
3. Kan «tunge» beregninger, som reduksjonen av S-polynomene, effektiviseres?

Vi starter med å se på punkt 2 i de første tre avsnittene, før vi ser raskt på punkt 1. Det siste punktet blir behandlet i F_4-avsnittet.

4.2 Overflødige S-polynomer 1: Buchbergers kriterier

Siden det blir mange S-polynomer og LCM'er (minste felles multiplum, se definisjon 2.5.3) i dette kapitlet, innfører vi litt forenklende notasjon. For f_i og f_j i F, la

$$\begin{aligned} S(i,j) &:= S(f_i, f_j) \\ L(i,j) &:= LCM(LM(f_i), LM(f_j)) \end{aligned}$$

Vi husker kriteriet som lå til grunn for Buchbergers algoritme, som har som konsekvens at S-polynomer som reduserer til 0, er uvesentlige for Gröbnerbasisen. Her følger et tilfelle der vi kan forutsi at S-polynomet reduserer til 0, uten å beregne det.

Lemma 4.2.1 (Buchbergers første kriterium)**.** *La f_1 og f_2 være polynomer i en endelig mengde $F \subseteq R$. Hvis $L(1,2) = LM(f_1)LM(f_2)$, har vi $S(1,2) \xrightarrow{F} 0$.*

Bevis. Anta, uten tap av generalitet, at $LC(f_1) = LC(f_2) = 1$. La $f_1' = f_1 - LT(f_1)$ og $f_2' = f_2 - LT(f_2)$. Det er klart at $LM(f_i) = f_i - f_i'$. Vi beregner S-polynomet av f_1 og f_2:

$$\begin{aligned} S(1,2) &= \frac{LM(f_1)LM(f_2)}{LM(f_1)} f_1 - \frac{LM(f_1)LM(f_2)}{LM(f_2)} f_2 \\ &= LM(f_2)f_1 - LM(f_1)f_2 \\ &= (f_2 - f_2')f_1 - (f_1 - f_1')f_2 \\ &= f_1'f_2 - f_2'f_1 \end{aligned}$$

Vi har altså, med notasjonen fra definisjon 2.5.1(b), at $S(1,2) = \sum_{i=1}^{s} a_i f_i$, der s er antall elementer i F. Vi mangler bare å vise at $LT(a_i f_i) \preceq LT(S(1,2))$ for $i = 1, 2$. Den eneste måten dette *ikke* kan være tilfelle på, er dersom de ledende leddene i $f_1'f_2$ og $f_2'f_1$ kansellerer hverandre. I så fall må vi ha $LM(f_1')LM(f_2) = LM(f_2')LM(f_1)$, og det går ikke siden vi har antatt at $LM(f_1)$ og $LM(f_2)$ er uten felles faktorer. Dermed reduserer $S(1,2)$ til 0 modulo F. ♣

Som navnet på lemmaet tilsier, ble dette kriteriet først formulert av Buchberger. Et annet av hans resultater er *Buchbergers andre kriterium* (lemma 4.2.4), som i [7] blir bevist ved hjelp av *syzygyer*. Her følger vi [22], som viser at dette kriteriet holder uten å ty til slike (syzygyer møter vi derimot i kapittel 5.6). Istedenfor må vi fire på kravet om at S-polynomene skal redusere til 0.

Definisjon 4.2.2. Anta at g_i og g_j er elementer i $G = \{g_1, \ldots, g_t\} \subseteq R$, slik at $LC(g_i) = LC(g_j) = 1$. Vi sier at S-polynomet $S(i,j)$ **reduserer svakt til 0 modulo** G dersom vi har elementer $a_1, \ldots, a_t$ i R slik at

$$S(i,j) = \sum_{l=1}^{t} a_l g_l$$

med $LT(a_l g_l) \prec L(i,j)$ for alle $1 \leq l \leq t$.

Merk. Dersom $S(i,j)$ reduserer til 0 modulo G, reduserer det også svakt til 0 modulo G, siden vi da har, fra definisjonen av $S(i,j)$, at

$$LT(a_l g_l) \preceq LT(S(i,j)) \prec L(i,j)$$

Med definisjonen av svak reduksjon kan vi lage en ny variant av teorem 2.5.5. Denne gangen gir vi også beviset.

Teorem 4.2.3. *La $G = \{g_1, \ldots, g_t\} \subseteq R$ der $LC(g_i) = 1$ for $1 \leq i \leq t$. Da er G en Gröbnerbasis for idealet $I = \langle G \rangle$ hvis og bare hvis S-polynomet $S(i,j)$ reduserer svakt til 0 modulo G for alle g_i og g_j i G.*

Bevis. $\Rightarrow$ følger direkte fra merknaden over.

$\Leftarrow$ Anta at alle S-polynomene reduserer svakt til 0 modulo G, men at G *ikke* er en Gröbnerbasis for I. Siden G ikke er en Gröbnerbasis, finnes et element f i I der $LT(f)$ ikke er delelig med noen $LT(g_i)$ der g_i er i G. Siden f er i $I = \langle G \rangle$, finnes $p_1, \ldots, p_t$ i R slik at

$$f = \sum_{i=1}^{t} p_i g_i \tag{4.1}$$

Vi definerer $q_i = g_i - LT(g_i) = g_i - LM(g_i)$ og skriver f som

$$f = \sum_{i=1}^{t} p_i LT(g_i) + \sum_{i=1}^{t} p_i q_i \tag{4.2}$$

Siden $LT(f)$ er et ledd på venstre side i (4.2), er den også på høyre side, men ikke i den første summen, siden $LT(g_i)$ ikke deler $LT(f)$. Altså må de leddene i denne summen som er større enn $LT(f)$, kansellere hverandre.

La $p^* = \max_i\{LM(p_i)LM(g_i)\}$, altså det største monomet på høyre side i (4.1). Vi ser at dette befinner seg i den første summen i (4.2). Altså har vi $p^* \succ LM(f)$, og det må finnes et annet ledd i den samme summen fra (4.2) som kansellerer med p^*. Med litt forenklende notasjon får vi at det finnes g og g' i G, og p og p' i R, slik at pg og $p'g'$ er med i (4.1) og

$$p^* = LM(p)LM(g) = LM(p')LM(g')$$

Merk at p^* er avhengig av hvilken representasjon for f vi valgte i (4.1) (en slik representasjon er ikke unik). Siden $\prec$ er en velordning, kan vi velge en representasjon av f som gir *minst mulig* p^*, og deretter en representasjon der denne p^* opptrer *færrest mulig* antall ganger.

Vi innfører følgende notasjon:

$$\begin{aligned} L &= LCM(LM(g), LM(g')) \\ u &= p - LT(p) \\ u' &= p' - LT(p') \end{aligned}$$

Siden $LM(g)$ og $LM(g')$ deler p^*, må L dele p^*. Altså finnes et monom τ slik at $p^* = \tau L$, det vil si at

$$LM(p) = \frac{\tau \cdot L}{LM(g)} \text{ og } LM(p') = \frac{\tau \cdot L}{LM(g')}$$

Vi bruker dette til å skrive om leddet pg fra (4.1):

$$\begin{aligned} pg &= LT(p)g + ug \\ &= LT(p)g - \frac{LC(p)}{LC(p')}p'g' + \frac{LC(p)}{LC(p')}p'g' + ug \\ &= LT(p)g - \frac{LC(p)}{LC(p')}LT(p')g' + \frac{LC(p)}{LC(p')}p'g' + ug - \frac{LC(p)}{LC(p')}u'g' \\ &= LC(p)\frac{\tau \cdot L}{LM(g)} \cdot g - LC(p)\frac{\tau \cdot L}{LM(g')} \cdot g' + \frac{LC(p)}{LC(p')}p'g' + ug - \frac{LC(p)}{LC(p')}u'g' \\ &= LC(p)\tau \cdot (\frac{L}{LM(g)} \cdot g - \frac{L}{LM(g')} \cdot g') + \frac{LC(p)}{LC(p')}p'g' + ug - \frac{LC(p)}{LC(p')}u'g' \\ &= LC(p)\tau \cdot S(g,g') + \frac{LC(p)}{LC(p')}p'g' + \text{mindre ledd} \end{aligned} \tag{4.3}$$

Vi har, fra antakelsen vår om at $S(g,g')$ reduserer svakt til 0 modulo G, det vil si at vi har c_i i R for $1 \leq i \leq t$ slik at

$$S(g,g') = \sum_{i=1}^{t} c_i g_i \text{ og } LT(c_i g_i) \prec L \text{ for alle } 1 \leq i \leq t \tag{4.4}$$

Det vil si at første ledd i siste linje i (4.3), nemlig $LC(p)\tau \cdot S(g,g')$, ikke har noen ledd som er større enn $LT(p'g')$. Altså har vi

$$pg = \frac{LC(p)}{LC(p')}p'g' + \text{ mindre ledd}$$

der alle de «mindre leddene» er i I, og vi kan lage en ny representasjon av f ved å skrive om de to leddene $pg + p'g'$ i summen (4.1) på følgende måte:

$$pg + p'g' = (\frac{LC(p)}{LC(p')}p'g' + \text{ mindre ledd }) + p'g' = (1 + \frac{LC(p)}{LC(p')})p'g' + \text{ mindre ledd}$$

Denne representasjonen inneholder en p^* mindre enn den opprinnelige, en motsigelse til valg av representasjon av f. Det kan dermed ikke eksistere noen slik f der ingen $LT(g_i)$ deler $LT(f)$, og G er en Gröbnerbasis for I. ♣

Merk.

1. Det kan synes uklart hvorfor det er avgjørende for beviset at $LT(c_i g_i) \prec L$. Dersom dette ikke var tilfelle, kunne $LC(p) \cdot \tau \cdot S(g,g')$ inneholdt større ledd enn $LT(p'g')$, og da kunne $LM(p'g') = p^*$ kansellert mot et annet ledd, og vi kunne ikke skrevet om summen (4.1) slik vi gjorde.
2. Vi ga ikke beviset for den opprinnelige versjonen av dette teoremet (teorem 2.5.5), men vi får også bevist dette ved å bytte ut linje (4.4) med

$$S(g,g') = \sum_{i=1}^{t} c_i g_i \text{ og } LT(c_i g_i) \preceq LT(S(g,g')) \prec L \text{ for alle } 1 \leq i \leq t$$

Vi kan nå gi Buchbergers andre kriterium.

Lemma 4.2.4 (Buchbergers andre kriterium)**.** *La f_i, f_j og f_k være elementer i $F = \{f_1, \ldots, f_s\} \subseteq R$. Anta at $LM(f_k)$ deler $L(i,j)$, og at både $S(i,k)$ og $S(j,k)$ reduserer svakt til 0 modulo F. Da reduserer også $S(i,j)$ svakt til 0 modulo F.*

Bevis. $LM(f_k)$ deler både $L(i,k)$ og $L(j,k)$, så vi har $L(i,k)|L(i,j)$ og $L(j,k)|L(i,j)$. La oss anta, uten tap av generalitet, at f_i, f_j og f_k er moniske. Vi får da

$$\begin{aligned}
S(i,j) =& \frac{L(i,j)}{LM(f_i)} f_i - \frac{L(i,j)}{LM(f_j)} f_j \\
=& \frac{L(i,j)}{LM(f_i)} \cdot \frac{L(i,k)}{L(i,k)} f_i - \frac{L(i,j)}{LM(f_j)} \cdot \frac{L(j,k)}{L(j,k)} f_j \\
=& (\frac{L(i,k)}{LM(f_i)} f_i - \frac{L(i,k)}{LM(f_k)} f_k) \cdot \frac{L(i,j)}{L(i,k)} + \frac{L(i,j)}{L(i,k)} \cdot \frac{L(i,k)}{LM(f_k)} f_k \\
& - (\frac{L(j,k)}{LM(f_j)} f_j - \frac{L(j,k)}{LM(f_k)} f_k) \cdot \frac{L(i,j)}{L(j,k)} - \frac{L(i,j)}{L(j,k)} \cdot \frac{L(j,k)}{LM(f_k)} f_k \\
=& S(i,k) \cdot \frac{L(i,j)}{L(i,k)} - S(j,k) \cdot \frac{L(i,j)}{L(j,k)}
\end{aligned}$$

Siden $S(i,k)$ og $S(j,k)$ reduserer svakt til 0 modulo F, har vi elementer a_l og b_l i R, for $1 \leq l \leq s$, slik at

$$S(i,k) = \sum_{l=1}^{s} a_l f_l \text{ der } LT(a_l f_l) \prec L(i,k)$$
$$S(j,k) = \sum_{l=1}^{s} b_l f_l \text{ der } LT(b_l f_l) \prec L(j,k)$$

Da er $S(i,j)$ en sum på formen

$$S(i,j) = \frac{L(i,j)}{L(i,k)} \cdot \sum_{l=1}^{s} a_l f_l - \frac{L(i,j)}{L(j,k)} \cdot \sum_{l=1}^{s} b_l f_l$$

der vi har

$$\frac{L(i,j)}{L(i,k)} \cdot LM(a_l f_l) \prec \frac{L(i,j)}{L(i,k)} L(i,k) = L(i,j)$$

og tilsvarende for $\frac{L(i,j)}{L(j,k)} \cdot LM(b_l f_l)$. Dermed reduserer $S(i,j)$ svakt til 0 modulo F. ♣

Mora skriver at svake reduksjoner av S-polynomer til 0 «ikke eksisterer, de er bare fiksjon; akkurat som enhjørninger, jeg kan ikke gi et eneste eksempel på en svak Gröbnerrepresentasjon som ikke er en Gröbnerrepresentasjon» [22, s. 96 – min oversettelse] – det vil si at dersom et S-polynom reduserer svakt til 0 modulo F, reduserer det også til 0 modulo F.

Ved å bruke lemmaet sparer vi oss for å redusere mange overflødige S-polynomer. Dersom vi allerede har vist at $S(i,k)$ og $S(j,k)$ reduserer svakt til 0, og vi har $S(i,j)$ som oppfyller kravene i lemmaet, vet vi at det reduserer svakt til null uten å utføre reduksjonen. Dette forklarer også hvorfor de svake reduksjonene «ikke finnes»: Når vi har funnet en Gröbnerbasis G ved hjelp av Buchbergers algoritme og lemmaet over, har vi, for alle S-polynomer $S(i,j)$, enten

- redusert $S(i,j)$ med divisjonsalgoritmen og enten fått 0 eller lagt til resultatet i Gröbnerbasisen, slik at vi nå har $S(i,j) \xrightarrow{G} 0$
- brukt Buchbergers første kriterium, da vet vi også at $S(i,j) \xrightarrow{G} 0$
- brukt Buchbergers andre kriterium, slik at vi vet at $S(i,j)$ reduserer svakt til 0. Men fra teorem 2.5.5 vet vi at $S(i,j)$ reduserer til 0 modulo G.

Siden vi benytter oss av informasjon om tidligere S-polynomer, må den nye algoritmen vår «huske» hva den har beregnet tidligere. For å løse dette problemet, lar vi køen Q fra algoritme 4.1 inneholde (i,j), der f_i og f_j er i F, istedenfor $S(i,j)$. Dersom vi skal beregne $S(i,j)$ og det finnes k slik at Buchbergers andre kriterium er oppfylt, må vi vite om $S(i,k)$ og $S(j,k)$ allerede er beregnet – dersom de *ikke* er det, kan vi ikke bruke kriteriet! Men dersom (i,k) og (j,k) ikke er i Q, vet vi at S-polynomene deres reduserer svakt til 0. Dette benytter vi oss av når vi lager en ny, forbedret utgave av Buchbergers algoritme (algoritme 4.4). For å gjøre koden mer oversiktlig, har vi gitt de to Buchberger-kriteriene hver sin underalgoritme, henholdsvis algoritme 4.2 og 4.3.

Korollar 4.2.5. *Den modifiserte versjonen av Buchbergers algoritme, gitt i algoritme 4.4, stopper etter et endelig antall steg, og returnerer en Gröbnerbasis for* $I = \langle f_1, \ldots, f_t \rangle$

Bevis. Beviset følger direkte fra beviset for den opprinnelige Buchbergers algoritme, Buchbergers to kriterier og kommentaren over. ♠

Algoritme 4.2 Krit-1

input: f_1 og f_2 i R
output: TRUE eller FALSE
if $L(i,j) = LM(f_i)LM(f_j)$ **then**
  return TRUE
else
  return FALSE
end if

Algoritme 4.3 Krit-2

input: $F = \{f_1, \ldots, f_s\}$, f_i og f_j i F og $Q \subseteq \{(i,j) | 1 \leq i < j \leq s\}$
output: TRUE eller FALSE
for f_k i $F \setminus \{f_i, f_j\}$ **do**
  if $LM(f_k) | L(i,j)$ og $(i,k) \notin Q$ og $(j,k) \notin Q$ **then**
    return TRUE
  end if
end for
return FALSE

Algoritme 4.4 Buchbergers algoritme, versjon 3

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
$\delta := s + 1$
$Q := \emptyset$
for $i := 1$ **to** $s - 1$ **do**
  for $j := i + 1$ **to** s **do**
    $Q := Q \cup \{(i, j)\}$
  end for
end for
while $Q \neq \emptyset$ **do**
  velg $(i, j) \in Q$
  $Q := Q \setminus \{(i, j)\}$
  if Krit-1$(f_i, f_j) =$ FALSE **and** Krit-2$(f_i, f_j, Q, F) =$ FALSE **then**
    $r := S(i, j)$ modulo F
    if $r \neq 0$ **then**
      $f_\delta := LC(r)^{-1} r$ # vil ha f monisk
      $F := F \cup \{f_\delta\}$
      for $k := 1$ **to** $\delta - 1$ **do**
        $Q := Q \cup \{S(i, \delta)\}$
      end for
      $\delta := \delta + 1$
    end if
  end if
end while
$G := F$
return G

4.3 Overflødige S-polynomer 2: Gebauer-Möller

I forrige avsnitt så vi at på bakgrunn av tidligere beregnede S-polynomer, kan vi avgjøre om nye er overflødige (det vil si at de reduserer til null) uten å utføre reduksjonen. Dette avsnittet tar for seg hvordan Buchbergers andre kriterium kan tilpasses slik at vi også tar hensyn til valg-funksjonen i algoritmen, altså der det står «velg $(i, j) \in Q$» i algoritme 4.4. Mer teori om dette kommer i neste avsnitt – enn så lenge bruker vi ukritisk følgende kriterium:

$$\text{velg } (i, j) \in Q \text{ slik at } L(i, j) \text{ er minimial med hensyn til } \prec$$

Dette kan implementeres slik at vi sorterer Q på nytt hver gang vi legger til nye elementer, men dette er tungvindt og kostbart. Gebauer og Möllers løsning var å holde antall elementer i Q så lavt som mulig. Tidligere plukket vi et element ut av Q, vurderte det og avgjorde eventuelt at det var overflødig. Nå vil vi gjøre denne vurderingen av paret (i, j) *før* vi eventuelt legger det til i Q.

Gebauer og Möllers resultater ble publisert i to artikler fra åttitallet [14] [15]. Her har vi imidlertid brukt Moras gjengivelse som kilde [22, kapittel 25.1]. Alle definisjoner og resultater er, med mindre annet er oppgitt, hentet derfra.

Vi starter med å definere en Gebauer-Möller-mengde. For enkelthets skyld innfører vi først følgende notasjon: La $F = \{f_1, f_2, \ldots, f_s\} \subseteq R$.

$$\mathcal{S}_s := \{(i,j) | 1 \leq i < j \leq s\}$$
$$L(i,j,k) := LCM(LM(f_i), LM(f_j), LM(f_k))$$

Merk at for et element (i,j) i $\mathcal{S}_s$ har vi alltid $i < j$, men i praksis hender det at vi ikke vet om i eller j er størst, eller at begge alternativer er mulige. Vi har passet på å kommentere de stedene der dette er tilfelle. For minste felles multiplum spiller ikke rekkefølgen på elementene noen rolle, mens for S-polynomer definerer vi at dersom $j < i$, er $S(i,j) := S(j,i)$.

Definisjon 4.3.1. La $G = \{g_1, \ldots, g_t\} \subseteq R = k[x_1, \ldots, x_n]$. Delmengden $\mathfrak{GM} \subseteq \mathcal{S}_t$ er en **Gebauer-Möller-mengde** for G dersom det for alle par (i,j) i $\mathcal{S}_t$ finnes:

- par $(i_1, j_1), \ldots, (i_\rho, j_\rho), \ldots, (i_r, j_r)$ i $\mathcal{S}_t$
- monomer $\tau_1, \ldots, \tau_r$ i R
- koeffisienter $c_1, \ldots, c_r$ i k

slik at det følgende er oppfylt:

(i) $S(i,j) = \sum_\rho c_\rho \tau_\rho S(i_\rho, j_\rho)$

(ii) $L(i,j) = \tau_\rho L(i_\rho, j_\rho)$ for alle $1 \leq \rho \leq r$

(iii) For alle $1 \leq \rho \leq r$ er *enten* (i_ρ, j_ρ) i $\mathfrak{GM}$, *eller* så er $L(i_\rho, j_\rho) = LM(g_{i_\rho}) LM(g_{j_\rho})$.

Vi kaller en representasjon som i punkt (i) av $S(i,j)$ som også oppfyller krav (ii) og (iii) for en **Gebauer-Möller-representasjon**, forkortet **GM-representasjon**, av $S(i,j)$.

Vi gir en kort forklaring på definisjonen over. Det den sier er at dersom $\mathfrak{GM}$ er en Gebauer-Möller-mengde for G, skal *alle* S-polynomer kunne uttrykkes som en kombinasjon av andre S-polynomer, monomer og koeffisienter (i). Alle S-polynomer $S(i_\rho, j_\rho)$ som brukes i denne representasjonen skal enten «være i» $\mathfrak{GM}$ (det vil si at (i_ρ, j_ρ) er i $\mathfrak{GM}$), eller de skal oppfylle Buchbergers andre kriterium (iii). I tillegg har vi kravet (ii), som skal vise seg å være nyttig:

Anta at $\mathfrak{GM}$ er en Gebauer-Möller-mengde for $G = \{g_1, \ldots, g_t\}$, og at vi har (i,j) i $\mathcal{S}_t$. La

$$S(i,j) = \sum_\rho c_\rho \tau_\rho S(i_\rho, j_\rho) \tag{4.5}$$

være en GM-representasjon av $S(i,j)$. Da har vi

$$\max_\rho \{\tau_\rho S(i_\rho, j_\rho)\} = \max_\rho \{\tau_\rho LM(S(i_\rho, j_\rho))\} \prec \max_\rho \{\tau_\rho L(i_\rho, j_\rho)\} = \max_\rho \{L(i,j)\} = L(i,j) \tag{4.6}$$

Dette er nødvendig for å vise det følgende resultatet.

Korollar 4.3.2. *La $G = \{g_1, \ldots, g_t\} \subseteq R$ der $LC(g_i) = 1$ for alle $1 \leq i \leq t$. Da er G en Gröbnerbasis for idealet $I = \langle G \rangle$ hvis og bare hvis det finnes en Gebauer-Möller-mengde $\mathfrak{GM}$ for G og $(i,j) \in \mathfrak{GM}$ medfører at $S(i,j)$ reduserer svakt til 0 modulo G.*

Bevis. $\Rightarrow$ Dersom G er en Gröbnerbasis, følger det fra teorem 4.2.3 og definisjonen over at $\mathcal{S}_t$ er en Gebauer-Möller-mengde for G, der alle S-polynomene reduserer svakt til 0 modulo G.

$\Leftarrow$ La (i,j) være i $\mathcal{S}_t$. Dersom (i,j) er i $\mathfrak{GM}$, reduserer den svakt til 0 (fra antakelsen). Vi antar derfor at $(i,j) \notin \mathfrak{GM}$. Men vi har en GM-representasjon (4.5) av $S(i,j)$ der alle $S(i_\rho, j_\rho)$ reduserer svakt til 0 (fra antakelsen eller fra Buchbergers første kriterium), og fra (4.6) reduserer også $S(i,j)$ svakt til 0. ♣

Målet vårt er nå å finne en Gebauer-Möller-mengde for G som er *mindre* enn $\mathcal{S}_t$. Men la oss først se på en ordning på $\mathcal{S}_t$, definert slik:

Definisjon 4.3.3. La $\prec$ være monomordningen som benyttes, og (i,j) og (i',j') to ulike par i $\mathcal{S}_t$. Vi sier at (i,j) **er mindre enn** (i',j'), eller $(i,j) < (i',j')$, dersom et av følgende er oppfylt:

(a) $L(i,j) \prec L(i',j')$

(b) $L(i,j) = L(i',j')$ og $j < j'$

(c) $L(i,j) = L(i',j')$ og $j = j'$ og $i < i'$

Vi ser at denne ordningen er tilpasset ønsket om å velge S-polynomer $S(i,j)$ der $L(i,j)$ er minst mulig. Det neste resultatet kan synes teknisk, men det blir avgjørende så snart det kombineres med definisjonen over.

Lemma 4.3.4. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, og la $1 \leq i,j,k \leq t$. Da er*

$$\frac{L(i,j,k)}{L(i,k)} S(i,k) - \frac{L(i,j,k)}{L(i,j)} S(i,j) + \frac{L(i,j,k)}{L(k,j)} S(k,j) = 0 \tag{4.7}$$

Bevis. Beviset følger direkte fra definisjonen av S-polynomer (definisjon 2.5.3). ♠

Vi kan skrive om på (4.7) til

$$\frac{L(i,j,k)}{L(i,j)} S(i,j) = \frac{L(i,j,k)}{L(i,k)} S(i,k) + \frac{L(i,j,k)}{L(j,k)} S(j,k)$$

Vi ønsker at $S(i,j)$ skal være overflødig, det vil si at $S(i,j)$ reduserer til 0 dersom $S(i,k)$ og $S(j,k)$ gjør det. Dersom disse tre S-polynomene er en del av en Gröbnerbasis-beregning, må vi altså ha plukket ut og tatt hånd om $S(i,k)$ og $S(j,k)$ før vi kommer til $S(i,j)$, slik at vi vet at $S(i,j)$ er overflødig. Det er uansett klart at vi må ha

$$\frac{L(i,j,k)}{L(i,j)} = 1, \text{ det vil si } L(i,j,k) = L(i,j) \tag{4.8}$$

Dessuten må vi ha plukket ut (i,k) og (k,j) fra Q før vi kommer til (i,j), det vil si at vi må ha

$$(i,j) > (i,k) \text{ og } (i,j) > (k,j) \tag{4.9}$$

Notasjonen skurrer litt foreløpig, siden vi ikke vet noe om størrelsesforholdet mellom i, j og k, men vi kan anta at $i < j$. (4.8) gir oss at både $LM(f_k)$, $L(i,k)$ og $L(k,j)$ deler $L(i,j)$. Fra (4.9)

får vi:

$$\begin{aligned}(i,k) < (i,j) &\iff \begin{cases} L(i,k) \prec L(i,j) \text{ eller} \\ L(i,k) = L(i,j) \text{ og } k < j \end{cases} \\ (j,k) < (i,j) &\iff \begin{cases} L(j,k) \prec L(i,j) \text{ eller} \\ L(j,k) = L(i,j) \text{ og } k < i \end{cases}\end{aligned} \tag{4.10}$$

Merk at det siste alternativet fra definisjon 4.3.3 forsvinner, siden vi har $i < j$. Vi har tre alternativer for k:

(a) $k > j$. Da er eneste mulige alternativ fra (4.10) at $L(i,k) \neq L(i,j)$ og $L(k,j) \neq L(i,j)$.

(b) $k < i$ og $L(j,k) = L(i,j)$ (vi kan ikke her si noe om hvorvidt $L(i,k)$ og $L(i,j)$ er like).

(c) $k < j$ og $L(j,k) \neq L(i,j)$ (se kommentar på (b)).

Vi ser at (b) og (c) kan deles inn i undertilfeller, men vi holder oss til disse tre. Det er klart at så lenge et av de tre tilfellene samt (4.8) er oppfylt, har vi

$$S(i,j) = \frac{L(i,j,k)}{L(i,k)} S(i,k) + \frac{L(i,j,k)}{L(j,k)} S(j,k) \tag{4.11}$$

og $L(i,j)$ er maksimalt. Hvis vi ikke går ut fra at (4.8) holder, hvor my ekstra informasjon må vi føye til tilfelle (a)–(c) for at (4.11) fortsatt skal holde? Svaret blir gitt i det følgende resultatet, som er en sterkere utgave av Buchbergers andre kriterium:

Korollar 4.3.5 (Buchbergers andre kriterium (sterk))**.** *La $G = \{g_1, \ldots, g_t\} \subseteq R$, og la $1 \leq i < j \leq t$. Dersom det finnes en k slik at $1 \leq k \leq t$ og en av følgende er oppfylt:*

(a) $i < j < k$, $L(i,j,k) = L(i,j)$ og $L(i,k) \neq L(i,j) \neq L(j,k)$

(b) $k < j$ og $L(k,j) | L(i,j) \neq L(k,j)$

(c) $k < i < j$ og $L(k,j) = L(i,j)$

Da er

$$S(i,j) = \frac{L(i,j,k)}{L(i,k)} S(i,k) + \frac{L(i,j,k)}{L(j,k)} S(j,k)$$

og dersom $S(i,k)$ og $S(j,k)$ reduserer svakt til 0, gjør også $S(i,j)$ det.

Bevis. Vi trenger bare å vise at i alle tilfellene er $L(i,j,k) = L(i,j)$. Dette følger direkte fra hver av betingelsene (a)-(c) (i det første er det også gitt eksplisitt). ♠

Legg merke til at vi ikke bruker verdien av k i beviset, og at korollaret ikke eksplisitt sier at $S(i,j)$ skal være maksimalt. Fra de foregående utgreingene vet vi imidlertid at dette er tilfelle. Kravene til k videreføres i neste definisjon.

Definisjon 4.3.6. S-polynomet $S(i,j)$, der (i,j) er i $\mathcal{S}_t$, er **redundant** dersom det finnes en k der $1 \leq k \leq t$ og et av følgende holder:

(a) $k > j$ og $L(i,j,k) = L(i,j)$ og $L(i,k) \neq L(i,j) \neq L(j,k)$

(b) $k < j$ og $L(j,k)|L(i,j) \neq L(j,k)$

Vi kjenner igjen (a) og (b) fra det forrige korollaret. Nå er det på tide å vende tilbake til Gebauer-Möller-mengdene. Det følgende resultatet er ikke hentet fra Mora, men spiller omtrent samme rolle som Moras lemma 25.1.8 [22, s. 261].

Lemma 4.3.7. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, og la $\mathfrak{GM}$ være en Gebauer-Möller-mengde for G. Anta at (i,j) er i $\mathfrak{GM}$ og at (i,j) er redundant. Da er*

$$\mathfrak{GM}' = \mathfrak{GM} \setminus \{(i,j)\}$$

også en Gebauer-Möller-mengde for G.

Bevis. Å ta ut (i,j) fra $\mathfrak{GM}$ påvirker to ting: Alle GM-representasjoner av S-polynomer $S(i',j')$ der $S(i,j)$ inngår, og selve GM-representasjonen av $S(i,j)$, som tidligere var en gyldig GM-representasjon i seg selv. Hvis vi kan finne en GM-representasjon av $S(i,j)$, vil begge disse problemene løses.

Hvis $L(i,j) = LM(g_i)LM(g_j)$ er vi ferdige, så vi antar dette ikke er tilfelle. Siden (i,j) er redundant, har vi (i,k) og (j,k) slik at (fra korollar 4.3.5)

$$S(i,j) = \frac{L(i,j,k)}{L(i,k)}S(i,k) + \frac{L(i,j,k)}{L(j,k)}S(j,k)$$

Vi må vise at krav (2) og (3) holder for denne representasjonen, der $c_1 = c_2 = 1$, $\tau_1 = \frac{L(i,j,k)}{L(i,k)}$, $\tau_2 = \frac{L(i,j,k)}{L(j,k)}$ og $(i_1,j_1) = (i,k)$ og $(i_2,j_2) = (j,k)$ (merk at rekkefølgen innad i parene er avhengig av hvilket tilfelle fra definisjonen av redundante elementer vi er i). Vi starter med krav (2):

$$\tau_1 L(i,k) = \frac{L(i,j,k)}{L(i,k)}L(i,k) = L(i,j,k) = L(i,j)$$

og vi får samme resultat for $\tau_2 L(j,k)$, så dette kravet er oppfylt. Vi har (i,k) og (j,k) ulik (i,j), og videre er $(i,k) < (i,j)$ og $(j,k) < (i,j)$, så $S(i,k)$ og $S(j,k)$ har en GM-representasjon fra $\mathfrak{GM}$, der $S(i,j)$ ikke er involvert. Altså er $\mathfrak{GM}'$ en Gebauer-Möller-mengde for G. ♣

Altså kan vi, ved å luke ut redundante (i,j)-par, krympe en Gebauer-Möller-mengde for G, og fra før vet vi at parene i denne mengden er tilstrekkelige til å bygge en Gröbnerbasis for $\langle G \rangle$. Det eneste vi mangler før vi kan skrive selve algoritmen, er svaret på spørsmålet: Hvordan kan vi utvide Gebauer-Möller-mengden når vi utvider G, slik at vi slipper å bygge den forfra hver gang?

Lemma 4.3.8. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, og la $\mathfrak{GM}_* \subseteq \mathcal{S}_{t-1}$ være en Gebauer-Möller-mengde for $G \setminus \{g_t\}$. Definer*

$$\mathsf{T} = \{L(j,t) | 1 \leq j < t\}$$

og la $\mathsf{T}' \subseteq \mathsf{T}$ være alle τ i T slik at enten

(1) det finnes en $\tau' \neq \tau$ i T slik at τ' deler τ, eller

(2) det finnes en i_τ der $1 \neq i_\tau < t$ slik at $LM(g_{i_\tau})LM(g_t) = L(i_\tau, t) = \tau$

For alle τ i $\mathsf{T} \setminus \mathsf{T}'$ velger vi en i_τ, der $1 \leq i_\tau < t$, slik at $L(i_\tau, t) = \tau$. Da er mengden

$$\mathfrak{GM} = \mathfrak{GM}_* \cup \{(i_\tau, t) | \tau \in \mathsf{T} \setminus \mathsf{T}'\}$$

en Gebauer-Möller-mengde for G.

Før vi beviser dette lemmaet, la oss tydeliggjøre hva det sier. Mengden T er alle *LCM*'er av ledende ledd i de «nye» (i, j)-parene, altså de parene som legges til $\mathcal{S}_{t-1}$ for å danne $\mathcal{S}_t$. Mengden T' består av redundante elementer i henhold til del (b) av definisjonen (del (a) kan ikke brukes til å ta ut elementer fra T, siden t er den største indeksen vi har), og dessuten de elementene hvis tilhørende S-polynom reduserer direkte til 0 fra Buchbergers første kriterium.

Når vi har tatt ut alle elementene i T' fra T, velger vi *ett* (j, t)-par for hver τ i $\mathsf{T} \setminus \mathsf{T}'$ (vi vet at minst et slikt par finnes, for vi konstruerte T ut fra slike par). Dette betyr at det kan finnes (j, t)-par som ikke tas ut i T', men som heller ikke legges til $\mathfrak{GM}$. At dette er OK, får vi se i beviset.

Bevis. Vi ser på et vilkårlig par (i, j) i $\mathcal{S}_t$, og viser at det tilhørende S-polynomet $S(i, j)$ har en GM-representasjon som gitt av definisjon 4.3.1. I utgangspunktet har vi to muligheter for j, nemlig $j < t$ og $j = t$. I det første tilfellet er vi i mål, siden $\mathfrak{GM}_*$ er en Gebauer-Möller-mengde. Vi antar derfor at vi har et par (i, t).

Dersom $L(i, t) = LM(g_i)LM(g_t)$, oppfylles kravene i definisjon 4.3.1 ($S(i, j)$ er da i seg selv en gyldig GM-representasjon).

Dersom det finnes en $j < t$ slik at $L(j, t)$ deler $L(i, t)$ og $L(i, t) \neq L(j, t)$, er $S(i, t)$ redundant, og forrige lemma forteller oss at disse kan plukkes ut.

Dersom ingen av tilfellene over holder, må det finnes et par (i_t, t) slik at (i_t, t) er i $\mathfrak{GM}$ og $L(i_t, t) = L(i, t)$. Da har vi to muligheter: Dersom $i = i_t$, er (i, t) i $\mathfrak{GM}$, og vi er ferdige. Har vi derimot $i \neq i_t$, har vi $L(i, t) = L(i_t, t) = L(i, i_t, t)$, slik at vi kan skrive om på $S(i, t)$:

$$\begin{aligned}
S(i,t) &= \frac{L(i,t)}{LM(g_i)} g_i - \frac{L(i,t)}{LM(g_t)} g_t \\
&= \frac{L(i,t)}{LM(g_i)} g_i - \frac{L(i_t,t)}{LM(g_{i_t})} g_{i_t} + \frac{L(i_t,t)}{LM(g_{i_t})} g_{i_t} - \frac{L(i,t)}{LM(g_t)} g_t \\
&= \frac{L(i,i_t,t)}{L(i,i_t)} \frac{L(i,i_t)}{LM(g_i)} g_i - \frac{L(i,i_t,i)}{L(i,i_t)} \frac{L(i,i_t)}{LM(g_{i_t})} g_{i_t} + S(i_t,t) \\
&= \frac{L(i,i_t,t)}{L(i,i_t)} S(i,i_t) + S(i_t,t)
\end{aligned}$$

(i_t, t) er i $\mathfrak{GM}$, og vi har $\frac{L(i,i_t,t)}{L(i_t,t)} L(i_t, i) = L(i, i_t, t) = L(i, t)$ og $L(i_t, t) = L(i, t)$, så denne representasjonen av $S(i, t)$ er gyldig i henhold til definisjon 4.3.1. ♣

Når vi lager en algoritme for å bygge disse Gebauer-Möller-mengdene, vil vi også bruke del (a) fra definisjon 4.3.6 til å *ta ut* elementer fra $\mathfrak{GM}_*$ som blir redundante i det vi legger til nye elementer. Dette påvirker ikke korrektheten av framgangsmåten gitt i lemma 4.3.8, så lenge vi passer på å ikke gå i ring (det vil si at vi ikke bruker elementet (i, s) i $\mathfrak{GM}_*$ til å forkaste (j, t) som redundant, for deretter å bruke (j, t) til å forkaste (i, s) fra $\mathfrak{GM}_*$).

Algoritmen kaller vi Oppdater(Q, G, g), fordi den oppdaterer Q når et nytt element g skal legges til. Merk at denne algoritmen kalles i to forskjellige situasjoner: Først når vi bygger Q fra inputen F, og deretter hver gang et nytt element g, som stammer fra en reduksjon av et S-polynom, skal legges til. Algoritme 4.6 gir den nye versjonen av Buchbergers algoritme. Under har vi gitt en detaljert linje for linje-forklaring av Oppdater.

Forklaring av algoritme 4.5: Oppdater

Tallene henviser til linjenummerene i algoritmen.

4–9: Bruker del (a) av definisjon 4.3.6 til å ta ut par i Q som blir redundante når vi legger til g_δ i G.

10–14: Lager alle par som inneholder δ og samler dem i mengden $\mathcal{S}^+$. På sikt vil vi lage $\mathcal{S}_\delta = Q$ fra $\mathcal{S}_{\delta-1}$ og $\mathcal{S}^+$.

15–20: Bruker del (b) av definisjon 4.3.6 til å ta ut par i $\mathcal{S}^+$ som er redundante på grunn av par i $\mathcal{S}_{\delta-1}$.

21–27: T er mengden som beskrevet i lemma 4.3.8, bortsett fra at vi allerede har tatt ut redundante elementer i henhold til krav (1) i lemmaet.

28–35: For alle τ i T lager vi mengden $\mathcal{S}(\tau)$, som består av alle par (i, δ) i $\mathcal{S}^+$ slik at $L(i, \delta) = \tau$. Dersom det, for en gitt τ, finnes et element (i_τ, δ) i $\mathcal{S}^+$ der $L(i_\tau, \delta) = LM(g_{i_\tau})LM(g_\delta)$, vil selvfølgelig $S(i_\tau, \delta)$ redusere til 0 modulo G, men det er heller ingen vits i å legge til noen *andre* par fra $\mathcal{S}(\tau)$ til Q: I lemma 4.3.8 tar vi ut alle slike τ fra T (krav (2)), og som vi så i beviset, får vi en Gebauer-Möller-mengde selv om det finnes elementer som verken tas ut av T eller legges til i Q.

Algoritme 4.5 Oppdater

input: $G = \{g_1, \ldots, g_{\delta-1}\}$, $Q_{gammel} \subseteq \mathcal{S}_{\delta-1}$, $g_\delta \notin G$
output: $Q \subseteq \mathcal{S}_\delta$
$Q := Q_{gammel}$
# tester om g_δ gjør tidligere par i Q redundante
for alle $(i,j) \in Q$ **do**
  if $L(i,j,\delta) = L(i,j)$, $L(i,\delta) \neq L(i,j) \neq L(j,\delta)$ **then**
    $Q := Q \setminus \{(i,j)\}$
  end if
end for
# lager alle mulige nye par
$\mathcal{S}^+ := \emptyset$
for $i := 1$ **to** $\delta - 1$ **do**
  $\mathcal{S}^+ := \mathcal{S}^+ \cup \{(i,\delta)\}$
end for
# ta ut redundante elementer fra $\mathcal{S}^+$
for $i := 1$ **to** $\delta - 1$ **do**
  if $\exists j \neq i$ s.a. $1 \leq j < \delta - 1$ og $L(j,\delta) | L(i,\delta) \neq L(j,\delta)$ **then**
    $\mathcal{S}^+ := \mathcal{S}^+ \setminus \{(i,\delta)\}$
  end if
end for
# bygger mengden T
$\mathsf{T} := \emptyset$
for $(i,\delta) \in \mathcal{S}^+$ **do**
  if $L(i,\delta) \notin \mathsf{T}$ **then**
    $\mathsf{T} := \mathsf{T} \cup \{L(i,\delta)\}$
  end if
end for
# vurderer hvilket (i,δ)-par der $(i,\delta) = \tau$ som skal være med
for alle $\tau \in \mathsf{T}$ **do**
  $\mathcal{S}(\tau) := \{(i,\delta) | (i,\delta) \in \mathcal{S}^+ \text{ og } L(i,\delta) = \tau\}$
  if $L(i,\delta) \neq LM(g_i)LM(g_\delta)$ for alle $(i,\delta) \in \mathcal{S}(\tau)$ **then**
    velg $(i,\delta) \in \mathcal{S}(\tau)$
    $Q := Q \cup \{(i,\delta)\}$
  end if
end for
return Q

Algoritme 4.6 Buchbergers algoritme, versjon 4

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
{initialiserer G og Q}
$G := \{f_1, f_2\}$
if $L(1,2) \neq LM(f_1)LM(f_2)$ **then**
  $Q := \{(1,2)\}$
else
  $Q := \emptyset$
end if
# bygger Q fra F
for $i := 3$ **to** s **do**
  $Q :=$ Oppdater(G, Q, f_i)
  $G := G \cup \{f_i\}$
end for
# vurderer et og et element i Q
$\delta := s + 1$
while $Q \neq \emptyset$ **do**
  velg $(i, j) \in Q$
  $Q := Q \setminus \{(i, j)\}$
  $r := S(i, j)$ modulo G
  if $r \neq 0$ **then**
    $f_\delta := LC(r)^{-1}r$ # vil ha f_δ monisk
    $Q :=$ Oppdater(G, Q, f_δ)
    $G := G \cup \{f_\delta\}$
    $\delta := \delta + 1$
  end if
end while
return G

4.4 Eksempel, og en kort analyse av Oppdater

For å sammenlikne den opprinnelige Buchberger-algoritmen og den siste versjonen, henter vi fram eksempel 2.5.7 fra kapittel 2. Vi vil finne en Gröbnerbasis for idealet generert av $F = \{f_1, f_2\}$ der $f_1 = x^3y + 1$ og $f_2 = xy^2 + y$ i $R = \mathbb{F}_5[x, y]$, gradert leksikografisk ordning med $x > y$. Da vi gjorde eksempelet forrige gang, brukte vi i praksis algoritme 4.1. Denne gangen bruker vi algoritme 4.6.

Vi starter med å sette $G = \{f_1, f_2\}$, og siden $L(1,2) = x^3y^2 \neq LM(f_1)LM(f_2) = x^4y^3$, initialiserer vi Q til å være $Q = \{(1,2)\}$. Deretter kjører vi `while`-løkken helt til Q er tom. Det viser seg at vi får 6 iterasjoner.

(1) Vi velger $(1,2)$ fra Q. Da får vi $r = -x^2y + y \neq 0$, det vil si at vi setter $f_3 = x^2y - y$, og kjører Oppdater(G, Q, f_3).

Merk at siden Q nå er tom, går linje 5–9 i algoritme 4.5 ut. Vi går direkte til å beregne $\mathcal{S}^+$:

$$\mathcal{S}^+ = \{(1,3),(2,3)\}$$

Vi har $L(1,3) = x^3y$ og $L(2,3) = x^3y^3$. Siden $L(1,3)|L(2,3) \neq L(1,3)$, tar vi ut $L(2,3)$ fra $\mathcal{S}^+$. Vi bygger nå mengden T:

$$\mathsf{T} = \{x^3y\}$$

Siden bare ett par gir det eneste elementet i T, og $L(1,3) \neq LM(f_1)LM(f_3)$, får vi

$$Q := Q \cup \{(1,3)\} = \{(1,3)\}$$

(2) Vi har igjen bare ett element i Q, så valget er enkelt: Vi velger $(1,3)$. Da får vi $r = xy + 1 \neq 0$, som gir $f_4 = xy + 1$. Vi oppdaterer Q, og starter igjen med å finne $\mathcal{S}^+$.

$$\mathcal{S}^+ = \{(1,4),(2,4),(3,4)\}$$

Vi har nå $L(1,4) = x^3y$, $L(2,4) = xy^2$ og $L(3,4) = x^2y$. Siden $L(3,4)|L(1,4) \neq L(3,4)$ tar vi ut $L(1,4)$ fra $\mathcal{S}^+$. Dette gir

$$\mathsf{T} = \{xy^2, x^2y\}$$

og siden ingen av de to parene oppfyller Buchbergers første kriterium, legger vi både $(2,4)$ og $(3,4)$ til i Q.

(3) Vi velger nå $(2,4)$ fra Q, siden denne gir minimal LCM. Det viser seg at S-polynomet $S(2,4)$ blir 0.

(4) Siden det igjen er bare ett element i Q, velger vi det: $(3,4)$, som gir $f_5 = x + y$. Da har vi

$$\mathcal{S}^+ = \{(1,5),(2,5),(3,5),(4,5)\}$$

Det viser seg at $L(4,5)$ deler samtlige av de andre LCM'ene, så vi fjerner alle unntatt denne fra $\mathcal{S}^+$. Buchbergers første kriterium oppfylles ikke, og vi har da $Q = \{(4,5)\}$.

(5) Vi velger $(4,5)$ og finner $f_6 = y^2 - 1$, som gir følgende:

$$\begin{aligned}
\mathcal{S}^+ &= \{(1,6),(2,6),(3,6),(4,6),(5,6)\} \\
L(1,6) &= x^3y^2 \\
L(2,6) &= xy^2 \\
L(3,6) &= x^2y^2 \\
L(4,6) &= xy^2 \\
L(5,6) &= xy^2
\end{aligned}$$

Vi kan forkaste $L(1,6)$ og $L(3,6)$ fra $\mathcal{S}^+$, men siden de andre tre parene har lik LCM, må vi foreløpig beholde alle sammen. Vi får $\mathsf{T} = \{xy^2\}$, men denne gangen har vi tre par å velge mellom (ingen av dem oppfyller Buchbergers første kriterium) når vi skal flette T og Q (linje 28–35). Vi har ikke noen retningslinjer for hvilket par vi bør velge, så vi tar det minste i henhold til definisjon 4.3.3, nemlig $(2,6)$. Da har vi $Q = \{(2,6)\}$.

(6) Vi ser på $(2,6)$, og får

$$S(2,6) = y + x \xrightarrow{G} 0$$

Vi legger altså ikke til noen nye elementer i G, eller nye par i Q, i denne iterasjonen. Siden Q nå er tom, er vi i mål, med følgende Gröbnerbasis for $\langle F \rangle$:

$$G = \{x^3y + 1, xy^2 + y, x^2y - y, xy + 1, x + y, y^2 - 1\}$$

Når vi sammenlikner med Gröbnerbasisen vi fikk da vi opprinnelig gjorde eksempelet, ser vi at de er identiske med unntak av $x^2 - 1$, som var med forrige gang, men dette elementet ble senere forkastet da vi lagde en minimal Gröbnerbasis. Større forskjeller er det imidlertid når vi sammenlikner hvor effektive de to algoritmene er: Denne gangen beregnet vi totalt seks S-polynomer, mot tidligere 21. Vi utførte bare fem reduksjoner, også langt færre enn forrige gang.

Til gjengjeld har vi gjort ganske mange andre operasjoner, som vi ikke gjorde tidligere. Før gjorde vi mange utregninger av S-polynomer og mange reduksjoner modulo G. Nå har vi laget mengden $\mathcal{S}^+$, beregnet mange LCM'er og sammenliknet disse. Vi har flyttet fokuset fra polynomer til monomer, og det gjør beregningene mye enklere. I en implementasjon vil monomer typisk implementeres som vektorer (grad-vektoren til monomet), og denne har lengde n der n er antall variabler i polynomringen. Når vi beregner $L(i, j)$, leser vi to vektorer samtidig og plukker ut det største elementet på hver posisjon – en operasjon som er $O(n)$. Tilsvarende er kompleksiteten $O(n)$ når vi tester om et monom deler et annet (dersom x^α deler x^β, må alle elementer i α være mindre enn eller lik tilsvarende element i β).

Mengden $\mathcal{S}^+$ vokser også lineært (den har δ elementer i det vi legger til det $\delta + 1$'te elementet i Gröbnerbasisen), skjønt det er klart at dersom Gröbnerbasisen vokser voldsomt, vil $\mathcal{S}^+$ også gjøre det. Å lage T kan også implementeres effektivt, og det bør være mulig å implementere slik at T og $\mathcal{S}(\tau)$ konstrueres samtidig for å spare tid. Når det gjelder den første `for`-løkken i Oppdater, som vi ikke brukte i eksempelet over, kjører denne like mange ganger som det er elementer i Q, og dersom G i inputen har δ elementer, er det maksimalt $\binom{\delta}{2}$ elementer i Q.

Uten å ha laget en slik implementasjon, eller å ha analysert dette veldig grundig, mener vi at det bør være mulig å lage en $O(\delta^2)$-implementasjon (eller bedre) av Oppdater, der δ er kardinaliteten til den foreløpige Gröbnerbasisen G. Gjennomsnittlig kjøretid vil antakeligvis være mindre.

Å beregne S-polynomer er ikke det vanskeligste, men det er klart at det er mer krevende enn operasjonene fra Oppdater, blant annet fordi koeffisienter er involvert. Divisjonsalgoritmen også mer krevende, og involverer mange polynomoperasjoner. I eksempelet over vurderte vi bare 35 % av de mulige S-polynomene, og det er klart at i de aller fleste tilfeller, vil vi spare mye tid på Gebauer-Möller-versjonen av Buchbergers algoritme. Mora skriver at kardinaliteten av Q typisk vil være 10–20 % av tidligere [22, s. 263].

4.5 Valg-strategier og kompleksitet

Selv om vi har fått bort mange overflødige beregninger, er den foreløpige algoritmen langt fra perfekt. Det er fortsatt mye vi ikke har tatt stilling til: Hvilke valgmuligheter har vi for «velg $(i, j) \in Q$»? Bør vi redusere S-polynomene fullstendig, eller stoppe når det ledende leddet ikke lenger lar seg redusere? Når det gjelder *kompleksiteten* til Buchbergers algoritme er det

vanskelig å gi generelle svar, men vi har inkludert noen setninger om dette avslutningsvis i dette delkapitlet.

Ingen av disse spørsmålene er lette å besvare, og flere av dem krever eksperimentering i stor skala. Dette har ikke vært målsettingen for denne oppgaven, så her må vi nøye oss med en rask gjennomgang av andres resultater.

Buchberger innførte kriteriet vi benyttet oss av i forrige avsnitt, nemlig å velge (i, j) slik at $L(i, j)$ er minimal. Dette vil gi rest ulik 0 på et tidligere stadium, slik at senere S-polynomer raskere reduserer til 0 [7]. Denne valgstrategien heter *normal valgstrategi* (normal selection strategy). Det finnes en variant som heter Sugar, men den gir ikke nødvendigvis bedre resultater [10].

Traverso og Donati utførte på slutten av åttitallet en rekke eksperimenter med Gröbnerbasisalgoritmer, der de gjorde mange kjente eksempler mens de varierte ulike parametere. De sammenliknet både CPU-tider, antall (i, j)-par som ble vurdert og antall reduksjoner som ble utført [26]. De fleste forsøkene ble gjort i polynomringer over kropper med karakteristikk 0, men det er også noen få testoppsett der kroppen har karakteristikk $p > 0$.

Når man studerer tallene fra disse eksperimentene, kan det synes som om følgende alternative valgstrategi fungerer godt: Istedenfor å sortere etter $L(i, j)$, sorterer vi etter «estimert ledende monom» av $S(i, j)$, definert som følger: La g_i og g_j være moniske elementer i G, og S_i og S_j det *nest største* monomet i henholdsvis g_i og g_j. Vi definerer da

$$S_{ij} = \max\{\frac{L(i,j)S_i}{LM(g_i)}, \frac{L(i,j)S_j}{LM(g_j)}\}$$

Dersom $S_i \neq S_j$, er $LM(S(i, j)) = S_{ij}$. Altså velger man (i, j) der S_{ij} er minst mulig. Traverso vurderte også å sortere etter $L(i, j)/GCD(i, j)$, men dette ga dårlige resultater. For noen av eksperimentene har det beste resultatet kommet med den vanlige normalstrategien.

Traverso konkluderte dessuten med at det lønner seg å redusere S-polynomene fullstendig, siden vi ellers får unødvendig mange nye S-par. Å redusere de foregående elementene med de nye (det vil si, å lage en redusert Gröbnerbasis) gjør visstnok ikke noe utslag. Dette ser vi nærmere på i det ikke-kommutative tilfellet (se kapittel 6.4).

I neste avsnitt, der vi tar for oss F_4-algoritmen, ser vi på en god måte å utføre selve reduksjonen av S-polynomene. Men også der finnes det en valgmulighet, som heller ikke er eksplisitt beskrevet i F_4, nemlig den følgende: Når vi reduserer f med $F = \{f_1, \ldots, f_s\}$, kan det være flere elementer i F som oppfyller $LT(f) = m \cdot LT(f_i)$. Hvilken f_i skal vi velge? Traverso har testet en rekke muligheter, og hevder at dette valger har langt mindre innflytelse på effektiviteten enn valg av (i, j)-par [26, s. 197]. Likevel kan det synes at følgende strategi ofte gir gode resultater: Velg f_i slik at f_i har kortest mulig lengde, det vil si minimalt antall ledd. Mora holder også en knapp på å velge det «eldste» polynomet, altså det som først ble lagt til i F [22].

Man må også ha i bakhodet at ulike monomordninger påvirker størrelsen på Gröbnerbasisene. Leksikografisk ordning gir ofte store Gröbnerbasiser, mens en ordning som kan være mer effektiv å bruke er *gradert revers leksikografisk ordning* [7, s. 58]. Dette gjenspeiles også hos Traverso. Det finnes algoritmer som tar hensyn til dette (se referanser i [7, s. 112]).

Kompleksitet

Vi har allerede sett hvordan Gebauer-Möller kan redusere kjøretiden til Buchbergers algoritme gjennom å redusere Q, men det største problemet med Buchbergers algoritme er at det generelt er vanskelig å forutsi form og ikke minst størrelse på den ønskede Gröbnerbasisen. Det er ingen tvil om at algoritmen har en dårlig *worst case*-kjøretid. Som vi skal se i kapittel 7, er det mulig å lage idealer hvis Gröbnerbasis er like vanskelig å beregne som det er å løse et gitt $\mathcal{NP}$-komplett problem.

Det er funnet noen øvre skranker for kjøretiden, som er eksponensielle i graden på polynomene i Gröbnerbasisen. For et ideal generert av homogene polynomer (det vil si polynomer der alle monomene har samme totale grad) finnes det en ganske god øvre skranke (se [10, s. 12] og [22, kapittel 23]), men å overføre denne til vilkårlige polynomer gir ofte et overestimat. En annen type idealer som kan gi «snillere» beregninger enn andre, er idealer generert av polynomer med to ledd – et eksempel på at en slik utregning kan være kort og grei, er eksempelet i forrige avsnitt. Dersom polynomene har flere ledd, vil de typisk få enda flere ledd etter hvert som vi beregner og reduserer S-polynomer, og kompleksiteten øker. Men heller ikke for disse idealene er vi garantert å raskt finne en Gröbnerbasis [5].

4.6 F_4-algoritmen

F_4-algoritmen er utviklet av franskmannen Faugère, og ble introdusert i en artikkel fra 1999 [10]. Algoritmen har blitt beskrevet som en forbedring av Buchbergers algoritme, der polynomreduksjonsteget er erstattet av en mer effektiv framgangsmåte. F_4 baserer seg også på teorem 2.5.5, og tar også med oppdateringsalgoritmen fra Gebauer og Möller (algoritme 4.5). Men når det kommer til å redusere polynomer modulo G, har F_4 to ess i ermet, nemlig 1) å benytte seg av lineæralgebrateknikker for spredte matriser (på engelsk *sparse*; det vil si matriser med få elementer ulik 0), og 2) å redusere flere S-polynomer om gangen.

Faugères notasjon er svært forskjellig fra den vi har brukt hittil, så når vi nå gir de sentrale definisjonene og algoritmene fra [10], vil vi tilpasse notasjonen etter de standarder vi har brukt tidligere. Bevisene (og til dels resultatene) gjengitt her er ikke hentet fra [10].

La oss først lære oss å oversette polynomer til matriser, og omvendt.

Definisjon 4.6.1. La $\mathcal{T}$ være mengden av alle mulige monomer i $R = k[x_1, \ldots, x_n]$, det vil si $\mathcal{T} = \{x_1^{\alpha_1} \cdots x_n^{\alpha_n} | (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_{\geq 0}^n\}$.

Vi har ikke tidligere hatt behov for å spesifisere mengden av monomer i R, men i kapittel 3 benytter vi den analoge ikke-kommutative definisjonen, kalt $\mathcal{B}$.

Definisjon 4.6.2. La $\mathcal{T}_M = (t_1, \ldots, t_m)$ være en ordnet mengde av elementer i $\mathcal{T}$, og la $V_{\mathcal{T}_M} = \text{Span}_k(\mathcal{T}_M)$. La $\{\epsilon_i\}_{i=1}^m$ være den kanoniske basisen for k^m, det vil si at $\epsilon_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ der 1 står på plass i. Vi definerer følgende avbildninger:

$$\phi_{\mathcal{T}_M} : V_{\mathcal{T}_M} \longrightarrow k^m \text{ ved } t_i \longmapsto \epsilon_i$$
$$\psi_{\mathcal{T}_M} : k^m \longrightarrow V_{\mathcal{T}_M} \text{ ved } \epsilon_i \longmapsto t_i$$

Eksempel 4.6.3. $\mathcal{T}_M = \{x^2, xy, y^2, x, y, 1\}$ i $k[x, y]$, gradert leksikografisk ordning med $x > y$. La $f = 3x^2 + 2y^2 - y + 8 \in V_{\mathcal{T}_M}$. Vi har

$$\phi_{\mathcal{T}_M}(f) = 3(1,0,0,0,0,0) + 2(0,0,1,0,0,0) - (0,0,0,0,1,0) + 8(0,0,0,0,0,1) = (3,0,2,0,-1,8)$$

Vi lager en $s \times m$-matrise M av et tuppel F bestående av s polynomer i $V_{\mathcal{T}_M}$ ved å la rad i være $\phi_{\mathcal{T}_M}(f_i)$, der f_i er det i'te polynomet i F. Tilsvarende kan vi lage s polynomer av en $s \times m$-matrise ved å bruke $\psi_{\mathcal{T}_M}$ på hver rad, forutsatt at $\mathcal{T}_M$ er kjent. I den videre teksten hopper vi «naturlig» mellom polynomer og matriser, uten å referere til de underliggende avbildningene.

Eksempel 4.6.4. La M være en $s \times m$-matrise over k, og sett $\mathcal{T}_M = \{x_1, \ldots, x_m\}$. Fra M kan vi da lage mengden $F \subseteq k[x_1, \ldots, x_m]$ der alle polynomene i F er lineære. Men la oss holde fokuset på M litt lenger. Med lineæralgebra (for eksempel Gauss-Jordan-eliminering) kan vi finne $\tilde{M}$, den reduserte trappeformen av M. La $\tilde{F}$ være den tilhørende mengden polynomer i $k[x_1, \ldots, x_m]$. Da er $\tilde{F}$ en redusert Gröbnerbasis for idealet $\langle F \rangle$ under leksikografisk ordning med $x_1 > \ldots > x_m$: Vi har at F er en Gröbnerbasis, siden de ledende leddene ikke har noen felles variabler. Av samme grunn kan elementene i f kun redusere hverandre lineært, og det er dette vi har gjort for å få $\tilde{F}$.

Fra kapittel 2.6 vet vi at å finne en Gröbnerbasis fungerer som en slags generalisering av Gauss-Jordan-eliminering. Men i et mer generelt tilfelle enn eksempelet over, kan elementene redusere hverandre mer enn bare lineært (hvis vi lager en matrise av $x^3 + 1$ og $x^2 + x$ vil ikke Gauss-Jordan-eliminering endre matrisen, men det er klart at $x^2 + x$ kan redusere $x^3 + 1$). Anta at vi har gitt en foreløpig Gröbnerbasis F (altså at F er resultatet etter en iterasjon av `while`-løkken i Buchbergers algoritme) og et nytt polynom f, som vi vil redusere modulo F. Hvordan kan vi få til dette ved hjelp av lineæralgebra? Ideen i F_4 er å gjøre det følgende:

1. Lag en matrise A fra F og f.
2. Reduser A til $\tilde{A}$, en matrise på redusert trappeform.
3. Finn redusert utgave av f fra $\tilde{A}$, og legg til denne i F.

Merk. I en matrise på redusert trappeform er alle pivotelementene 1. Dette tilsvarer at vi ikke bare reduserer f modulo F, men at vi også gjør f monisk. Siden det er akkurat dette vi vil (tidligere har vi først redusert f, deretter gjort den monisk), er det ikke et problem, men med notasjonen vi har brukt hittil blir det egentlig feil å si at det polynomet vi leser av fra $\tilde{A}$, er f redusert modulo F.

Vi skal se på konstruksjonen av A. La $T(f) = \{x^\alpha \in \mathcal{T} | x^\alpha \text{ er et monom i } f\}$, og legg til f som første rad i A. Når vi reduserer f modulo F, starter vi typisk med å finne en f_i i F slik at $LT(f_i)|LT(f)$ (for strategier for hvilken f_i vi velger, se forrige avsnitt). Deretter erstatter vi f med $f := f - \frac{LT(f)}{LT(f_i)} f_i$ og begynner forfra igjen.

Når vi har funnet en ny f som over, får vi sannsynligvis flere elementer i $T(f)$, som også må reduseres. Løsningen blir å simulere utvidingen av $T(f)$ før reduksjonen starter: Hver gang vi reduserer (en versjon av) f, legger vil til alle monomer (unntatt det ledende) fra $\frac{LT(f)}{LT(f_i)} f_i$ i $T(f)$. Samtidig legger vi til $\frac{LT(f)}{LT(f_i)} f_i$ som en rad i A. Vi tar ut elementene fra $T(f)$ etter hvert som vi

vurderer dem, og passer på å ikke legge til elementer vi allerede har vurdert. Slik holder vi på til $T(f)$ er tom, og A er ferdig.

Nå skal vi generalisere «algoritmen» beskrevet over slik at vi kan redusere *flere* polynomer om gangen. Ny konstruksjon av A er gitt i algoritme 4.7.

Algoritme 4.7 Konstr-A

1: **input:** $F = \{f_1, \ldots, f_l\}$ som skal reduseres modulo G, og $G = \{g_1, \ldots, g_s\}$
2: **output:** A, en matrise
3: $T(F) := \{x^\alpha \in \mathcal{T} | x^\alpha$ er et monom i et polynom i $F\}$
4: $B := F$
5: **while** $T(F) \neq \emptyset$ **do**
6: $\quad m := \max_\prec T(F)$ # m er maksimalt med hensyn til $\prec$
7: $\quad T(F) := T(F) \setminus \{m\}$
8: $\quad$ **if** $LM(g_i)$ deler m **then**
9: $\quad\quad m' := \frac{m}{LM(g_i)}$
10: $\quad\quad B := B \cup \{m' \cdot g_i\}$
11: $\quad\quad T(F) := T(F) \cup \{$alle monomer i $m' \cdot g_i$ unntatt $LM(m' \cdot g_i)\}$
12: $\quad$ **end if**
13: **end while**
14: $A :=$matrise(B)
15: **return** A

Merk.

1. Årsaken til at vi alltid velger det største monomet i $T(F)$ (linje 6), er at slik unngår vi å legge til elementer i $T(F)$ som vi allerede har sett på og tatt ut av $T(F)$. Det finnes også andre måter å håndtere dette på.
2. Matrise(B) lager en matrise av elementene i B, slik vi har sett tidligere.
3. Det er klart at Konstr-A terminerer, siden idealene generert av monomer som tas ut av $T(F)$ danner en strengt økende kjede av idealer, og ringen $k[x_1, \ldots, x_n]$ er noethersk.

Eksempel 4.6.5. Vi har gitt $G = \{g_1, g_2\} = \{x^2 - 3x, y + 1\}$ og $F = \{f\} = \{x^3 + 2xy + y^2 + 1\}$, gradert leksikografisk ordning med $x > y$. Vi skal bruke algoritme 4.7 til å konstruere en matrise A. Vi starter med å initialisere $T(F)$ og A: Som første rad i A setter vi f. Merk at A er en matrise med 10 kolonner, siden det ledende monomet i f er x^3 og det finnes 10 monomer som er mindre enn eller lik x^3 med den valgte ordningen. $T(F)$ er foreløpig alle monomene i f, i synkende rekkefølge $T(F) = \{x^3, xy, y^2, 1\}$. Vi velger et og et element i $T(F)$, og utvider eventuelt $T(F)$ og A for hver iterasjon.

(1) $m = x^3$
Vi har $LM(g_1)|m$, så vi setter $m' = \frac{x^3}{x^2} = x$ og legger til x^2 til $T(F)$ og $m' \cdot g_1$ til A. Vi har nå $T(F) = \{x^2, xy, y^2, 1\}$, og polynomene i A er $\{f, x^3 - 3x^2\}$.

(2) $m = x^2$
Vi må igjen bruke g_1, og vi får $m' = \frac{x^2}{x^2} = 1$. Da må vi legge til x i $T(F)$ og $m' \cdot g_1 = x^2 - 3x$ i A. Nå er $T(F) = \{xy, y^2, x, 1\}$.

(3) $m = xy$

Nå må vi velge g_2, som gir $m' = \frac{xy}{y} = x$. Vi legger til $m' \cdot g_2 = xy + x$ i A, men vi legger ikke til noe mer i $T(F)$, siden denne allerede inneholder x.

(4) $m = y^2$
Igjen velger vi g_2, og $m' = \frac{y^2}{y} = y$. Da legger vi til y i $T(F)$ og $m' \cdot g_2 = y^2 + y$ i A.

(5) $m = x$
Verken $LM(g_1)$ eller $LM(g_2)$ deler x, så her skjer det ingen ting.

(6) $m = y$
Vi bruker g_2 og får $m' = \frac{y}{y} = 1$. Vi har allerede 1 i $T(F)$, men vi må legge til $m' \cdot g_2 = y + 1$ i A.

(7) $m = 1$
Denne kan vi heller ikke gjøre noe med.

Vi har nå tømt hele $T(F)$, og er ferdige. Matrisen A ser slik ut:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Kolonnene i A representerer henholdsvis $(x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y, 1)$. Det er tre kolonner som kun består av nuller – disse kan naturligvis fjernes, så lenge man vet hvilket monom hver kolonne representerer.

Reduksjonsalgoritmen er gitt i algoritme 5.5. Legg merke til outputen.

Algoritme 4.8 Reduksjon

1: **input:** $F = \{f_1, \ldots, f_l\}$ som skal reduseres modulo G, og $G = \{g_1, \ldots, g_s\}$
2: **output:** $\tilde{A}^+$, en endelig mengde polynomer (færre enn l)
3: $A :=$ Konstr-$A(F, G)$
4: $\tilde{A} :=$ redusert trappeform av A
5: $\tilde{A}^+ := \{f \in \tilde{A} | LM(f) \notin LM(A)\}$
6: **return** $\tilde{A}$

Lemma 4.6.6. *La f være et polynom i $R = k[x_1, \ldots, x_n]$, og la $F = \{f\}$. La G være en endelig delmengde av R, og $\tilde{A}^+ =$* Reduksjon(F, G)*, der* Reduksjon *er algoritme 5.5. Da har vi et av følgende:*

(a) $\tilde{A}^+ = \{r\}$, og det finnes et monom r' slik at $f \xrightarrow{G} r'$, der ingen ledd i r' er divisible med $LT(g_i)$ for noen g_i i G, og slik at $r = LC(r')^{-1}r'$

(b) $\tilde{A}^+ = \emptyset$, og $f \xrightarrow{G} 0$

Bevis. Redusert trappeform av en matrise er unik, uavhengig av metoden vi bruker for å finne den og rekkefølgen vi gjør de mulige operasjonene i.

La A være matrisen som returneres av Konstr-A på input F og G. Radene i A er på formen $m \cdot g_i$, der g_i er i G og m er i $\mathcal{T}$, i tillegg til den første raden, som er f. Merk at med unntak av f-raden, har ingen av radene samme ledende monom.

Når vi reduserer A, starter vi med å få bort pivotelementet til f ved hjelp av 2. rad. Dersom 2. rad er $m \cdot g_i$, tilsvarer dette å redusere f med g_i. Slik fortsetter vi til vi ikke lenger kan redusere f med andre rader. Det er klart at dersom 1. rad nå er polynomet r', har vi $f \xrightarrow{G} r'$. Siden A er konstruert slik at alle monomer som dukker opp i første rad i løpet av denne prosessen, enten er fullstendig redusert modulo G eller så finnes det også som pivot i en annen rad i A, finnes det ingen g_i i G slik at $LT(g_i)$ deler noen ledd i r, så 1. rad har pivot i en kolonne der alle andre elementer er 0.

Vi fullfører reduksjonen med å gange 1. rad med $LC(r')^{-1}$, og betegner første rad r. Det er klart at selv om vi reduserer de andre radene med hverandre, endrer det ikke plasseringen av pivotene. Dersom rad 1 kun er 0'er, returneres $\emptyset$. Hvis ikke, returneres r. ♣

Eksempel 4.6.7. I forrige eksempel konstruerte vi en matrise A ved hjelp av Konstr-A-algoritmen. Vi skal nå redusere A, og se om vi kan lese av den reduserte verdien av f (gjort monisk). Med vanlig Gauss-Jordan-eliminering får vi, dersom vi gjør beregningene i kroppen $\mathbb{F}_{11}$:

$$\tilde{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Siden de fem siste radene i A allerede var redusert (ikke fullstendig redusert) og med pivoter lik 1, er de nesten identiske med de fem siste radene i $\tilde{A}$. Den eneste raden som har en pivot som ikke var i A er den øverste, og Reduksjon returnerer dermed $\tilde{A}^+ = \{x + 5\}$. Dersom vi reduserer f modulo G på «gamlemåten» får vi $7x + 2$, og når vi ganger med 7^{-1} modulo 11, som er -3, får vi ganske riktig $x + 5$ som svar.

Vi kan nå gi en pseudokode for selve F_4-algoritmen. Merk at i Faugères originale artikkel benyttes ikke S-polynomene direkte. Det er mulig dette gir noen små beregnbarhetsmessige fordeler (man kan da regne med noe mindre polynomer), men det er ikke her styrken til F_4 ligger. Algoritmen er gitt i algoritme 4.9.

Merk. Dersom vi velger Q_d til å være *ett* element i Q, har vi Buchbergers algoritme, siden vi allerede har vist at $\tilde{A}^+$ da kun består av det valgte S-polynomet modulo G (dersom det reduserer til 0, har vi $\tilde{A}^+ = \emptyset$ og ingenting skjer).

Teorem 4.6.8. *Algoritmen F_4, som gjengitt i algoritme 4.9, terminerer og returnerer en Gröbnerbasis G slik at $\langle G \rangle = \langle f_1, \ldots, f_s \rangle$.*

Bevis. Vi har vist at dersom kun ett polynom utgjør input'en F til Reduksjon, returneres det korrekt reduserte polynomet modulo G. Vi antar nå at input'en F består av flere S-polynomer. På samme måte som i forrige bevis får vi at Reduksjon reduserer S-polynomene modulo G, men S-polynomene kan i tillegg redusere *hverandre*. Hadde vi tatt et S-polynom om gangen,

og lagt til hver normalform ulik 0 til G før vi reduserte det neste, kunne tidligere beregnede S-polynomer bidratt til å redusere de neste. Dette er altså ikke et problem.

Imidlertid har vi at S-polynomene som reduseres samtidig, kun vil kunne redusere hverandre *lineært*. Dette medfører at vi kan få overflødige elementer i $\tilde{A}^+$ og dermed i Gröbnerbasisen, men dette påvirker ikke korrektheten til algoritmen.

Resten av teoremet følger fra tidligere resultater (teorem 2.5.6 og resultatene fra avsnitt 4.3 der Oppdater-algoritmen gjennomgås). ♣

Hvorfor F_4?

Hva er det som gjør at F_4 er bedre enn Buchbergers? Vi må først understreke at for *små* eksempler, der Buchbergers gir et raskt resultat, er ikke F_4 nødvendigvis bedre. Men F_4 har et fortrinn når det gjelder mange større problemer. Siden matrisen A er en spredt matrise, kan vi bruke reduksjonesteknikker som er langt raskere enn Gauss-eliminering [10, avsnitt 3]. Vi sparer også tid på å redusere flere S-polynomer samtidig. Selve konstruksjonen av A er også effektiv, siden antall iterasjoner i `while`-løkken der er begrenset av antall monomer som er mindre enn $LM(\max_{\prec}\{F\})$.

Merk at siden vi velger flere par fra Q i gangen, slipper vi unna et valg: Dersom vi ønsker å velge $(i, j) \in Q$ med $L(i, j)$ minimal (som er det Faugère har fått best resultat med, og det som ligger til grunn for Gebauer-Möller), og det er flere par som har denne minimale graden, plukker vi ut alle på en gang.

Faugère lagde, ikke lenge etter, en helt ny Gröbnerbasisalgoritme med navnet F_5. Denne er tema for neste kapittel.

Algoritme 4.9 F_4

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
initialiserer G og Q
$G := \{f_1, f_2\}$
if $L(1,2) \neq LM(f_1)LM(f_2)$ **then**
    $Q := \{(1,2)\}$
else
    $Q := \emptyset$
end if
# bygger Q fra F
for $i := 3$ **to** s **do**
    $Q :=$ Oppdater(G, Q, f_i)
    $G := G \cup \{f_i\}$
end for
# vurderer elementene i Q
$\delta := s + 1$
$d := 0$
while $Q \neq \emptyset$ **do**
    $d := d + 1$
    velg $\emptyset \neq Q_d \subseteq Q$
    $Q := Q \setminus Q_d$
    # lager S-polynomer av parene vi valgte
    $S := \emptyset$
    for $(i, j) \in Q_d$ **do**
        $S := S \cup \{S(f_i, f_j)\}$
    end for
    $\tilde{A}^+ :=$ Reduksjon(S, G)
    for $h \in \tilde{A}^+$ **do**
        $f_\delta := h$
        $Q :=$ Oppdater(G, Q, f_δ)
        $G := G \cup \{f_\delta\}$
        $\delta := \delta + 1$
    end for
end while
return G

5 F_5-algoritmen

5.1 Innledning

Selv om Gebauer-Möller-forbedringen av Buchbergers algoritme gjør at vi kan se bort fra mange S-polynomer, er algoritmen fortsatt ikke optimal, siden det blant S-polynomene vi ser på typisk er mange som reduserer til 0. Faugères F_5-algoritme er konstruert rundt et helt nytt kriterium, som gjør at dersom inputen er på en spesiell form, vil vi kunne sortere ut – a priori – *alle* S-polynomer som reduserer til 0. Algoritmen ble introdusert i en artikkel fra 2002 [9]. Selv om den skiller seg radikalt fra både Buchbergers algoritme og forbedringene av denne som vi så på i forrige kapittel, har F_5 ifølge Mora store fellestrekk med den såkalte *Staggered linear basis*-algoritmen som omhandles i [22, kapittel 25.4].

Dessverre er Faugères originale artikkel svært lite detaljert, og det er videre en feil i termineringsbeviset. Selv om F_5 er relativt ny, er det allerede skrevet flere artikler og avhandlinger som går mer i dybden på algoritmen og det nye kriteriet (eller, som vi senere skal se, de nye *kriteriene*). Flere artikler er også under arbeid/i preprint, og det gjenstår å se om noen av disse har klart å rette opp Faugères termineringsbevis. Vi har brukt tre artikler i tillegg til den originale [9] i arbeidet med denne teksten: En diplomoppgave av Stegers fra 2006 [25], en doktorgradsavhandling av Gash fra 2008 [13] og en artikkel av Eder fra 2008 [8]. De to første omhandler hele algoritmen, mens Eders tekst er et alternativt bevis av de to nye kriteriene.

Vi har forsøkt å legge notasjonen så nær som mulig mot Faugères egen (naturligvis tilpasset notasjonen vi har brukt tidligere i oppgaven), men enkelte ganger har det vært fordelaktig å forenkle definisjoner slik det er gjort hos for eksempel Eder. Der dette er gjort, eller der vi har laget vår egen notasjon, har vi opplyst om dette. Alle resultater som er hentet fra en av de nevnte tekstene er også utstyrt med kildehenvisning. Når det gjelder selve algoritmene, er de – med unntak av tilpassing til «vår» notasjon – så godt som like som hos Faugère.

Når vi tilpasser framgangsmåten for å finne en Gröbnerbasis etter de nye kriteriene, får vi en algoritme som skiller seg fra Buchbergers på mange punkter. F_5-algoritmen har følgende egenskaper som vi ikke finner hos Buchbergers algoritme, dersom input'en er $F = \{f_1, \ldots, f_s\}$ i ringen $R = k[x_1, \ldots, x_n]$:

- To helt nye kriterier, som erstatter teorem 2.5.5, benyttes.
- F_5 er iterativ, i den forstand at den først finner en Gröbnerbasis for $\langle f_s \rangle$, deretter for $\langle f_{s-1}, f_s \rangle$ og så videre til $\langle f_1, \ldots, f_s \rangle$.
- Alle polynomer som benyttes i beregningene er utstyrt med tilleggsinformasjon i form av en *signatur* (dette er den mest iøynefallende forskjellen).
- Reduksjonsprosedyren er annerledes.

Innledningsvis nevnte vi at polynomene i input'en F må være på en «spesiell» form for at vi ikke skal ha noen reduksjoner til 0. For det første må f_i'ene være *homogene* (det vil si at alle monomene i f_i har samme totale grad). For det andre må $(f_1, \ldots, f_s)$ være en *regulær følge* (merk at F_5 fungerer for vilkårlige mengder av homogene polynomer, men regularitet er nødvendig for å garantere at ingen S-polynomer reduserer til 0). En definisjon av regulære følger (vi skal bruke en annen definisjon senere) er gitt hos Stegers [25, s. 14]:

Definisjon 5.1.1. En følge $(f_1, \ldots, f_s)$ i $R \setminus \{0\}$ er en **regulær følge** dersom $\langle f_1, \ldots, f_s \rangle \neq R$ og for ingen i, der $1 \leq i \leq s$, er f_i en nulldivisor i $R/\langle f_{i+1}, \ldots, f_s \rangle$. Den siste egenskapen kan også uttrykkes slik: La g være et vilkårlg element i R. Dersom gf_i er et element i $\langle f_{i+1}, \ldots, f_s \rangle$, må g også være et element i $\langle f_{i+1}, \ldots, f_s \rangle$.

Kravet om homogenitet er ikke en like stor innskrenkning som det kan virke, siden det finnes algoritmer som, ved å legge til en ekstra (homogeniserings)variabel til R, kan konvertere et vilkårlig polynom til et homogent polynom. Vi kan utføre beregninger (for eksempel finne en Gröbnerbasis med F_5-algoritmen) på en mengde homogeniserte polynomer, og deretter «avhomogenisere» resultatet. Hvorfor og hvordan dette fungerer skal vi ikke gå i detalj på her. Homogenisering av polynomer er tema i [22, kapittel 20.5].

Faugère skriver innledningsvis i sin artikkel at F_5 er basert på *hovedsyzygier*. For å forstå beviset for kriteriene som brukes i F_5, må vi altså vite hva syzygyer er (vi husker at vi unngikk disse i diskusjonen rundt Buchbergers andre kriterium). Siden både beviset for F_5-kriteriene og syzygy-teorien er nokså omfattende, har vi samlet de fleste bevisene i et avsnitt til slutt i kapitlet. Dette medfører at vi bruker enkelte ikke beviste resultater underveis.

Oppbyggingen av dette kapitlet er derfor som følger: Først ser vi på «tilleggsinformasjonen» (signaturen) til polynomene, og deretter på de nye kriteriene. Videre tar vi for oss reduksjonsdelen av F_5-algoritmen, før vi gjør et eksempel for å vise hvordan algoritmen fungerer i praksis. Til slutt gjør vi deler av beviset for de nye kriteriene. Pseudokoden er spredt utover de tre kommende delkapitlene. Siden F_5-algoritmen er større og mer kompleks enn de tidligere algoritmene vi har sett på, består den av flere delalgoritmer, og for å skape minst mulig forvirring har vi kalt den av disse delalgoritmene som er «skjelettet» i F_5 for F5. Når vi skriver F5 refererer vi altså til en konkret delalgoritme (nærmere bestemt algoritme 5.2), mens når vi skriver F_5, snakker vi om «hele» algoritmen.

5.2 Signerte polynomer

Alle polynomer som opptrer underveis i F_5-algoritmen er såkalte *signerte polynomer*, det vil si at de er utstyrt med en *signatur*. Denne signaturen er et element i R^s, der s er antall polynomer i input'en F. Etter hvert må vi også utføre beregninger i R^t, der t er antall elementer i output'en G. Derfor gjør vi de første generelle definisjonene for R-modulen R^m, der $m \geq 2$. Definisjonene er hentet fra Faugère [10].

Definisjon 5.2.1. La $\{\epsilon\}_{i=1}^m$ være den kanoniske basisen for R^m. Et R^m**-monom** er et element i R^m på formen $t\epsilon_i$, der t er et monom i R og $1 \leq i \leq m$. Vi definerer **indeksen** til et R^m-monom som $\text{index}(t\epsilon_i) = i$.

Dersom $t_i\epsilon_i$ og $t_j\epsilon_j$ er to R^m-monomer, ordner vi dem etter regelen

$$t_i\epsilon_i < t_j\epsilon_j \iff \begin{cases} i > j \text{ eller} \\ i = j \text{ og } t_i \prec t_j \end{cases}$$

der $\prec$ er monomordingen vi bruker på R. Siden R^m er en R-modul, har vi definert multiplikasjon mellom monomer i R og R^s-monomer: La u være et monom i R og la $t\epsilon_j$ være et R^s-monom. Da definerer vi

$$u(t\epsilon_i) = (ut)\epsilon_i$$

Polynom-signaturene er R^m-monomer, men før vi definerer disse, må vi generalisere indekser og ordninger til R^m. Et element i R^m kan skrives entydig som

$$\mathbf{g} = (g_1, \ldots, g_m) = \sum_{i=1}^{m} g_i\epsilon_i$$

Vi definer indeks og ledende ledd for $\mathbf{g}$ som følger:

Definisjon 5.2.2. La $\mathbf{g} = (g_1, \ldots, g_m)$ være et element i R^m, og la i være slik at $g_i \neq 0$ og $g_j = 0$ for alle $1 \leq j < i$. Da definerer vi $\text{index}(\mathbf{g}) = i$. Det **ledende monomet** til $\mathbf{g}$ er da gitt som $LM(\mathbf{g}) = LM(g_i)\epsilon_i$.

Nå ser vi på R-modulen R^s, der s er antall elementer i et fiksert s-tuppel $F = (f_1, \ldots, f_s)$ i R^s. Vi lar F bestå av homogene polynomer, og vi kan tenke på F som input'en til F_5-algoritmen. Vi definerer en viktig avbildning:

Definisjon 5.2.3. La F være som over. Da definerer vi evalueringsavbildningen $v_F : R^s \longrightarrow R$ ved

$$(g_1, \ldots, g_s) \mapsto \sum_{i=1}^{s} g_i f_i$$

Faugère kaller denne avbildningen for v, men det er lurt å ta med indeksen F, siden vi senere skal se på tilsvarende evalueringsavbildninger for andre mengder enn F. Når vi nå definerer signerte polynomer, bruker vi en litt forenklet definisjon av Faugères opprinnelige. Vår definisjon er nærmere Eders definisjon (se [8, Remark 2.3 (b)] for Eders begrunnelse til å forenkle Faugères definisjon), men med en forskjell som vi forklarer i den påfølgende kommentaren.

Definisjon 5.2.4. La $r = (\mathcal{S}(r), \text{poly}(r))$ være i $R^s \times R$. Da er r et **signert polynom** dersom det finnes en $\mathbf{g}$ i R^s slik at

$$\mathcal{S}(r) = LM(\mathbf{g}) \text{ og } v_F(\mathbf{g}) = \text{poly}(r)$$

Indeksen til r er j der $\mathcal{S}(r) = m\epsilon_j$. Vi har $\text{index}(r) = \text{index}(\mathbf{g})$. Dersom $G = \{r_1, \ldots, r_t\}$ er en mengde av signerte polynomer, definerer vi $\text{poly}(G) = \{\text{poly}(r_1), \ldots, \text{poly}(r_t)\}$. Vi sier at $\mathcal{S}(r)$ er *signaturen* til r, mens $\text{poly}(r)$ er *polynomet* til r.

Merk. Eder har beholdt Faugères «admissible»-egenskap i sin definisjon, mens vi lar denne egenskapen være fullstendig innbakt i definisjonen (et tuppel $r = (\mathcal{S}(r), \text{poly}(r))$ er admissible hvis det finnes en $\mathbf{g}$ med egenskapene over). Vi må derfor vise at alle signerte polynomer

som produseres av F_5 er *veldefinerte* istedenfor admissible. Dette kommer vi tilbake til i lemma 5.2.6.

Vi definerer multiplikasjon mellom et monom u i R og et signert polynom $r = (\mathcal{S}(r), \text{poly}(r))$ som følger:

$$ur = (u\mathcal{S}(r), u\text{poly}(r))$$

der $u\mathcal{S}(r)$ allerede er definert, og $u\text{poly}(r)$ er vanlig multiplikasjon i R.

Vi skal nå definere S-polynomer for signerte polynomer, men la oss først fiksere en forenklende notasjon som vi skal bruke gjennom hele kapitlet.

- Dersom r_i er et signert polynom, skriver vi p_i for $\text{poly}(r_i)$.
- Dersom r_i og r_j er to signerte polynomer med $\mathcal{S}(r_i) > \mathcal{S}(r_j)$, skriver vi $L(i,j)$ for $LCM(LM(p_i), LM(p_j))$. Videre lar vi

  $$u_i = \frac{L(i,j)}{LM(p_i)} \text{ og } u_j = \frac{L(i,j)}{LM(p_j)}$$

  som gjør at vi kan skrive $S(p_i, p_j) = \frac{1}{LC(p_i)} u_i p_i - \frac{1}{LC(p_j)} u_j p_j$.

Merk at p_i kun avhenger av r_i, mens u_i også avhenger av r_j (vi snakker aldri om u_i dersom det ikke går klart fram at det er et *par* (r_i, r_j) vi ser på). Vi kan nå definere S-polynomet mellom r_i og r_j, en definisjon som ikke er eksplisitt gitt hos Faugère, men det er den hos Eder:

Definisjon 5.2.5. La r_i og r_j være to signerte polynomer der $u_i\mathcal{S}(r_i) > u_j\mathcal{S}(r_j)$. Da er S-polynomet mellom r_i og r_j definert som

$$S(r_i, r_j) = (u_i\mathcal{S}(r_i), S(p_i, p_j))$$

Graden til et S-polynom defineres som $deg_{\text{tot}}(L(i,j))$.

Vi må nå vise at S-polynomet gitt over er et veldefinert signert polynom.

Lemma 5.2.6. *La r_i og r_j være to signerte polynomer med $u_i\mathcal{S}(r_i) > u_j\mathcal{S}(r_j)$. Da er $S(r_i, r_j) = (u_i\mathcal{S}(r_i), S(p_i, p_j))$ et signert polynom.*

Bevis. La $\text{index}(r_i) = k_i$ og $\text{index}(r_j) = k_j$. Siden r_i og r_j er signerte polynomer, har vi $\mathbf{g} = (g_1, \ldots, g_s)$ og $\mathbf{h} = (h_1, \ldots, h_s)$ slik at

$$\begin{aligned} v_F(\mathbf{g}) &= \text{poly}(r_i) \quad \text{og} \quad LM(\mathbf{g}) = \mathcal{S}(r_i) \\ v_F(\mathbf{h}) &= \text{poly}(r_j) \quad \text{og} \quad LM(\mathbf{h}) = \mathcal{S}(r_j) \end{aligned}$$

Vi definerer et element $\mathbf{q}$ der

$$q_l = \begin{cases} u_i g_l & \text{for } 1 \leq l < k_j \\ u_i g_l - u_j h_l & \text{for } k_j \leq l \leq s \end{cases}$$

Merk at vi har $k_j \geq k_i$. Vi har nå at

$$\begin{aligned} v_F(\mathbf{q}) = \sum_{l=1}^{s} q_l f_l &= \sum_{l=1}^{k_j-1} u_i g_l f_l + \sum_{l=k_j}^{s} (u_i g_l - u_j h_l) f_l \\ &= \sum_{l=1}^{s} u_i g_l f_l - \sum_{l=k_j}^{s} u_j h_l f_l \\ &= u_i \sum_{l=1}^{s} g_l f_l - u_j \sum_{l=1}^{s} h_l f_l \\ &= u_i p_i - u_j p_j \\ &= S(p_i, p_j) \end{aligned}$$

Dersom $k_j > k_i$, har vi $LM(\mathbf{q}) = u_i LM(g_{k_i})\epsilon_{k_i} = u_i \mathcal{S}(r_i) = \mathcal{S}(S(r_i, r_j))$. Dersom $k_j = k_i$, får vi samme resultat siden $LM(u_i\mathbf{g}) \succ LM(u_j\mathbf{h})$. Altså er $S(r_i, r_j)$ et signert polynom. ♣

Vi vet nå hvordan vi lager nye signerte polynomer, men hvor får vi de første fra? Faugère gir en entydig definisjon av hvordan man finner signaturen til et polynom i R, men denne er vanskelig å bruke i praksis. Vi tar heller utgangspunkt i hvordan F_5 lager signerte polynomer, og bruker en rekursiv definisjon:

- For f_i i $F = (f_1, \ldots, f_s)$ setter vi $r_i = (\epsilon_i, f_i)$.
- Senere signerte polynomer konstrueres kun i form av S-polynomer.

Det er lett å verifisere at $r_i = (\epsilon_i, f_i)$ er et gyldig signert polynom: Vi har $\mathbf{g} = \epsilon_i$ der $v_F(\mathbf{g}) = f_i$ og $LM(\mathbf{g}) = \epsilon_i$.

Vi gir nå den iterative delen av F_5-algoritmen, kalt Ytre F5 (algoritme 5.1), og selve F_5-sløyfen, kalt F5 (algoritme 5.2). Variabelen Regler i Ytre F5 skal vi komme tilbake til. Når det gjelder F5, minner den mye om F_4 (den reduserer også flere elementer i gangen), men de tre underalgoritmene LagPar, SPol og Reduksjon er ulikt alt vi har sett tidligere. Legg merke til at valg-strategien i F_5 kommer tydelig fram i F5: Vi velger (i, j)-par der graden til $S(r_i, r_j)$ er minimal. Vi skal ikke drøfte dette nærmere i denne oppgaven.

Merk. F5 får *alle* de tidligere Gröbnerbasisene som er beregnet i tidligere iterasjoner som input. Hos Faugère er det bare G_{k+1} som sendes til F5 fra Ytre F5, men dette er ikke tilstrekkelig, siden vi senere kanskje må finne normalformer modulo G_j, der $j > k+1$. Mer om dette følger. Implementeringen kan gjøres effektivt ved å la G være en global variabel som inneholder alle elementene vi legger til i Gröbnerbasisen, og la $G_s, \ldots, G_1$ peke til elementer i denne (Stegers gjør det samme i sin pseudokode [25, kapittel 3.3]). Vi går ikke nærmere inn på disse detaljene.

Algoritme 5.1 Ytre F5

input: $F = \{f_1, \ldots, f_s\}$ homogene polynomer
output: $G = \{g_1, \ldots, g_t\}$
initialiserer
Regler := []
for $i := 1$ **to** s **do**
    Regler[i] := [] # Regler består av s tomme lister samlet i én liste
end for
$r_s := (\epsilon_s, LC(f_s)^{-1} f_s)$
$G_s := \{r_s\}$
# lager iterativt Gröbnerbasiser $G_{s-1}, \ldots, G_1$
for $i := s - 1$ **downto** 1 **do**
    $G_i :=$ F5$(i, f_i, [G_{i+1}, \ldots, G_s])$
end for
return G_1

Algoritme 5.2 F5

input: i – et heltall, f_i – et signert polynom, $[G_{i+1}, \ldots, G_s]$ – Gröbnerbasiser
output: $G_i = \{r_i, \ldots, r_{n_i}\}$
initialiserer
$r_i := (\epsilon_i, LC(f_i)^{-1} f_i)$
$G_i := G_{i+1} \cup \{r_i\}$
$Q := \emptyset$
# fyller Q med par
for $r \in G_{i+1}$ **do**
    $Q := Q \cup \{\text{LagPar}(i, r_i, r_j, [G_{i+1}, \ldots, G_s])\}$
end for
# vurderer elementene i Q
while $Q \neq \emptyset$ **do**
    $d :=$ minimal grad blant elementene i Q
    $Q_d := \{q \in Q | deg(q) = d\}$
    $Q := Q \setminus Q_d$
    # lager S-polynomer av parene vi valgte
    $S_d :=$ SPol(Q_d)
    # reduserer alle S-polynomene samtidig
    $R_d :=$ Reduksjon$(S_d, G_i, [G_{i+1}, \ldots, G_s])$
    # legger nye elementer til i G_i og finner nye par
    for $r \in R_d$ **do**
        for $r' \in G_i$ **do**
            $Q := Q \cup \{\text{LagPar}(i, r, r', [G_{i+1}, \ldots, G_s])\}$
        end for
        $G_i := G_i \cup \{r\}$
    end for
end while
return G_i

5.3 Nye kriterier

I Faugères opprinnelige artikkel er det *ett* kriterium som blir presentert som «det nye Gröbnerbasiskriteriet». Men det er også et annet kriterium, som vi kan kalle «omskrivbarkriteriet» (jf. Eder, [8, definisjon 3.2]), som brukes i algoritmen. Faugère beviser aldri dette kriteriet, men han bruker det like fullt. Enkelte formuleringer i [9] kan tyde på at omskrivbarkriteriet ikke var ment som et kriterium for Gröbnerbasiser, men at det snarere var ment å benyttes til å skrive om på S-polynomer med tanke på mest mulig gjenbruk av tidligere beregninger.

Definisjonen av *normaliserte par* er gitt hos Faugère, mens definisjonen av *ikke-omskrivbare par* kun er gitt implisitt gjennom algoritmene. Vi gjengir Eders eksplisitte definisjon [8, definisjon 3.2].

Definisjon 5.3.1. La u være et monom i R og r et signert polynom som beregnes i en iterasjon av Ytre F5, der $\mathcal{S}(r) = t\epsilon_k$. Vi sier at ur er **normalisert** dersom ut ikke kan reduseres modulo G_{k+1}.

Et par (r_i, r_j) av slike signerte polynomer, der $\mathcal{S}(r_i) = t_i\epsilon_{k_i}$ og $\mathcal{S}(r_j) = t_j\epsilon_{k_j}$, er normalisert dersom både $u_i r_i$ og $u_j r_j$ er normaliserte.

Definisjon 5.3.2. La u være et monom i R og r et vilkårlig signert polynom i G_k med $\mathcal{S}(r) = t\epsilon_i$. Vi sier at ur er **omskrivbart** dersom det finnes tidligere elementer r_v og r_w i G_k med tilhørende S-polynom $S(r_v, r_w)$ der $\mathcal{S}(S(r_v, r_w)) = t_{v,w}\epsilon_i$ og $t_{v,w}$ deler ut.

Et par (r_i, r_j) av signerte polynomer i $G_k \times G_k$ er **ikke omskrivbart** dersom verken $u_i r_i$ eller $u_j r_j$ er omskrivbare.

Merk. En av de mer forvirrende tingene med F_5-algoritmen er måten elementene i G_i er nummerert på. Alle elementene har sin unike subskrift (de heter r_1, r_2 og så videre), og elementene vi legger til på starten av iterasjonen, har *lavere* subskrift enn *noen* av de foregående elementene. Merk at de øvrige elementene som legges til G_i *ikke* får denne subskriften i F5-algoritmen. Vi skal senere se at dette «navnet» blir gitt elementene i algoritmene som henholdsvis lager S-polynomer (SPol, algoritme 5.4) og reduserer dem (ToppReduksjon, algoritme 5.6). Når det gjelder *indeksen* til r_i, har vi $\text{index}(r_i) = i$ kun for $1 \leq i \leq s$. Anta at vi tester de to kriteriene på et par (r_i, r_j) underveis i den k'te iterasjonen av F5. Når vi tester for normaliserthet bruker vi en Gröbnerbasis fra en tidligere iterasjon. I definisjonen av omskrivbarhet vil vi teste mot *alle* S-polynomer (med samme indeks som det nye) vi tidligere har sett på, enten de er fra iterasjonen vi nå er inne i, eller tidligere iterasjoner.

Før vi gir det nye teoremet som knytter sammen disse definisjonene og Gröbnerbasiser, må vi lage en ny definisjon av (svak) reduksjon til 0, der vi bruker signerte polynomer. Faugère bruker andre navn og annen notasjon [9, Definition 1], men vi skriver definisjonene slik at de likner mest mulig på det vi har brukt før.

Definisjon 5.3.3. La $S(r_i, r_j)$ være et S-polynom. Vi sier at $S(r_i, r_j)$ **reduserer til 0 modulo** $G = \{r_1, \ldots, r_t\}$ dersom

$$S(p_i, p_j) = \sum_{l=1}^{t} a_l p_l \tag{5.1}$$

der a_l er i R, og vi har $LM(a_l p_l) \prec LM(S(p_i, p_j))$ og dessuten $LM(a_l)\mathcal{S}(r_l) \leq \mathcal{S}(S(r_i, r_j))$ for alle $1 \leq l \leq t$.

Har vi, for summen (5.1) over, at $LM(a_l p_l) \prec L(i, j)$, og samme krav til $\mathcal{S}(S(r_i, r_j))$ som før, sier vi at $S(r_i, r_j)$ **reduserer svakt til 0 modulo** G.

Det følgende teoremet er det nye Gröbnerbasiskriteriet, som skal erstatte teorem 2.5.5.

Teorem 5.3.4. *La $F = (f_1, \ldots, f_s)$ være et s-tuppel av homogene polynomer i R, og $G = \{r_1, \ldots, r_t\}$ en mengde signerte polynomer i $R^s \times R$ der $r_i = (\epsilon_i, f_i)$ for $1 \leq i \leq s$ og de resterende r_i'ene er bygd som S-polynomer av tidligere elementer i G. Anta at alle par (r_i, r_j) i $G \times G$ som er normaliserte og ikke omskrivbare reduserer svakt til 0 modulo G. Da er* $\mathrm{poly}(G)$ *en Gröbnerbasis for $I = \langle F \rangle$.*

Vi gir et *delvis* bevis av dette resultatet i avsnitt 5.6. Hos Faugère er kravet om at parene skal være ikke omskrivbare utelatt, men som han beviser, er det tilstrekkelig å kun kreve normaliserthet for å konkludere med at G er en Gröbnerbasis. Derimot vil kjøretiden til F_5 øke dersom vi ser bort fra omskrivbarkriteriet [13, s. 57]. Vi skal se på ulike formuleringer av Gröbnerbasiskriteriene i avsnitt 5.6.

Vi må teste begge kriteriene når vi skal lage nye S-polynomer. I praksis tester vi om et par er normalisert før vi legger det til i Q (altså i algoritmen LagPar), mens hvorvidt paret er omskrivbart testes i algoritmen SPol. Merk at dersom vi, i en av disse algoritmene, oppdager et par som ikke svarer til et av kriteriene, *forkaster* vi dette paret – vi forsøker ikke å «normalisere» eller «omskrive» det.

La oss se på hvordan omskrivbar-testen utføres i F_5. Vi husker at i Ytre F5 fantes linjen

$$4\text{: Regler} = []$$

Variabelen Regler oppdateres hver gang vi lager et nytt S-polynom, og er dermed «registeret» av S-polynomer vi trenger for å teste om et tidligere S-polynom gjør et nytt omskrivbart. For å gjøre dette så effektivt som mulig, består Regler av s lister, der et S-polynom med indeks i registreres i liste nummer i i Regler. Når vi har et nytt S-polynom (som ikke er blitt forkastet) på formen

$$r_l = S(r_i, r_j) = (m\epsilon_k, u_i p_i - u_j p_j)$$

legger vi til regelen $m \longrightarrow l$ i den k'te lista i Regler. Når vi tester om et S-polynom $S(r_i, r_j)$ med $\mathrm{index}(r_i) = k_i$ og $\mathrm{index}(r_j) = k_j$ er omskrivbart, gjør vi som følger: Vi plukker ut reglene på plass k_i i Regler, og går gjennom monomene i disse. For hvert monom tester vi om det deler $u_i r_i$. Deretter gjentar vi for reglene på plass k_j og $u_j r_j$. Hvis vi finner en regel som gjør (r_i, r_j) omskrivbart, forkaster vi dette S-polynomet.

Det er til sammen tre algoritmer hos Faugère som håndterer reglene og selve omskrivingstesten, ved navn AddRule, Rewritten og Rewritten? [9]. Her gir vi ingen pseudokoder for disse, men lar navnene NyRegel og Omskrivbar utføre de samme oppgavene, der NyRegel(r) resulterer i en ny regel som beskrevet over, og

$$\mathrm{Omskrivbar}(u, r_k) = \begin{cases} \text{TRUE dersom } ur_k \text{ kan omskrives} \\ \text{FALSE hvis ikke} \end{cases}$$

Pseudokode for de to algoritmene LagPar og SPol er gitt i henholdsvis algoritme 5.3 og algoritme 5.4.

Algoritme 5.3 LagPar

input: k – et heltall, (r_i, r_j) – to signerte polynomer, $[G_{k+1}, \ldots, G_s]$ – Gröbnerbasiser
output: Et 4-tuppel (u_i, r_i, u_j, r_j) eller $\emptyset$
# sjekke at rekkefølgen på r_i og r_j er riktig
if $\mathcal{S}(u_i r_i) < \mathcal{S}(u_j r_j)$ **then**
    bytt r_i og r_j
end if
# plukke ut monomene fra signaturene
$t_i \epsilon_{k_i} = \mathcal{S}(r_i)$
$t_j \epsilon_{k_j} = \mathcal{S}(r_j)$
# sjekke for normaliserthet
if $\overline{u_i t_i}^{G_{k_i+1}} = u_i t_i$ **then**
    if $\overline{u_j t_j}^{G_{k_j+1}} = u_j t_j$ **then**
        return (u_i, r_i, u_j, r_j)
    end if
end if
return $\emptyset$

Algoritme 5.4 SPol

input: $P = \{(u_i, r_i, u_j, r_j)\}$ endelig liste
output: En liste med signerte S-polynomer (kan være tom)
initialiserer
$S := \emptyset$
la $N - 1$ være subskrift for det siste elementet i G_i
for $p = (u_i, r_i, u_j, r_j) \in P$ **do**
    if Omskrivbar$(u_i, r_i) =$ FALSE **and** Omskrivbar$(u_j, r_j) =$ FALSE **then**
        $N := N + 1$
        $r_N := (u_i \mathcal{S}(r_i), u_i p_i - u_j p_j)$
        NyRegel(r_N)
        $S := S \cup \{r_N\}$
    end if
end for
Sorter S etter økende signatur
return S

Variabelen N som dukker opp i SPol-algoritmen er «navnet» på det *siste* elementet i G_i (det vil si det elementet i G_i som har høyest subskrift). Vi har ikke gitt noen forklaring på hvordan SPol-algoritmen kjenner verdien av N – dette er en implementeringsdetalj. Stegers har løst dette ved å la både G_i og N være globale variabler [25]. Merk også at siden vi sorterer S-polynomene, vil de ikke nødvendigvis legges til G_i i rekkefølgen gitt av subskriftene.

Det kan synes ulogisk å gi r_l «navnet» l allerede i algoritmen SPol – dersom dette hadde vært Buchbergers algoritme hadde vi ikke gitt dette S-polynomet en plass i G før vi hadde sett om det reduserte til 0. Men dersom det ikke opptrer noen reduksjoner til 0 i F_5, som vi senere

skal se at er tilfelle for noen input'er, vil ikke dette være et problem. (Også dersom det *er* reduksjoner til 0 går det an å implementere F5 slik at det ikke er noe problem å gi r_l en plass i G før det blir redusert, men vi skal som vanlig ikke dvele ved slike implementeringssdetaljer.)

Merk. I Faugères opprinnelige versjon av LagPar [9, CritPair] er det to feil som er rettet opp her (og hos Stegers og Gash, se [25, kapittel 3.3] og [13, kapittel 3.4.1]):

1. Hos Faugère er linje 4 i LagPar testen $\mathcal{S}(r_i) < \mathcal{S}(r_j)$. Dette er ikke nok til å konkludere $\mathcal{S}(u_i r_i) < \mathcal{S}(u_j r_j)$, som er kravet fra definisjonen av S-polynomer.
2. Faugère reduserer både $u_i t_i$ og $u_j t_j$ med G_{k+1} i LagPar (det er derfor han, som nevnt tidligere, bare har G_{k+1} som input til F5). Men siden dette ikke nødvendigvis oppfyller kravet i definisjonen av normaliserthet, må vi åpne for at vi også må redusere med andre (tidligere) Gröbnerbasiser.

Det gjenstår å vise hva som skjer i reduksjonsdelen av algoritmen. Målet er å vise at alle S-polynomer som defineres i F_5-algoritmen er normaliserte og ikke omskrivbare, og at ingen par som både er normaliserte og ikke omskrivbare *ikke* blir til S-polynomer. Vi gir dette resultatet i teorem 5.6.4 i avsnitt 5.6.

5.4 Reduksjon av S-polynomer

Som nevnt tidligere er reduksjonen av S-polynomer i F_5-algoritmen annerledes enn reduksjonen vi er vant med fra før (altså den vanlige divisjonsalgoritmen, *alle* hittil definerte elementer i den uferdige Gröbnerbasisen betraktes som potensielle divisorer av S-polynomet som skal reduseres). Nå må vi ta nye hensyn for å hindre at det dukker opp nye S-polynomer som ikke er normaliserte eller omskrivbare, i tillegg til at de reduksjonene må være gyldige i henhold til definisjon 5.3.3. Dette gir utslag i form av tre «overraskende» resultater:

1. Vi reduserer aldri alle leddene i et S-polynom på en gang, vi reduserer kun det ledende leddet (*toppreduksjon*).
2. En slik toppreduksjon kan returnere *ingen*, *ett* eller *to* polynomer.
3. Ikke alle elementer i G_i er gyldige divisorer for alle (eller noen) S-polynomer.

Før vi går nærmere inn på dette, gir vi de tre algoritmene som utgjør reduksjonsdelen av F_5. Algoritmen Reduksjon (algoritme 5.5) er den ytre løkken, ToppReduksjon (algoritme 5.6) gjør selve toppreduksjonen og FinnDivisor (algoritme 5.7) finner, om mulig, et element i G_i som kan brukes til toppreduksjonen. Den siste av disse tilsvarer Faugères IsReducible [9], men vi har valgt å bruke Stegers mer selvforklarende navn [25].

La oss gå gjennom de tre algoritmene i tur og orden. Vi starter med å se på ToppReduksjon, og forklarer de forskjellige «typene» output denne kan ha. Input'en til ToppReduksjon er polynomet r_i som skal toppreduseres, mengden G som kan brukes til toppreduksjon og de tidligere Gröbnerbasisene (disse er med kun fordi de må sendes videre til FinnDivisor). Output'en er et tuppel (h_1, S') der minst en av h_1 og S' er $\emptyset$. De fire alternativene for (h_1, S'), som hver tilhører en av de fire `return`-linjene i ToppReduksjon, er

Algoritme 5.5 Reduksjon

1: **input:** S, G_k – endelige lister signerte polynomer, $[G_{k+1}, \ldots, G_s]$ – Gröbnerbasiser
2: **output:** Endelig liste signerte polynomer
3: # initialiserer
4: $\tilde{S} := \emptyset$
5: # reduserer til alt er ferdig redusert
6: **while** $S \neq \emptyset$ **do**
7: $\quad h :=$ minimalt element i S med hensyn til signaturen
8: $\quad S := S \setminus \{h\}$
9: $\quad h := \overline{h}^{G_{k+1}}$ # normalformen av h i $R/\langle G_{k+1}\rangle$
10: $\quad (h_1, S') :=$ ToppReduksjon$(h, G \cup \tilde{S}, [G_{k+1}, \ldots, G_s])$
11: $\quad \tilde{S} := \tilde{S} \cup \{h_1\}$
12: $\quad S := S \cup S'$
13: **end while**
14: **return** $\tilde{S}$

(a) $(\emptyset, \emptyset)$ dersom p_i er 0 (linje 5).

(b) $((\mathcal{S}(r_i), LC(p_i)^{-1}p_i), \emptyset)$ dersom $LM(p_i)$ ikke kan reduseres modulo G (linje 11).

(c) $(\emptyset, \{r_i := (\mathcal{S}(r_i), LC(p_i)^{-1}p_i - up_j)\})$ dersom $LM(p_i) = uLM(p_j)$ og r_j er en gyldig divisor i henhold til kravene i FinnDivisor. Dessuten må den nye r_i være et veldefinert S-polynom (linje 20).

(d) $(\emptyset, \{r_i, r_N := (u\mathcal{S}(r_j), up_j - LC(p_i)^{-1}p_i)\})$ dersom r_j er en gyldig divisor som i tilfelle (c), men $\mathcal{S}(r_i) < u\mathcal{S}(r_j)$ (linje 26).

Den første delen av tuplet er enten $\emptyset$ eller et ferdig redusert S-polynom (tilfelle (b) – merk at de resterende leddene av r_i ikke nødvendigvis er fullstendig redusert modulo G). Den andre delen av tuplet, dersom den ikke er tom, skal reduseres videre (linje 12 i Reduksjon). At dette virker for tilfelle (c) og at r_i definert i linje 19 er et veldefinert signert polynom er opplagt, men (d) trenger en nærmere forklaring.

Siden $\mathcal{S}(r_i) < u\mathcal{S}(r_j)$ kan vi ikke redusere r_i med r_j, for dette vil gå imot signaturkravet i definisjon 5.3.3 av svak reduksjon til 0. Den nye r_i vil heller ikke nødvendigvis være et veldefinert signert S-polynom. Vi innfører derfor r_N, som er veldefinert (lett å vise), og som vi kan tenke på som en «kopi» av r_i, redusert med r_j. Å *ikke* lage en slik kopi, og heller ikke redusere r_i med r_j, er ikke et alternativ: Når vi til slutt lager en Gröbnerbasis av polynomene i G, fjerner vi all informasjon om signaturene. Vi har at $r_i - ur_j$ er i $\langle \text{poly}(G)\rangle$, men uten r_N er vi ikke garantert at dette elementet har normalform 0 i $R/\langle \text{poly}(G)\rangle$.

Neste gang vi har r_i som input til ToppReduksjon vil ikke r_j være en gyldig divisor fra FinnDivisor. Dette følger av at $\mathcal{S}(r_N) = u\mathcal{S}(r_j) = \mathcal{S}(ur_j)$, så ur_j er nå omskrivbar. Det kan imidlertid være det finnes en annen (gyldig) divisor av r_i.

Merk. Hvorfor alle testene i FinnDivisor? Uten dem kan det være at vi lager ikke-normaliserte S-polynomer i linje 24 i ToppReduksjon. Den første av disse testene, der vi tester $ut_j\epsilon_{k_j} \neq t_i\epsilon_{k_i}$, sørger for at r_N blir veldefinert.

Algoritme 5.6 ToppReduksjon

1: **input:** r_i – signert polynom, G – endelig liste av signerte polynomer, $[G_{k+1}, \ldots, G_s]$ – Gröbnerbasiser
2: **output:** et tuppel (a, b) der minst en av a og b er $\emptyset$
3: **if** $p_i = 0$ **then**
4: # da er ikke input til Ytre F5 en regulær følge
5: **return** $(\emptyset, \emptyset)$
6: **end if**
7: # forsøker å finne en gyldig divisor av $LM(p_i)$
8: $r :=$ FinnDivisor$(r_i, G, [G_{k+1}, \ldots, G_s])$
9: # hvis det ikke finnes en slik divisor
10: **if** $r = \emptyset$ **then**
11: **return** $((\mathcal{S}(r_i), LC(p_i)^{-1}p_i), \emptyset)$
12: **end if**
13: # hvis det finnes en divisor
14: $r_j := r$ # vi trenger «navnet» til r
15: $u := \frac{LM(r_i)}{LM(r_j)}$
16: # teste om reduksjonen er gyldig
17: **if** $u\mathcal{S}(r_j) < \mathcal{S}(r_i)$ **then**
18: # hvis ja: utføre topp-reduksjonen og returnere nye r_i for videre reduksjon
19: poly$(r_i) := LC(p_i)^{-1}p_i - up_j$
20: **return** $(\emptyset, \{r_i\})$
21: **else**
22: # hvis nei: lage et nytt S-polynom
23: la $N - 1$ være subskrift for det siste elementet i G_i
24: $r_N := (u\mathcal{S}(r_j), up_j - LC(p_i)^{-1}p_i)$
25: NyRegel(r_N)
26: **return** $(\emptyset, \{r_i, r_N\})$
27: **end if**

Det følgende lemmaet vil bli brukt i flere senere beviser.

Lemma 5.4.1. *Alle S-polynomer som konstrueres i iterasjon k av F5 har indeks k.*

Bevis. Det er klart at r_k som initialiseres i linje 4 i F5 har indeks k. Første gang LagPar kalles er $r_i = r_k$, mens index$(r_j) > k$, så vi har $\mathcal{S}(u_i r_i) > \mathcal{S}(u_j r_j)$ i linje 4. Vi har altså fortsatt $r_i = r_k$ i output'en (dersom paret (r_k, r_j) er normalisert, ellers er det ingen output). I SPol byttes ikke rekkefølgen på r_k og r_j, så dersom paret også er ikke omskrivbart, har S-polynomet mellom r_k og r_j indeks k. Tilsvarende returnerer LagPar og SPol kun S-polynomer med indeks k (eller $\emptyset$) dersom minst ett av de signerte polynomene i input'en til LagPar har indeks k.

Når vi reduserer i linje 19 i ToppReduksjon, endres ikke signaturen. Det gjør den derimot hvis linje 24 blir utført, men da har vi $u\mathcal{S}(r_j) > \mathcal{S}(r_i)$ og siden r_i har indeks k, må r_j ha indeks mindre eller lik k. Ingen polynomer i iterasjon k har indeks mindre enn k, så index$(r_j) = k$ og det nye S-polynomet r_N har også indeks k. Resten av beviset følger ved induksjon. ♣

Algoritme 5.7 FinnDivisor

1: **input:** $r_i = (t_i\epsilon_{k_i}, p_i)$, G og $[G_{k+1}, \ldots, G_s]$
2: **output:** Et signert polynom eller $\emptyset$
3: # går gjennom G på leting etter en divisor av $LM(p_i)$
4: **for** $r_j = (t_j\epsilon_{k_j}, p_j) \in G$ **do**
5: **if** $LM(p_i) = uLM(p_j)$ **then**
6: # tester om divisoren er gyldig
7: **if** $ut_j\epsilon_{k_j} \neq t_i\epsilon_{k_i}$ **and** Omskrivbar$(u, r_j) =$ FALSE **and** $\overline{ut_j}^{G_{k_j+1}} = ut_j$ **then**
8: **return** r_j
9: **end if**
10: **end if**
11: **end for**
12: **return** $\emptyset$

Vi gir nå to lemmaer der det siste viser at reduksjonen i F_5 fungerer. Dette resultatet er hentet fra Stegers [25, s. 35]. Det første lemmaet er med fordi det brukes i beviset av det andre.

Lemma 5.4.2. *La r_i og r_j være to vilkårlige polynomer i output'en G_1 fra Ytre F5. Da har r_i og r_j forskjellig signatur.*

Bevis. Anta at r_i og r_j ble lagt til i iterasjon k av $F5$. Da har vi, fra lemma 5.4.1, at $\mathcal{S}(r_i) = t_i\epsilon_k$ og $\mathcal{S}(r_j) = t_j\epsilon_k$. Anta at r_i ble lagt til før r_j. Dersom $t_i = t_j$, ville r_j vært omskrivbar, men dette ble testet enten i SPol eller i FinnDivisor. Da må vi ha $t_i \neq t_j$. ♣

Lemma 5.4.3. *La $F = (f_1, \ldots, f_s)$ være en følge av homogene polynomer i $R \setminus \{0\}$. Da holder det følgende for ethvert kall* Reduksjon$(S_d, G_k, [G_{k+1}, \ldots, G_s])$ *i en vilkårlige iterasjon k av F5, der Ytre F5 har F som input:*

(1) Reduksjon *terminerer etter et endelig antall steg.*

(2) Dersom $\tilde{S}$ er resultatet av Reduksjon$(S_d, G_k, [G_{k+1}, \ldots, G_s])$*, reduserer alle S-polynomene i S_d svakt til 0 modulo $G_k \cup \tilde{S}$, der G_k er identisk med den G_k som var med i input'en til Reduksjon.*

Bevis. (1) Vi viser først at Reduksjon terminerer. La h være polynomet vi velger i linje 7 i Reduksjon. Merk at h er den eneste muligheten vi har i linje 7, siden ingen S-polynomer har lik signatur (lemma 5.4.2). Etter linje 9 lar vi h betegne normalformen modulo G_{k+1} av den h vi startet med. Vi har fire muligheter for output'en fra ToppReduksjon (linjenumrene refererer til ToppReduksjon):

(a) $(\emptyset, \emptyset)$ dersom poly$(h) = 0$ (linje 5).

(b) $(\{h\}, \emptyset)$ dersom $LM(\text{poly}(h))$ er i normalform modulo G (linje 11).

(c) $(\emptyset, \{h'\})$ dersom h ble (topp-)redusert til h' (linje 20).

(d) $(\emptyset, \{h, r_N\})$ dersom reduksjonen måtte utføres på en «kopi» av h (linje 26).

Dersom (a) eller (b) er tilfelle, er h ferdig redusert og mengden S får ingen nye elementer (linje 12 i Reduksjon). Dersom h' legges til i S, vil h' bli valgt på nytt i linje 7 i neste iterasjon av `while`-løkken (tilfelle (c)), siden $\mathcal{S}(h') = \mathcal{S}(h)$. Dersom (d) inntreffer og h legges tilbake i

S, blir h valgt i linje 7 i neste iterasjon siden $\mathcal{S}(r_N) > \mathcal{S}(h)$.

Det er klart at dersom kun tilfelle (a), (b) og (c) inntreffer, vil algoritmen terminere, siden det er et endelig antall mulige divisorer av h og et endelig antall elementer i S_d.

Anta at (d) er tilfelle. Vi har sett at vi i neste iterasjon av `while`-løkken får en annen (eller ingen) divisor av h fra FinnDivisor. Det nye polynomet r_N vil heller ikke være en mulig divisor, siden $LM(r_N) < LM(h)$ (fra konstruksjonen av r_N), og $LM(r_N)$ og $LM(h)$ har samme totale grad (dette følger også fra konstruksjonen av r_N og det faktum at h er homogen). Siden det er et endelig antall mulige divisorer, vil h til slutt enten redusere til 0 (tilfelle (a)), eller bli tatt ut av S og lagt til i $\tilde{S}$ (tilfelle (b)). Dermed må Reduksjon terminere.

(2) Vi skal nå vise at output'en fra Reduksjon gjør det vi ønsker. La $r = (\mathcal{S}(r), p)$ være det minimale (med hensyn til $\mathcal{S}(r)$) elementet i S_d i input'en til Reduksjon. r kommer fra et S-polynom, si $S(r_i, r_j) = (u_i\mathcal{S}(r_i), u_i r_i - u_j r_j)$. Dersom r reduserer til 0 i Reduksjon, er påstand (2) automatisk oppfylt. Vi antar derfor at r reduserer til $r' = (\mathcal{S}(r), p')$ der $p' \neq 0$ og $LM(p') < LM(p)$. Merk at grunnet linje 11 i ToppReduksjon er p' monisk. Da finnes det en koeffisient c i kroppen k, polynomer m_q i R og en delmengde $M \subseteq G_k$ slik at

$$p = cp' + \sum_{q \in M} m_q q \tag{5.2}$$

Som vist i del (1) av dette beviset er de eneste elementene som kan legges til i $G \cup \tilde{S}$ under reduksjonen av r være r_N'er som defineres i linje 24 i ToppReduksjon, og disse kan *ikke* brukes til å redusere r. Vi har, under antakelsen om at $p \neq cp'$, at

- $\mathcal{S}(r') = \mathcal{S}(S(r_i, r_j)) = u_i\mathcal{S}(r_i)$ siden signaturen ikke endrer seg når vi reduserer r.
- $LM(\sum_{q \in M} m_q q) = LM(p) < u_i r_i$, så dermed er $LM(m_q q) < u_i r_i$ for alle q i M.

Dermed reduserer r svakt til 0 modulo $G_k \cup \{r'\}$. Men senere i gjennomkjøringen av Reduksjon kan det være at også elementene i M blir redusert, så vi må vise at r reduserer svakt til 0 modulo $G_k \cup \tilde{S}$. Vi antar at det finnes et polynom ψ i M som reduserer til et element $\psi' \neq \psi$, det vil si at

$$\psi = c'\psi' + \sum_{\phi \in M'} m_\phi \phi \tag{5.3}$$

Som over har vi at $\mathcal{S}(c'\psi' + \sum_{\phi \in M'} m_\phi) = \mathcal{S}(\psi)$ og at $LM(c'\phi) < LM(\psi)$ og $LM(m_\phi \phi) \leq LM(\psi)$. Dette gjelder selv om p' er et av elementene i M'. Vi erstatter ψ med (5.3) i (5.2):

$$p = cp' + (c'\psi + \sum_{\phi \in M'} m_\phi \phi) m_\psi + \sum_{q \in M \setminus \{\psi\}} m_q q$$

Signaturen endres ikke, og vi har $LM(c'\psi' + \sum_{\phi \in M'} m_\phi \phi) m_\psi = \psi m_\psi < u_i r_i$. Altså har vi at r reduserer svakt til 0 modulo $G_k \cup \tilde{S}$.

Vi valgte r til å være det minimale elementet i S_d, men argumentet for senere elementer i S_d er nesten identisk og er derfor utelatt herfra (la r være polynomet som velges i linje 7 i Reduksjon i den neste iterasjonen, og så videre). ♣

5.5 Eksempel

Faugère bruker et eksempel der input'en er $F = (f_1, f_2, f_3)$,

$$f_1 = yz^3 - x^2t^2$$
$$f_2 = xz^2 - y^2t$$
$$f_3 = x^2y - z^2t$$

Beregningene foregår i ringen $R = k[x, y, z, t]$, og ordningen er revers gradert leksikografisk ordning med $x > y > z > t$ (at ordningen er revers betyr at vi sammenlikner monomvektorene fra høyre istedenfor fra venstre). I Faugères tekst [9, avsnitt 8] vises beregningene for å finne G_1, der G_2 og G_3 allerede er kjent. Her ser vi på hvordan vi finner G_2 og G_3 (merk at *hele* dette eksempelet også er gitt hos Gash [13, avsnitt 3.4.2]).

Den første algoritmen som kalles er Ytre F5. Der setter vi

$$\text{Regler} = [[], [], []]$$
$$r_3 = (\epsilon_3, x^2y - z^2t)$$
$$G_3 = \{r_3\}$$

Deretter entrer vi `for`-løkken med $i = 2$. Algoritmen F5 initialiseres ved

$$r_2 = (\epsilon_2, xz^2 - y^2t)$$
$$G_2 = \{r_2, r_3\}$$
$$Q = \emptyset$$

Når vi skal fylle Q med par, har vi bare ett mulig par å velge. Vi kjører LagPar$(2, r_2, r_3, G_3)$. De to polynomene r_2 og r_3 kommer i riktig rekkefølge med hensyn til signaturen, så vi går videre til normaliseringstesten. Vi har $L(2, 3) = x^2yz^2$, som gir $u_2 = xy$ og $u_3 = z^2$. Videre er $t_2 = t_3 = 1$, og verken xy eller z^2 kan reduseres med G_3 (ingen av dem er delelige med x^2y), så LagPar returnerer (xy, r_2, z^2, r_3).

I F5 går vi nå inn i `while`-løkken. Det er bare ett element i Q, og vi prøver å lage S-polynom av dette elementet ved å kjøre SPol(Q). Siden det ikke finnes noen tidligere S-polynomer, kan ikke u_2r_2 eller u_3r_3 være omskrivbare, så vi returnerer S-polynomet

$$r_4 = (u_2\mathcal{S}(r_2), u_2r_2 - u_3r_3) = (xy\epsilon_2, -xy^3t + z^4t)$$

Vi lager også en regel $xy \longrightarrow 4$ i den andre lista i Regler.

Tilbake i F5 er turen kommet til reduksjon. Vi har bare ett S-polynom å redusere, så vi kjører Reduksjon$(\{r_4\}, G_2, [G_3])$. Vi ser at r_4 allerede er redusert modulo G_3, så vi går til linje 10 i Reduksjon og kaller ToppReduksjon$(r_4, G_2, [G_3])$. Men verken $LM(p_2)$ eller $LM(p_3)$ er divisorer av $LM(p_4)$, så det eneste som skjer er at vi ganger p_4 med -1 slik at når vi er tilbake i F5 igjen, legger vi til $r_4 = (xy\epsilon_2, xy^3t - z^4t)$ i G_2.

Men før vi legger til r_4, finner vi nye par å legge til i Q. Det er to alternativer, så vi kjører LagPar to ganger:

Den første gangen er input'en til LagPar$(2, r_4, r_2, G_3)$. Vi har $L(4,2) = xy^3z^2t$, som gir $u_4 = z^2$ og $u_2 = y^3t$. Siden $\mathcal{S}(u_4r_4) > \mathcal{S}(u_2r_2)$, bytter vi ikke rekkefølgen på r_4 og r_2. Siden verken $u_4t_4 = xyz^2$ eller $u_2t_2 = y^3t$ lar seg redusere med $LM(g_3)$, returneres (z^2, r_4, y^3t, r_2).

Den neste input'en til LagPar er $(2, r_4, r_3, G_3)$. Nå er $L(4,3) = x^2y^3t$, og dermed er $u_4 = x$ og $u_3 = y^2t$. Vi bytter ikke rekkefølge på r_4 og r_2. Men dette paret er ikke normalisert, siden

$$u_4t_4 = x \cdot xy = x^2y = LM(g_3)$$

Dermed returneres $\emptyset$, og vi har $Q = \{(z^2, r_4, y^3t, r_2)\}$.

Nå er vi inne i den andre iterasjonen av `while`-løkken i F5. Vi har ett element i Q, og dette sender vi som input til SPol. Vi må teste om u_4t_4 eller u_2t_2 er omskrivbare. Begge disse har indeks 2, så vi må sjekke mot den ene regelen ($xy \longrightarrow 4$) som ligger på plass 2 i Regler. Vi ser at $u_4t_4 = z^2xy$, som forsåvidt er delelig med xy, men xy kommer fra regelen $xy \longrightarrow 4$ og r_4 kan ikke gjøre seg selv omskrivbart. Videre er $u_2t_2 = y^3t$, som ikke er delelig med xy. Altså lager vi det nye S-polynomet

$$r_5 = (u_4\mathcal{S}(r_4), u_4r_4 - u_2r_2) = (xyz^2\epsilon_2, -z^6t + y^5t^2)$$

Vi legger også til en ny regel på plass 2 i Regler: $(xyz^2, 5)$

Nå skal vi redusere r_5. Det ledende leddet i r_5 er z^6t, som ikke er delelig med noen tidligere elementer. Dermed skjer det samme som sist: Vi returnerer $r_5 = (xyz^2\epsilon_2, z^6t - y^5t^2)$, og tilbake i F5 legges denne til i G_2. Dessuten må vi se på nye par:

LagPar kjøres tre ganger, med mulige par mellom r_5 og hvert av de tre tidligere elementene i G_2. Første input er $(2, r_5, r_2, G_3)$. Her har vi $u_5t_5 = x^2yz^2 = z^2LM(g_3)$, så LagPar returnerer $\emptyset$.

Neste input er $(2, r_5, r_3, G_3)$. Vi har $u_5t_5 = x^3y^2z^2 = xyz^2LM(g_3)$, så LagPar returnerer $\emptyset$ denne gangen også. Siste input, $(2, r_5, r_4, G_3)$, gir samme resultat, siden $u_5t_5 = x^2y^4z^2 = y^3z^2LM(g_3)$. Da forblir Q tom, og `while`-løkken i F5 stopper.

Den ferdige Gröbnerbasisen er poly$(G_2) = \{xz^2 - y^2t, x^2y - z^2t, xy^3t - z^4t, z^6t - y^5t^2\}$. Merk at det var ingen reduksjoner til 0.

Resten av eksempelet (det vil si, den siste iterasjonen av F5, der man finner G_1) er gitt hos Faugère og Gash. Denne siste iterasjonen er også den mest omfattende, og her vil også omskrivbarkriteriet benyttes.

5.6 Teorien bak F_5

I dette avsnittet skal vi se på (deler av) beviset for teorem 5.3.4, og vi skal vise at F_5-algoritmen, dersom den terminerer, resulterer i en Gröbnerbasis for $\langle F \rangle$. Videre skal vi vise at dersom F er en regulær følge, har vi ingen reduksjoner til 0. Til slutt skal vi si litt om hvorvidt algoritmen terminerer. Men før vi starter med bevisene, trenger vi en grunnleggende innføring i *syzygyer*. Vi åpner med å definere hva en syzygy er.

Definisjon 5.6.1. La $F = (f_1, \ldots, f_m)$ være et ordnet m-tuppel i R^m. Da er m-tuppelet $\mathbf{s} = (s_1, \ldots, s_m)$ en **syzygy** på F dersom

$$v_F(\mathbf{s}) = \sum_{i=1}^{m} s_i f_i = 0$$

Videre definerer vi $\text{Syz}(F) = \{\mathbf{s} \in R^m | \mathbf{s} \text{ er en syzygy på } F\}$. Det er klart at $\text{Syz}(F)$ er en R-modul der vi har definert, for $\mathbf{s}$ og $\mathbf{s}'$ i $\text{Syz}(F)$, og f i R,

$$\begin{aligned} \mathbf{s} + \mathbf{s}' &= (s_1, \ldots, s_m) + (s_1', \ldots, s_m') = (s_1 + s_1', \ldots, s_m + s_m') \\ f \cdot \mathbf{s} &= (fs_1, \ldots, fs_m) \end{aligned}$$

Faktisk har vi brukt syzygyer før – vi nevnte at S'en i ordet S-polynom står for syzygy, og S-polynomene er syzygyer på $LT(G)$ der G er en foreløpig Gröbnerbasis. Tidligere har vi forsøkt, gjennom ymse kriterier, å finne en minimal genererende mengde for $\text{Syz}(LT(G))$. For mer om Buchbergers kriterier uttrykt og utledet ved syzygier, se [7, kapittel 2.9].

F_5 er imidlertid ikke ute etter å finne en basis for $\text{Syz}(LT(G))$. Derimot relaterer den syzygyer på F og reduksjoner til 0, nærmere bestemt syzygyer på formen

$$\mathbf{s}_{i,j} = f_j \epsilon_i - f_i \epsilon_j$$

Dette er de såkalte **hovedsyzygyene** (også kalt *trivielle* syzygyer) i $\text{Syz}(F)$. Vi kaller R-modulen generert av disse for $\text{PSyz}(F)$.

Vi gjengir nå den versjonen av teorem 5.3.4 som vi skal bevise. Merk at denne versjonen av teoremet er identisk med Faugères Theorem 1. Vi bruker imidlertid ikke Faugères bevis, men Eders [8, Theorem 3.3]. For bevis for at omskrivbarkriteriet også er et Gröbnerbasiskriterium og at det kan brukes sammen normaliseringskriteriet, se enten Eders bevis – som følger samme «oppskrift» som beviset under, eller Gash' bevis [13, Theorem 3.4.1]. Gash gir tidligere i sin avhandling et bevis for Faugères teorem, og lar omskrivbarkriteriet få sitt eget resultat som lyder omtrent slik: *Dersom et S-polynom som skal legges til G_k returnerer TRUE når det blir sent som input til algoritmen Omskrivbar, trenger vi ikke legge til dette S-polynomet.* Dette er på en måte en vel så god framstilling som hos Eder, som vi har tatt vårt teorem 5.3.4 fra, men på den andre siden virker det også fornuftig å ha med begge kriteriene i hovedteoremet, siden begge brukes.

Teorem 5.6.2. *La $F = (f_1, \ldots, f_s)$ være et s-tuppel av homogene polynomer i R, og $G = \{r_1, \ldots, r_t\}$ en mengde signerte polynomer i $R^s \times R$ der $r_i = (\epsilon_i, f_i)$ for $1 \leq i \leq s$ og de resterende r_i'ene er bygd som S-polynomer av tidligere elementer i G. Anta at alle normaliserte par (r_i, r_j) i $G \times G$ reduserer svakt til 0 modulo G. Da er* $\text{poly}(G)$ *en Gröbnerbasis for $I = \langle F \rangle$.*

Bevis. Anta at betingelsen i teoremet er oppfylt, og la (r_i, r_j) være et par i $G \times G$ som ikke er normalisert. Vi må vise at $S(r_i, r_j)$ likevel reduserer svakt til 0 modulo G. Vi kan anta at $\mathcal{S}(u_i r_i) > \mathcal{S}(u_j r_j)$. Hvis ikke, bytter vi om på r_i og r_j. Siden (r_i, r_j) ikke er normalisert, er $u_i r_i$ og/eller $u_j r_j$ ikke normalisert.

Vi har at de s første elementene i G er $r_l = (\epsilon_l, f_l)$. Vi antar at de resterende elementene er lagt til i rekkefølge – det vil si at når vi legger til r_δ, er den på plass δ i G. Vi må kanskje

renummerere elementene i G for å oppfylle dette, men det skjer uten tap av generalitet. Vi har en fiksert ordning på G, og kan definere en evalueringsavbildning lik den vi har sett tidligere for F:

$$\begin{aligned} v_G : \ R^t &\longrightarrow R \\ g &\mapsto \sum_{l=1}^{t} g_l p_l \end{aligned}$$

Videre lar vi n_δ være antall elementer i Gröbnerbasisen før iterasjonen der r_δ ble lagt til. Vi antar altså at vi gjør som F_5 og lager Gröbnerbasiser for $\langle f_s \rangle, \langle f_{s-1}, f_s \rangle, \ldots, \langle f_1, \ldots, f_s \rangle$. Vi må bruke disse Gröbnerbasisene for å teste om et par er normalisert (eller utnytte at det ikke er det). Vi antar at (r_i, r_j) dukker opp i iterasjon k av F5, det vil si at r_i har indeks k (lemma 5.4.1). Vi antar også at det er $u_i r_i$ som ikke er normalisert (dersom vi heller velger $u_j r_j$ får vi omtrent samme framgangsmåte, se [8, Remark 3.4 (e)]).

Vi har $r_i = (t_i \epsilon_k, p_i)$. Siden $u_i r_i$ ikke er normalisert, finnes r_v i G_{k+1} slik at $LM(p_v)$ deler $u_i t_i$, det vil si at det finnes et monom λ i R slik at

$$\lambda LM(p_v) = u_i t_i$$

Vi skriver opp en hovedsyzygy basert på dette:

$$\mathbf{s}_{v,k} = p_v \epsilon_k - p_k \epsilon_v$$

Dette er en syzygy på G, og dermed et element i R^t. Merk at vi er interessert i $\mathbf{s}_{v,k}$ og ikke $\mathbf{s}_{v,i}$. Vi har $LM(\mathbf{s}_{v,k}) = LM(p_v \epsilon_k)$

Vi trenger en syzygy til. Dersom $1 \leq i \leq s$ må vi ha $k = i$, og vi bruker den trivielle syzygyen

$$\mathbf{s}_i = \epsilon_k - \epsilon_k$$

Dersom $i > s$, er r_i resultatet av en reduksjon av et S-polynom $u_\alpha r_\alpha - u_\beta r_\beta$. Dette gir oss en syzygy

$$u_\alpha \epsilon_\alpha - u_\beta \epsilon_\beta - \epsilon_i$$

på G, men dersom r_α og/eller r_β også stammer fra S-polynomer, kan vi fortsette å nøste oss bakover helt til vi får en syzygy

$$\mathbf{s}_i = \sum_{l=k}^{n_i+1} a_l \epsilon_l - \epsilon_i$$

der alle S-polynomene vi bruker er laget av par der et av elementene er i G_{k+1} og det andre er r_k. Merk at $\mathbf{s}_i$ har indeks k. Dessuten har vi $LM(\mathbf{s}_i) = \mathcal{S}(r_i)$, og vi har

$$LM(u_i \mathbf{s}_i) = u_i \mathcal{S}(r_i) = u_i t_i \epsilon_k = \lambda LM(\mathbf{s}_{v,k})$$

Resten av beviset holder for begge variantene av $\mathbf{s}_i$, men merk at for den første varianten ($\mathbf{s}_i = \epsilon_k - \epsilon_k$) vil flere av leddene i syzygyen $\mathbf{s}$, som vi snart skal definere, bli 0.

La $p'_v = p_v - LT(p_v)$ og $a'_k = a_k - LT(a_k)$. Vi lager nå en ny syzygy $\mathbf{s}$ av $\mathbf{s}_{v,k}$ og $\mathbf{s}_i$.

$$\begin{aligned}\mathbf{s} = \lambda \mathbf{s}_{v,k} - u_i s_i &= \lambda p_v \epsilon_k - \lambda p_k \epsilon_v - u_i a_k \epsilon_k - \sum_{l=k+1}^{n_i+1} u_l a_l \epsilon_k + u_i \epsilon_i \\ &= (\lambda p'_v - u_i a'_k)\epsilon_k - \sum_{l=k+1}^{n_i+1} u_l a_l \epsilon_k - \lambda p_k \epsilon_v + u_i \epsilon_i \\ &= \sum_{l=k}^{n_i+1} b_l \epsilon_l + u_i \epsilon_i\end{aligned}$$

I den siste summen har vi $b_k = \lambda p'_v - u_i a'_k$ og $b_l = u_l a_l$ for $k < l < v$ og $v < l \leq n_i + 1$, og $b_v = u_v a_v - \lambda p_k$. Om signaturene til leddene i $\mathbf{s}$ vet vi at:

(i) $LM(\lambda p'_v - u_i a'_k)\mathcal{S}(r_k) < u_i \mathcal{S}(r_i)$ siden $\text{index}(r_i) = \text{index}(r_k) = k$ og $u_i \mathcal{S}(r_i) = u_i t_i \epsilon_k$, men $u_i t_i$ er kansellert i $\lambda p_v - u_i a_k$, så $LM(\lambda p'_v - u_i a'_k) < u_i t_i$.

(ii) $LM(u_l a_l)\mathcal{S}(r_l) < u_i \mathcal{S}(r_i)$ siden $\text{index}(r_l) = l > k$.

(iii) $LM(\lambda p_k)\mathcal{S}(r_v) < u_i \mathcal{S}(r_i)$ siden $\text{index}(r_v) > k$.

Det vil si at $\text{index}(\mathbf{s}) = k$ og $LM(\mathbf{s}) = u_i \epsilon_i$. Siden $\mathbf{s}$ er en syzygy, har vi

$$v_G(\mathbf{s}) = \sum_{l=k}^{n_i+1} b_l p_l + u_i p_i = 0 \tag{5.4}$$

Da må alle leddene i $v_G(\mathbf{s})$ kansellere mot et annet ledd. La l være slik at $b_l LM(p_l)$ er maksimal i (5.4). Da finnes også en l' slik at $b_{l'} p_{l'}$ er i $v_G(\mathbf{s})$ og

$$LM(b_l p_l) = LM(b_{l'} p_{l'})$$

Det vil si at det finnes et multippel av S-polynomet $S(r_l, r_{l'})$ i $v_G(\mathbf{s})$, der $u_l \mathcal{S}(r_l) < u_i \mathcal{S}(r_i)$ og $u_{l'} \mathcal{S}(r_{l'}) < u_i \mathcal{S}(r_i)$. (Husk at u_i er fiksert gjennom hele beviset, den «hører til» S-polynomet mellom r_i og r_j.)

Dersom $S(r_l, r_{l'})$ ikke er normalisert, kan vi gjøre det samme som med $S(r_i, r_j)$, og erstatte det med en v_G-sum der verken signaturene eller de ledende leddene er større enn hos $S(r_l, r_{l'})$. Hvis *ikke*, kan de reduseres til 0 modulo G, for vi har antatt at dette er tilfelle for alle normaliserte S-polynomer.

Slik fortsetter vi med å skrive om på S-polynomer (vi velger alltid maksimale ledd i summen (5.4)) helt til det maksimale leddet er $u_i LM(p_i)$, og vi må ha et ledd $u_{l_0} p_{l_0}$ i Σ-uttrykket i $v_G(\mathbf{s})$ som oppfyller

$$LM(u_{l_0} p_{l_0}) = LM(u_i p_i)$$

Vi vil før eller siden finne et slikt element, siden prosessen beskrevet over må slutte etter et endelig antall steg. Da har vi et multippel av $S(r_i, r_{l_0})$, og enten er $u_{l_0} r_{l_0} = u_j r_j$, ellers er dette ikke tilfelle. Vi må se på begge mulighetene.

(1) Anta at $u_{l_0} r_{l_0} \neq u_j r_j$. Da har vi, for polynomer q_l i R,

$$\sum_{l=k}^{n_i+1} q_l p_l - u_{l_0} p_{l_0} + u_i p_i = 0$$

og det finnes et monom v_1 i R slik at

$$u_i p_i - u_{l_0} p_{l_0} = v_1 S(p_i, p_{l_0})$$

Altså har vi at

$$v_1 S(p_i, p_{l_0}) = \sum_{l=k}^{n_i+1} q_l p_l$$

Men vi har også at $u_{l_0} LM(p_{l_0}) = u_j LM(p_j)$, så vi har et monom v_2 og polynomer q_l' i R slik at

$$v_2 S(p_j, p_{l_0}) = \sum_{l=k}^{n_i+1} q_l' p_l$$

Da har vi at

$$\begin{aligned} S(p_i, p_j) &= u_i p_i - u_j p_j \\ &= u_i p_i - u_{l_0} p_{l_0} + u_{l_0} p_{l_0} - u_j p_j \\ &= v_1 S(p_i, p_{l_0}) + v_2 S(p_j, p_{l_0}) \\ &= \sum_{l=k}^{n_i+1} (q_l + q_l') p_l \end{aligned}$$

Vi må kontrollere at signaturene til $(q_l + q_l')p_l$ oppfyller kravet i definisjon 5.3.3 (svak reduksjon til 0), men det følger fra (i)-(iii) som vi hadde tidligere i beviset. Altså reduserer $S(p_i, p_j)$ svakt til 0 modulo G i dette tilfellet.

(2) Anta at $u_{l_0} r_{l_0} = u_j p_j$. Da har vi elementer q_l i R slik at

$$\sum_{l=k}^{n_i+1} q_l p_l - u_j p_j + u_i p_i = 0$$

$$S(p_i, p_j) = -\sum_{l=k}^{n_i+1} q_l p_l$$

Vi ser at $S(r_i, r_j)$ reduserer svakt til 0 også i dette tilfellet. ♣

Beviset over er et konstruktivt bevis, så vi kan demonstrere hvordan det fungerer på et eksempel.

Eksempel 5.6.3. I eksempelet i avsnitt 5.5 så vi at paret (r_5, r_2) ble forkastet i iterasjon 2 av F5, siden paret ikke var normalisert. Vi hadde

$$(u_5, r_5, u_2, r_2) = (x, (xyz^2\epsilon_2, -z^6 t + y^5 t^2), z^4 t, (\epsilon_2, xz^2 - y^2 t))$$

der $u_5 r_5$ ikke var normalisert, siden

$$u_5 LM(p_5) = x^2 y z^2 = z^2 \cdot LM(p_3)$$

Med notasjonen fra beviset har vi $i = 5$, $j = 2$, $k = 2$, $v = 3$ og $\lambda = z^2$. Den første syzygyen vi trenger er hovedsyzygyen

$$\begin{aligned}\mathbf{s}_{3,2} &= p_3\epsilon_2 - p_2\epsilon_3 \\ &= (x^2y - z^2t)\epsilon_2 - (xz^2 - y^2t)\epsilon_3 \\ &= x^2y\epsilon_2 - z^2t\epsilon_2 - xz^2\epsilon_3 + y^2t\epsilon_3\end{aligned}$$

For å finne syzygyen $\mathbf{s}_5$ må vi se på S-polynomet r_5 kommer fra, som var $z^2r_4 - y^3tr_2$. Den første delen av dette polynomet, r_4, kommer også fra et S-polynom, nemlig $xyr_2 - z^2r_3$. Dette gir

$$\begin{aligned}r_5 &= z^2r_4 - y^3tr_2 \\ &= z^2(xyr_2 - z^2r_3) - y^3tr_2 \\ &= xyz^2r_2 - z^4r_3 - y^3tr_2\end{aligned}$$

Altså får vi syzygyen

$$\mathbf{s}_5 = xyz^2\epsilon_2 - z^4\epsilon_3 - y^3t\epsilon_2 - \epsilon_5$$

Vi ser at vi har $u_5LM(\mathbf{s}_5) = \lambda LM(\mathbf{s}_{3,2})$, og vi lager en ny syzygy

$$\begin{aligned}\mathbf{s} &= u_5\mathbf{s}_5 - \lambda\mathbf{s}_{3,2} \\ &= xy^3t\epsilon_2 - y^2z^2t\epsilon_3 + z^4t\epsilon_2 - x\epsilon_5\end{aligned}$$

Flere ledd ble kansellert i $\mathbf{s}$ (det ene var selvfølgelig $u_5LM(\mathbf{s}_5)$, det andre var $\lambda LM(\mathbf{s}_{2,3})$). Vi får nå, ved å bruke evalueringsavbildningen v_G, at

$$\begin{aligned}xy^3tp_2 - y^2z^2tp_3 + z^4tp_2 - xp_5 = 0 \\ S(p_5, p_2) = y^2z^2tp_3 - xy^3tp_2\end{aligned}$$

Vi har $y^2z^2t\mathcal{S}(r_3) = y^2z^2t\epsilon_3 < x^2yz^2\epsilon_2 = \mathcal{S}(S(r_5, r_2))$ og $xy^3t\mathcal{S}(r_2) = xy^3t\epsilon_2 < \mathcal{S}(S(r_5, r_2))$. Når det gjelder de største monomene i hvert «ledd» i summen over, har vi

$$\begin{aligned}xy^3tLM(p_2) &= x^2y^3z^2t \\ y^2z^2tLM(p_3) &= x^2y^3z^2t \\ z^4tLM(p_2) &= xz^6t \\ xLM(p_5) &= xz^6t\end{aligned}$$

Det største leddet er $x^2y^3z^2t$, og vi har et multippel av $S(p_2, p_3)$, nærmere bestemt

$$y^2tS(p_2, p_3) = -y^2tp_4$$

Merk at $S(p_2, p_3)$ er normalisert (ellers ville ikke r_4 vært et element i G). Dette gir oss

$$-y^2tp_4 + xy^3tp_2 - xp_5 = 0$$

Siden $y^2tLM(p_4) = xy^5t^2 < xz^6t$, er det største monomet i denne summen $xy^3LM(p_2) = u_5LM(p_5)$. Siden $xy^3tp_2 = u_2p_2$, er vi i tilfelle (2) i beviset, og vi ender opp med

$$S(p_5, p_2) = -y^2tS(p_2, p_3) = y^2tp_4$$

Dermed reduserer $S(p_5, p_2)$ svakt til 0 modulo G.

Vi beviser nå at F_5-algoritmen, dersom den terminerer, resulterer i en korrekt Gröbnerbasis for $\langle F \rangle$, der $F = (f_1, \ldots, f_s)$ er et s-tuppel med homogene polynomer i R. Vi antar at teorem 5.3.4 holder, det vil si at det er ok å ta ut omskrivbare par.

Teorem 5.6.4. *La G_1 være output av Ytre F5 (algoritme 5.1) på input $F = (f_1, \ldots, f_s)$ der alle f_i'ene er homogene. Da er* $\text{poly}(G_1)$ *en Gröbnerbasis for $\langle F \rangle$.*

Bevis. La F være input'en som beskrevet over, og la G_1 være output'en. Følgende er oppfylt:

- Siden vi legger til (ϵ_i, f_i) på starten av hver iterasjon av F5, har vi $F \subseteq \text{poly}(G_1)$. Dermed er $\langle \text{poly}(G_1) \rangle = \langle F \rangle$.
- Alle par som returneres av LagPar er normaliserte, på grunn av testene i linje 11 og 12. Alle par som forkastes er ikke normaliserte.
- Alle S-polynomer som returneres av SPol er normaliserte, og alle par som forkastes er omskrivbare.
- Nye S-polynomer som lages i linje 24 av ToppReduksjon er normaliserte og ikke omskrivbare, på grunn av testene i linje 7 i FinnDivisor. De er også veldefinerte S-polynomer på grunn av testen $ut_j\epsilon_{k_j} \neq t_i\epsilon_{k_i}$ i FinnDivisor og testen i linje 17 i ToppReduksjon.
- Fra lemma 5.4.3 vet vi at alle par som sendes til Reduksjon reduserer svakt til 0 modulo G_1.

Altså reduserer alle normaliserte og ikke omskrivbare S-polynomer i $\text{poly}(G_1)$ til 0 modulo $\text{poly}(G_1)$, så $\text{poly}(G_1)$ er en Gröbnerbasis for $\langle F \rangle$. ♣

Vi skal nå bevise den «mest elegante» egenskapen ved F_5-algoritmen, som er årsaken til at algoritmen er så effektiv – nemlig at ingen reduksjoner til 0 utføres, gitt at input'en er en regulær følge. Men først må vi gi en ny definisjon på regulære følger i R^m. Denne definisjonen er ekvivalent med definisjon 5.1.1 som vi ga tidligere [25, Theorem 3.4].

Definisjon 5.6.5. En følge $F = (f_1, \ldots, f_m)$ i R^m er en **regulær følge** dersom $\text{Syz}(F) = \text{PSyz}(F)$.

Det omtalte resultatet gir vi i form av et lemma og et teorem, som begge er hentet fra Gash [13], henholdsvis fra Theorem 3.2.1 og Theorem 3.4.3. Det første henter deler av beviset fra et bevis hos Faugère.

Lemma 5.6.6. *La $r = (u\epsilon_i, p)$ og $r' = (u'\epsilon_i, p)$ være to signerte polynomer der $u'\epsilon_i < u\epsilon_i$. La $(f_1, \ldots, f_s)$ være en regulær følge. Da er ikke r normalisert.*

Bevis. Vi har $\mathbf{g}$ og $\mathbf{g}'$ i R^s slik at

(i) $v_F(\mathbf{g}) = v_F(\mathbf{g}') = p$

(ii) $LM(\mathbf{g}) = u\epsilon_i$

(iii) $LM(\mathbf{g}') = u'\epsilon_i$

(iv) $\text{index}(\mathbf{g}) = \text{index}(\mathbf{g}') = i$

Siden $v_F(\mathbf{g} - \mathbf{g}') = 0$, har vi at $\mathbf{g} - \mathbf{g}'$ er i $\mathrm{Syz}(F) = \mathrm{PSyz}(F)$. La $\mathbf{s}$ være en syzygy i $\mathrm{PSyz}(F)$ der $\mathrm{index}(\mathbf{s}) = i$. Da er

$$\begin{aligned}
\mathbf{s} &= \sum_{k=i}^{s} \sum_{j=k+1}^{s} \lambda_{k,j} s_{k,j} \\
&= \sum_{k=i}^{s} \sum_{j=k+1}^{s} \lambda_{k,j} f_j \epsilon_k - \sum_{k=i}^{s} \sum_{j=k+1}^{s} \lambda_{k,j} f_k \epsilon_j \\
&= \sum_{j=k+1}^{s} \lambda_{i,j} f_j \epsilon_i + \sum_{k=i+1}^{s} \sum_{j=k+1}^{s} \lambda_{k,j} f_j \epsilon_k - \sum_{k=1}^{s} \sum_{j=k+1}^{s} \lambda_{k,j} f_k \epsilon_j \\
&= \sum_{j=i+1}^{s} \lambda_{i,j} f_j \epsilon_i + \sum_{k=i+1}^{s} q_k \epsilon_k
\end{aligned}$$

Nå er $LM(\mathbf{s})$ med i den første summen over, det vil si at dersom $LM(\mathbf{s}) = u''\epsilon_i$, finnes det en $f_{j'}$ i F med $i < j' \leq s$ slik at $LM(f_{j'})$ deler u''. Siden $\mathbf{s}$ er en *vilkårlig* syzygy i $\mathrm{PSyz}(F)$ med indeks i, holder dette for $\mathbf{g} - \mathbf{g}'$ også. Altså finnes det en $i < j' \leq s$ slik at $LM(f_{j'})$ deler u, så r er ikke normalisert. ♣

Vi kan nå vise at dersom input'en er regulær, er det ikke noen reduksjoner til 0 i F_5.

Teorem 5.6.7. *Dersom F_5-algoritmen kjøres med input F, der F er en regulær følge i R^s, får vi ingen reduksjoner til 0.*

Bevis. Anta at vi har $r = (u\epsilon_k, p)$ som skal legges til G. Anta videre at r reduserer til 0 modulo G. Det vil si at vi har elementer p_j i $\mathrm{poly}(G)$ og polynomer q_j i R slik at

$$p^* = LC(p)^{-1} p - \sum q_j p_j = 0$$

Vi vet at r er normalisert (dette ble testet i LagPar), og fra FinnDivisor vet vi at for alle monomer u' som finnes i q_j er $u'r_j$ normalisert. Signaturen til p^* er enten $\mathcal{S}(r) = u\epsilon_k$ eller $\mathcal{S}(u'r_j)$ for en u' som beskrevet over.

Anta først at $\mathcal{S}(p^*) = \mathcal{S}(u'r_j)$. Da er $\mathcal{S}(u'r_j) > u\epsilon_k$, og vi har $\mathcal{S}(r_j) = t_j\epsilon_k$. Altså er $u't_j > u$. Nå må det finnes $\mathbf{g}$ og $\mathbf{g}'$ i R^s slik at det følgende holder:

(i) $v_F(\mathbf{g}) = v_F(\mathbf{g}') = p = \sum q_j p_j$

(ii) $LM(\mathbf{g}) = u\epsilon_k$

(iii) $LM(\mathbf{g}') = u't_j\epsilon_k$

(iv) $\mathrm{index}(\mathbf{g}) = \mathrm{index}(\mathbf{g}') = k$

Vi har $\mathbf{g} - \mathbf{g}'$ i $\mathrm{Syz}(F)$, og siden F er regulær, har vi $\mathbf{g} - \mathbf{g}'$ i $\mathrm{PSyz}(F)$. Men fra lemma 5.6.6 vet vi at $u'r_j$ ikke er normalisert. Dette er en motsigelse, og r kan ikke ha redusert til 0 modulo G.

I det andre tilfellet kan vi også finne $\mathbf{g}$ og $\mathbf{g}'$ med egenskapene over, men nå er motsigelsen at r ikke er normalisert. ♣

Merk. Dersom noen av reduksjonene som inngår i $\sum q_j p_j$ kommer fra *normalformberegningen* i linje 9 i ToppReduksjon, har vi ingen garanti for at $u'r_j$ er normalisert. Men dersom $u'r_j$

kommer fra en slik reduksjonen, er $\text{index}(r_j) > k = \text{index}(r)$, så vi kan *ikke* ha $\mathcal{S}(p^*) = \mathcal{S}(u'r_j)$.

Til slutt skal vi si litt om hvorvidt F_5 terminerer. Faugère viser i [9] at dersom det ikke er noen reduksjoner til 0 i F_5, terminerer algoritmen etter et endelig antall steg. Han hevder videre at F_5-algoritmen kan modifiseres slik at den er garantert å terminere på alle (homogene) input'er. Termineringsbeviset bygger på følgende antakelse [9, Theorem 2]:

Anta at alle f_i'ene i input'en F er homogene, og at det ikke forekommer noen reduksjoner til 0. La R_d være resultatet av $\text{Reduksjon}(f_i, G_i, [G_{i+1}, \ldots, G_s])$ for vilkårlige d, j og og i. Da er

$$\langle LT(G_i)\rangle \neq \langle LT(G_i \cup R_d)\rangle \tag{5.5}$$

Det er lett å se hvordan denne egenskapen medfører at F_5 terminerer (beviset vil likne på beviset for at Buchbergers algoritme terminerer, se teorem 2.5.6). Imidlertid *holder ikke* nødvendigvis (5.5) for alle mulige d, j og i. Et moteksempel er gitt hos Gash [13, s. 66]. Gash vier et vedlegg [13, Appendix A] til å vise hvor i Faugères opprinnelige bevis (eller snarere i Stegers detaljerte, men ufullstendige versjon [25, Conjecture 3.31]) feilen ligger.

Gash forsøker også å vise, med andre resultater til hjelp, at F_5 faktisk terminerer. Hans endelige bevis baserer seg på en formodning. En fordel ved Gash' termineringsbevis er at det ikke er avhengig av at ingen S-polynomer reduserer til 0. Vi viser til Gash [13, kapittel 3.5] for formodningen og konsekvensen av denne som garanterer terminering. På John Perrys hjemmeside (http://www.math.usm.edu/perry/research.html) er en uutgitt artikkel ved navn *Modifying Faugère's F5 algorithm to ensure termination* nevnt, denne er skrevet av Perry, Eder og Gash og er sendt inn til ISSAC10.

Gash har også laget en forbedret versjon av F_5, samt en «ny» algoritme F_5T (en hybrid mellom F_5 og Buchbergers algoritme), men igjen henviser vi leseren til Gash' avhandling.

6 Algoritmer for ikke-kommutative Gröbnerbasiser

6.1 Innledning

I kapittel 4 så vi hvordan vi kan modifisere Buchbergers algoritme for ikke-kommutative polynomringer, for å oppnå en mer effektiv algoritme. I dette kapitlet skal vi se hvordan vi kan gjøre mye av det samme med den ikke-kommutative versjonen av Buchbergers algoritme, som vi foreløpig kjenner fra algoritme 3.2. Som før lar vi R betegne polynomringen $k\langle x_1, \ldots, x_n\rangle$, der x_i'ene er ikke-kommuterende variabler og k er en kropp. Vi husker også at $\mathcal{B}$ er alle strenger som kan lages av x_i'ene, altså $\mathcal{B} = \{x_1, \ldots, x_n\}^*$.

Vi så i kapittel 2 og 3 at ikke-kommutative Gröbnerbasiser ble definert og beregnet på en måte som var nært beslektet med de kommutative. Så lenge vi passet på tosidig divisjon og multiplikasjon, var det meste likt. Men det var to forskjeller mellom den kommutative og den ikke-kommutative Buchberger-algoritmen som kanskje kan medføre problemer når vi nå går fra en «teoretisk» til en effektiv algoritme, problemer vi ikke møtte i det kommutative tilfellet:

1. Siden vi ser på idealer i en ring som ikke er noethersk, kan vi risikere at idealene vi er interesserte i har uendelig Gröbnerbasis, så algoritmen vil ikke alltid stoppe.
2. For en mengde $F = \{f_1, \ldots, f_s\}$ i den kommutative ringen $k[x_1, \ldots, x_n]$, er alle mulige S-polynomer gitt av $\{S(i,j) | 1 \leq i < j \leq s\}$, en endelig mengde med $\binom{s}{2}$ elementer. De ikke-kommutative overlappene virker mer uhåndterlige: To monomer kan ha flere (eller ingen) overlapper, og et monom kan også overlappe seg selv.

Når det gjelder det første punktet, er det beste vi kan håpe på, å lage en algoritme som, dersom den stopper, returnerer en Gröbnerbasis (algoritme 3.2 er en slik algoritme). Det er ikke mulig å, på generell basis, avgjøre hvorvidt et ideal I i R har endelig Gröbnerbasis. Vi kan vise at det har en slik ved å finne den, og i enkelte tilfeller kan vi også bevise at et ideal har en uendelig, redusert Gröbnerbasis (se kapittel 9). Men å lage en generell algoritme for dette problemet, lar seg ikke gjøre – et bevis for dette finnes hos Mora ([21, s. 4]).

Det andre «problemet» over er langt mer angripelig. Fra definisjonen av en overlapp (definisjon 3.5.1) på formen

$$l\text{tip}(f)r = \lambda\text{tip}(g)\rho$$

har vi at $\text{tip}(f)$ deler verken λ eller ρ, og $\text{tip}(g)$ deler verken l eller r. Dette medfører at to monomer kun har et endelig antall overlapper.

I forrige kapittel innførte vi mengden $\mathcal{S}_t$ for å holde styr på S-polynomene. Den ikke kommutative analogen blir, av grunner nevnt i punkt 2 over, litt mer komplisert. Istedenfor par (i,j), må vi bruke *blokkeringer*.

Definisjon 6.1.1. La $F = \{f_1, \ldots, f_s\}$ være en mengde polynomer i R. Anta at vi har f_i og f_j i F, med $i \geq j$, slik at det finnes en overlapp

$$l\text{tip}(f_i)r = \lambda\text{tip}(f_j)\rho$$

i henhold til definisjon 3.5.1. Da definerer vi **blokkeringen**

$$\sigma = (l, i, r; \lambda, j, \rho)$$

et 6-tuppel i $\mathcal{B} \times \mathbb{Z}_s \times \mathcal{B} \times \mathcal{B} \times \mathbb{Z}_s \times \mathcal{B}$. Mengden av alle blokkeringer på F er gitt ved

$$\mathcal{O}_F := \{(l, i, r; \lambda, j, \rho) \mid i \geq j \text{ og } l\text{tip}(f_i)r = \lambda\text{tip}(f_j)\rho \text{ er en overlapp}\}$$

Videre definerer vi **overlappsrelasjonen av en blokkering** σ som

$$O(\sigma) = \frac{1}{\text{Ctip}(f_i)} l f_i r - \frac{1}{\text{Ctip}(f_j)} \lambda f_j \rho$$

Vi skriver om algoritme 3.2 slik at den kun beregner hver overlappsrelasjon én gang. Resulatet er gitt i algoritme 6.1.

De fleste resultatene og definisjonene i dette kapitlet er hentet fra Mora [21] eller Cohen [6], men de er typisk omskrevet noe for at den endelige algoritmen (algoritme 6.3) skal bli mest mulig lik den kommutative. De siste resultatene før algoritmen (redundante elementer, lage nye grunnmengder) er ikke-kommutative tilpasninger av stoffet fra forrige kapittel. Dersom det ikke er presisert noe opphav for resultater eller definisjoner, er de et resultat av en slik «oversettelse». Når det gjelder definisjonen av blokkeringer gitt over, er den fra [21] (en liknende definisjon finnes hos [6], men der benyttes $\text{tip}(g_i) \preceq \text{tip}(g_j)$ til fordel for $i \geq j$), men vi har utvidet den med kravene til en overlappsrelasjon. Dessuten har vi satt $i \geq j$ istedenfor det «vanlige» $i \leq j$, noe som også skiller seg fra det vi brukte i det kommutative tilfellet (kun av estetiske grunner).

6.2 Ikke-kommutativ analog av Gebauer-Möller

Det er lett å se at en ikke-kommutativ analog av Buchbergers første kriterium (lemma 4.2.1) er overflødig, siden dette kriteriet ivaretas av punkt 3 i overlapp-definisjonen.

Merk. I enkelte tekster ([21], [6]) er overlapper/blokkeringer definert uten dette punktet, så disse bruker en ikke-kommutativ versjon av Buchbergers første kriterium.

Istedenfor å finne en ikke-kommutativ analog av Buchbergers andre kriterium, går vi direkte til å oversette Gebauer-Möller (som i det kommutative tilfellet var mer effektiv enn Buchbergers andre kriterium). Vi startet det forrige Gebauer-Möller-kapitlet med å definere Gebauer-Möller-mengder. Her benytter vi oss istedenfor av *grunnmengder* av blokkeringer.

Definisjon 6.2.1. La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$, og la $I = \langle G \rangle$. Vi har da definert mengden $\mathcal{O}_G$ av alle blokkeringer på G. Vi sier at en delmengde $Q \subseteq \mathcal{O}_G$ er en **grunnmengde** for G dersom den følgende implikasjonen holder:

$$\text{Hvis alle blokkeringer } \sigma \text{ i } Q \text{ er slik at } O(\sigma) \xrightarrow{G} 0, \text{ så er } G \text{ en Gröbnerbasis for } I. \tag{6.1}$$

Algoritme 6.1 Buchbergers algoritme for ikke-kommutative polynomringer, versjon 2

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
$\delta := s + 1$
$Q := \emptyset$
for $i := 1$ **to** s **do**
  for $j := 1$ **to** i **do**
    for alle blokkeringer $\sigma = (l, i, r; \lambda, j, \rho)$ **do**
      $Q := Q \cup \{\sigma\}$
    end for
  end for
end for
while $Q \neq \emptyset$ **do**
  velg $\sigma \in Q$
  $Q := Q \setminus \{\sigma\}$
  $r := O(\sigma)$ modulo F
  if $r \neq 0$ **then**
    $f_\delta := \mathrm{Ctip}(r)^{-1} \cdot r$ # vil ha f_δ monisk
    $F := F \cup \{f_\delta\}$
    for $k := 1$ **to** δ **do**
      for alle blokkeringer $\sigma = (l, \delta, r; \lambda, k, \rho)$ **do**
        $Q := Q \cup \{\sigma\}$
      end for
    end for
    $\delta := \delta + 1$
  end if
end while
$G := F$
return G

Merk. Definisjonen er nesten identisk med Cohens *basic sets*, men der består Q av overlappsrelasjoner, ikke blokkeringer [6, def. 8].

Eksempel 6.2.2. $\mathcal{O}_G$ er en grunnmengde for $G = \{g_1, \ldots, g_t\}$, siden G er en Gröbnerbasis hvis og bare hvis $O(\sigma)$ reduserer til 0 modulo G for alle σ i $\mathcal{O}_G$ (teorem 3.5.3).

Korollar 6.2.3. *La G og Q være som i definisjonen over. Dersom Q er en grunnmengde for G, og $Q = \emptyset$, er G en Gröbnerbasis for $\langle G \rangle$.*

Bevis. Siden Q ikke har noen elementer, er første del av implikasjonen (6.1) automatisk oppfylt, og siden denne implikasjonen er sann, er G en Gröbnerbasis for $\langle G \rangle$. ♣

Videre trenger vi en definisjon av svak reduksjon til 0. Vi husker fra definisjon 3.3.9 at en overlappsrelasjon

$$h = \frac{1}{\mathrm{Ctip}(g_i)} l g_i r - \frac{1}{\mathrm{Ctip}(g_j)} \lambda g_j \rho$$

reduserer til 0 modulo $G = \{g_1, \ldots, g_t\}$ dersom

$$h = \sum_l u_l g_l v_l \text{ (endelig sum)}$$

for u_l og v_l i $\mathcal{B}$, der $u_l \text{tip}(g_l) v_l \preceq \text{tip}(h)$ for alle l. Den følgende definisjonen er hentet fra [21]:

Definisjon 6.2.4. Overlappsrelasjonen

$$h = \frac{1}{\text{Ctip}(g_i)} l g_i r - \frac{1}{\text{Ctip}(g_j)} \lambda g_j \rho$$

reduserer svakt til 0 modulo G dersom det finnes elementer u_l og v_l i $\mathcal{B}$ slik at

$$h = \sum_l u_l g_l v_l \text{ (endelig sum)}$$

der $u_l \text{tip}(g_l) v_l \prec l\text{tip}(g_i) r (= \lambda \text{tip}(g_j) \rho)$ for alle l.

Vi ser at dette speiler det kommutative tilfellet, og dette gir oss følgende Gröbnerbasis-kriterium:

Teorem 6.2.5. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$, og la $I = \langle G \rangle$. Da er G en Gröbnerbasis for I hvis og bare hvis alle overlappsrelasjoner $O(\sigma)$, der σ er i $\mathcal{O}_G$, reduserer svakt til 0 modulo G.*

Bevis. Det er faktisk dette vi bruker i beviset av teorem 3.5.3, så dette er allerede bevist. Merk at dersom en overlappsrelasjon reduserer til 0 modulo G, reduserer den også svakt til 0 modulo G. ♣

Vi fortsetter å følge ruten fra forrige kapittel, med følgende resultat:

Korollar 6.2.6. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$, og la $I = \langle G \rangle$. Da er G en Gröbnerbasis for I hvis og bare hvis det finnes en grunnmengde av blokkeringer på G, kalt Q, der $O(\sigma)$ reduserer svakt til 0 for alle σ i Q.*

Bevis. $\Rightarrow$ Dersom G er en Gröbnerbasis, er $\mathcal{O}_G$ en grunnmengde for G, der alle overlappsrelasjonene reduserer svakt til 0 modulo G.

$\Leftarrow$ Anta at Q er en grunnmengde for G, og at for alle σ i Q har vi $O(\sigma) \xrightarrow{G} 0$. Da er G per definisjon en Gröbnerbasis for I. ♣

Så langt har oversettingen gått greit, men nå støter vi på et problem. For kommutative S-polynomer hadde vi alltid (lemma 4.3.4) at

$$\frac{L(i,j,k)}{L(i,k)} S(i,k) - \frac{L(i,j,k)}{L(i,j)} S(i,j) + \frac{L(i,j,k)}{L(k,j)} S(k,j) = 0$$

og dette la grunnlaget for definisjonen av redundante S-polynomer. Men dette resultatet lar seg nødig oversette til den ikke-kommutative verden. For en blokkering $(l, i, r; \lambda, j, \rho)$ har vi at $l\text{tip}(g_i) r$ er den ikke-kommutative varianten av $L(i,j)$, men hva skal $L(i,j,k)$ oversettes til?

At vi har to blokkeringer mellom henholdsvis (g_i, g_j) og (g_j, g_k) gir oss ikke alltid en overlapp mellom $\text{tip}(g_i)$ og $\text{tip}(g_k)$. Og hva skal vi gjøre med brøkene i formelen over? Divisjon er ikke entydig definert i $\mathcal{B}$, men vi sier likevel (for u og v i $\mathcal{B}$) at u deler v dersom

$$v = aub$$

for monomer a og b i $\mathcal{B}$. Vi gir en også en definisjon på at to *blokkeringer* deler hverandre:

Definisjon 6.2.7. La $\sigma_1 = (l_1, i, r_1; \lambda_1, j, \rho_1)$ og $\sigma_2 = (l_2, i, r_2; \lambda_2, k, \rho_2)$ være to blokkeringer på $G = \{g_1, \ldots, g_t\}$ der $1 \leq j, k < i \leq t$. Vi sier at σ_1 **deler** σ_2 dersom det eksisterer w_1 og w_2 i B slik at $l_2 = w_1 l_1$ og $r_2 = r_1 w_2$.

Eksempel 6.2.8. La $F = \{f_1, f_2, f_3\} \subseteq k\langle x, y\rangle$ der

$$\begin{aligned} f_1 &= xyyxy + xy \\ f_2 &= yx - x \\ f_3 &= yy - y \end{aligned}$$

med lengde-leksikografisk ordning med $x > y$. Blant blokkeringene i $\mathcal{O}_G$ finner vi

$$\begin{aligned} \sigma_1 &= (xy, 2, y; 1, 1, 1) \\ \sigma_2 &= (y, 3, 1; 1, 2, x) \end{aligned}$$

Vi ser at $xy = x \cdot y$ og $y = 1 \cdot y$, så σ_2 deler σ_1.

Vi gir nå et kriterium for å gjenkjenne en type overflødige blokkeringer. Lemmaet er en generalisering av et resultat hos Cohen [6, s. 8].

Lemma 6.2.9. *La σ_1 og σ_2 være som i definisjonen over. Dersom σ_1 deler σ_2, finnes det en blokkering σ_3 slik at dersom σ_1 og σ_3 reduserer svakt til 0 modulo G, gjør også σ_2 det.*

Bevis. Vi har $\sigma_1 = (l_1, i, r_1; \lambda_1, j, \rho_1)$ og $\sigma_2 = (l_2, i, r_2; \lambda_2, k, \rho_2)$, som gir fire muligheter for l_1 og r_1 (fra definisjonen av overlapp):

(1) $l_1 = 1$ og $r_1 \neq 1$

(2) $l_1 \neq 1$ og $r_1 = 1$

(3) $l_1 \neq 1$ og $r_1 \neq 1$

(4) $l_1 = 1$ og $r_1 = 1$

Her skal vi ta for oss det første av disse tilfellene, men de resterende tre kan vises med tilsvarende argumenter.

Vi antar at vi har $l_1 = 1$ og $r_1 \neq 1$. Da er $\rho_1 = 1$ og vi har

$$\text{tip}(g_i) r_1 = \lambda_1 \text{tip}(g_j)$$

Dessuten har vi $w_1 \text{tip}(g_i) r_1 w_2 = \lambda_2 \text{tip}(g_k) \rho_2$ slik at enten

(a) $w_1 = 1$ og $\rho_2 = 1$, eller

(b) $w_1 \neq 1$ og $\lambda_2 = \rho_2 = 1$

I tilfelle (a) har vi

$$\lambda_1 \text{tip}(g_j) w_2 = \lambda_2 \text{tip}(g_k)$$

Dersom λ_1 deler λ_2, har vi en λ_3 i $\mathcal{B}$ slik at $\lambda_2 = \lambda_1 \lambda_3$ og

$$\text{tip}(g_j) w_2 = \lambda_3 \text{tip}(g_k)$$

som gir blokkeringen $\sigma_3 = (1, j, w_2; \lambda_3, k, 1)$. Hvis vi ser på overlappsrelasjonene, har vi

$$\begin{aligned} O(\sigma_1) &= g_i r_1 - \lambda_1 g_j \\ O(\sigma_2) &= g_i r_1 w_2 - \lambda_2 g_k \\ O(\sigma_3) &= g_j w_2 - \lambda_3 g_k \end{aligned}$$

Vi antar at $O(\sigma_1)$ og $O(\sigma_3)$ reduserer svakt til 0 modulo G, det vil si at vi har

$$\begin{aligned} O(\sigma_1) &= \sum_l u_l g_l v_l \ \text{ der } \ u_l \text{tip}(g_l) v_l \prec \text{tip}(g_i) r_1 \ \text{ og} \\ O(\sigma_3) &= \sum_p u_p g_p v_p \ \text{ der } \ u_p \text{tip}(g_p) v_p \prec \lambda_3 \text{tip}(g_k) \end{aligned}$$

Vi ser på $O(\sigma_2)$:

$$\begin{aligned} O(\sigma_2) &= g_i r_1 w_2 - \lambda_2 g_k \\ &= g_i r_1 w_2 - \lambda_1 g_j w_2 + \lambda_1 g_j w_2 - \lambda_1 \lambda_3 g_k \\ &= (g_i r_1 - \lambda_1 g_j) w_2 + \lambda_1 (g_j w_2 - \lambda_3 g_k) \\ &= O(\sigma_1) w_2 + \lambda_1 O(\sigma_3) \\ &= \sum_l u_l g_l v_l w_2 + \sum_p \lambda_1 u_p g_p v_p \end{aligned}$$

der vi har, fra definisjonen av svak reduksjon til 0, at

$$\begin{aligned} u_l \text{tip}(g_l) v_l w_2 &\prec \text{tip}(g_i) r_1 w_2 \ \text{ og} \\ \lambda_1 u_p \text{tip}(g_p) v_p &\prec \lambda_1 \lambda_3 \text{tip}(g_k) = \lambda_2 \text{tip}(g_k) \end{aligned}$$

så $O(\sigma_2)$ reduserer svakt til 0 modulo G.

Dersom λ_1 ikke deler λ_2, har vi at λ_2 deler λ_1, slik at $\lambda_1 = \lambda_2 \lambda_3$ og

$$\lambda_3 \text{tip}(g_j) w_2 = \text{tip}(g_k)$$

Dette gir $O(\sigma_2) = O(\sigma_1) w_1 + \lambda_2 O(\sigma_3)$, og resten følger som over.

Tilfelle (b) gir oss $\sigma_3 = (w_1 \lambda_1, j, w_2; 1, k, 1)$ og $O(\sigma_2) = w_1 O(\sigma_1) w_2 + O(\sigma_3)$. ♣

Vi ser at dette minner oss om lemma 4.3.4, men at det er mindre generelt. Vi kan også sammenlikne med definisjon 4.3.6 av redundante S-polynomer (den ikke-kommutative analogen blir gitt senere), og da ser vi at det vi har gjort likner på del (b) herfra. Vi skal nå gi et resultat som «svarer til» del (a) av denne definisjonen, hentet fra [21, s. 17].

Lemma 6.2.10. *La $\sigma_1 = (l, i, r; \lambda, k, \rho)$ være en blokkering på G, der $G = \{g_1, \ldots, g_t\} \subseteq R$ og $\mathrm{Ctip}(g_s) = 1$ for $1 \leq s \leq t$. Anta at minst en av $\{l, r\}$ og minst en av $\{\lambda, \rho\}$ er 1. La g_j være i G slik at $j > i \geq k$, og anta at det finnes $w_1 \neq 1$ og $w_2 \neq 1$ i $\mathcal{B}$ slik at*

$$l\mathrm{tip}(g_i)r = w_1\mathrm{tip}(g_j)w_2$$

Da finnes blokkeringer σ_2 og σ_3 slik at dersom σ_2 og σ_3 reduserer svakt til 0 modulo G, gjør også σ_1 det.

Bevis. Vi antar at $l = \rho = 1$, det vil si at $\mathrm{tip}(g_i)r = \lambda\mathrm{tip}(g_k)$ (dersom vi heller antar $r = \lambda = 1$, gjøres et symmetrisk bevis). Vi har også gitt en g_j i G med $j > i \geq k$, og $w_1 \neq 1 \neq w_2$ i $\mathcal{B}$ slik at $\mathrm{tip}(g_i)r = w_1\mathrm{tip}(g_j)w_2$. Vi har fire muligheter for hvordan w_1 og w_2 er bygd opp i forhold til $\mathrm{tip}(g_i)r$ og $\lambda\mathrm{tip}(g_k)$. I alle tilfellene finnes L og R i $\mathcal{B}$ slik at henholdsvis

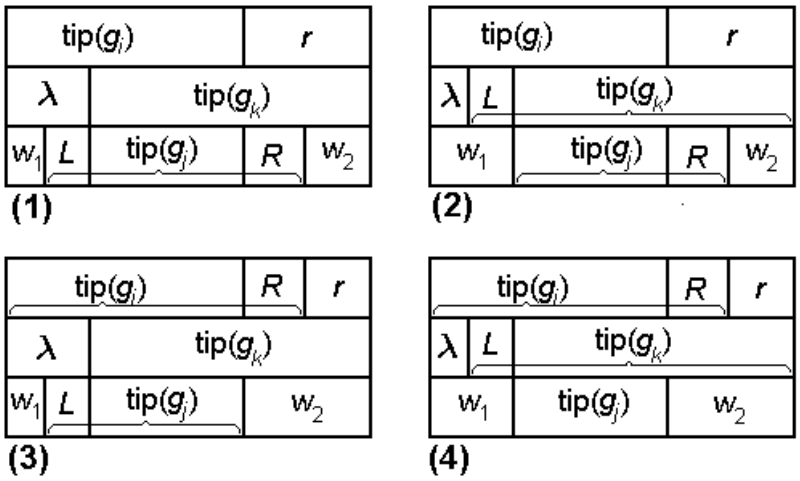
(1) $\mathrm{tip}(g_i)R = w_1\mathrm{tip}(g_j)$ og $L\mathrm{tip}(g_k) = \mathrm{tip}(g_j)w_2$

(2) $\mathrm{tip}(g_i)R = w_1\mathrm{tip}(g_j)$ og $\mathrm{tip}(g_k) = L\mathrm{tip}(g_j)w_2$

(3) $\mathrm{tip}(g_i) = w_1\mathrm{tip}(g_j)R$ og $L\mathrm{tip}(g_k) = \mathrm{tip}(g_j)w_2$

(4) $\mathrm{tip}(g_i) = w_1\mathrm{tip}(g_j)R$ og $\mathrm{tip}(g_k) = L\mathrm{tip}(g_j)w_2$

De fire mulighetene er illustrert på figur 6.1.



Figur 6.1: Muligheter for overlapp.

De to likningene i hvert av punktene (1)–(4) danner hver sin blokkering/overlappsrelasjon. Her ser vi kun på tilfelle (3) – man følger en liknende framgangsmåte for de andre tilfellene (de er gjengitt i [21]). Vi har

$$\sigma_1 = (1, i, r; \lambda, k, 1)$$
$$\sigma_2 = (w_1, j, R; 1, i, 1)$$
$$\sigma_3 = (1, j, w_2; L, k, 1)$$

som gir

$$\begin{aligned} O(\sigma_1) &= g_i r - \lambda g_k \\ &= g_i r - w_1 g_j w_2 + w_1 g_j w_2 - w_1 L g_k \\ &= -(w_1 g_j R r - g_i r) + w_1 (g_j w_2 - L g_k) \\ &= -O(\sigma_2) r + w_1 O(\sigma_3) \end{aligned}$$

Anta at $O(\sigma_2)$ og $O(\sigma_3)$ reduserer svakt til 0 modulo G, det vil si at vi har

$$\begin{aligned} O(\sigma_2) &= \sum_l u_l g_l v_l \text{ der } u_l \text{tip}(g_l) v_l \prec \text{tip}(g_i) \\ O(\sigma_3) &= \sum_p u_p g_p v_p \text{ der } u_p \text{tip}(g_p) v_p \prec L\text{tip}(g_k) \end{aligned}$$

Da er

$$\begin{aligned} \text{tip}(O(\sigma_2) r) &\prec \text{tip}(g_i) r \text{ og} \\ \text{tip}(w_1 O(\sigma_3)) &\prec w_1 L \text{tip}(g_k) = \lambda \text{tip}(g_k) \end{aligned}$$

så $O(\sigma_1)$ reduserer svakt til 0 modulo G. ♣

Som vi ser, tilsvarer dette del (b) i den kommutative definisjonen av redundante S-polynomer. Vi formaliserer dette for ikke-kommutative blokkeringer med følgende definisjon:

Definisjon 6.2.11. La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$. En blokkering $\sigma = (l_1, i, r_1; \lambda_1, j, \rho_1)$ på G er **redundant** dersom det finnes en k der $1 \leq k \leq t$ og et av følgende holder:

(a) $k > i$, minst en av $\{l_1, r_1\}$ og minst en av $\{\lambda_1, \rho_1\}$ er 1 og det finnes $w_1 \neq 1$ og $w_2 \neq 1$ i $\mathcal{B}$ slik at
$$l_1 \text{tip}(g_i) r_1 = w_1 \text{tip}(g_k) w_2$$

(b) $k < i$ og det finnes en blokkering $\sigma' = (l_2, i, r_2; \lambda_2, k, \rho_2)$ på G slik at σ' deler σ

Det neste resultatet følger direkte fra definisjon 6.2.1, lemma 6.2.9 og lemma 6.2.10.

Korollar 6.2.12. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$, og la Q være en grunnmengde for G. Anta at vi har σ i Q der σ er redundant. Da er $Q \setminus \{\sigma\}$ også en grunnmengde for G.*

Vi fortsetter med følgende lemma, som forteller oss hvordan vi går fram for å lage en grunnmengde for $G \cup \{g_t\}$, gitt en grunnmengde for G.

Lemma 6.2.13. *La $G = \{g_1, \ldots, g_t\} \subseteq R$, der $\text{Ctip}(g_i) = 1$ for $1 \leq i \leq t$, og la $Q_* \subseteq \mathcal{O}_{G \setminus \{g_t\}}$ være en grunnmengde for $G \setminus \{g_t\}$. Vi definerer to mengder $\mathcal{O}^+$ og $\mathcal{O}^-$:*

$$\begin{aligned} \mathcal{O}^+ &= \{\textit{alle blokkeringer } (l, t, r; \lambda, j, \rho)\} \\ \mathcal{O}^- &= \{\sigma \in \mathcal{O}^+ \mid \textit{det finnes } \sigma' \in \mathcal{O}^+ \textit{ der } \sigma' \textit{ deler } \sigma\} \end{aligned}$$

Da er mengden

$$Q = Q_* \cup (\mathcal{O}^+ \setminus \mathcal{O}^-)$$

en grunnmengde for G.

Bevis. Det er klart at $Q \cup \mathcal{O}^+$ er en grunnmengde for $G \setminus \{g_t\}$. Elementene vi fjerner med $\mathcal{O}^-$ er reduntante i henhold til del (b) av definisjonen, og forrige korollar gir oss at $(Q \cup \mathcal{O}^+) \setminus \mathcal{O}^- = Q \cup (\mathcal{O}^+ \setminus \mathcal{O}^-)$ er en grunnmengde for $G \setminus \{g_t\}$. ♣

Merk. Det er verdt å merke seg at hver selv-overlapp vil telles to ganger (som $\sigma_1 = (l, t, r; \lambda, t, \rho)$ og som $\sigma_2 = (\lambda, t, \rho; l, t, r)$), men det holder å kreve at den ene reduserer til 0 (siden $O(\sigma_1) = -O(\sigma_2)$). Dette er inkludert i algoritmen vi snart gir, se neste avsnitt.

Nå kan vi, som i det kommutative tilfellet, lage en oppdateringsalgoritme og en ny versjon av Buchbergers algoritme, gitt i henholdsvis algoritme 6.2 og algoritme 6.3. Vi ser at en viktig forskjell mellom den ikke-kommutative Buchbergers og den kommutative, er initialiseringen av Q: Nå må vi ta hensyn til selv-overlapper. Linje 12–17 i Oppdater sørger for at hver selv-overlapp blir med bare én gang.

Deler av algoritmen er her gjengitt i en veldig «udetaljert» form, særlig gjelder dette delen som innebærer å finne alle nye blokkeringer (linje 11) i Oppdater. Mora har en noe mer utskrevet versjon [21, s. 16–17], men for en fullstendig implementasjon, se vedlegg C. Implementasjonen følger algoritmene i dette kapitlet så nært som mulig, men her er prosedyren for å ekskludere doble selv-overlapper plassert i underalgoritmen som finner nye blokkeringer (istedenfor i Oppdater).

Algoritme 6.2 Oppdater

input: $G = \{g_1, \ldots, g_{\delta-1}\}$, $Q_{gammel} \subseteq \mathcal{O}_G$, $g_\delta \notin G$
output: $Q \subseteq \mathcal{O}_{G \cup \{g_\delta\}}$
$Q := Q_{gammel}$
# tester om g_δ gjør tidligere elementer i Q redundante
for alle $\sigma = (l, i, r; \lambda, j, \rho) \in Q$ **do**
  if $1 \in \{l, r\}$ og $1 \in \{\lambda, \rho\}$ og $l\text{tip}(g_i)r = w_1\text{tip}(g_k)w_2$, $w_1 \neq 1 \neq w_2$ **then**
    $Q := Q \setminus \{\sigma\}$
  end if
end for
# lager alle nye blokkeringer
$\mathcal{O}^+ :=$ {alle blokkeringer $(l, \delta, r; \lambda, j, \rho)$}
# ta ut overflødige selv-overlapper fra $\mathcal{O}^+$
for alle $\sigma = (l, \delta, r; \lambda, \delta, \rho\} \in \mathcal{O}^+$ **do**
  if $\exists \sigma' = (\lambda, \delta, \rho; l, \delta, r) \in \mathcal{O}^+$ **then**
    $\mathcal{O}^+ := \mathcal{O}^+ \setminus \{\sigma'\}$
  end if
end for
# ta ut redundante elementer fra $\mathcal{O}^+$
for alle σ i $\mathcal{O}^+$ **do**
  if $\exists \sigma' \neq \sigma$ der σ' deler σ **then**
    $\mathcal{O}^+ := \mathcal{O}^+ \setminus \{\sigma\}$
  end if
end for
$Q := Q \cup \mathcal{O}^+$
return Q

Algoritme 6.3 Ikke-kommutativ Buchbergers algoritme, versjon 3

```
1: input: F = {f_1, ..., f_s}
2: output: G = {g_1, ..., g_t}
3: # initialiserer G og Q
4: G := {f_1}
5: Q := Oppdater(G, ∅, f_1)
6: # bygger Q fra F
7: for i := 2 to s do
8:    Q := Oppdater(G, Q, f_i)
9:    G := G ∪ {f_i}
10: end for
11: # vurderer et og et element i Q
12: δ := s + 1
13: while Q ≠ ∅ do
14:    velg σ ∈ Q
15:    Q := Q \ {σ}
16:    r := O(σ) modulo G
17:    if r ≠ 0 then
18:       f_δ := Ctip(r)^{-1} r   # vil ha f_δ monisk
19:       Q := Oppdater(G, Q, f_δ)
20:       G := G ∪ {f_δ}
21:       δ := δ + 1
22:    end if
23: end while
24: return G
```

6.3 Eksempel og sammenlikning med kommutativ

Vi gjentar eksempel 3.5.5, nå med den siste versjonen av Buchbergers algoritme. Vi har $F = \{f_1, f_2, f_3\}$ der

$$\begin{aligned} f_1 &= xyx - x \\ f_2 &= yx - x \\ f_3 &= xx - y \end{aligned}$$

og ordningen er lengde-leksikografisk i $k\langle x, y\rangle$ med $y > x$.

Vi starter med å initialisere G og Q (linje 3–5). Vi setter $G = \{f_1\}$ og kjører Oppdater($G, \emptyset, f_1$). Siden Q foreløpig er tom, går vi rett til å lage alle mulige blokkeringer (linje 11):

$$\begin{aligned} \mathcal{O}^+ := \{&\sigma_1 = (1, 1, yx; xy, 1, 1), \\ &\sigma_2 = (xy, 1, 1; 1, 1, yx)\} \end{aligned}$$

Siden σ_1 og σ_2 egentlig er samme overlapp, tar vi bort σ_2 fra $\mathcal{O}^+$ (linje 15 i Oppdater). Vi kan ikke gjøre mer, og går derfor videre til f_2 etter å ha satt $Q := \{\sigma_1\}$.

Vi har $\text{tip}(f_2) = yx$, og siden

$$1 \cdot \text{tip}(f_1) \cdot yx = xyxyx = x \cdot \text{tip}(f_2) \cdot yx$$

tar vi ut σ_1 fra Q (linje 7). Vi finner blokkeringer for f_2:

$$\begin{aligned}\mathcal{O}^+ := \{&\sigma_3 = (x, 2, 1; 1, 1, 1),\\ &\sigma_4 = (1, 2, yx; y, 1, 1)\}\end{aligned}$$

Ingen av disse deler hverandre, og Q er tom, og $(x, 1) \neq (1, yx)$. Vi setter dermed

$$Q := Q \cup \{\sigma_3, \sigma_4\} = \{\sigma_3, \sigma_4\}$$

og går til f_3. Vi har $\text{tip}(f_3) = xx$, som ikke deler $x \cdot \text{tip}(f_2)$ eller $\text{tip}(f_2) \cdot yx$. Blokkeringer for f_3 er:

$$\begin{aligned}\mathcal{O}^+ := \{&\sigma_5 = (1, 3, x; x, 3, 1),\\ &\sigma_6 = (x, 3, 1; 1, 3, x),\\ &\sigma_7 = (xy, 3, 1; 1, 1, x),\\ &\sigma_8 = (1, 3, yx; x, 1, 1),\\ &\sigma_9 = (y, 3, 1; 1, 2, x)\}\end{aligned}$$

Vi ser at σ_9 deler σ_7, så vi tar ut σ_7 fra $\mathcal{O}^+$. Videre er σ_5 og σ_6 like, så vi tar ut σ_6. Dette gir

$$Q := Q \cup \mathcal{O}^+ = \{\sigma_3, \sigma_4, \sigma_5, \sigma_8, \sigma_9\}$$

Nå begynner vi å plukke elementer fra Q (`while`-løkken i Buchbergers som starter i linje 13). Vi plukker først $\sigma_3 = (x, 2, 1; 1, 1, 1)$ og beregner overlappsrelasjonen:

$$O(\sigma_3) = x \cdot (yx - x) - (xyx - x) = -xx + x \xrightarrow{G} -y + x = r$$

som gir $f_4 := -1 \cdot r = y - x$. Vi kjører Oppdater($G, Q, f_4$) og ser at vi kan forkaste σ_4 og σ_8 fra Q, siden

$$\begin{aligned}yxyx &= yx \cdot y \cdot x \ \ (\sigma_4)\\ xxyx &= xx \cdot y \cdot x \ \ (\sigma_8)\end{aligned}$$

Vi beregner blokkeringer med f_4:

$$\begin{aligned}\mathcal{O}^+ := \{&\sigma_{10} = (x, 4, x; 1, 1, 1),\\ &\sigma_{11} = (1, 4, x; 1, 2, 1)\}\end{aligned}$$

Siden σ_{11} deler σ_{10}, forkaster vi σ_{10} fra $\mathcal{O}^+$. Vi har nå

$$Q := \{\sigma_5, \sigma_9, \sigma_{11}\}$$

Fra eksempel 3.5.5 vet vi at både $O(\sigma_5)$, $O(\sigma_9)$ og $O(\sigma_{11})$ reduserer til 0 modulo G, så Oppdater blir ikke kjørt flere ganger. Resultatet er en Gröbnerbasis $G = \{f_1, f_2, f_3, f_4\}$.

Da vi gjorde eksempelet forrige gang, beregnet vi samtlige 11 overlappsrelasjoner, hvorav to ble 0 direkte og de resterende ni ble redusert modulo G. Denne gangen beregnet vi bare fire overlappsrelasjoner (der alle ble redusert modulo G), en merkbar forbedring.

I kapittel 4.4 argumenterte vi for at operasjonene som ble utført av den kommutative Oppdater-algoritmen var «raske» i forhold til polynomoperasjonene. Er dette også tilfelle for den ikke-kommutative versjonen? Vi kan ikke lenger implementere monomer som vektorer, og å sammenlikne to strenger («sammenlikne» betyr her å se om den ene deler den andre) er litt tyngre enn å sammenlikne vektorer – se [18, kapittel 4] for mer om hvordan dette kan gjøres effektivt. Men når vi reduserer et polynom modulo G, må vi uansett gjøre mange slike sammenlikninger – å få ned antall overlappsrelasjoner er tilsynelatende besparende.

Polynomoperasjoner innebærer også at polynomer må sorteres (etter valgt ordning), og dette er også en «tung» operasjon (merk at dette er avhengig av hva slags sorteringsalgoritme som benyttes, og det er mulig å lage spesielle implementasjoner dersom polynomene ikke er spredte, men tette).

Selv om $\mathcal{O}^+$-mengdene kan være større enn de kommutative $\mathcal{S}^+$-mengdene, og dessuten er mer krevende å lage, er de alltid endelige. Vi hadde

$$|\mathcal{S}^+| = \binom{t}{2}$$
$$\text{mens nå er } |\mathcal{O}_G| \leq (\binom{t}{2} + 1) \cdot \alpha^2 = \kappa$$

der α er lengden på den største tuppen i G. Merk at κ vanligvis er mye større enn $|\mathcal{O}_G|$, men antall operasjoner som utføres for å finne $\mathcal{O}_G$, er nøyaktig κ. Når G har $\delta - 1$ elementer og vi skal finne $\mathcal{O}^+$, utfører vi maksimalt $\delta \cdot \alpha^2$ operasjoner (færre dersom noen tupper er kortere enn α), og dersom α ikke er alt for stor, er ikke det så mye mere enn de δ operajonene som ga $\mathcal{S}^+$.

6.4 Tuppredusering eller midt-overlapper?

Vi definerer en **midt-overlapp** som en blokkering på formen $\sigma = (1, i, 1; \lambda, j, \rho)$ eller $\sigma = (l, i, r; 1, j, 1)$ der henholdsvis verken λ eller ρ, eller l eller r, er 1.

I enkelte tekster, som [16] og [23], benyttes ikke midt-overlapper i det hele tatt, og man kan vise at teorem 3.5.3 da holder for en *tuppredusert* mengde G. Dersom man modifiserer Buchbergers algoritme slik at den foreløpige Gröbnerbasisen hele tiden er tuppredusert, kan man altså se bort fra midtoverlapper. At en tuppredusert Gröbnerbasis er nødvendig for at dette skal være mulig, illustreres av det følgende eksempelet.

Eksempel 6.4.1. Vi ser på idealet $I = \langle f_1, f_2 \rangle$ i ringen $R = k\langle x, y, z, v \rangle$ med lengde-leksikografisk ordning, $x > y > z > v$ og der

$$f_1 = yxyz - y$$
$$f_2 = xyzz - v$$

Det finnes bare én overlappsrelasjon mellom $\text{tip}(f_1)$ og $\text{tip}(f_2)$, nemlig

$$O(y \cdot f_2, f_1 \cdot z) = yxyzz - yv - yxyzz + yz = yz - yv$$

Dette gir $f_3 = yz - yv$, som gir oss én ny overlappsrelasjon, dersom vi ikke tillater midt-overlapper:

$$O(yx \cdot f_3, f_1) = yxyz - yxyv - yxyz + y = -yxyv + y$$

Vi får $f_4 = yxyv - y$, som ikke overlapper med noen av de foregående f_i'ene. Kan vi da konkludere med at $F = \{f_1, f_2, f_3, f_4\}$ er en Gröbnerbasis for I? Vi lar $p = zxyzzy$, et element i R, og prøver å finne normalformen av p i R/I.

$$p \xrightarrow{f_2} zvy = p'$$

men vi har også

$$p \xrightarrow{f_3} zxyvzy = p''$$

Verken p' eller p'' kan reduseres videre, og $p' \neq p''$. Altså er ikke F en Gröbnerbasis. Vi beregner den eneste midt-overlappen i F:

$$O(x \cdot f_3 \cdot z, f_2) = xyzz - xyvz - xyzz + v = -xyvz + v$$

Vi legger til $f_5 = xyvz - v$ til F, og ser at f_5 ikke overlapper med tidligere f_i'er. Teorem 3.5.3 gir at F er en Gröbnerbasis, og vi har

$$p'' \xrightarrow{f_5} zvy = p'$$

Eksempelet viste at vi kan ikke utelukke midt-overlapper, selv ikke når vi starter med en tuppredusert mengde, som $\{f_1, f_2\}$ over jo var. Men en tuppredusert mengde har ingen midt-overlapper. Faktisk er det å beregne en midt-overlapp nært beslektet med å redusere et element med det overlappende: La g_1 og g_2 være to polynomer hvis tupper har en midt-overlapp med overlappsrelasjon

$$g_2 - lg_1r$$

Skal vi redusere g_2 med g_1, vil vi gjøre akkurat det samme! Så dersom vi modifiserer algoritmen vår slik at G hele tiden er en tuppredusert mengde, kan vi la være å lete etter midt-overlapper, siden det ikke vil være noen (alle erstattes av en reduksjon som vist over). Det kan virke som om det er vilkårlig om vi velger den ene eller den andre strategien, men fullt så enkelt er det ikke.

Å lage en redusert Gröbnerbasis vil meget mulig gi oss en *mindre* Gröbnerbasis, noe vi kanskje kan spare tid på. Videre kan det tenkes at der den vanlige algoritmen ville produsert stadig nye elementer i en uendelig, ikke tuppredusert Gröbnerbasis, vil en modifisert algoritme kanskje finne en endelig, tuppredusert Gröbnerbasis. Dette er et godt argument for å tuppredusere underveis.

Men når vi tuppreduserer alle de tidligere elementene i F med det nye elementet f_δ, får vi «nye» polynomer som kan

1. tuppredusere hverandre

2. gi opphav til nye blokkeringer

mens de tidligere blokkeringene kanskje er utdaterte, for eksempel hvis vi har blokkeringen $(l, i, r; \lambda, j, \rho)$ i Q og f_i reduserer til 0 med f_δ. Altså må vi oppdatere Q hver gang vi reduserer et element i G. Dessuten, dersom f_i lar seg redusere modulo $F \setminus \{f_i\}$ etter å ha lagt til f_δ til F, og vi deretter reduserer f_j modulo $F \setminus \{f_j\}$, kan det være at den «nye» f_i kan redusere den «nye» f_j – og så videre. Dette er vanskelig å forutsi (selv om det ikke er en uendelig prosess, siden $\prec$ er en velordning), og kan fort bli veldig tid- og ressurskrevende.

GAP-pakken *GBNP* (se http://www.win.tue.nl/ amc/pub/grobner/doc.html) bygger på en versjon av (ikke-kommutativ) Buchbergers algoritme som er beskrevet hos Cohen [6]. Selv om Cohen i de innledende resultatene bruker reduserte Gröbnerbasiser, skifter han til å heller bruke midt-overlapper i selve algoritmen, nettopp for å unngå den rekursive prosessen beskrevet over.

Et annet system som bruker tuppredusering er Opal, et program som ikke lenger er tilgjengelig, men som bygger på teorien presentert av Keller i hans doktorgradsavhandling fra 1996 [18]. Keller introduserer to alternative metoder for å holde G tuppredusert. Anta at det nye elementet h som skal legges til G er slik at $\mathrm{tip}(h)$ deler $\mathrm{tip}(g)$ der g er et tidligere element i G. Da kan de to metodene oppsummeres slik:

1. Vi forkaster g fra G, men beholder alle blokkeringer der g er involvert.
2. Vi sletter alle blokkeringer som inneholder g, reduserer g med h og finner nye blokkeringer for den reduserte g

Fra Kellers testresultater kan det synes som om den andre strategien fungerer best (i alle fall for det ene eksempelet det er testet ut på).

6.5 Videre modifikasjoner

I kapittel 6.2 så vi på en ikke-kommutativ analog av Gebauer-Möller-strategiene. Vi husker fra kapittel 4.3 at disse baserte seg på at det lønner seg å velge minimale elementer i Q når algoritmen sier «velg $(i, j) \in Q$», og minimale elementer i Q betyr i det ikke-kommutative tilfellet en blokkering $(l, i, r; \lambda, j, \rho)$ i Q der $l\mathrm{tip}(f_i)r$ er minimal.

I det kommuative tilfellet hadde vi Traverso og Donatis eksperimenter [26] å ta utgangspunkt i, både for valgstrategien over og for andre spørsmål. Tilsvarende eksperimenter for ikke-kommutative Gröbnerbasiser ble utført av Keller og publisert i hans før nevnte doktoravhandling [18]. I tillegg til eksperimentering rundt tuppreduserte Gröbnerbasiser, som vi allerede har gjengitt, tar Keller for seg valgstrategier og håndtering av overflødige blokkeringer. Han har også eksperimentert med forskjellige tillatelige ordninger, men det skal vi ikke gå nærmere inn på her.

Når det gjelder valgstrategi (hvilken blokkering skal vi velge fra Q?) har Keller sett på to alternativer:

1. Normal valgstrategi: Gitt en tillatelig ordning $\prec$ som brukes i beregningene, velg den blokkeringen $(l, i, r; \lambda, j, \rho)$ der $l\mathrm{tip}(g_i)r$ er minimal med hensyn til $\prec$.

2. Kortest-strategi: Velg $(l, i, r; \lambda, j, \rho)$ slik at $l\text{tip}(g_i)r$ har kortest *lengde*.

Generelt ga den andre strategien bedre resultat enn den første.

Håndtering av overflødige (redundante) blokkeringer kan også gjøres på to måter, analogt med kommutativ der vi tok for oss to slike strategier – Buchbergers 2. kriterium og Gebauer-Möller. I det ikke-kommutative tilfellet har vi bare sett på sistnevnte alternativ, men det er også dette som fungerer best ifølge Kellers eksperimenter. Et tredje alternativ er en hybridløsning som kombinerer begge disse mulighetene, men Keller fant i flere av eksperimentene ingen signifikant forskjell mellom denne løsningen og «ren» Gebauer-Möller, mens den i andre tilfeller var litt bedre [18, s. 59–62].

Å beregne kompleksiteten av den ikke-kommutative versjonen av Buchbergers algoritme er ikke lettere enn for den kommutative, og dessuten har vi som kjent problemet med at noen idelaer kan ha uendelig Gröbnerbasis for noen (eller alle) tillatelige ordninger.

Avslutningsvis i dette kapitlet tar vi for oss F_4-algoritmen og hvordan den kan tilpasses en ikke-kommutativ setting. Dersom man modifiserer konstr-A-algoritmen (algoritme 4.7) til å se på ikke-kommutative monomer, vil man få en A-matrise med samme egenskaper som i det kommutative tilfellet. Siden den eneste forskjellen på den kommutative og den ikke-kommutative divisjonsalgoritmen er tosidig divisjon, og dette tas hånd om av modifikasjonen, vil både lemma 4.6.6 og teorem 4.6.8 kunne overføres til å gjelde idealer i $k\langle x_1, \ldots, x_n\rangle$.

Vi gir her den modifiserte utgaven av Konstr-A (algoritme 6.4), og et eksempel på hvordan den fungerer. Reduksjon-algoritmen vil være lik som før, mens den ikke-kommutative F_4 er gitt i algoritme 6.5.

Algoritme 6.4 Konstr-A

input: $F = \{f_1, \ldots, f_l\}$ som skal reduseres modulo G, og $G = \{g_1, \ldots, g_s\}$
output: A, en matrise
$T(F) := \{p \in \mathcal{B} | p$ er et monom i et polynom i $F\}$
$B := F$
while $T(F) \neq \emptyset$ **do**
  $m := \max_{\prec} T(F)$ # m er maksimalt med hensyn til $\prec$
  $T(F) := T(F) \setminus \{m\}$
  if $m = u \cdot \text{tip}(g_i) \cdot v$ for $g_i \in G$ **then**
    $B := B \cup \{ug_iv\}$
    $T(F) := T(F) \cup \{$alle monomer i ug_iv unntatt $\text{tip}(ug_iv)\}$
  end if
end while
$A :=$matrise(B)
return A

Eksempel 6.5.1. Vi ser på polynomene

$$\begin{aligned} f &= yxzx + xzy + yxz + x \\ g_1 &= xz - y \\ g_2 &= yx - z \end{aligned}$$

i ringen $\mathbb{Z}_5\langle x, y, z\rangle$, lengde-leksikografisk ordning med $x > y > z$. Vi ønsker å redusere f med $G = \{g_1, g_2\}$. Dersom vi reduserer på «gamlemåten», får vi med gitt rekkefølge på G at

$$f \xrightarrow{G} 2yy + yz + x = f'$$

Merk at vi også kan ha $f \xrightarrow{G} yy + yz + zz + x = f''$, dersom vi bytter rekkefølge på g_1 og g_2 i reduksjonen. Vi skal se at Konstr-A gir en matrise A slik at f' eller f'' kan lese av den reduserte trappeformen til A.

Vi initialiserer $T(F)$ og B, og får

$$T(F) = \{yxzx, xzy, yxz, x\}$$
$$B = \{f\}$$

Deretter kjører vi `while`-løkken til $T(F)$ er tom.

(1) $m = yxzx$
Vi har $yxzx = y\text{tip}(g_1)x$, så vi legger til $yg_1x = yxzx - yyx$ til B og yyx til $T(F)$. Nå er $T(F) = \{xzy, yxz, yyx, x\}$.

(2) $m = xzy$
Vi har $xzy = tipg_1y$, så vi legger til $g_1y = xzy - yy$ til B og yy til $T(F)$. Nå er $T(F) = \{yxz, yyx, yy, x\}$.

(3) $m = yxz$
Vi har $yxz = ytipg_1$, så vi legger til $yg_1 = yxz - yy$ til B, men yy er allerede i $T(F)$, så ingenting nytt legges til her.

(4) $m = yyx$
Vi har $yyx = y\text{tip}(g_2)$, så vi legger til $yg_2 = yyx - yz$ til B, og yz til $T(F)$. Nå er $T(F) = \{yy, yz, x\}$.

(5) $m = yy$
Lar seg ikke dele med $\text{tip}(g_1)$ eller $\text{tip}(g_2)$.

(6) $m = yz$
Samme som over.

(7) $m = x$
Samme som over, og nå er $T(F)$ tom.

B består av fem polynomer, og vi bygger A slik at hver kolonne svarer til et monom i $T(B)$, i synkende rekkefølge.

$$T(B) = \{yxzx, xzy, yxz, yyx, yy, yz, x\}$$
$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

Vi finner den reduserte trappeformen av A:

$$\tilde{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

Vi ser at $\tilde{A}^+ = \{f \in \tilde{A} | \text{tip}(f) \notin \text{Tip}(A)\} = (0,0,0,0,2,1,1)$, som svarer til polynomet

$$f' = 2yy + yz + x$$

som var det vi ville fram til. Merk at siden det er tyngre å regne med strenger enn med vektorer, sparer vi mer tid her enn i det ikke-kommutative tilfellet på å bruke lineæralgebra framfor divisjonsalgoritmen – i alle fall dersom vi har gode reduksjonsalgoritmer for spredte matriser til rådighet.

Algoritme 6.5 Ikke-kommutativ F_4

input: $F = \{f_1, \ldots, f_s\}$
output: $G = \{g_1, \ldots, g_t\}$
initialiserer G og Q
$G := \{f_1\}$
$Q :=$ Oppdater$(G, \emptyset, f_1)$
# bygger Q fra F
for $i := 2$ **to** s **do**
    $Q :=$ Oppdater(G, Q, f_i)
    $G := G \cup \{f_i\}$
end for
# vurderer et og et element i Q
$\delta := s + 1$
$d := 0$
while $Q \neq \emptyset$ **do**
    $d := d + 1$
    velg $\emptyset \neq Q_d \subseteq Q$
    $Q := Q \setminus Q_d$
    # lager overlappsrelasjoner av blokkeringene vi valgte
    $O := \emptyset$
    for $\sigma \in Q_d$ **do**
        $O := O \cup \{O(\sigma)\}$
    end for
    $\tilde{A}^+ :=$Reduksjon(O, G)
    for $h \in \tilde{A}^+$ **do**
        $f_\delta := h$
        $Q :=$ Oppdater(G, Q, f_δ)
        $G := G \cup \{f_\delta\}$
        $\delta := \delta + 1$
    end for
end while
return G

Del III
Polly Cracker-kryptografi

«(...) Du kan for eksempel bytte om to bokstaver, du kan skrive et ord baklengs, sette bokstavene i gal rekkefølge, eller bare en og to, og så begynne på begynnelsen av ordet igjen, du kan, som i dette tilfellet, bytte ut bokstavene med zodiakale tegn, mens de virkelige bokstavene de skal forestille får en annen nummerorden, og du kan bytte om nummerrekken ved hjelp av et annet alfabet – .»

«Hvilket av disse systemene har Venanzio brukt?»

«Vi er nødt til å prøve dem alle, og enda fler. Men den første regel for dechiffrering av et budskap er denne: du må forsøke å gjette hva det er budskapet vil si.»

fra *Rosens navn* av Umberto Eco

7 Kommutativ Polly Cracker

7.1 Innledning

Polly Cracker er et offentlig nøkkel-kryptosystem, introdusert i 1994 av Fellows og Koblitz [11]. Ideen deres var å bruke beregningsmessig vanskelige kombinatoriske problemer (se kapittel 1.2) som grunnlag for et kryptosystem, med den hensikt å gjøre det like vanskelig for en kryptanalyst å finne den hemmelige nøkkelen, som det er å finne løsningen på det kombinatoriske problemet den er bygd på.

Omtrent samtidig som Fellows og Koblitz konstruerte Polly Cracker (deres opprinnelige versjon er den som i denne teksten er omtalt som «spesialtilfelle», og dette kan beskrives også uten bruk av Gröbnerbasiser), skrev Boo Barkee et al. en artikkel om et system som i oppsett er identisk med denne tekstens «generelle» Polly Cracker [1]. Derfor går Polly Cracker-systemer basert på Gröbnerbasiser også under navnet «Barkee-systemer». Koblitz generaliserte også Polly Cracker-systemene ved hjelp av Gröbnerbasiser [19], og selv om oppsettene til Koblitz og Barkee i dette tilfellet kan synes identiske, er det likevel en sentral forskjell på dem. Mer om dette i avsnitt 7.2.2.

Polly Cracker-systemer er fortsatt på forskningsstadiet, og de er ikke implementert. Blant de åpenbare ulempene med systemene er at korte klartekster blir lange chiffertekster, og at vi mangler en algoritme for å lage vanskelige kombinatoriske problemer med en kjent løsning. Likevel er Polly Cracker av matematisk interesse, særlig når man utvider konseptet fra polynomringer over endelige kropper til ikke-kommutative frie algebraer, noe vi skal komme tilbake til i de påfølgende kapitlene.

Dette kapitlet tar for seg det formelle oppsettet av kommutativ Polly Cracker, samt sikkerhet og svakheter ved systemet. Vi ser på før nevnte spesialtilfelle av Polly Cracker, og et konkret eksempel på hvordan vi kan bruke et kjent $\mathcal{NP}$-komplett problem (ikke et av de tre problemene Koblitz demonstrerer i [19]) til å gjøre den offentlige nøkkelen «innbruddssikker».

7.2 Oppsett

Arne og Bjarne skal kommunisere som vist på figur 1.3. Katla, kryptanalysten, får tak i Bjarnes krypterte meldinger, og gjør alt hun kan for å knekke dem.

La $\mathbb{F}$ være en endelig kropp og T en endelig variabelmengde. La $I \subseteq \mathbb{F}[T]$ være et ideal.

Hemmelig nøkkel: G, en Gröbnerbasis for I.
Offentlig nøkkel: $B = \{q_i\}_{i=1}^{s}$, der q_i er i I. La $J = \langle B \rangle$, et ideal i $\mathbb{F}[T]$ inneholdt i I.
Meldingsrom: $M \subseteq \{m \in \mathbb{F}[T] | m \xrightarrow{G} m\}$. M må være endelig.

Chiffertekstrom: $C \subseteq \mathbb{F}[T]$.

Kryptering: $c = p + m$, der $p = \sum_{j=1}^{s} h_j q_j$, et element i J (Vi velger h_j 'ene fra $\mathbb{F}[T]$).

Dekryptering: redusere c modulo G.

Merk. Vi gjør følgende observasjoner:

1. Vi vet at alle idealer I har en endelig Gröbnerbasis (teorem 2.4.4).
2. Meldingsrommet M må, av opplagte grunner, være kjent. Elementene i

$$N = \{m \in \mathbb{F}[T] | m \xrightarrow{G} m\}$$

   utgjør en representerende mengde for faktorringen $\mathbb{F}/I$ (det vil si at for $m \neq m'$ i N, vil $m + I \neq m' + I$. Dette holder siden $a + I = a' + I$ hvis og bare hvis $\overline{a}^G = \overline{a'}^G$. Videre kan alle $a + I$ skrives som $m + I$ med m i N, siden $a \xrightarrow{G} m$ der m er i N). Man kan spørre seg om kjennskap til N gjør at det er mulig å gjette I, og dermed kunne finne G. Dette skal vi komme tilbake til senere, men merk at vi kan velge M på en slik måte at vi røper lite om N.
3. Krypteringsalgoritmen er ikke deterministisk. Dersom vi krypterer samme melding to ganger, vil vi ikke få den samme chifferteksten. Dette er en fordel, særlig dersom de aktuelle meldingene er få – f.eks. {ja, nei}.
4. Dekrypteringen fungerer siden p er i J, og dermed i I, så $p \xrightarrow{G} 0$. Vi får altså $c \xrightarrow{G} m$.

7.2.1 Spesialtilfelle

La $T = \{t_1, \ldots, t_n\}$, og la $\overline{y}$ være i $\mathbb{F}^n$, der $\overline{y} = (y_1, \ldots, y_n)$. Da er $G = \{t_1 - y_1, \ldots, t_n - y_n\}$ en Gröbnerbasis for idealet $I = \langle G \rangle$. Den hemmelige nøkkelen blir rett og slett $\overline{y}$, og siden elementene i I er alle polynomer i $\mathbb{F}[T]$ som blir 0 evaluert i $\overline{y}$, er dekrypteringen nå gitt av $c(\overline{y}) = m$. Polynomene i $\mathbb{F}[T]$ som ikke lar seg redusere modulo G er elementene i $\mathbb{F}$, altså de konstante polynomene i $\mathbb{F}[T]$, så vi får $M = \mathbb{F}$.

Merk. Spesiell Polly Cracker over $\mathbb{F}_2$ gir meldingsrommet $\{0, 1\}$. Dette gjør anvendelsene noe upraktiske.

La oss se på problemet med hvorvidt kjennskap til meldingsrommet kan avsløre I, ved hjelp av et eksempel.

Eksempel 7.2.1. Katla kjenner ikke den hemmelige vektoren $\overline{y}$, men vet at meldingsrommet er $M = \mathbb{F}$, og at dette er alle polynomer som ikke kan reduseres med den hemmelige Gröbnerbasisen.

Imidlertid vil enhver vektor $\overline{z}$ i $\mathbb{F}^n$ gi opphav til et ideal $I_{\overline{z}} = \langle \{t_i - z_i | i = 1, \ldots, n\} \rangle$. Disse idealene er ikke nødvendigvis like, selv om de deler mengden av polynomer i $\mathbb{F}[T]$ som ikke lar seg redusere.

Så, selv om Katla lett finner en Gröbnerbasis $G' = \{t_1 - z_1, \ldots, t_n - z_n\}$ slik at $m \xrightarrow{G'} m$, vil ikke dette medføre at $c = p + m \xrightarrow{G'} m$, siden vi ikke har noen grunn til å tro at p er i $\langle G' \rangle$ og at p dermed reduserer til 0 modulo G'.

7.2.2 Tett eller spredt?

Fellows og Koblitz argumenterte for et *spredt* system, det vil si at polynmene som brukes har *få* koeffisienter ulik null. Dette for å øke kompleksiteten på et angrep kalt *lineæralgebraangrep* (se avsnitt 7.4.1). Barkee et al. tok utgangspunkt i et *tett* system da de viste at dette var usikkert mot lineæralgebraangrep og delvis Gröbnerbasis-angrep (avsnitt 7.4.4), og avsluttet artikkelen sin med følgende utfordring:

> (...) leseren kan tenke seg (kanskje med en viss grunn) at et spredt system kan fungere. Vi tror (kanskje ubegrunnet) at spredthet vil gjøre systemet enklere å knekke. [1, s. 5 – min oversettelse]

I en artikkel fra 2008, fjorten år etter at ufordringen over ble gitt, viste Caboara et al. at spredte Polly Cracker-systemer er sårbare for et «spesialsydd» lineæralgebraangrep [5] (se mer om dette angrepet i avsnitt 7.4.3).

7.3 Konstruksjon av B

B er den offentlige nøkkelen, og det er en sammenheng mellom denne og den hemmelige nøkkelen. Det er viktig å konstruere B slik at ingen informasjon om den hemmelige nøkkelen lekker ut, og at selve B på ingen måte kan brukes til å dekryptere meldinger.

7.3.1 For spesialtilfellet

La oss først se på spesialtilfellet av Polly Cracker, beskrevet i avsnitt 7.2. Vi har altså en hemmelig nøkkel, $\overline{y}$, og en offentlig nøkkel, B, der alle q_j i B har det til felles at $q_j(\overline{y}) = 0$. Dersom Katla klarer å finne et punkt $\overline{z}$ i $\mathbb{F}^n$ der $\langle B \rangle$ forsvinner (altså et felles nullpunkt for alle q_j i B), kan hun dekryptere alle mulige meldinger, siden $c(\overline{z}) = p(\overline{z}) + m(\overline{z}) = 0 + m = m$. Vi må altså konstruere B slik at det er *vanskelig* å finne et slikt punkt.

Det er her Koblitz' og Fellows opprinnelige idé kommer inn. La *Felles nullpunkt* være det følgende problemet: Gitt en endelig mengde polynomer B i $\mathbb{F}[T]$, finnes det et punkt $\overline{y}$ i $\mathbb{F}^{|[T]|}$ slik at $q(\overline{y}) = 0$ for alle q i B? Koblitz gir følgende teorem [19]:

Teorem 7.3.1. *La L være et $\mathcal{NP}$-komplett søkeproblem. Da finnes en polynomisk tid-reduksjon τ fra L til problemet Felles nullpunkt i $\mathbb{F}[T]$, slik at dersom ρ er en instans av L og x en mulig løsning, har vi*

$$x \text{ er en løsning av } \rho \iff \tau(x) \text{ er et felles nullpunkt for polynomene i } \tau(\rho)$$

Merk. Dette teoremet høres kanskje rart og lite intuitivt ut – Koblitz gir heller ikke noe fullt bevis i [19], men en skisse av beviset er gitt i [11] – men dersom man har kjennskap til Cook-Levin-teoremet, som sier at *SAT* er $\mathcal{NP}$-komplett [24], virker det ikke så merkelig. En instans av *SAT* kan lett reduseres til polynomer over $\mathbb{F}_2$, og siden *SAT* er $\mathcal{NP}$-komplett, kan alle problemer i klassen $\mathcal{NP}$ reduseres til *SAT*.

Korollar 7.3.2. *La ρ være en instans av et $\mathcal{NP}$-komplett søkeproblemet som vi kjenner en løsning av, og la B være den tilhørende mengden polynomer gitt av reduksjonen fra teoremet over. En løsning av ρ reduserer da til $\overline{y}$ i $\mathbb{F}^n$, slik at $q(\overline{y}) = 0$ for alle q i B. Dersom vi setter $\overline{y}$ som hemmelig nøkkel og B som offentlig nøkkel i spesialtilfellet av Polly Cracker, vil det være uoverkommelig for en kryptanalyst å finne et punkt der $\langle B \rangle$ forsvinner i polynomisk tid.*

Bevis. Anta at vi har en algoritme med polynomisk kjøretid som finner et nullpunkt $\overline{z}$ i $\mathbb{F}^n$ slik at $q_j(\overline{z}) = 0$ for alle q i B. Da vil $\overline{z}$ svare til en løsning av ρ, og vi har dermed en polynomisk tid-algoritme for å løse et $\mathcal{NP}$-komplett problem! Forutsatt at antakelsen $\mathcal{P} \neq \mathcal{NP}$ holder, er dette umulig. ♣

Eksempel 7.3.3. Et Polly Cracker-system bygd på det $\mathcal{NP}$-komplette problemet *Clique* (reduksjonen er delvis basert på en reduksjon *Clique* $\rightarrow$ *SAT* hentet fra [12]).

Clique: Gitt en urettet graf $\mathfrak{G} = (V, E)$, finnes det en delmengde $C \subseteq V$ med $|C| = K$ slik at for alle par (i, j) i $C \times C$ der $i \neq j$, er (i, j) i E (altså er C en delmengde av hjørnene som alle har en kant til hverandre)?

Reduksjon: Sett $|V| = N$. Vi lar $B \subseteq \mathbb{F}_2[T]$, der $T = \{t_{i,r} | 1 \leq i \leq N, 1 \leq r \leq K\}$. Vi setter $t_{i,r} = 1$ dersom hjørne i er det r'te hjørnet i C, og 0 ellers. La $B = B_1 \cup B_2 \cup B_3 \cup B_4$, der

$$
\begin{aligned}
B_1 &= \{t_{1,r} + t_{2,r} + \ldots - t_{N,r} + 1 \mid 1 \leq r \leq K\} \\
B_2 &= \{t_{i,r} \cdot t_{i,s} \mid 1 \leq i \leq N,\ 1 \leq r < s \leq K\} \\
B_3 &= \{t_{i,r} \cdot t_{j,r} \mid 1 \leq i < j \leq N,\ 1 \leq r \leq K\} \\
B_4 &= \{t_{i,r} \cdot t_{j,s} \mid 1 \leq i < j \leq N,\ (i,j) \in \overline{E},\ 1 \leq r,\ s \leq K,\ r \neq s\}
\end{aligned}
$$

Vi ser følgende:

$B_1 = 0 \iff$ minst et hjørne er hjørne nr. r i C
$B_2 = 0 \iff$ ingen hjørner er både hjørne r og s i C
$B_3 = 0 \iff$ maks ett hjørne er hjørne nr. r
$B_4 = 0 \iff$ dersom hjørnene i og j ikke har en kant mellom seg i $\mathfrak{G}$, kan de ikke være i C

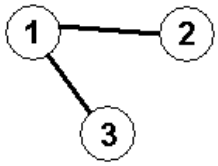
Altså er en løsning av *Clique*-instansen også en løsning av $\langle B \rangle = 0$, og omvendt. Da har vi oppnådd det vi ville, og vi setter B til offentlig nøkkel og en løsning $\overline{y}$ av *Clique* til hemmelig nøkkel.

Merk. Observasjoner gjort for dette spesifikke eksempelet:

1. Vi vet ikke hvordan dette gjøres i praksis. Det er trivielt å konstruere en graf med en «klikk» i, men vi har ingen algoritme for å konstruere slike grafer der det også er ikke-trivielt å *finne* en vilkårlig klikk.

2. En annen klikk enn vår hemmelige nøkkel gir en annen Gröbnerbasis for et annet ideal, med kan like fullt brukes til dekryptering siden den gir et nullpunkt i $\langle B \rangle$. Altså trenger vi en graf der det er vanskelig å finne *noen* klikk i.

Eksempel 7.3.4. Vi skal se på en konkret instans av *Clique*. Vi har følgende graf $\mathfrak{G} = (V, E)$:



Vi studerer klikker av størrelse 2. Vi trenger en klikk til å være hemmelig nøkkel, la oss velge klikken $(1, 2)$. Vi får seks variabler $t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}, t_{3,1}, t_{3,2}$. Den hemmelige vektoren blir da $\overline{y} = (1, 0, 0, 1, 0, 0)$. Likningene som utgjør den offentlige nøkkelen er:

$$\begin{aligned}
B_1 &= \{t_{1,1} + t_{2,1} + t_{3,1} + 1,\ t_{1,2} + t_{2,2} + t_{3,2} + 1\} \\
B_2 &= \{t_{1,1}t_{1,2},\ t_{2,1}t_{2,2},\ t_{3,1}t_{3,2}\} \\
B_3 &= \{t_{1,1}t_{2,1},\ t_{1,1}t_{3,1},\ t_{2,1}t_{3,1},\ t_{1,2}t_{2,2},\ t_{1,2}t_{3,2},\ t_{2,2}t_{3,2}\} \\
B_4 &= \{t_{2,1}t_{3,2},\ t_{2,2}t_{3,1}\}
\end{aligned}$$

Vi ser at likningene B_3 i dette tilfellet er reduntante, så vi setter $B = B_1 \cup B_2 \cup B_4$. Merk at dette er et lite eksempel i en liten graf, og at det ikke er noe problem å løse dette systemet – det finnes bare 2^6 mulige variabeltilordninger. Men systemet er likevel ganske stort i forhold til den lille grafen, så man kan se for seg at i et større og mer realistisk eksempel, blir det svært mange likninger.

7.3.2 For det generelle tilfellet

Nå vender vi tilbake til det generelle tilfellet av Polly Cracker, der den hemmelige nøkkelen er en Gröbnerbasis. En fordel med dette framfor spesialtilfellet er at meldingsrommet blir større. Men hvordan skal vi her sikre B? Og hva skal vi sikre den *mot*?

Lemma 7.3.5. *Vi har et Polly Cracker-system som beskrevet innledningsvis i avsnitt 7.2. Dersom en kryptanalyst har gitt en Gröbnerbasis G' for $J = \langle B \rangle \subseteq I = \langle G \rangle$, og $c = p + m$ er en kryptert melding med p i J og m i M, kan hun dekryptere c siden $c \xrightarrow{G'} m$.*

Bevis. Vi har $p \xrightarrow{G'} 0$, siden p er i J. Anta $c \xrightarrow{G'} m'$. Da vil enten $m = m'$, ellers så har vi klart å redusere m' med G'. La $G = \{g_1, \ldots, g_t\}$ og $G' = \{g'_1, \ldots, g'_{t'}\}$. Siden $G' \subseteq I$, har vi, for alle $1 \leq i \leq t'$, $LT(g'_i) = h \cdot LT(g_j)$ for et monom h i $\mathbb{F}[T]$ og en $1 \leq j \leq t$.

Hvis vi har redusert m med G', har vi, for et monom f i $\mathbb{F}[T]$ og en $1 \leq i \leq t'$:

$$LT(m) = f \cdot LT(g'_i) = f \cdot h \cdot LT(g_j)$$

og da kan m reduseres modulo G' også. Dette er en motsigelse, og vi har $m = m'$. ♣

Dermed vet vi hva vi må passe på, nemlig å konstruere B slik at det ikke er mulig å finne en Gröbnerbasis for $\langle B \rangle$ i polynomisk tid. Før vi går videre, la oss se på et av Koblitz' eksempler på et generelt Polly Cracker-system [19]:

Eksempel 7.3.6. La $T = \{t_1, \ldots, t_n\}$, og se på polynomringen $\mathbb{F}[T]$ med gradert leksikografisk monomordning. La $\overline{y}$ i $\mathbb{F}^n$ være en hemmelig vektor. Velg et positivt heltall $d \geq 1$. Da danner alle polynomer på formen

$$\{\prod_{i=1}^{n}(t_i - y_i)^{\alpha_i} | \sum_{i=1}^{n} \alpha_i = d\}$$

en Gröbnerbasis G for idealet $I = \langle G \rangle$, der I er idealet i $\mathbb{F}[T]$ som består av alle polynomer av total grad minst d, og som blir 0 evaluert i $\overline{y}$. Vi har $LT(G) = \{t_1^{\alpha_1} \ldots t_n^{\alpha_n} | \sum_{i=1}^{n} \alpha_i = d\}$, altså alle monomer av grad d. Siden alle f i I er slik at $deg_{\text{tot}}(LT(f)) \geq d$, finnes alltid et monom i $LT(G)$ som deler $LT(f)$. Altså er G en Gröbnerbasis for I.

I eksempelet over har vi igjen en hemmelig vektor, $\overline{y}$, som vi kanskje kan sette til å være løsningen av et $\mathcal{NP}$-komplett problem. For å få $\langle B \rangle \subseteq I$ må vi passe på at alle polynomene i B har grad $\geq d$. En nyttig observasjon dersom vi jobber over $\mathbb{F}_2$, er at

$$t = 1 \iff t^2 = 1 \iff t^3 = 1 \iff \ldots$$

Slik kan vi for eksempel modifisere B gitt av *Clique* i eksempel 7.3.3, og få et større meldingsrom enn $\mathbb{F}_2$.

Men en sikker $\overline{y}$ hjelper ikke dersom Katla lett kan finne en Gröbnerbasis for $\langle B \rangle$. Imidlertid viser det seg at dette er umulig å oppnå i polynomisk tid: Dersom man lager en Gröbnerbasis for $\langle B \rangle$ (ved hjelp av polynomene i B), vil vi også kunne løse likningene i B (se kapittel 2.6). Ergo er det minst like vanskelig å lage en Gröbnerbasis for $\langle B \rangle$, som det er å løse det $\mathcal{NP}$-komplette problemet $\overline{y}$ er en løsning av. Vi konkluderer med at den hemmelige nøkkelen er sikret mot «innbrudd» via den offentlige nøkkelen.

Eksempel 7.3.7. Vi ser igjen på *Clique*-eksempelet fra forrige avsnitt. Sett $x_1 = t_{1,1}$, $x_2 = t_{1,2}$, $x_3 = t_{2,1}$, $x_4 = t_{2,2}$, $x_5 = t_{3,1}$ og $x_6 = t_{3,2}$. Vi ser at likningene i B er identiske med likningene i eksempel 2.6.4 fra kapittel 2.6. Den gangen fant vi en Gröbnerbasis for det som tilsvarer idealet $\langle B \rangle$ her, og brukte denne til å finne alle de mulige løsningene. Når vi sammenlikner med figur 7.3.4, ser vi at de fire løsningene samsvarer med de fire mulige klikkene i grafen: $(1,2), (2,1), (1,3)$ og $(3,1)$.

7.4 Angrep

Konklusjonen på slutten av forrige avnsitt virket lovende, men dessverre har Polly Cracker-systemet andre svakheter. Vi har tatt for oss konstruksjonen av B, den offentlige nøkkelen. Imidlertid gjør man også en annen konstruksjon når man bruker Polly Cracker, nemlig når man lager $p = \sum_{j=1}^{s} h_j q_j$ i krypteringen. Selv om det finnes alternativer for å forsøke å sikre denne også, er de ikke regnet som sikre, og disse alternativene blir kunstige og lite velfungerende i praksis.

Alle angrepene som presenteres her er såkalte *chiffertekstangrep* (se kapittel 1.3). De to *valgt chiffertekst-angrepene* som står omtalt i forbindelse med ikke-kommutativ Polly Cracker (se kapittel 8.4.3) kan også brukes her, med stort hell. De er utelatt herfra siden lineæralgebra-angrepene også er så effektive at Polly Cracker må regnes som svært usikkert selv uten at kryptanalysten har midlertidig tilgang til dekrypteringsprogrammet.

7.4.1 Lineæralgebraangrep

Lineæralgebraangrepene er like gamle som ideen om Polly Cracker, omtalt både av Fellows og Koblitz [11] og av Barkee [1]. Barkee viste at de er svært effektive på tette systemer, mens Koblitz og Fellows foreslo å bruke spredte systemer, noe som gir få kanselleringer i chiffer-teksten og dermed et stort system det er vanskelig å løse. Vi introduserer først det generelle lineæralgebraangrepet (for spesialtilfellet og det generelle tilfellet), før vi ser på de forbedrede og mer spesialtilpassede versjonene.

For spesialtilfellet over $\mathbb{F}_2$

Fellows og Koblitz [11] tar for seg en «banal» versjon av lineæralgebraangrepet, laget for spesialtilfellet av Polly Cracker over $\mathbb{F}_2$. Vi befinner oss nå i samme setting som i avsnitt 7.2.1. Katla har slått kloa i en kryptert melding, som hun vet er på formen $c = \sum_{j=1}^{s} h_j q_j + m$, og videre vet hun at m er i $\{0, 1\}$. Med andre ord finnes polynomer f_i i $\mathbb{F}_2[T]$ slik at vi har *enten* $\sum_{i=1}^{s} f_i q_i = c$, hvis $m = 0$, *eller* $\sum_{i=1}^{s} f_i q_i + 1 = c$ hvis $m = 1$. Alt Katla trenger å gjøre er å gjette den totale graden til f_i'ene, og deretter prøve å løse det første tilfellet som et vanlig lineært likningssystem med koeffisientene i f_i'ene som de ukjente.

Å gjette graden til koeffisientpolynomene kan gjøres slik: Vi vet den totale graden til q_j, og vi vet den totale graden til c. Vi gjetter at den totale graden til f_i er differansen mellom disse to, og skriver f_i som en k-lineærkombinasjon av alle monomer med total grad mindre eller lik resultatet.

Merk. Kan det skje at $c = \sum_{i=1}^{s} f_i q_i + 1 = \sum_{i=1}^{s} f_i' q_i = c'$? Nei, for da vil dekrypteringen være tvetydig. Vi har at $c(\overline{y}) = m$. Likningen over ville gitt $c(\overline{y}) = 1 = c'(\overline{y}) = 0$, en selvmotsigelse.

For generell Polly Cracker

Vi skal se hvordan Katla kan utvide lineæralgebraangrepet til å gjelde for generell Polly Cracker. Vi starter med å se på c, og sorterer ut alle leddene som er i M. Disse leddene samler vi i polynomet m'. Deretter setter vi, som i sted, $c = \sum_{i=1}^{s} f_i q_i + m$, og forsøker å finne f_i'er som løser systemet. Finner vi disse, finner vi m. Vi må først gjette graden på f_i'ene på samme måte som over. Vi får et lineært likningssystem der de ukjente er koeffisientene i f_i'ene. Men i motsetning til over, da vi kun hadde to alternativer for m, blir dette litt mer komplisert.

Vi deler likningssettet i to. Først samler vi opp leddene av $\sum_{i=1}^{s} f_i q_i$ som er i M, og setter dette

lik m''. Det første likningssettet vi ser på, er

$$\sum_{i=1}^{s} f_i q_i - m'' = c - m'$$

Fra dette finner vi forhåpentligvis flere av koeffisientene. Deretter bruker vi disse til å løse $m'' + m = m'$.

Eksempel 7.4.1. Gradert leksikografisk ordning, kryptering over $\mathbb{F}[x, y]$ der $x > y$ og $\mathbb{F}$ har kardinalitet større enn 2. Vi har

$$\begin{aligned} G &= \{(x-1)^2, (x-1)(y+1), (y+1)^2\} \\ &= \{x^2 - 2x + 1, xy + x - y + 1, y^2 + 2y + 1\} \end{aligned}$$

Vi kan vise at G er en Gröbnerbasis for idealet I, som består av alle polynomer i $\mathbb{F}[x, y]$ med total grad ≥ 2 og som blir 0 evaluert i $(1, -1)$. M er alle polynomer av grad < 2 i $\mathbb{F}[x, y]$.

G er altså hemmelig nøkkel. Den offentlige nøkkelen setter vi til

$$B = \{q_1, q_2\} = \{x^2 + y, x^2 - 1\}$$

Meldingen vi vil kryptere er $m = x$. Vi krypterer og får

$$c = \sum_{j=1}^{2} h_j q_j = xq_1 + yq_2 + x = x^3 + x^2 y + xy + x - y$$

Her er $m' = x - y$, og vi ser at $m \neq m'$.

Katla gjetter graden av f_j'ene til å være 1, det vil si $f_1 = a_1 x + a_2 y + a_3$ og $f_2 = b_1 x + b_2 y + b_3$. Hun løser ut $f_1 q_1 + f_2 q_2 + m = (a_1 + b_1)x^3 + (a_2 + b_2)x^2 y + (a_3 + b_3)x^2 + a_1 xy + a_2 y^2 - b_1 x + (a_3 - b_2)y - b_3 + m = c$.

Den første likningen blir altså

$$\begin{aligned} f_1 q_1 + f_2 q_2 - m'' &= c - m' \\ (a_1 + b_1)x^3 + (a_2 + b_2)x^2 y + (a_3 + b_3)x^2 + a_1 xy + a_2 y^2 &= x^3 + x^2 y + xy \end{aligned}$$

mens den andre blir

$$m'' + m = -b_1 x + (a_3 - b_2)y - b_3 + m = x - y = m'$$

Det første lineære systemet blir

$$\begin{cases} a_1 & = & 1 & & \\ a_2 & = & 0 & & \\ a_1 + b_1 & = & 1 & \implies & b_1 = 1 - 1 = 0 \\ a_2 + b_2 & = & 1 & \implies & b_2 = 1 - 0 = 1 \\ a_3 + b_3 & = & 0 & & \end{cases}$$

Vi står igjen med

$$-b_1x + (a_3 - b_2)y - b_3 + m = m' = x - y$$

som gir oss at

$$m = x - y + b_1x - a_3y + b_2y + b_3 = x - a_3y + b_3$$

Vi har at $a_3 = -b_3$, så $a_3 = 0, b_3 = 0$ er en løsning. Siden vi har funnet en løsning, vet vi at $m = x$.

Men hva hadde skjedd dersom leddene med høyest grad kansellerte under krypteringen? Da hadde vi ikke fått noen løsning av systemet, og vi måtte økt graden av f_i'ene.

Ulempen med dette angrepet er at det har høy kjøretid, særlig dersom polynomene som brukes i krypteringen er spredte.

7.4.2 Intelligent lineæralgebraangrep

Dette angrepet ble først lansert av Lenstra, og er gjengitt hos Rai [23]. Vi har satt det opp her med Rais notasjon. Vi har også inkludert et par ganske omfattende eksempler. I det første går angrepet på skinner, mens i det neste skjærer det seg. Men la oss først innføre noen begreper. Settingen vi befinner oss i, er beskrevet på starten av avsnitt 7.2.

Katla kjenner $c = \sum_{j=1}^{s} h_j q_j + m$, der m er i M. Vi definerer følgende:

- $\mathcal{C}$ = mengden av monomer som finnes i c
- $\mathcal{Q}_j$ = mengden av monomer som er i q_j

Angrepet bygger på følgende antakelse: For alle monomer d som er i h_j, har vi $d \cdot Q_j \cap \mathcal{C} \neq \emptyset$. Med andre ord, for et slikt monom d, bør det finnes minst et monom δ i q_i, slik at $d \cdot \delta$ er i $\mathcal{C}$.

Lenstra sier at det er «lett» å finne disse d'ene i polynomisk tid. Vi ser på to eksempler.

Eksempel 7.4.2. Vi bruker nesten samme skjema som i eksempel 7.4.1. Vi beholder G, men denne gangen setter vi $B = \{q_1, q_2\} = \{x^2 + y, y^2 - 1\}$. Anta at vi krypterer $m = x$ på følgende måte:

$$c = \sum_{j=1}^{2} h_j q_j = (x+1)q_1 + (y^3 - 1)q_2 + x = y^5 + x^3 - y^3 + x^2 + xy - y^2 + x + y + 1$$

Det vi gjør, er å for hver q_i gå gjennom alle leddene i c, og skille ut de monomene som kan være i $d \cdot \mathcal{Q}_j \cap \mathcal{C}$. Vi har $\mathcal{Q}_1 = \{x^2, y\}$, $\mathcal{Q}_2 = \{y^2, 1\}$, $\mathcal{C} = \{y^5, x^3, y^3, x^2, xy, y^2, x, y, 1\}$.

q_1 :	$\mathcal{C}$		$d? \cdot q, q \in \mathcal{Q}_1$	q_2 :	$\mathcal{C}$		$d? \cdot q, q \in \mathcal{Q}_2$
	y^5	$=$	$y^4 \cdot y$		y^5	$=$	$y^3 \cdot y^2 = y^5 \cdot 1$
	x^3	$=$	$x \cdot x^2$		$(x^3$	$=$	$x^3 \cdot 1)$
	y^3	$=$	$y^2 \cdot y$		y^3	$=$	$y \cdot y^2 = y^3 \cdot 1$
	x^2	$=$	$1 \cdot x^2$		$(x^2$	$=$	$x^2 \cdot 1)$
	xy	$=$	$x \cdot y$		xy	$=$	$xy \cdot 1$
	y^2	$=$	$y \cdot y$		$(y^2$	$=$	$1 \cdot y^2 = y^2 \cdot 1)$
	x	nei			x	$=$	$x \cdot 1$
	y	$=$	$1 \cdot y$		$(y$	$=$	$y \cdot 1)$
	1	nei			1	$=$	$1 \cdot 1$

Når vi følger prosedyren for q_1, finner vi at vi har to mulige d'er som går igjen to ganger (og de *skal* finnes nøyaktig to ganger i tabellen, siden det er to ledd i q_1): x og 1. Når vi ser på q_2, utelukker vi de fire monomene i $\mathcal{C}$ som tilhørte disse koeffisientene. Av de gjenværende finner vi også her to mulige d'er som går igjen to steder, nemlig y^3 og 1. Vi ser at det eneste monomet i $\mathcal{C}$ som ikke befinner seg i noen av $d \cdot Q_j \cap \mathcal{C}$ er x, og vi gjetter at dette er den hemmelige meldingen.

Dette verifiserer vi gjennom å finne koeffisientene i h_i'ene, og ser at det stemmer med c. Vi gjetter at

$$h_1 = ax + b \text{ og } h_2 = dy^3 + e \text{ og } m = fx$$

og løser ut:

$$(ax+b)(x^2+y) + (dy^3+e)(y^2-1) + fx = dy^5 + ax^3 + dy^3 + bx^2 + axy + ey^2 + by - e + fx$$

Setter vi dette lik c, ser vi at $a = 1$, $b = 1$, $d = 1$, $e = -1$ og $f = 1$ gir en løsning. Altså har vi at $m = x$.

Merk. Den siste utregningen vi gjorde i eksempelet, kan anses som en «forenklet» versjon av lineæralgebraangrepet, bare med mye mindre likninger. Likningene ble funnet gjennom en «intelligent» vurdering av c og B, derav «intelligent lineæralgebraangrep».

Framgansmåten over fører ikke alltid fram på første forsøk, for eksempel kan vi få problemer dersom flere $h_i q_i$'er inneholder samme monom, eller dersom et monom fra en $h_i q_i$ kanselleres og dermed ikke finnes i c. Det neste eksempelet viser en kryptert melding der dette har vært tilfelle.

Eksempel 7.4.3. Denne gangen har det oppstått kanselleringer under krypteringen. Vi har samme G, B og m som sist, men krypterer med andre polynomer. Denne gangen setter vi

$$c = \sum_{j=1}^{2} h_j q_j = (x+1)q_1 + (y^3+x^2)q_2 + x = y^5 + x^2y^2 + x^3 - y^3 + xy + x + y$$

Dette gir $\mathcal{Q}_1 = \{x^2, y\}$, $\mathcal{Q}_2 = \{y^2, 1\}$, $\mathcal{C} = \{y^5, x^2y^2, x^3, y^3, xy, y^2, x, y\}$. Vi setter opp tabeller tilsvarende som i forrige eksempel.

q_1 :	$\mathcal{C}$		$d? \cdot q, q \in \mathcal{Q}_1$	q_2 :	$\mathcal{C}$	$d? \cdot q, q \in \mathcal{Q}_2$
	y^5	$=$	$y^4 \cdot y$		y^5	$=$ $y^3 \cdot y^2 = y^5 \cdot 1$
	x^2y^2	$=$	$y^2 \cdot x^2$		$(x^2y^2$	$=$ $x^2 \cdot y^2?)$
	x^3	$=$	$x \cdot x^2$		$(x^3$	$=$ $x^3 \cdot 1)$
	y^3	$=$	$y^2 \cdot y$		$(y^3$	$=$ $y \cdot y^2 = y^3 \cdot 1)$
	xy	$=$	$x \cdot y$		$(xy$	$=$ $xy \cdot 1)$
	x	nei			x	$=$ $x \cdot 1$
	y	$=$	$1 \cdot y$		y	$=$ $y \cdot 1$

Vi ser at for q_2 finner vi ingen fornuftige muligheter for d. Vi sitter igjen med for mange ledd som *ikke* er i meldingsrommet.

I eksempelet over fant vi ikke m like lett som i det forrige, men vi kan bruke en mer «standardisert» versjon av angrepet, slik det er presentert hos Levy-dit-Vehel [20]. Dette er en slags mellomting av lineæralgebraangrep og intelligent lineæralgebraangrep («uintelligent intelligent lineæralgebraangrep»?).

Vi har gitt $\mathcal{C}$ og $\mathcal{Q}_j$ som før, og definerer dessuten

$$\mathcal{D} = \{\text{monom } d \mid \exists q \in \bigcup_{j=1}^{s} \mathcal{Q}_j, \exists p \in \mathcal{C} \text{ s.a. } p = dq\}$$

Deretter gjetter vi h_i'ene til å være

$$h_i = \sum_{d \in \mathcal{D}_i} k_d d$$

der de ukjente er k_d'ene (elementer i $\mathbb{F}$), og $\mathcal{D}_i$ er definert som

$$\mathcal{D}_i = \{d \in \mathcal{D} \mid deg_{\text{tot}}(d) \leq deg_{\text{tot}}(c) - deg_{\text{tot}}(q_i)\}$$

Dersom vi får at $\mathcal{D}_i$ inneholder *alle* mulige monomer som er begrenset av den gitte graden, er dette nøyaktig det samme som det «vanlige» lineæralgebraangrepet.

Eksempel 7.4.4. I forrige eksempel har vi

$$\mathcal{D} = \{y^5, x^2y^2, y^4, x^3, x^2y, y^3, xy, y^2, x, y, 1\}$$
$$\mathcal{D}_1 = \{x^3, x^2y, y^3, xy, y^2, x, y, 1\} = \mathcal{D}_2$$

Vi ser at monomene xy^2 og x^2 ikke er med, så dette er lettere enn et vanlig lineæralgebraangrep.

Hvis dette angrepet ikke fører fram (vi kan, som i lineæralgebraangrepet, prøve å øke graden i tilfelle det har vært kanselleringer), betyr det at det finnes monom(er) i h_i'ene som *ikke* er i $\mathcal{D}$. Da kan man utvide til et generelt lineæralgebraangrep, men dette vil øke kompleksiteten.

7.4.3 Lineæralgebraangrep på spredte systemer

Som nevnt tidligere, argumenterte Fellows og Koblitz for at spredte systemer vil øke kompleksiteten til lineæralgebraangrepene så mye at de ikke ble effektive, mens Barkee et al. gjettet at dette ikke ville hjelpe, snarere tvert imot. Caboara, Caruso og Traverso introduserte, i

forbindelse med presentasjonen av en helt ny Polly Cracker-variant [5], følgende angrep på spredt Polly Cracker:

Det er slik at *alle* monomene som opptrer i q_i'ene, h_i'ene, c og beregninger forøvrig utgjør en relativt *liten* mengde (også på grunn av spredtheten, samt at beregningene skal kunne utføres relativt raskt). Altså skjer alle beregninger i et $\mathbb{F}$-vektorrom utspent av denne mengden av monomer. Vi vil prøve å finne en slik liten mengde S', slik at vi kan utføre *tett* lineæralgebra i $\text{Span}_{\mathbb{F}}(S')$.

La $S = \{\text{alle monomer som er i } c\}$. Vi har $S \subseteq S'$. Videre vet vi at idealet I (generert av den hemmelige nøkkelen) består av tre typer polynomer: monomer (med koeffisient), $\mathbb{F}$-lineærkombinasjoner av *to* monomer, og polynomer med minst tre ledd. Vi antar først at B kun består av polynomer med minst tre ledd. Siden systemet er spredt, er det stor sannsynlighet for det følgende: Dersom vi har to slike polynomer f_1 og f_2, og et monom $\mathbb{X}$, vil *maksimalt ett* monom kansellere i summen $f_1 + \mathbb{X}f_2$.

Vi bygger S' på følgende måte: Vi setter først $S' = S$, og legger til alle monomer $\mathbb{X}$ slik at $\mathbb{X}q_i$ er i S' for en q_i i B. Under antakelsen vi nettopp gjorde vil S' ha begrenset kardinalitet. Dersom vi for eksempel har gitt $B = \{q_1, q_2\}$ der q_1 og q_2 har tre ledd, og c har tolv ledd, kan vi regne med at h_1 og h_2 har maks tre ledd hver, eventuelt seks ledd til sammen.

Deretter utfører vi «vanlig» lineæralgebra, på det (relativt lille) tette systemet vi ender opp med.

Men hva om B også inneholder polynomer med ett eller to ledd? La $I_{1,2}$ være idealet generert av alle slike polynomer i B. Ettleddspolynomene representerer ingen «fare», det er lett å finne Gröbnerbasis for et ideal generert av slike. Dersom det også er lett å finne en Gröbnerbasis $G_{1,2}$ for $I_{1,2}$, kan vi flytte fokus fra $R = \mathbb{F}[x_1, \ldots, x_n]$ til $R_{1,2} = R/I_{1,2}$. Med $G_{1,2}$ behandler vi alle polynomer f i $R_{1,2}$ på følgende vis:

- Dersom f inneholder et monom i $G_{1,2}$ (anta alle elementene i $G_{1,2}$ er moniske, så de er på formen $\mathbb{X}$ eller $\mathbb{X} - k\mathbb{Y}$), ta ut det aktuelle leddet fra f. Dette siden monomet er 0 i $R_{1,2}$.
- Dersom f inneholder et monom $\mathbb{X}$ slik at $\mathbb{X} - k\mathbb{Y}$ er i $G_{1,2}$, erstatt $\mathbb{X}$ med $k\mathbb{Y}$.

Dette forenkler arbeidet med å finne S', siden c og polynomene i B har blitt forenklet.

Redningen (altså det som forkludrer angrepet) må bli det følgende: Kan det finnes polynomer på formen $k_1\mathbb{X} - k_2\mathbb{Y}$ slik at idealet generert av disse, er slik at det er vanskelig å finne en Gröbnerbasis for? Polynomer på denne formen kalles *binomer*, og denne typen Polly Cracker-systemer er beskrevet i [5]. Per dags dato (2010) er Caboara, Caruso og Traversos Polly Cracker-variant, kalt *Lattice Polly Cracker*, et av få Polly Cracker-systemer det fortsatt er knyttet håp til (ifølge [20]).

7.4.4 Angrep med en delvis Gröbnerbasis

Dette er et angrep som ble introdusert av Barkee et al. [1]. Det baserer seg på følgende resultat:

Proposisjon 7.4.5. *Gitt $J = \langle f_1, \cdots, f_l \rangle \subseteq I \subseteq R$ og en h i R med grad mindre eller lik D slik at $h - N_J(h) = \sum_{i=1}^{l} p_i f_i$ med $deg(p_i f_i) \leq D$. ($N_J(h)$ er normalformen av h i R/J.)*

Vi kjører en modifisert utgave av Buchbergers algoritme på $\{f_1, \ldots, f_l\}$ der ingen beregninger som involverer polynomer av grad $\geq D$ blir utført, og får som resultat en mengde G'. Da har vi $h \xrightarrow{G'} N_J(h)$.

Merk. Hvis G er en Gröbnerbasis for J slik at $G' \subseteq G$, har vi $h \xrightarrow{G} N_J(h)$ (proposisjon 2.4.5). Men bare polynomer i G av grad $\leq \deg(h) \leq D$ kan redusere h, så $h \xrightarrow{G'} N_J(h)$.

Vi har sett at man kan konstruere den offentlige nøkkelen, B, i et Polly Cracker-system slik at det er umulig å finne en Gröbnerbasis for $\langle B \rangle$ i polynomisk tid (forutsatt $\mathcal{P} \neq \mathcal{NP}$). Men det er ikke dermed sagt at vi ikke kan finne en *delvis* Gröbnerbasis, begrenset av en grad D, som i teoremet over. Anta at vi finner en slik delvis Gröbnerbasis, G', for $\langle B \rangle$. Katla har mottatt en kryptert melding, c, og beregner $h = \overline{c}^{G'}$. Dersom vi har h i M, er hun i mål (siden dekrypteringen er entydig og $G' \subseteq G$ der G er hele Gröbnerbasisen for $\langle B \rangle$). Hvis ikke, må det ha vært noen kanselleringer i krypteringen, og Katla øker D og prøver igjen.

Barkee hevder at kompleksiteten i dette angrepet er $O(\tau^4)$, der τ er antall ledd i c med grad mindre eller lik D. Dette forutsetter et tett system, og at man ikke begynner forfra med S-polynomer når man øker D, men snarere bruker dem man beregnet tidligere og dessuten lagrer S-polynomer av grad $D + 1$.

Tilsvarende angrep i det ikke-kommutative tilfellet blir tatt opp i de to siste kapitlene i denne oppgaven.

8 Ikke-kommutativ Polly Cracker

8.1 Innledning

Ikke-kommutativ Polly Cracker ble introdusert av Rai i hans doktoravhandling fra 2004 [23]. Noen senere artikler (som [4]) følger opp denne avhandlingen, men generelt er det gjort lite forskning på området, og det som er gjort, følger Rais premisser. Rais Polly Cracker-system er på et tidlig stadium, og mye av det bygger på empiriske data innsamlet gjennom prøving og feiling. Der Koblitz og Fellows lagde et system hvis sikkerhet var basert på kompleksiteten til $\mathcal{NP}$-komplette problemer, har Rai forsøkt å utnytte det faktum at det i ikke-kommutative polynomringer finnes idealer som ikke har endelig Gröbnerbasis, ved å sette slike idealer som offentlig nøkkel.

Dette kapittelet tar for seg det generelle oppsettet for ikke-kommutative Polly Cracker-systemer, og noen angrep vi bør sikre oss mot. Rais systemer kommer vi tilbake til i kapittel 10.

8.2 Oppsett

Et generelt ikke-kommutativt Polly Cracker-system over $\mathbb{F}\langle X\rangle$, der $\mathbb{F}$ er en endelig kropp og X en endelig mengde ikke-kommuterende variabler, er gitt ved:

Hemmelig nøkkel: G, en Gröbnerbasis for idealet $I = \langle G\rangle \subseteq \mathbb{F}\langle X\rangle$.
Offentlig nøkkel: $B = \{q_i\}_{i=1}^{s}$, der q_i er i I. La $J = \langle B\rangle$, et ideal i $\mathbb{F}\langle X\rangle$ inneholdt i I.
Meldingsrom: $M \subseteq \mathrm{Span}(\mathrm{NonTip}(I))$. M må være endelig.
Chiffertekstrom: $C \subseteq \mathbb{F}\langle X\rangle$.
Kryptering: $c = p + m$, der $p = \sum_{i=1}^{s}\sum_{j=1}^{k_i} F_{ij}q_iH_{ij}$ er i J og F_{ij} og H_{ij} er i $\mathbb{F}\langle X\rangle$.
Dekryptering: Redusere c modulo G.

For kommentarer om kryptering og dekryptering, se merknad 3. og 4. til det kommutative Polly Cracker-oppsettet.

Merk. I [23] skriver Rai at meldingsrommet M er en delmengde av $\mathrm{NonTip}(I) = \mathcal{B} \setminus \mathrm{Tip}(I)$, en mengde som kun består av monomer. I praksis bruker han likevel polynomer med koeffisienter i $\mathbb{F}$ som meldinger. Jamfør kommutativ Polly Cracker skal meldingsrommet være en delmengde av

$$\{f \in \mathbb{F} | f \xrightarrow{G} f\}$$

og denne mengden er den samme som $\mathrm{Span}(\mathrm{NonTip}(I))$ (se beviset av korollar 3.4.8). I praksis er M ofte gitt ved en mengde monomer A i $\mathrm{NonTip}(I)$, slik at $M = \mathrm{Span}(\mathrm{NonTip}(A))$. Vi skal se på noen nødvendige, sikkerhetsmessige krav til meldingsrommet i avsnitt 8.4.3.

8.3 Konstruksjon av B

Lemma 8.3.1. *Gitt et Polly Cracker-system over $\mathbb{F}\langle X\rangle$, med hemmelig nøkkel G, der $\langle G\rangle = I \subseteq \mathbb{F}\langle X\rangle$, og offentlig nøkkel B, der $\langle B\rangle = J \subseteq I$. Dersom en kryptanalyst har en Gröbnerbasis G' for J, kan hun dekryptere enhver melding $c = p + m$ der p er i J og m er i M, siden $c \xrightarrow{G'} 0$.*

Bevis. Se lemma 7.3.5. ♠

Som før må altså B konstrueres slik at det er umulig å finne en (endelig) Gröbnerbasis for J i polynomisk tid. Men det er to ting til vi må passe på i denne sammenhengen. Ved å benytte Buchbergers algoritme på mengden B, vil vi finne de første, om ikke alle, elementene i en Gröbnerbasis G' for J. Derfor må B oppfylle følgende krav:

(i) Elementene i G' er ikke på en forutsigbar form. Selv om vi ikke klarer å liste alle elementene, vet vi hvordan alle ser ut. (En analogi: Vi kan regne med alle heltall, selv om det er uendelig mange av dem.) I kapittel 9.2 skal vi se noen eksempler på idealer som *ikke* oppfyller dette kravet.

(ii) Elementene i G', etter hvert som vi beregner dem, har ikke økende tupper. Vi husker teorem 7.4.5, som sier at for å redusere et element f til normalformen $N(f)$, trenger vi bare å beregne de elementene g_i i Gröbnerbasisen der $LT(g_i) \preceq LT(f)$. Resultatet overføres til ikke-kommutative Gröbnerbasiser (følger fra korollar 3.4.8), og vi har at mengden $\{g \in G | \text{tip}(g) \prec \text{tip}(f)\}$ er endelig for en gitt f i $\mathbb{F}\langle X\rangle$. Dersom vi bruker Buchbergers algoritme på $\langle B\rangle$, og vet at for hvert element vi finner, er tuppen større enn hos de foregående, kan vi alltid finne normalformen av f.

I tillegg må B være slik at det ikke er mulig å hente ut noen informasjon om G ved å studere B (og eventuelt M). Ideer til hvordan man lager sikre offentlige nøkler kommer vi tilbake til når vi ser på ulike ikke-kommutative Polly Cracker-systemer i kapittel 10.

8.4 Angrep

Vi skal ta for oss alle, unntatt ett, av angrepene fra kommutativ Polly Cracker-kapittelet. Det «farligste» angrepet vi så på tidligere var *angrep med en delvis Gröbnerbasis* (avsnitt 7.4.4). I forrige avsnitt satte vi noen krav til B, som er slik at dersom de er oppfylt, vil ikke dette angrepet fungere. Disse kravene er på ingen måte trivielle å oppfylle – vi vil komme tilbake til dette i kapittel 10.

Angrepene vi ser på her, kan deles i to typer. De første, lineæralgebraangrepene, har som mål å dekryptere enkeltmeldinger (chiffertekstangrep). De andre er valgt chiffertekstangrep, noe som utnytter andre svakheter ved systemet, men de er like fullt verdt å sikre seg mot.

8.4.1 Lineæralgebraangrep

Som i det kommutative tilfellet, er kryptanalysten Katla kjent med:

- den offentlige nøkkelen $B = \{q_i\}_{i=1}^{s}$
- meldingsrommet M
- en chiffertekst $c = p + m$, der $p = \sum_{i=1}^{s} \sum_{j=1}^{k_i} F_{ij} q_i H_{ij}$

På samme måte som tidligere, kan Katla gjøre en fornuftig gjetting på den maksimale graden (eller, mer formelt, *lengden* av tuppen) til F_{ij}'ene og H_{ij}'ene. Men der stopper også analogien. Som vi så i eksempel 3.2.2, kan ikke alltid doble summer gjøres om til enkle. Vi ender opp med å se på uttrykk av typen $u \cdot q_i \cdot v$, der u og v er monomer i $\mathcal{B}$. Monomene u og v kan gå igjen i flere slike ledd, men med ulike koeffisienter. Dermed er det umulig å sette opp et likningssystem på samme måte som vi gjorde sist.

Hvis Katla *kjenner* formen på (det vil si monomene i) F_{ij}'ene og H_{ij}'ene, blir situasjonen en annen. Men er det trolig at hun gjør det? I kommutativ Polly Cracker blir polynomene som brukes til kryptering valgt så godt som tilfeldig (med mulig unntak av noen grep for å forhindre nettopp lineæralgebraangrep). I Rais forslag til systemer, derimot, er en eller begge av disse betingelsene ofte oppfylt:

(i) Formen på F_{ij}'ene og H_{ij}'ene er offentlig kjent.

(ii) Kun enkle summer er brukt i krypteringen, det vil si $p = \sum_{i=1}^{s} F_i q_i H_i$.

I begge disse tilfellene kan vi sette opp likningssystemer der koeffisientene av F_{ij} og H_{ij} er de ukjente (for type (ii) er mange av disse lik null). I det andre tilfellet kan systemet bli for stort til at det er løselig, mens i det første tilfellet er dette sannsynligvis ikke tilfelle. Men ingen av systemene blir til lineære, da koeffisienten i et ledd vil bestå av (summer av) multiplum av koeffisienter fra både F'er og H'er. Dette er grunnen til at ikke-kommutativ Polly Cracker regnes som sikret mot lineæralgebraangrep. Det finnes likevel unntak, som det følgende tilfellet, generelt beskrevet av Rai [23], viser.

Et vellykket lineæralgebraangrep

Anta at vi kjenner formen på F_{ij}'ene og H_{ij}'ene brukt til kryptering av en melding m. Dersom disse er «dumt» satt opp, vil angrepet beskrevet i følgende eksempel gi oss m.

Eksempel 8.4.1. Dette er et lite realistisk eksempel, men illustrerer angrepet godt. La $B = \{q_1, q_2\} = \{x + 1, y + 2\} \subseteq \mathbb{F}\langle x, y\rangle$ der $\mathbb{F} = \mathbb{Z}_{29}$, og la meldingsrommet være $M = \mathbb{F}$. Vi vil kryptere meldingen $m = 9$. La

$$\begin{array}{lcllcl} F_{11} & = & y+1 & & F_{21} & = & x+y \\ H_{11} & = & 3 & & H_{21} & = & 2x \\ F_{12} & = & 3x & & H_{22} & = & 4 \end{array}$$

Vi krypterer og får

$$c = 2xyx + 2yyx + 7xx + 7yx + 6x + 7y + 20$$

Katla lar a, b, d, e, f, g, h og i være de ukjente koeffisientene og setter opp følgende uttrykk:

$$(ay + b)q_1 d + exq_1 + (fx + gy)(y + 2)hx + (y + 2)i = c \tag{8.1}$$

Hun løser det opp og får likningssettet

$$\begin{cases} fh & = 2 \\ gh & = 2 \\ e + 2fh & = 7 \\ ad + 2gh & = 7 \\ bd + e & = 6 \\ ad + i & = 7 \\ bd + 2i + m & = 20 \end{cases}$$

Dette er et ikke-lineært likningssett med 7 likninger og 9 ukjente, og burde derfor vært vanskelig å løse. Imidlertid er ikke Katla interessert i verdiene $a, \ldots, i$, så lenge hun finner m. Hun substituerer $\delta_1 = fh$, $\delta_2 = gh$, $\delta_3 = ad$ og $\delta_4 = bd$ og får det følgende lineære likningssystemet med 7 likninger og 7 ukjente:

$$\begin{cases} \delta_1 & = 2 \\ \delta_2 & = 2 \\ e + 2\delta_1 & = 7 \\ \delta_3 + 2\delta_2 & = 7 \\ \delta_4 + e & = 6 \\ \delta_3 + i & = 7 \\ \delta_4 + 2i + m & = 20 \end{cases}$$

Det er enkelt å finne løsningen $m = 9$ av dette systemet.

Mottiltak. Man må øke antall ukjente uten å øke antall likninger. Dette gjøres ved å sørge for at flere $F_{ij} \cdot q_i \cdot H_{ij}$-ledd bidrar til monomene i c, for eksempel ved å bruke noen (F_{ij}, H_{ij})-par flere ganger men med ulike koeffisienter.

Eksempel 8.4.2. Vi legger til $(F_{13}, H_{13}) = (2y + 3, 5)$ i krypteringen. Da må Katla ta med leddet $(j \cdot y + k)q_1 \cdot l$ i (8.1). Hun setter $\delta_5 = jl$ og $\delta_6 = kl$. Likningssystemet hun nå må løse har også 7 likninger, men 9 ukjente.

Dette angrepet er altså enkelt å slå tilbake, men det viser at valget av F_{ij} og H_{ij} ikke kan være *helt* tilfeldig.

8.4.2 Intelligent lineæralgebraangrep

Dette angrepet er ikke avhengig av at angriperen kjenner formen på F_{ij} og H_{ij}, og er slik kanskje et mer realistisk scenario enn det forrige. Det intelligente lineæralgebraangrepet er presentert i kapittel 7.4.2. Det ikke-kommutative tilfellet skiller seg fra det kommutative ved at vi nå har tosidig multiplikasjon.

Vi har nå to muligheter med hensyn til F_i'ene og H_i'ene: Enten er formen på dem kjent, ellers er den ikke det. For å ta det første først: I dette tilfellet mangler vi bare koeffisientene, og vi er

i samme situasjon som ved det vanlige lineæralgebraangrepet. Problemet med at likningene ikke er lineære, er vedvarende.

Dersom formen på F_i'ene og H_i'ene er ukjent, kan man bruke et angrep nesten identisk med det kommutative intelligente lineæralgebraangrepet for å finne (gjette) formen på F_i'ene og H_i'ene. Kanselleringer og sammenslåinger kan igjen skape problemer, og det er vanskeligere å gjette, siden vi har flere muligheter:

Eksempel 8.4.3. Vi har monomet $xxyy$ i c, og vet at monomet y finnes i q_1. Hvordan kan y finnes i $xxyy$?
Kommutativt tilfelle: Én mulighet, nemlig $x^2y \cdot y$
Ikke-kommutativt tilfelle: To muligheter, $xx \cdot y \cdot y$ og $xxy \cdot y \cdot 1$

En ikke-kommutativ variant av Levy-dit-Vehels generaliserte intelligente lineæralgebraangrep (se avsnitt 7.4.2) er også mulig, men koeffisientene blir ikke lettere å finne, siden vi fortsatt får et ikke-lineært system. Vi konkluderer med at ikke-kommutativ Polly Cracker er, såfremt man tar hensyn til mottiltaket fra avsnitt 8.4.1, sikret mot lineæralgebraangrep.

Merk. Polynomene som brukes i krypteringen må være av et slikt antall og av en slik lengde at ikke-lineære systemer oppnådd ved intelligent lineæralgebraangrep faktisk *er* vanskelige å løse.

8.4.3 Valgt chiffertekst-angrep

De følgende angrepene er hentet fra en artikkel av Rai og Bulygin [4], en av få tekster som følger opp [23]. Denne artikkelen omhandler også et tredje angrep, men da mottiltaket her innebærer å fjerne seg fra ideene bak offentlig nøkkel-kryptografi (se kapittel 1.3), har vi utelatt dette angrepet fra denne teksten.

Angrep 1. Dette angrepet fungerer dersom følgende kriterier er oppfylt:

(i) Den hemmelige nøkkelen G er en endelig redusert Gröbnerbasis $G = \{g_1, \ldots, g_t\}$.

(ii) Kryptanalysten kjenner tuppen av alle elementene i G.

(iii) Kryptanalysten har midlertidig tilgang til dekrypteringsprogrammet (uten å kjenne den hemmelige nøkkelen).

Merk. Hvor realistisk er det at krav (ii) er oppfylt? I Rais systemer er dette regelen heller enn unntaket. Faktisk er dette tilfelle også for spesialtilfellet av kommutativ Polly Cracker (se 7.2.1), der vi har $LT(g_i) = t_i$. Disse systemene er svært sårbare for denne type angrep.

Slik fungerer angrepet: Anta at krav (i)–(iii) er oppfylt, og at kryptanalysten Katla sender følgende «meldinger» til dekryptering: $\{\text{tip}(g_1), \text{tip}(g_2), \ldots, \text{tip}(g_t)\}$. Dette gir, siden dekryptering er reduksjon med en Gröbnerbasis, elementene $\{N(\text{tip}(g_1)), \ldots, N(\text{tip}(g_t))\}$. Definer $g_i' = \text{tip}(g_i) - N(\text{tip}(g_i))$, og sett $G' = \{g_1', \ldots, g_t'\}$. Vi har

$$\text{tip}(g_i') = \text{tip}(g_i) \text{ og}$$

$$N(g_i') = N(\text{tip}(g_i)) - N(N(\text{tip}(g_i))) = N(\text{tip}(g_i)) - N(\text{tip}(g_i)) = 0$$

Dette medfører henholdsvis $\langle \text{Tip}(G') \rangle = \langle \text{Tip}(G) \rangle$ og $G' \subseteq \langle G \rangle$, hvilket betyr at G' er en Gröbnerbasis for $\langle G \rangle$. Siden $f \xrightarrow{G'} N(f)$ for alle f i $\mathbb{F}\langle X \rangle$, kan vi bruke G' til å dekryptere en hvilken som helst melding som blir sendt.

Et annet alternativ er å velge monomer $m_1, \ldots, m_t$ i meldingsrommet, og deretter dekryptere $\{m_1 + g_1, \ldots, m_t + g_t\}$. Siden vi har

$$N(m_i + \text{tip}(g_i)) = m_i + N(\text{tip}(g_i))$$

oppnår vi samme resultat som før ved å sette $g' = \text{tip}(g_i) - (N(\text{tip}(g_i)) - m_i)$.

Mottiltak. Vi må ha noen krav til konstruksjonen av M og G. La $A \subsetneq \text{NonTip}(I)$, og sett $M = \{f \in \text{Span}(A) \mid l(\text{tip}(f)) \leq d\}$ for en d i $\mathbb{N}$. La G være slik at for alle g_i i G, er ikke $\text{tail}(g_i)$ i meldingsrommet M. Dekrypteringsalgoritmen må implementeres slik at den returnerer en feilmelding eller den uberørte krypterte chifferteksten ved input c og output $N(c) \notin M$.

Siden G er en redusert Gröbnerbasis, har vi at

$$N(\text{tip}(g_i)) = -\text{tail}(g_i) \notin M$$

Dersom kryptanalysten forsøker å sende inn $\text{tip}(g_i)$ til dekryptering, får hun altså en feilmelding i retur. Prøver hun å dekryptere $m_i + \text{tip}(g_i)$ gir det heller ikke ønsket resultat, siden $m_i - \text{tail}(g_i) \notin M$.

Hva ville skjedd dersom dette angrepet ble gjennomført mot et system der den hemmelige nøkkelen var en Gröbnerbasis, men ikke en *redusert* sådan? Anta dette er tilfelle for en minimal Gröbnerbasis G, og at en kryptanalyst sender $\text{tip}(g_i)$ for en g_i i G gjennom dekrypteringsprogrammet. Nå kan det være at $\text{tip}(\text{tail}(g_i)) = u \cdot \text{tip}(g_j) \cdot v$ for $g_j \neq g_i$ i G og u, v i $\mathcal{B}$. Blant monomene i resultatet av reduksjonen med g_j finner vi $u \cdot \text{tip}(g_j) \cdot v$. Slik fortsetter vi til vi har normalformen av $\text{tip}(g_i)$, for siden G er en endelig Gröbnerbasis og $\text{tip}(g_i)$ har endelig lengde, stopper prosessen etter et endelig antall reduksjoner.

For det første må vi passe på at $u\text{tail}(g_i)v$ ikke er i M for alle g_i i G og alle u, v i $\mathcal{B}$. Videre kan det ikke gå an at de leddene i $u\text{tail}(g_i)v$ som ikke er i M, kansellerer mot de leddene i $u'\text{tail}(g_j)v'$ som ikke er i M, og så videre. Rai og Bulygin foreslår at minst et monom $b_i \notin M$ finnes i g_i, der $ub_iv \notin M$ for alle u, v i $\mathcal{B}$, men det bør være flere enn et slikt monom i hver g_i for at sjansene for kansellering er liten. Dette blir fort komplisert, så en redusert Gröbnerbasis er anbefalt.

Det følgende angrepet er mer generelt og krever ikke at angriperen kjenner tuppene i den hemmelige nøkkelen, det holder å vite hvilken tillatelig ordning som blir brukt.

Angrep 2. Anta at følgende kriterier er oppfylt:

(i) Den hemmelige nøkkelen G er en endelig redusert Gröbnerbasis $G = \{g_1, \ldots, g_t\}$.

(ii) Kryptanalysten vet hvilken tillatelig ordning som brukes i dekrypteringen.

(iii) Kryptanalysten har midlertidig tilgang til dekrypteringsprogrammet (uten å kjenne den hemmelige nøkkelen).

(iv) Den største tuppen i B er større enn eller lik den største tuppen i G:

$$\max_{q_i \in B}\{\text{tip}(q_i)\} = T \succeq \max_{g_i \in G}\{\text{tip}(g_i)\}$$

Punkt (iv) er vanligvis oppfylt, siden q_i er laget ved hjelp av g_i'ene. Vi skal se nærmere på hvorvidt angrepet fungerer dersom punktet *ikke* er oppfylt senere.

Slik fungerer angrepet: Kryptanalysten finner den største tuppen i B, kalt T, og definerer mengden $\tau = \{t \in \mathcal{B} | t \preceq T\}$. Dette er en endelig mengde ($\prec$ er den tillatelige ordningen som brukes i dekrypteringen), og vi vet at $\text{Tip}(G) \subseteq \tau$. Kryptanalysten bruker deretter dekrypteringsprogrammet til å beregne $N(t)$ for alle t i τ. Det er to mulige resultater:

(1) $N(t) = t$. Da er $t \notin \text{Tip}(I)$, det vil si $t \notin \text{Tip}(G)$.

(2) $N(t) \neq t$. Da er t i $\text{Tip}(I)$, det vil si at $t = u \cdot \text{tip}(g) \cdot v$ for en g i G og u og v i $\mathcal{B}$.

Vi forkaster alle tuppene som oppfyller punkt (1), og lar

$$G' = \{t - N(t) | t \in \tau \text{ og } N(t) \neq t\}$$

Vi har $\text{Tip}(G') \subseteq \langle \text{Tip}(G) \rangle$ og $\text{Tip}(G) \subseteq \text{Tip}(G')$, så $\langle \text{Tip}(G') \rangle = \langle \text{Tip}(G) \rangle$. Videre er $N(t - N(t)) = 0$ for alle $t - N(t)$ i G', så $G' \subseteq \langle G \rangle$. Dermed har vi funnet en Gröbnerbasis G' for $\langle G \rangle$, slik at vi kan beregne normalformer i $\mathbb{F}\langle X \rangle / \langle G \rangle$ – med andre ord dekryptere alle meldinger.

Merk at dersom $l(T) = \delta$, er antall elementer i τ $O(n^\delta)$, der n er antall variabler i ringen beregningene utføres i. Altså vokser kjøretiden til angrepet eksponensielt med lengden av T.

Mottiltak. Anta at krav (i)–(iv) i beskrivelsen av angrepet er oppfylt. Vi ser at for å forhindre dette angrepet, kan vi gjøre det samme som i forrige mottiltak.

Angrep 2 forutsetter at T er større enn, eller lik, den største tuppen i den hemmelige nøkkelen. Rai og Bulygin hevder at dette alltid er tilfelle, men dette stemmer ikke – det er ofte mulig å konstruere B slik at vi får kanselleringer i hver q_i (for eksempel ved hjelp av overlapper i G). Vi antar at dette er tilfelle, det vil si at vi har en Gröbnerbasis G og en offentlig nøkkel $B \subset \langle G \rangle$, der

$$\max_{q_i \in B}\{\text{tip}(q_i)\} = T \prec \max_{g_i \in G}\{\text{tip}(g_i)\} = p \tag{8.2}$$

Angrepet gjennomføres som før, og kryptanalysten oppnår en tuppredusert mengde G' der $G' \subseteq \langle G \rangle$. Vi antar at G er redusert. Da vet vi at

$$\langle \text{Tip}(G') \rangle \subsetneq \langle \text{Tip}(G) \rangle$$

Dette følger av at p ikke kan være på formen $u'\text{tip}(g')v'$ for g' i G, siden vi vet at $\text{tip}(g') = u\text{tip}(g)v$ for en g i G, og G er redusert. Ergo har kryptanalysten funnet en mengde G' slik at $\langle G' \rangle \neq I$. Det er to muligheter for G':

(1) G' er en Gröbnerbasis for $\langle G' \rangle = I' \subsetneq I$. Dersom dette er tilfelle, vil kryptanalysten kanskje narres til å tro at hun har funnet en Gröbnerbasis for I, men som ikke virkeligheten ikke kan

brukes til dekryptering. Eller kan den det? Dersom alle q_i'ene i B reduserer til 0 modulo G', vil G' dekryptere korrekt.

(2) G' er ikke en Gröbnerbasis. Da forstår kryptanalysten hva som har skjedd, og øker T i angrepet. Hun vil da (før eller siden) enten finne en Gröbnerbasis for I, eller havne i tilfelle (a).

9 Idealer med uendelig Gröbnerbasis

9.1 Innledning

Som nevnt i forrige kapittel, er en viktig egenskap ved Rais ikke-kommutative Polly Cracker-systemer at den offentlige nøkkelen skal generere et ideal med uendelig Gröbnerbasis [23]. I dette kapitlet skal vi ta for oss noen slike idealer, selv om alle ikke er egnet for bruk i kryptografi. Dette gjelder spesielt de tre idealene vi ser på i det første avsnittet. I tillegg skal vi se på noen teknikker som i enkelte tilfeller kan brukes for å finne normalformer i slike idealer (disse teknikkene er ikke generaliserte, og er derfor begrenset til noen små og veldig konkrete eksempler). Selve kryptosystemene basert på noen av idealene vi skal se på blir presentert i neste kapittel.

Vi benytter notasjonen introdusert i kapittel 3, der k betegner en kropp, og $\mathcal{B} = \{x_1, \ldots, x_n\}^*$.

9.2 Tre idealer med uendelig Gröbnerbasis

Vi skal her se på tre hovedidealer som alle har en uendelig Gröbnerbasis under visse forutsetninger. Alle bevisene følger samme tankegang: Vi viser at alle elementer i Gröbnerbasisen har en bestemt form ved å vise at alle overlappsrelasjoner mellom to elementer på denne formen danner et nytt på samme form.

Idealet $\langle xx - xy \rangle$ i $k\langle x, y \rangle$

Vi ser på idealet $I = \langle xx - xy \rangle$ i $k\langle x, y \rangle$, gitt en vilkårlig tillatelig ordning $\prec$ der $x > y$. Da har I en uendelig Gröbnerbasis for alle slike ordninger.

Bevis. La $g_1 = xx - xy$. Siden $x > y$, er $\text{tip}(g_1) = xx$, og siden g_1 overlapper seg selv, er ikke $\{g_1\}$ en Gröbnerbasis. Vi får

$$O(g_1 \cdot x, x \cdot g_1) \overset{\{g_1\}}{\rightarrow} xyx - xyy = g_2$$
$$O(g_1 \cdot yx, x \cdot g_2) \overset{\{g_1, g_2\}}{\rightarrow} xyyx - xyyy = g_3$$

Allerede etter to overlapper ser vi et mønster. Vi vil vise at $G = \{g_i | g_i = xy^{i-1}x - xy^i, i \geq 1\}$ er en Gröbnerbasis for $\langle xx - xy \rangle$. Vi viser ved induksjon at to elementer g_i og g_j i G må gi opphav til et nytt element i G, nærmere bestemt g_{i+j}.

(1) For $i = 1$ vet vi at g_1 har én selvoverlapp, som gir oss g_2.

(2) Anta at vi har $g_k = xy^{k-1}x - xy^k$ i G for $k \leq m$. Vi ser på g_i og g_j, der i og j er mindre enn eller lik m, men $i + j$ kan være større enn m. Det er bare to overlapper mellom g_i og g_j, og på grunn av symmetrien mellom dem, trenger vi bare å se på den ene:

$$O(g_i \cdot y^{j-1}x, x \cdot y^{i-1}g_j) = xy^{i-1}xy^j - xy^{i+j-1}x = h$$

Siden $x > y$, er $\text{tip}(h) = xy^{i-1}xy^j$. Vi ser at $\text{tip}(h) = \text{tip}(g_i)y^j$, så vi kan redusere h med g_i og få $h' = -xy^{i+j-1}x - xy^{i+j}$. Dersom $i + j - 1 \leq m$, finnes h allerede i G. Dersom $i + j > m$, finnes ingen elementer i G som reduserer h'. Vi setter $h'' = -1 \cdot h'$ og ser at h'' er på formen g_{i+j}, og vi legger dette elementet til i Gröbnerbasisen. Ved å velge i og j slik at $i + j > m$, kan vi alltid finne nye elementer, så $\langle xx - xy \rangle$ har en uendelig Gröbnerbasis på denne formen. ♣

Idealer på formen $\langle xyx - Axz \rangle$ i $k\langle x, y, z \rangle$

A er en konstant i k. Dette idealet har en uendelig Gröbnerbasis for alle tillatelige ordninger der $y \geq z$. På samme måte som over kan vi vise at $\langle xyx - Axz \rangle$ har en uendelig Gröbnerbasis, denne gang på formen $G = \{g_i | g_i = xz^{i-1}yx - Axz^i, i \geq 1\}$. Vi får alltid $\text{tip}(g_i) = xz^{i-1}yx$, siden $y \geq z$, siden kriteriene *(i)* og *(ii)* for tillatelige ordninger gir

$$y \geq z \overset{(i)}{\Rightarrow} xz^{i-1}y \succeq xz^i \overset{(i)}{\Rightarrow} xz^{i-1}yx \succeq xz^ix \overset{(ii)}{\Rightarrow} xz^{i-1}yx \succ xz^i$$

Merk. Vi ser lett at G er en *redusert* Gröbnerbasis for dette idealet. I utgangspunktet kan vi tenke oss at selv om vi finner en generell, uendelig Gröbnerbasis for et ideal, kan det også finnes en annen Gröbnerbasis som er endelig. Men den reduserte Gröbnerbasisen til et ideal er unik (gitt en bestemt ordning), og ingen annen Gröbnerbasis kan ha færre elementer enn den reduserte (det ville vært en motsigelse av proposisjon 3.4.11).

Idealer på formen $\langle xTx + AWx \rangle$ i $k\langle x_1, \ldots, x_n \rangle$

Igjen er A en konstant i k, mens x er et element i $\{x_1, \ldots, x_n\}$ og T og W er monomer i $\mathcal{B}$ med følgende egenskaper:

(i) $T \succeq W$

(ii) T og W overlapper verken hverandre eller seg selv

(iii) Verken T eller W starter på eller slutter med x

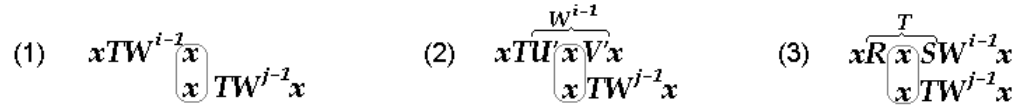
Da har $I = \langle xTx + AWx \rangle$ en uendelig redusert Gröbnerbasis for alle tillatelige ordninger der punkt (i) holder.

Bevis. Vi setter $g_1 = xTx + AWx$. Beregninger av de første overlappsrelasjonene gir $g_2 = xTWx + AW^2x$ og $g_3 = xTW^{i-1}x + AW^ix$. På grunnlag av dette vil vi vise at I har en uendelig Gröbnerbasis på formen $G = \{g_i | g_i = xTW^{i-1}x + AW^ix, i \geq 1\}$. Som i forrige eksempel kan vi fastslå tuppen av g_i, siden

$$T \succeq W \Rightarrow xTW^{i-1}x \succeq xW^ix \Rightarrow xTW^{i-1}x \succ W^ix$$

Vi har g_1 i G. Anta at vi har g_k i G for alle $k \leq m$. Vi ser på g_i og g_j i G, der i og j er mindre eller lik m. Vi ønsker å vise at den eneste mulige overlappen mellom g_i og g_j gir oss et element på formen g_{i+j}, som ikke kan reduseres av de tidligere elementene i G. Da må G være en uendelig, redusert Gröbnerbasis for $\langle g_1 \rangle$.

Vi har fire muligheter for hvordan g_i kan overlappe g_j (for hvordan g_j kan overlappe g_i, bare bytt om på i og j). Tre av disse ((1)–(3)) er illustrert på figur 9.1, den siste (4) er der $\text{tip}(g_i)$ er en (tosidig) ekte delstreng av $\text{tip}(g_j)$. Vi ønsker å vise at (1) er *eneste* mulighet, ved å vise at (2), (3) og (4) strider mot en eller flere av egenskap (i)–(iii) for T og W.



Figur 9.1: Tre av fire mulige overlapper.

(2) Her finnes en x inne i W^{i-1}, hvilket medøfer at det finnes en x i W. Altså kan vi skrive $W = UxV$ for monomer U og V i $\mathcal{B}$, der U og V ikke kan være 1, siden W ikke starter eller slutter med x. Vi studerer de mulighetene vi har for T dersom dette er tilfelle:

(a) Vi har $T \succeq W$, så vi kan ha $T = VW^bxZ$ for en Z i $\mathcal{B}$ slik at $Z \neq 1$ og Z ikke slutter på x. Siden W ikke deler T, må vi ha $b = 0$ slik at $T = VxZ$.

(b) $T \succ V$, så $T = V$ er ikke en mulighet. Men vi kan ha $T = VZ$, der $W = ZX$ for et monom X.

Det finnes ikke flere muligheter. (a) gir oss $W \cdot xZ = Ux \cdot T$, som motsier (ii) i kravene for T og W. (b) gir tilsvarende $T \cdot X = V \cdot W$, også en motsigelse av (ii). Altså er ikke overlapp (2) mulig.

(3) Nå er x'en som er utgangspunkt for overlappen gjemt inne i T, slik at vi får $T = RxS$ for monomer R og S i $\mathcal{B}$, som begge er ulik 1. Igjen ser vi på mulighetene vi har for T.

(a) $T \succ S$ og $T \succeq W$, så vi kan ha $T = SxZ$, der $Z \neq 1$ og Z ikke slutter på x.

(b) $T \succ S$, så $T = S$ går ikke. Men vi kan ha $T = SZ$, der $W = ZX$ for et monom X.

(a) gir oss en overlapp i form av $T \cdot xZ = Rx \cdot T$, en motsigelse av (ii), som sier at T ikke kan overlappe seg selv. (b) gir $T \cdot Z = Rx \cdot T$, også en motsigelse. Dermed er heller ikke overlapp (3) mulig.

(4) Vi viser vi at $\text{tip}(g_{i-1})$ ikke deler $\text{tip}(g_{j-1})$ for noen $1 < i < j$. La oss anta det motsatte, nemlig at det finnes monomer U og V i $\mathcal{B}$ (minst én av U og V må være ulik 1) slik at $\text{tip}(g_{j-1}) = U \cdot \text{tip}(g_{i-1}) \cdot V$. Igjen deler vi problemet opp i flere tilfeller.

(a) $U = 1$. Dette gir $xTW^jx = xTW^ixV$ der $i < j$. Dette medfører $W^jx = W^ixV$, og hvis vi nå forkorter med W^i fra venstre på hver side, får vi $W^{i-j}x = xV$. Siden W ikke skal begynne med x, er dette en motsigelse.

(b) $U \neq 1$. Vi har $xTW^jx = UxTW^jxV$, som gir at $U = xU'$ der $U' \neq 1$ og U' ikke starter med x (siden T ikke starter med x). Vi forkorter med x fra venstre på hver side og får $TW^jx =$

$U'xTW^jxV$. Vi har to muligheter for T:

(b1) $T = X$, der $U' = XY$ for et monom Y i $\mathcal{B} \setminus \{1\}$.

(b2) $T = U'xS$, der $T = SR$ for et monom R i $\mathcal{B} \setminus \{1\}$.

Tilfelle (b1) gir $W^jx = YxTW^ixV$. Dersom $YxT = W^k$ for en $k < j$, får vi en overlapp mellom W og T. Hvis ikke, har vi $YxTF = W^k$ for en $k < j$ og et monom F slik at $W = FH$. Da overlapper W seg selv. Tilfelle (b2) gir at T overlapper seg selv ($T \cdot R = U'x \cdot T$). Altså er ikke (4) en gyldig overlapp.

Da står vi bare igjen med overlapp **(1)**. Det er klart at denne er gyldig, siden vi ikke trenger å gjøre noen antakelser om T og W. Vi beregner overlappsrelasjonen

$$O(g_i \cdot TW^{j-1}x, xTW^{i-1} \cdot g_j) = AW^ixTW^{j-1}x - AxTW^{i+j-1}x = h$$

For å redusere h, må vi vite hva som er tuppen av h. Begge leddene består av nøyaktig de samme variablene, men i forskjellige rekkefølger. Dessuten har vi ikke spesifisert noen tillatelig ordning, og vi vet hvilken variabel x symboliserer, eller hva W starter med. Altså må vi se på to muligheter for tuppen.

(a) $\text{tip}(h) = W^ixTW^{j-1}x$. Da kan vi redusere h med g_j og få

$$h \overset{\{g_j\}}{\to} -AxTW^{i+j-1}x + A^2W^{i+j}x = h'$$

Vi har $\text{tip}(h) = xTW^{i+j-1}x$. Vi ganger med konstanten $-A^{-1}$ og får $g_{i+j} = xTW^{i+j-1}x + AW^{i+j}x$.

(b) Tilsvarende får vi at dersom $\text{tip}(h) = xTW^{i+j-1}x$ kan vi også redusere med g_j, og vi får samme g_{i+j} som over.

Vi har $\text{tip}(g_{i+j}) = xTW^{i+j-1}x$. Det gjenstår å vise at g_{i+j} er et element i den *reduserte* Gröbnerbasisen for I. Vi har allerede vist at $\text{tip}(g_{i-1})$ ikke deler $\text{tip}(g_{j-1})$ for noen $1 < i < j$. Vi konkluderer derfor med at g_{i+j} ikke lar seg redusere med noen tidligere elementer. Den er altså et element i Gröbnerbasisen for I, og videre er G en redusert, uendelig Gröbnerbasis for dette idealet. ♣

Merk. Rais definisjon av overlapp mellom f og g ekskluderer overlapper der $\text{tip}(f) = l\text{tip}(g)r$, der $\text{tip}(f) \succ \text{tip}(g)$ og $l \neq 1 \neq r$. Dersom vi tar bort denne typen overlapper fra krav (ii) for T og W, kan vi fortsatt vise at det ønskede resultatet holder. Da bruker vi argumentasjonen for at overlapper på formen (4) ikke eksisterer til å vise at g_{i+j} er (tupp)redusert.

Liknende eksempler

Rai [23] viser at idealene $\langle xyx - xy\rangle$ i $k\langle x, y\rangle$ og $\langle xTx + AxW\rangle$ i $k\langle x_1, \ldots, x_n\rangle$, med samme krav på T og W som over, også har uendelige Gröbnerbasiser på en bestemt form. Med symmetriske argumenter kan man også vise det samme om $\langle xyx - yx\rangle$ i $k\langle x, y\rangle$ og $\langle xyx + Azx\rangle$ i $k\langle x, y, z\rangle$.

Hvorfor man ikke bør bygge kryptosystemer på disse idealene

Ingen av disse idealene oppfyller kravene til en offentlig nøkkel gitt i avsnitt 7.3, siden de har forutsigbar Gröbnerbasis med jevnt økende tupper. Så hvorfor har vi tatt oss bryet med dette kapitlet, dersom kryptosystemer basert på disse idealene vil være usikre? For det første fordi de foregående avsnittene illustrerer hvordan man kan vise at et ideal har uendelig Gröbnerbasis. For det andre skal vi bruke dem som grunnlag for eksemplene i neste avsnitt. Men som vi snart skal se, så finnes det også idealer med uendelig Gröbnerbasis av en atskillig mindre forutsigbar art.

9.3 Bytte av tillatelig ordning og strengomskrivingsystemer

I dette avsnittet gir vi noen eksempler på teknikker som, i de gitte eksemplene, gjør det mulig å beregne normalformer i I uten noen kjennskap til Gröbnerbasisen for I under den gitte ordningen. Vi har ikke greid å generalisere noen av disse «angrepene», men det første av dem er relativt lett å unngå. Det andre kan synes noe mer uforutsigbart.

Vi starter med å se på *bytte av tillatelig ordning*. Anta at vi har gitt et ideal I og en tillatelig ordning $\prec$, slik at det er stor sannsynlighet for at den reduserte Gröbnerbasisen G for I med hensyn til $\prec$ er uendelig. Dersom det finnes en annen tillatelig ordning $\prec_1$ som gir en endelig Gröbnerbasis G_1 for I, er det mulig å bruke denne til å finne normalformen av et polynom i R med hensyn til den opprinnelige ordningen?

Eksempel 9.3.1. Vi har vist at idealet $I = \langle xyx - 3xz \rangle$ i $\mathbb{F}_7 \langle x, y, z \rangle$ har uendelig Gröbnerbasis for en ordning $\prec$ der $y \geq z$. Vi skifter til en vektordning $\prec_1$ der $w(x) = w(y) = 1$ og $w(z) = 3$. Dette gir at $xz \succ xyx$, så

$$g = ((-3)^{-1}(xyx - 3xz) = xz + 2xyx$$

genererer I. Siden g ikke har noen selvoverlapper, er $G_1 = \{g\}$ en Gröbnerbasis for I. Vi ser nå på en f i R, der

$$f = xzzzyxxzyx$$

Siden vi kjenner formen på Gröbnerbasisen G for I under den opprinnelige ordningen, kan vi beregne at

$$N_{\prec}(f) = 2xzzzzxzz$$

Vi bruker nå G_1 til å finne $N_{\prec_1}(f)$.

$$N_{\prec_1}(f) = 2xyxyxyxyxxyxyx$$

Hvis vi nå reduserer $N_{\prec_1}(f)$ *fra høyre* med $xyx - 3xz$, det opprinnelige generatorpolynomet for I, får vi

$$N_{\prec_1}(f) \longrightarrow 2xzzzzxzz = N_{\prec}(f)$$

Som nevnt over har vi ikke bekreftet at denne framgangsmåten alltid gjelder. Imidlertid er det mulig å finne idealer som har uendelig Gröbnerbasis under *alle* tillatelige ordninger – vi ser på et slikt ideal i neste avsnitt. Videre utforskninger av idealer med uendelig Gröbnerbasis kan leses i [17].

Den neste teknikken er såkalte *strengomskrivingsystemer* (String Rewrite Systems, SRS). Vi skal ikke gå så nærme inn på SRS i denne oppgaven, men man kan være klar over at SRS er nært beslektet med Gröbnerbasiser for ikke-kommutative binomidealer (se [18, kapittel 2.6]). Kort fortalt er et strengomskrivingsystem $\mathcal{R}$ over $\mathcal{B}$ et sett med *omskrivingsregler* på formen $l \to r$, der l og r er monomer i $\mathcal{B}$. Vi viser med eksempelet under hvordan vi, ved å innføre en ny variabel og et slikt system bestående av én regel, kan finne en endelig Gröbnerbasis – om enn for «feil» ideal – som vi bruker til å finne normalformer på omtrent samme måte som i forrige eksempel.

Eksempel 9.3.2. Vi ser på $I = \langle g = xTx + AWx \rangle$, et ideal i $k\langle x_1, \ldots, x_n \rangle$ under forutsetningene gitt i forrige avsnitt. Dette idealet har en uendelig Gröbnerbasis G for en ordning $\prec$ der forutsetningene holder.

Vi innfører nå en ny variabel $z \notin \{x_1, \ldots, x_n\}$ der $z \prec W$ og $xz \prec Wx$, og lar $\mathcal{B}_1$ være alle strenger i $(\{x_1, \ldots, x_n\} \cup \{z\})^*$. Vi definerer et strengomskrivingsystem på $\mathcal{B}_1$ med én regel:

$$Tx \to z$$

Vi setter $g_1 = Tx - z$ i ringen $R_1 = k\langle x_1, \ldots, x_n, z \rangle$, og lar J være idealet i R_1 generert av $\{g, g_1\}$. Vi har

$$g = xTx + AWx \xrightarrow{\{g_1\}} xz + AWx$$

der $\operatorname{tip}(xz + AWx) = Wx$. Vi setter $g_2 = Wx + A^{-1}xz$, så

$$\langle g, g_1 \rangle = \langle g_1, g_2 \rangle$$

Siden W og T ikke overlapper, er $G_1 = \{g_1, g_2\}$ en redusert Gröbnerbasis for J.

La f være elementet xTW^2xTx i R. Vi vet at

$$N_{R/I}(f) = A^2W^4x$$

Dersom vi overfører f til R_1 og reduserer modulo G_1, får vi

$$N_{R_1/J}(f) = A^{-2}xzzzz$$

Hvis vi «tilbakeskriver» omskrivingen $Tx \to z$, det vil si at vi setter $z = Tx$, får vi

$$A^{-2} \to A^{-2}xTxTxTxTxTx \in R$$

Vi reduserer nå dette polynomet *fra venstre* (motsatt av forrige eksempel) med $g = xTx + AWx$ og får

$$A^{-2}xTxTxTxTxTx \xrightarrow{\{g\}} A^2W^4x = N_{R/I}(f)$$

Andre eksempler gir riktig resultat dersom vi avslutningsvis reduserer *fra høyre*.

Strengomskrivingsystemer er grundig beskrevet i [3], dog uten referanser til Gröbnerbasiser. Moras bevis for at det er umulig å finne en generell algoritme for å avgjøre hvorvidt et ideal har endelig Gröbnerbasis (se kapittel 6) bygger også på resultater fra strengomskrivingteori.

9.4 Et mer egnet ideal med uendelig Gröbnerbasis

Siden hovedidealene i avsnitt 9.2 viste seg å være uegnet til bruk i Polly Cracker-kryptosystemer, må vi, dersom vi vil følge Rais prinsipper, se oss rundt etter andre idealer med uendelig, og også uforutsigbar, Gröbnerbasis. Rai har lagt stor vekt på følgende resultat:

Proposisjon 9.4.1. *La $I = \langle g_1, g_2 \rangle$ være et ideal i $k\langle x, y, z\rangle$ gitt ved $g_1 = xzy + yz$ og $g_2 = yzx + zy$. Uansett hvilken tillatelig ordning vi velger, har I en uendelig Gröbnerbasis.*

Idé for beviset. Hele beviset for proposisjon 9.4.1 er gitt hos Rai [23, s. 29–34]. Her skal vi nøye oss med å beskrive ideen bak, og gå gjennom et av tilfellene i beviset. Merk at dette tilfellet er symmetrisk med det første av tilfellene som er gjennomgått hos Rai, men hos sistnevnte finnes en feil, som er rettet opp i den nedenstående teksten.

For å bevise at I har uendelig Gröbnerbasis under *alle* tillatelige ordninger, må vi dele problemet i tre tilfeller:

(1) $\text{tip}(g_1) = xzy$ og $\text{tip}(g_2) = yzx$

(2) $\text{tip}(g_1) = xzy$ og $\text{tip}(g_2) = zy$

(3) $\text{tip}(g_1) = yz$ og $\text{tip}(g_2) = yzx$

Merk at vi *ikke* kan ha $\text{tip}(g_1) = yz$ og $\text{tip}(g_2) = zy$, siden dette går imot kriteriene for tillatelige ordninger: Anta dette er tilfelle. Siden $\text{tip}(g_1) = yz \succ xzy$, er $yz \succ zy$. Men $\text{tip}(g_2) = zy \succ yzx$, så $zy \succ yz$. Siden $zy \neq yz$, har vi en motsigelse.

Altså har vi tre tilfeller vi må se på. Det holder å vise at i hvert av disse tilfellene finnes det en uendelig kjede av elementer i I, hvis tupper ikke er delelige med en endelig delmengde av $\text{Tip}(I)$. (At $\text{tip}(f)$, for et polynom f i $k\langle x, y, z\rangle$, er delelig med en endelig mengde monomer, $X \subseteq \mathcal{B}$, er det samme som at $\text{tip}(f)$ reduserer til 0 modulo X, altså $\text{tip}(f) \xrightarrow{X} 0$).

Hvorfor er dette tilstrekkelig? Anta at det finnes en slik kjede av polynomer, $F = \{f_1, f_2, \ldots\}$, og at vi også har en endelig redusert Gröbnerbasis G for I. Da har vi

$$\begin{aligned} \langle \text{Tip}(G)\rangle &= \langle \text{Tip}(I)\rangle \text{ fra definisjonen av Gröbnerbasis, og} \\ \text{Tip}(F) \subseteq \text{Tip}(I) &\subseteq \langle \text{Tip}(I)\rangle \text{ siden } F \subseteq I \end{aligned}$$

Dette gir $\text{Tip}(F) \subseteq \langle \text{Tip}(G)\rangle$, det vil si at for en vilkårlig f i F, finnes monomer a_{ij} og b_{ij} i $\mathcal{B}$ slik at

$$\text{tip}(f) = \sum_{i=1}^{|G|} \sum_{j=1}^{t_i} a_{ij}\text{tip}(g_i)b_{ij}, \text{ eller } \text{tip}(f) \xrightarrow{\text{Tip}(G)} 0$$

Altså har vi at $\text{tip}(f)$ *er* delelig med en endelig delmengde av $\text{Tip}(I)$, en motsigelse til hvordan F er definert. Da må antakelsen om at I har en endelig Gröbnerbasis være feil.

La oss se på hvordan vi kan gå fram for tilfelle (1), der $\text{tip}(g_1) = xzy$ og $\text{tip}(g_2) = yzx$. For å finne den ønskede uendelige kjeden F, beregner vi noen overlapper i I. Vi skal ikke forsøke å beregne *alle*, og vi skal heller ikke prøve å redusere noen underveis, og det er heller ikke mulig å si hva som er tuppen i de nye polynomene uten å gjøre noen flere antakelser. Det vesentlige

er at disse polynomene er elementer i I, og det er de uavhengig av ordning. I beregningene bruker vi lengde-leksikografisk ordning med $x \succ y$.

$$
\begin{aligned}
O(g_1 \cdot zx, xz \cdot g_2) &= yzzx - xzzy \implies g_3 = xzzy - yzzx \\
O(g_3 \cdot zx, xzz \cdot g_2) &= -yzzxzx - xzxxy \implies g_4 = yzzxzx + xzzzy \\
O(g_1 \cdot z^2xzx, xz \cdot g_4) &= yzzzxzx - xzxzzzy \implies g_5 = (xz)^2z^2y - yz^2(zx)^2 \\
O(g_5 \cdot zx, (xz)^2z^2g_2) &= -yz^2(zx)^3 - (xz)^2z^3y \implies g_6 = yz^2(zx)^3 + (xz)^2z^3y
\end{aligned}
$$

Vi ser at vi får polynomer på formen

$$
\begin{aligned}
g_{2n} &= yz^{n-1}(zx)^n + (xz)^{n-1}z^ny \text{ for } n \geq 2 \\
g_{2n+1} &= (xz)^nz^ny - yz^n(zx)^n \text{ for } n \geq 2
\end{aligned}
$$

At alle slike g_{2n} og g_{2n+1}, med $n \geq 2$, er i I, kan vises med induksjon (slik vi gjorde med hovedidealene i de foregående avsnittene).

Det er tilstrekkelig å vise at én av de uendelige kjedene over har den egenskapen vi er ute etter. Med en vilkårlig tillatelig ordning $\succ$, der $xzy \succ yz$ og $yzx \succ zy$, kan både første og andre ledd i disse polynomene være tuppen. La oss se på $F = \{g_{2n} | n \geq 2\}$, der $\text{tip}(g_{2n}) = yz^{n-1}(zx)^n$ for $n \geq 2$. Målet er å vise at elementene i F ikke lar seg dele med en endelig delmengde av $\text{Tip}(I)$.

Anta at vi har en endelig delmengde $X \subseteq \text{Tip}(I)$ slik at for alle g_{2n} i F, finnes en p' i X slik at $\text{tip}(g_{2n}) = up'v$ med u og v i $\mathcal{B}$. Siden F er uendelig og X er endelig, må det finnes (minst) et element p i X der p deler uendelig mange elementer i $\text{Tip}(F)$ (hvis ikke, er F endelig eller X uendelig). For at p skal dele uendelig mange monomer på formen $yz^{n-1}(zx)^n$, må p være en av følgende:

(a) $p = yz^k, k \geq 0$

(b) $p = z^k(zx)^mz^l$, minst én av k og m ulik 0 og $l \in \{0, 1\}$

Vi ser raskt at p ikke kan ha formen (b) – siden alle ledd i g_1 og g_2 inneholder y, må alle monomer i I og dermed i $\text{Tip}(I)$ også gjøre det. Altså har vi $p = yz^k$ for en $k \geq 0$. Alle monomer i I må av tilsvarende grunn som over inneholde z, og dermed har vi $k \geq 1$.

Siden p er i $\text{Tip}(I)$, finnes monomer u_{ij} og v_{ij} i $\mathcal{B}$ og koeffisienter α_{ij} i k slik at

$$
f = \sum_{i=1}^{2} \sum_{j=1}^{s_i} \alpha_{ij}u_{ij}g_iv_{ij} \text{ og } \text{tip}(f) = p \tag{9.1}
$$

Vi forsøker å finne to endelige mengder av tupler $W_i = \{(u_{ij}, v_{ij})\}, i \in \{1, 2\}$ og $j \in \{1, 2, \ldots, s_i\}$, og dermed f som i (9.1), slik at $\text{tip}(f) = p$. Vi ser at for å få med leddet yz^k, må vi ha med $(1, z^{k-1})$ i W_1, og da får vi også med leddet $xzyz^{k-1}$ i f. Siden $xzy \succ yz$, er $xzyz^{k-1} \succ yz^k$, og siden $xzyz^{k-1}$ ikke er tuppen av f, må dette leddet kanselleres i f.

Vi prøver å kansellere ved hjelp av g_1, ved å inkludere (xz, z^{k-2}) i W_1. Dette gir oss foreløpig et polynom som inneholder

$$
1 \cdot g_1 \cdot z^{k-1} - xz \cdot g_1 \cdot z^{k-2} = -(xz)^2yz^{k-2} + yz^k \tag{9.2}
$$

For å avgjøre hva som er det største leddet av disse to, og til å hjelpe oss senere, trenger vi følgende lemma:

Lemma 9.4.2. *Gitt en tillatelig ordning $\prec$ slik at $xzy \succ yz$ og $yzx \succ zy$. Da er*
(a) *$(xz)^m yz^{k-m} \succ yz^k$ for alle $k \geq 1$ og $1 \leq m \leq k$*
(b) *$(xz)^{m-1}x^n yzx^n z^{k-m} \succ yz^k$ for alle $k \geq 1$ og $1 \leq m \leq k$ samt $n \geq 1$*
(c) *$(xz)^{m-1}x^l zyx^n z^{k-m} \succ yz^k$ for k, m som over og $l \geq 2$ og $n = l - 1$*

Bevis. **(a)** Det er klart at det holder for $m = 1$ (se tidligere). Anta, for en $m \geq 1$, at $(xz)^m yz^{k-m} \succ yz^k$. Vi ser på $m + 1$. Kriterium (i) for tillatelige ordninger (definisjon 3.3.1) gir:

$$\begin{aligned} & xzy \succ yz \\ \implies & (xz)^{m+1}y \succ (xz)^m yz \\ \implies & (xz)^{m+1}yz^{k-m-1} \succ (xz)^m yz^{k-m} \succ yz^k \end{aligned}$$

Altså holder det for alle $1 \leq m \leq k$.

(b) og **(c)** Vi bruker først (a) og at $yzx \succ zy$ til å vise at (b) holder for $n = 1$:

$$\begin{aligned} & yzx \succ zy \\ \implies & (xz)^{m-1}x \cdot yzx \cdot z^{k-m} \succ (xz)^{m-1} \cdot zy \cdot z^{k-m} = (xz)^{m-1}yz^{k-m} \overset{(a)}{\succ} yz^k \\ \implies & (xz)^{m-1}xyzxz^{k-m} \succ yz^k \end{aligned}$$

Det vil si at (b) holder for $n = 1$. Deretter bruker vi dette, samt at $xzy \succ yz$, til å vise at (c) holder for $n = 1$ og $l = n + 1 = 2$.

$$\begin{aligned} & xzy \succ yz \\ \implies & (xz)^{m-1}x \cdot xzy \cdot xz^{k-m} \succ (xz)^{m-1}x \cdot yz \cdot xz^{k-m} \overset{(b),n=1}{\succ} yz^k \\ \implies & (xz)^{m-1}x^2 ztxz^{k-m} \succ yz^k \end{aligned}$$

Så (c) holder for $l = 2$ og $n = 1$. Ved induksjon kan man vise at dersom (c) holder for l og n, holder (b) for $l = n + 1$, og at dersom (b) holder for n. holder (c) for n og $l = n + 1$. ♣

Vi må altså, som en følge av (a) i lemmaet over, sørge for at leddet $(xz)^2 yz^{k-2}$ i (9.2) kansellerer i f. Vi forsøker å bare bruke g_1, og må da legge til $((xz)^2, z^{k-3})$ i W_1. Da får vi også med $(xz)^3 yz^{k-3} + yz^k$ i f, og fra lemmaet over vet vi at det første leddet er større enn det andre og må derfor kanselleres. Vi ser at dersom vi skal fortsette å bare bruke g_1, vil vi fortsette å legge til $((xz)^{m-1}, z^{k-m+1})$ i W_1. Dette stopper til slutt med $((xz)^{k-1}, 1)$, som gir leddet $(xz)^k y$, som også er større enn yz^k. Altså kan vi ikke uttrykke f kun ved g_1. Vi må bruke g_2 til å kansellere et ledd på formen $(xz)^m yz^{k-m}$. Den eneste måten dette kan gjøres på, er å legge til $((xz)^{m-1}x, z^{k-m})$ i W_2 og få

$$(xz)^{m-1}xyzxz^{k-m} + yz^k \tag{9.3}$$

Fra lemmaet over vet vi at det første leddet er større enn det andre, og det må derfor kanselleres. Det lar seg ikke gjøre med g_2, så vi må tilbake til g_1 igjen. Vi legger til $((xz)^{m-1}x, xz^{k-m})$ i W_1, og får med leddet $(xz)^{m-1}x^2 zyxz^{k-m}$, som fra lemmaet er større enn yz^k. Dette leddet

kan bare kanselleres ved hjelp av g_2, så vi legger til $((xz)^{m-1}x^2, xz^{k-m})$ i W_2. Nå må vi kvitte oss med $(xz)^{m-1}x^2yzx^2z^{k-m}$, og det må vi gjøre med g_1.

Vi ser at vi må bruke g_1 og g_2 annenhver gang, og hver gang må vi legge til en x henholdsvis til høyre i u_{1j} og til venstre i $v_{2(j+1)}$. Siden representasjonen av f skulle være en endelig sum, har vi en motsigelse, siden dette mønsteret fortsetter i all evighet. Det finnes altså ingen p som deler uendelig mange elementer i $\text{Tip}(F)$.

Videre må man gjøre tilsvarende for tilfellet der $\text{tip}(g_{2n}) = (xz)^{n-1}z^ny$, og deretter gå tilbake til de andre alternativene for $\text{tip}(g_1)$ og $\text{tip}(g_2)$. ♠

Det viser seg at når man begynner å beregne elementer i en Gröbnerbasis for $I = \langle g_1, g_2\rangle$, blir den langt mindre forutsigbar enn kjeden av elementer i I som vi så på over. Ikke bare kommer det elementer på stadig nye former etter hvert som man beregner flere, men også lengden på tuppene øker og minker (i motsetning til å bare øke, som var tilfelle for idealene vi har sett på tidligere). Noen av mønstrene i G er gjengitt hos Rai, som for eksempel

- $yz^nx(zx)^{n-1} + (xz)^{n-1}z^ny$, for $n > 1$
- $zyzy^{n-1}x + y^{n-2}yzzy$, for $n > 1$
- $yzzyx^nzx + x^nyzzy$, for $n \geq 1$

Dette lover bra for bruk av dette idealet i kryptografi. Resultatet kan generaliseres noe med følgende korollar:

Korollar 9.4.3. *Gitt tre polynomer A, B og C i $k\langle x_1, \ldots, x_n\rangle$ med*

$$A = \prod_{i=1}^{n} x_i$$
$$B = x_1(\prod_{i=2}^{n-1} \rho(x_i))x_n$$
$$C = x_1(\prod_{i=2}^{n-1} \sigma(x_i))x_n$$

der ρ og σ er ulike, ikke-trivielle permutasjoner på $\{x_2, \ldots, x_{n-1}\}$. La $g_1 = ACB + BC$ og $g_2 = BCA + CB$. Da har ikke $I = \langle g_1, g_2\rangle$ en endelig Gröbnerbasis under noen tillatelige ordninger.

Beviset følger direkte fra 9.4.1, siden A, B og C overlapper verken seg selv eller hverandre.

9.5 Rais formodning

Denne formodningen, som ikke er bevist av Rai, danner grunnlaget for det tilsynelatende sikreste av kryptosystemene i Rais avhandling.

Formodning 9.5.1. *La $F = \{f_1, \ldots, f_m\}$ være en endelig delmengde av $R = k\langle x_1, \ldots, x_n\rangle$ slik at $\text{tip}(g_i) = T$ for alle $1 \leq i \leq m$, og la $\alpha = l(T)$. Sett N til å være antall strenger (monomer) av lengde $\alpha - 1$ som finnes i samtlige av g_i'ene. Anta at følgende krav er oppfylt:*

(i) $m \approx \frac{N}{2}$

(ii) N *er tilnærmet* $\frac{1}{3}$ *av og strengt mindre enn* $\frac{1}{2}$ *av det totale antallet mulige strenger av lengde* $\alpha - 1$*, det vil si*

$$N \approx \frac{1}{3}n^{\alpha-1} \text{ og } N < \frac{1}{2}n^{\alpha-1}$$

Da er det stor sannsynlighet for at den reduserte Gröbnerbasisen for $\langle F \rangle$ *er uendelig.*

Formodningen fokuserer på lengden av ord, og Rai skriver videre at elementene i F er på formen

$$f_l = T + \sum_{i=1}^{N} a_{li} W_i + \sum_{j=1}^{s_l} b_{lj} V_{lj}$$

der a_{il} og b_{jl} er i k, og $l(T) = \alpha$, $l(W_i) = \alpha - 1$ for alle l, og de N W_i'ene finnes i alle g_k'ene slik at N oppfyller kravene i formodningen. De resterende monomene i f_l har de ifølge Rai lengde mindre eller lik α, men med en vektordning kan det godt finnes V_{li}'er med lengde større enn α selv om $T \succ V_{li}$ for alle i og l. Dette er imidlertid ikke av praktisk betydning.

Når vi reduserer F på jakt etter en Gröbnerbasis, vil bare maks ett element beholde T som tupp. Tuppen hos de resterende $m - 1$ elementene er meget sannsynlig en av W_i'ene, og siden omtrent $1/3$ av alle mulige ord av lengde $\alpha - 1$ er blant disse, er det stor sannsynlighet for at de inneholder overlapper. Videre vil neppe alle disse redusere til 0, siden N er begrenset slik at mindre enn halvparten av de mulige ordene opptrer.

Hva som ender opp som ny tupp etter at vi har redusert bort den opprinnelige, er avhengig av monomene i elementet som *ikke* er blant de N felles-monomene. Formodningen stiller ingen krav til disse, og de bør derfor ikke ha noen betydning. Et stort eksempel hos Rai [23, s. 37/Appendix A] understøtter dette. Koeffisientene som benyttes i elementene i F, derimot, bør være helt tilfeldige, ellers kan vi ende opp med tilfeller som det følgende:

Eksempel 9.5.2. Det finnes idealer som oppfyller kravene i formodningen og likevel har endelig redusert Gröbnerbasis. Se for eksempel på idealet $I = \langle F \rangle \subseteq k\langle x, y, z \rangle$, lengde-leksikografisk ordning med $x > y > z$ og $F = \{f_1, f_2, f_3, f_4\}$ der

$$\begin{aligned}
f_1 &= xyxz + xyy + xzy + yxz + yyy + yzy + zyx + zyz \\
f_2 &= xyxz + 2xyy + xzy + yxz + yyy + yzy + zyx + zyz \\
f_3 &= xyxz + 3xyy + xzy + yxz + yyy + yzy + zyx + zyz \\
f_4 &= xyxz + 4xyy + xzy + yxz + yyy + yzy + zyx + zyz
\end{aligned}$$

Vi har $n = 3$ og $\text{Tip}(F) = \{xyxz\}$, så $\alpha = 4$. Det er $N = 8$ monomer av lengde $\alpha - 1 = 3$ som er med i alle f_i'ene, og vi ser at $N = 8 \approx \frac{1}{3}3^{4-1} = 9$ og $N < \frac{1}{2}3^{4-1} = 13,5$ og $m = 4 = \frac{N}{2}$. Men når vi reduserer mengden F, får vi følgende:

$$\begin{aligned}
&f_2 \xrightarrow{\{f_1\}} xyy = f_2' \\
&f_3 \xrightarrow{\{f_1\}} 2xyy \xrightarrow{\{f_2\}} 0 \text{ og } \quad f_4 \xrightarrow{\{f_1\}} 3xyy \xrightarrow{\{f_2\}} 0
\end{aligned}$$

Og siden mengden $\{f_1, f_2'\}$ er redusert, og $\text{tip}(f_1)$ og $\text{tip}(f_2')$ ikke overlapper, har I en endelig, redusert Gröbnerbasis.

10 Noen kryptosystemer basert på resultatene i kapittel 9

10.1 Innledning

Dette kapitlet er en gjennomgang av Rais forslag til kryptosystemer [23]. Systemene bygger på resulatene fra forrige kapittel, og sikkerheten er basert på prøving og feiling snarere enn matematiske beviser. For alle systemene ser vi raskt på hvordan vi kan sikre at

- den hemmelige nøkkelen ikke kan skaffes ved hjelp av den offentlige
- krypteringen kan gjøres på en slik måte at det kun er ved hjelp av den hemmelige nøkkelen det er mulig å dekryptere korrekt

I avsnitt 10.5 kaster vi et kritisk blikk på de potensielt sikre av disse systemene.

Alle beregninger foregår, dersom annet ikke er nevnt, i polynomringen $R = \mathbb{F}\langle x_1, \ldots, x_n\rangle$ der $\mathbb{F}$ er en endelig kropp. Som før har vi $\mathcal{B} = \{x_1, \ldots, x_n\}^*$

10.2 Systemer basert på proposisjon 9.4.1

Vi ønsker å ha en offentlig nøkkel der egenskapene til idealet $\langle xzy + yz, yzx + zy\rangle$ blir utnyttet. Å sette $q_1 = xzy + yz$ og $q_2 = yzx + zy$ som offentlig nøkkel er imidlertid dumt – siden den hemmelige nøkkelen g skal være en faktor i begge disse, må vi dermed ha $g = z$ eller $g = y$, og dette er uheldig siden g da kan leses rett av fra den offentlige nøkkelen. La oss derfor heller velge et hemmelig punkt i $\mathbb{F}^3$, si (a, b, c), og sette $g = z - c$, $f = x - a$ og $h = y - b$. La $G = \{g\}$. Det er klart at G er Gröbnerbasis for $I = \langle G\rangle$. La videre

$$q_1 = fgh + hg = xzy + yz - cxy - bxz - azy + bcx + (ac - c)y + (ab - b)z + (bc - abc)$$
$$q_2 = hgf + gh = yzx + zy - bzx - cyx - ayz + bcx + (ab - b)z + (ac - c)y + (bc - abc)$$

Idealet $\langle B\rangle$ har en uendelig redusert Gröbnerbasis [23, s. 42].

Å la den hemmelige nøkkelen være et element på denne formen ($g = z - c$) gjør dekrypteringen svært rask: Sett $z = c$ i den krypterte meldingen. Det at nøklene er bestemt av et punkt i $\mathbb{F}^n$ minner oss om Koblitz' og Fellows' opprinnelige Polly Cracker-systemer.

Merk. Som vi så i kapittel 8 er noen av angrepene på ikke-kommutativ Polly Cracker avhengig av å kjenne formen på den hemmelige nøkkelen. Dersom, i dette tilfellet, formen på G er kjent, og bare konstanten c er ukjent, er dette systemet fullstendig ubrukelig, siden c kan leses av som koeffisienten til xy i q_1 og yx i q_2.

Skjule den hemmelige nøkkelen

Vi har allerede slått fast at dersom konstruksjonen av G og B er kjent (noe det er rimelig å anta at er tilfellet), er ikke G sikret i det hele tatt. Imidlertid går det an å modifisere polynomene g, f og h slik at dette problemet forsvinner. Istedenfor å bruke moniske polynomer, velger vi konstanter a', b' og c' i $\mathbb{F}$ og setter $g = cz - c'$, $f = ax - a'$ og $h = by - b'$. Systemet en kryptanalyst må løse for å finne c og c' er ikke lineært. Merk at dekrypteringen fortsatt er effektiv, da vi setter $z = c^{-1}c'$.

Sikker kryptering

Meldingsrommet M vil være en delmengde av $\{f \in R | f \xrightarrow{\{g\}} f\} = \{f \in R |$ ingen monomer i f inneholder $z\}$. Vi krypterer en melding m fra M ved å velge polynomer F_{ij} og H_{ij} i R og sette c til en endelig sum

$$c = \sum\sum F_{ij} q_i H_{ij} + m$$

Et problem som melder seg, observert av Rai gjennom testing, er at med mindre $\text{tip}(c)$ er rimelig stor, finner vi ofte riktig m ved å redusere c med B. Et annet problem er at en delvis Gröbnerbasis for I også vil kunne gi riktig m. Rai introduserer tre metoder for kryptering som sikrer at $\bar{c}^B$ er ulik m:

(1) Velg F_{ij} og H_{ij} slik at $\text{tip}(c)$ blir på en form der mønsteret $\ldots xzyzxzyzxzy \ldots$ gjentar seg. Vi ser at hver gang vi velger å redusere med q_1, velger vi samtidig bort en mulighet for å redusere med q_2, og siden B ikke er en Gröbnerbasis, vil de ulike valgene gi forskjellig sluttresultat. For å få mange nok slike valg til at sannsynligheten for å slumpe til å velge riktig hele veien blir liten, må $\text{tip}(c)$ være ganske stor.

(2) Vi utvider B med elementer $\{q_3, \ldots, q_r\}$ hvis tupper har «overlappende overlapper» – det vil si at for enhver q_i i $B \setminus \{q_1, q_2\}$ finnes q_j og q_k i B slik at

$$\begin{aligned} \text{tip}(q_i) &= W_1 W_2 W_3 \\ \text{tip}(q_j) &= W_2 W_3 W_4 \\ \text{tip}(q_k) &= W_3 W_4 W_5 \end{aligned}$$

der W_i er i $\mathcal{B}$ for $1 \leq i \leq 5$. Når vi krypterer, velger vi en q_i i B og tilhørende q_i og q_j, og setter

$$c = \alpha q_i W_4 W_5 + \beta W_1 q_j W_5 - (\alpha + \beta) W_1 W_2 q_k$$

der α, β er i k. Vi ser at leddene som inneholder tuppene av q_i, q_j og q_k kanselerer, og reduksjon med B vil ikke føre fram.

(3) Vi velger en ny offentlig nøkkel $Q \subseteq J = \langle B \rangle$, der $Q = \{F_1, \ldots, F_t\}$ slik at $\text{tip}(f_i) = T$ for alle $1 \leq i \leq t$ og en T i $\mathcal{B}$. For kryptering velger vi konstanter $\alpha_1, \ldots, \alpha_t$ i $\mathbb{F}$ slik at $\sum_{i=1}^{t} \alpha_i = 0$, og setter

$$c = \sum_{i=1}^{t} \alpha_i F_i + m$$

Dette kan skrives om til

$$c = \sum_{i=1}^{t} \alpha_i T + \sum_{i=1}^{t} \alpha_i \text{tail}(F_i) + m = \sum_{i=1}^{t} \alpha_i \text{tail}(F_i) + m$$

det vil si at tuppene av F_i'ene ikke finnes i c, og reduksjon med Q gir ikke m.

Til slutt forkaster vi hele dette systemet, siden Rais forsøk med Opal [18] viser at en delvis Gröbnerbasis kan redusere c til korrekt m, uansett valg av krypteringsteknikk.

10.3 Systemer basert på korollar 9.4.3

Vi utvider systemet fra forrige avsnitt med tanke på korollar 9.4.3, til et system over $k\langle x_1, \ldots, x_n\rangle$. La

$$Z = \prod_{i=1}^{n} x_i$$
$$X = x_1(\prod_{i=2}^{n-1} \rho(x_i))x_n$$
$$Y = x_1(\prod_{i=2}^{n-1} \sigma(x_i))x_n$$

der ρ og σ er ulike, ikke-trivielle permutasjoner på $\{x_1, \ldots, x_n\}$. La videre a_i, b_i og c_i være i $\mathbb{F} \setminus \{0\}$ for $0 \leq i \leq n$, og definer

$$f = X + \sum_{i=1}^{n} a_i x_i + a_0,\ h = Y + \sum_{i=1}^{n} b_i x_i + b_0 \text{ og } g = Z + \sum_{i=1}^{n} c_i x_i + c_0$$

Vi setter g til hemmelig nøkkel, og som offentlig nøkkel setter vi $B = \{q_1, q_2\}$ der $q_1 = fgh + hg$ og $q_2 = hgf + gh$. Fra korollar 9.4.3 vet vi at $J = \langle B\rangle$ har uendelig Gröbnerbasis.

Vi har mange alternativer for meldingsrommet. Rai foreslår $M = \{$alle lineære polynomer i $\mathbb{F}\langle x_1, \ldots, x_n\rangle\}$.

Skjule den hemmelige nøkkelen

Er det mulig å lese av konstantene $c_0, \ldots, c_n$ fra q_1 og q_2? Blant leddene i q_1 finnes c_0XY og c_iXx_iY for alle $1 \leq i \leq n$, så svaret på dette spørsmålet er *ja*. Derfor må vi, som i forrige avsnitt, sette tilfeldige konstanter foran X, Y og Z, slik at vi får et ikke-lineært system. Alle leddene i q_1 og q_2 får da en koeffisient som er et produkt av tre elementer i $\mathbb{F}$.

Sikker kryptering

I motsetning til forrige system, har ikke Rai fått til å lage en delvis Gröbnerbasis for J ($n = 6$, eksperimentering i Opal, [23, s. 44]). Dermed kan ikke krypteringsmetode **(2)** fra forrige avsnitt brukes, men metode **(1)** og **(3)** fungerer også i dette tilfellet. Altså har vi kanskje oppnådd et anvendelig system, så lenge sikkerhetskravene i kapittel 8 er innfridd.

10.4 Systemer baser på formodning 9.5.1

Vi skal se på systemer der den hemmelige nøkkelen er på formen $G = \{g\} = \{W + \sum_{i=1}^{n} a_i x_i + a_0\}$ der a_i er i $\mathbb{F}$ for $0 \leq i \leq n$, og W er en streng i $\mathcal{B}$ som inneholder samtlige $x_1, \ldots, x_n$ uten å overlappe seg selv. Det er klart at $\text{tip}(g) = W$, uavhengig av hvilken tillatelig ordning som er valgt, og at G er en Gröbnerbasis for $I = \langle G \rangle$.

Den offentlige nøkkelen er på formen $B = \{q_i = f_i g h_i | f_i, h_i \in R\}_{i=1}^{s}$ der f_i og h_i er slik at idealet $\langle B \rangle$ oppfyller kravene i formodning 9.5.1. La $\text{tip}(f_i) = W_f$ for alle f_i, og $\text{tip}(h_i) = W_h$ for alle h_i. La $T = W_f W W_h$. Vi ser at $\text{Tip}(B) = \{T\}$, altså at alle q_i'ene har samme tupp. Som meldingsrom kan vi velge for eksempel alle lineære polynomer i R, eller alle polynomer i én bestemt variabel, begrenset av en grad D for en D i $\mathbb{N}$.

Når man skal konstruere f_i'ene og h_i'ene slik at alle kravene holder, er det enklest å ha alle f_i'ene på samme form og alle h_i'ene på samme form, og kun variere koeffisientene.

Skjule den hemmelige nøkkelen

Som i de foregående kryptosystemforslagene bør ikke polynomene i den hemmelige nøkkelen være moniske.

Sikker kryptering

Rai skisserer to forslag til hvordan krypteringen kan foregå.

(1) Bruk teknikk (3) fra 10.2, men nå trenger vi ikke velge en ny offentlig nøkkel – kravet om at $\text{tip}(q_i) = T$ for alle i er allerede oppfylt.

Merk. Dette fungerte ikke i Rais forsøk med Opal, siden Opal automatisk tuppreduserer alle mengder – også B, slik at maksimalt ett element i B fortsatt hadde tupp T da krypteringen fant sted!

(2) Velg tilfeldige polynomer F_i og H_i i R slik at vi har $\text{tip}(F_i) \cdot \text{tip}(H_i) \geq T$ for alle $1 \leq i \leq t$. Sett

$$c = \sum_{i=1}^{t} F_i q_i H_i + m$$

Etter all sannsynlighet vil c la seg redusere med B, men ideen er at, dersom c er reduserbar med en q_i i B, vil $\overline{c}^{\{q_i\}}$ ha *flere* ledd enn c. Slik vil ikke resultatet av reduksjonen gi m.

10.5 En kritisk vurdering av Rais Polly Cracker-systemer

Systemet fra avsnitt 10.2 ble forkastet av Rai, så det ser vi bort fra her.

Sikkerheten i systemene fra avsnitt 10.3 og 10.4 bygger på uendelige Gröbnerbasiser, det vil si at det skal være *umulig* for en kryptanalyst å finne Gröbnerbasisen hun trenger for å redusere chiffertekstene korrekt. Når det gjelder systemet fra 10.3 har det en garantert uendelig Gröbnerbasis for $\langle B \rangle$, og Rai har ikke klart å finne en «delvis Gröbnerbasis» med Opal som kan brukes av en kryptanalyst.

Akkurat hva Rai legger i at han ikke har klart å finne en delvis Gröbnerbasis er noe uklart, men vi antar det betyr at alle endelige mengder $G' \subset G$, der G er en (uendelig) Gröbnerbasis for $\langle B \rangle$, som er funnet med Opal, har vært ubrukelige til å redusere en chiffertekst c til den samsvarende meldingen m. Vi har to innvendinger til argumentet om at systemene som bygger på korollar 9.4.3 er sikre fordi Rai ikke har funnet en brukbar delvis Gröbnerbasis:

1. Eksperimenteringen med Opal har kun foregått i 24 timer. Kunne vi fått andre resulater med mere tid?
2. Kan det finnes andre og bedre systemer enn Opal, som beregner delvise Gröbnerbasiser raskere? Vi vet at mange kommutative, Gröbnerbasisrelaterte problemer som tidligere var regnet som for krevende å løse, ble løst med Faugères F_4- og F_5-algoritmer.

En annen tung innvending mot Rais systemer er mangelen på tilfeldighet. Vi vet fra kapittel 8 at desto mere som er kjent om formen på polynomene som benyttes i krypteringen, desto mer sårbart er systemet for ulike typer lineæralgebraangrep. Noe av styrken ved det opprinnelige Polly Cracker-systemet var den store graden av nettopp tilfeldigheter – det ble stilt få krav til krypteringen og denne var ikke-deterministisk i høyeste grad.

Mangelen på tilfeldighet gjelder også for den offentlige nøkkelen. Hos Rai krever konstruksjonen av denne ofte polynomer av en bestemt struktur som oppfyller diverse krav, og man kan se for seg at å generere nye slike for hvert element i B kan være kostbart. Det koker ned til at det eneste som velges tilfeldig er koeffisientene. Vi så i avsnitt 10.3 at all sikkerheten ligger i disse konstantene, siden formen på polynomene er kjent. (Til sammenlikning var den offentlige nøkkelen i de opprinnelige, kommutative systemene bygd direkte på et $\mathcal{NP}$-komplett problem, og ikke konstruert med utgangspunkt i den hemmelige nøkkelen.)

Et siste motargument går ut på at Rais systemer typisk har en hemmelig nøkkel på formen $G = \{g\}$ og offentlig nøkkel med polynomer $q_i = f_i g h_i$ (systemet fra avsnitt 10.4 oppfyller dette). Dette åpner opp for effektive *faktoriseringsangrep*, der man enten direkte eller med en annen metode forsøker å finne g ved å faktorisere q_i'ene. Levy-dit-Vehel et al. [20] foreslår følgende faktoriseringsangrep:

Anta at vi har hemmelig nøkkel $G = \{g\}$ og en offentlig nøkkel $B = \{q_i\}_{i=1}^{s}$, der $q_i = f_i g h_i$ for f_i og h_i i R. Da kan man finne g ved å betraktene q_i'ene som elementer i $\mathbb{F}[x_1, \ldots, x_n]$, det vil si i den kommutative polynomringen med samme variabler som R. Her kan man beregne største felles divisor (GCD) av alle q_i'ene, faktorisere og forsøke å sende resultatet tilbake til R. Merk at q_i'ene kan bli 0 i den kommutative versjonen, da er de til ingen nytte i angrepet. (Grunnen til at vi gjør de fleste beregningene i den kommutative ringen istedenfor den ikke-kommutative, er at algoritmene her er mye mer effektive.)

Eksempel 10.5.1. Vi har gitt $B = \{q_1, q_2\}$ over $\mathbb{F}_{331}\langle x, y\rangle$, lengde-leksikografisk ordning med $x > y$. Vi vet at $q_i = f_i g h_i$, men ikke noe om formen på g, annet enn at dette er et system av typen fra 10.4. Anta at vi har

$$q_1 = 6xxyxx + 12yxyxx + 3xxxx + 9xyxx + 6yxxx + 18yyxx + 15xxx + 30yxx$$
$$q_2 = 4yxxyy + 2yxxy + 6yxyy + 10yxy$$

Vi finner de kommutative versjonene av q_1 og q_2, henholdsvis $\overline{q_1}$ og $\overline{q_2}$:

$$\overline{q_1} = 6x^4y + 12x^3y^2 + 3x^4 + 15x^3y + 18x^2y^2 + 15x^3 + 30x^2y$$
$$\overline{q_2} = 4x^2y^3 + 2x^2y^2 + 6xy^3 + 10xy^2$$

Vi beregner $GCD(\overline{q_1}, \overline{q_2}) = x(2xy + x + 3y + 5) = \overline{g}$. Vi vet at g deler den korrekte, ikke-kommutative varianten av $\overline{g}$, og dette gir følgende mulige valg for g:

- $g = x$
- $g = 2xy + x + 3y + 5$
- $g = 2yx + x + 3y + 5$
- $g = 2xxy + xx + 3xy + 5x$
- $g = 2xxy + xx + 3yx + 5x$
- $g = 2xyx + xx + 3xy + 5x$
- $g = 2xyx + xx + 3yx + 5x$
- $g = 2yxx + xx + 3xy + 5x$
- $g = 2yxx + xx + 3yx + 5x$

Vi kan, ved en rask sammenlikning med q_i'ene, utelukke de to mulighetene der $\text{tip}(g) = xyx$. At den hemmelige nøkkelen skulle være $g = x$ er helt usannsynlig, så den kan vi også utelukke. Samtlige av de andre mulighetene er uten selv-overlapper, men det krever ikke mye arbeid å teste ut alle seks og vise at bare den første, $g = 2xy + x + 3y + 5$, er slik at q_1 og q_2 kan faktoriseres på formen $q_i = f_i g h_i$.

Merk. Den hemmelige nøkkelen g som ble funnet i eksempelet over kan virke «liten» og dermed urealistisk, men faktum er at Rai benytter hemmelige nøkler på denne formen i sine største eksempler (han benytter også kroppen $\mathbb{F}_{331}$). Imidlertid benytter han noe større polynomer i krypteringen, og også flere elementer i den offentlige nøkkelen. Uten å ha eksperimentert med dette, virker det rimelig å anta at flere elementer i B ikke vil gjøre angrepet verre, men snarere enklere, siden det er mindre sannsynlig at GCD'en man finner av de kommutative polynomene inneholder «ekstra» faktorer, slik det var i eksempelet vi nettopp så på.

Eksempel 10.5.2. Vi avslutter dette kapitlet med et lite eksperiment: La oss se om vi klarer å finne g fra forrige eksempel ved hjelp av direkte faktorisering av de ikke-kommutative polynomene. Denne gangen lar vi beregningene foregå i ringen $R = \mathbb{F}_{29}\langle x, y\rangle$, men vi lar q_1

og q_2 være de samme. Modulo 29 har vi nå

$$q_1 = 6xxyxx + 12yxyxx + 3xxxx + 9xyxx + 6yxxx - 11yyxx - 14xxx + yxx$$
$$q_2 = 4yxxyy + 2yxxy + 6yxyy + 2xyy + 10yxy + xy + 3yy + 5y$$

Vi kan anta at

$$\text{tip}(q_i) = \text{tip}(f_i)\text{tip}(g)\text{tip}(h_i)$$

(unntaket er dersom q_i'ene er konstruert slik at tuppene kansellerer). Vi får da to likninger

$$\text{tip}(q_1) = xxyxx = \text{tip}(f_1)\text{tip}(g)\text{tip}(h_1)$$
$$\text{tip}(q_2) = yxxyy = \text{tip}(f_2)\text{tip}(g)\text{tip}(h_2)$$

Vi ser at $\text{tip}(g)$ har lengde to eller tre. Dersom lengden er tre, har vi enten $\text{tip}(g) = xxy$ eller $\text{tip}(g) = yxx$. Vi tar utgangspunkt i den første og ser hvor langt vi kommer med faktoriseringen. Vi får altså

$$q_1 = (6xxy + 12yxy + 3xx + 9xy + 6yx - 11yy - 14x + y)xx = a \cdot h_1'$$
$$q_2 = y(4xxy + 2xx + 6xy + 10x + 3)y + (2xyy + xy + 5y) = f_2' \cdot b_1 \cdot h_2' + b_2'$$

Men vi er ikke i mål enda, for uttrykkene i parantesene er ikke på samme form, til tross for at

$$\text{tip}(a) = \text{tip}(b_1) = xxy$$

Hvis vi forsøker å faktorisere videre med utgangspunkt i dette, må vi velge $\text{tip}(g)$ til å være xy (vi kan ikke velge xx, siden denne ikke finnes i $\text{tip}(b_2)$). Et skritt til med faktorisering gir da

$$q_1 = x(6xy + 3x + 9y - 14)xx + y(12xy + 6x - 11y + 1)xx$$
$$q_2 = yx(4xy + 2x + 6y + 10)y + (2xy + x + 3y + 5)y$$

Nå har vi alle parantesuttrykkene på samme form, og vi antar derfor at g er på formen $g = \alpha xy + \beta x + \gamma y + \delta$, og at vi har $f_1 = ax + by$, $h_1 = cxx$, $f_2 = dyx + e$ og $h_2 = fy$. Det eneste som gjenstår å finne, er koeffisientene, og siden det ikke er noen grunn til å anta at verken g eller f_i'ene og h_i'ene er moniske, får vi et ikke-lineært likningssystem, nærmere bestemt systemet

$$\begin{array}{rcr} ac\alpha &=& 6 \\ ac\beta &=& 3 \\ ac\gamma &=& 9 \\ ac\delta &=& -14 \end{array} \qquad \begin{array}{rcr} bc\alpha &=& 12 \\ bc\beta &=& 6 \\ bc\gamma &=& -11 \\ bc\delta &=& 1 \end{array} \qquad \begin{array}{rcr} df\alpha &=& 4 \\ df\beta &=& 2 \\ df\gamma &=& 6 \\ df\delta &=& 10 \end{array} \qquad \begin{array}{rcr} ef\alpha &=& 2 \\ ef\beta &=& 1 \\ ef\gamma &=& 3 \\ ef\delta &=& 5 \end{array}$$

Man legger umiddelbart merke til at tallene i den første kolonnen er halvparten av tallene i den andre (modulo 29), og at tallene i den siste kolonnen er halvparten av tallene i den tredje. Altså antar vi

$$b = 2a \text{ og } d = 2e$$

og dessuten at $f = c = 1$. Vi ser at vi, under antakelsene over, har $e\beta = 1$. Denne likningen kan ha mange løsninger, men den enkelste er $e = \beta = 1$, og vi ser at vi da får en komplett løsning av likningssystemet, med $e = 1$, $d = 2$, $a = 3$ (siden $a \cdot 1 = 3$) og $b = 6$. Men det vi er mest interessert i, er de ukjente koeffisientene i g, og vi ser at vi har

$$\alpha = 2, \beta = 1, \gamma = 3 \text{ og } \delta = 5$$

Vi har dermed faktorisert q_1 og q_2 og funnet den hemmelige nøkkelen $g = 2xy + x + 3y + 5$.

Bibliografi

[1] Barkee, B., D. Can, J. Ecks, T. Moriarty og R. Ree: *Why you cannot even hope to use Gröbner bases in public key cryptology: An open letter to a scientist who failed and a challenge to those who have not yet failed*. Journal of Symbolic Computation, 18, 1994.

[2] Bhattacharya, P.B., S.K. Jain og S.R. Nagpaul: *Basic Abstract Algebra*. Cambridge University Press, 1994.

[3] Book, R.V. og F. Otto: *String-rewriting systems*. Springer, 1993.

[4] Bulygin, S. og T. Rai: *Noncommutative Polly Cracker-type cryptosystems and chosen-ciphertext security*, 2008.

[5] Caboara, M., F. Caruso og C. Traverso: *Gröbner bases for public key cryptography*. I *ISSAC '08: Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, 2008.

[6] Cohen, A.M.: *Non-commutative polynomial computations*. 2008. En beskrivelse av hovedalgoritmene i GAP-pakken GBNP.

[7] Cox, D., J. Little og D. O'Shea: *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[8] Eder, C.: *On The Criteria Of The F5 Algorithm*. 2008.

[9] Faugère, J.C.: *A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)*. ACM Press, 2002.

[10] Faugère, J.C.: *A new efficient algorithm for computing Gröbner bases (F_4)*. Journal of Pure and Applied Algebra, 139, 1999.

[11] Fellows, M. og N. Koblitz: *Combinatorial cryptosystems galore!* Contemporary Math, 168, 1994.

[12] Fortnow, L. og B. Gasarch: *Reductions To SAT*, desember 2006. Hentet fra http://blog.computationalcomplexity.org/, september 2009.

[13] Gash, J.M.: *On Efficient Computation of Groebner Bases*. PhD-avhandling, Indiana University, 2008.

[14] Gebauer, R. og H. Möller: *A new implementation of Buchberger's algorithm*. 1987.

[15] Gebauer, R. og H. Möller: *On an installation of Buchberger's algorithm*. Journal of Symbolic Computation, 6, 1988.

[16] Green, E.: *Noncommutative Gröbner Bases, and Projective Resolutions*. Progress in Mathematics, 173, 1999.

[17] Green, E., T. Mora og V. Ufnarovski: *The Non-Commutative Gröbner Freaks*. 2000.

[18] Keller, B.J.: *Algorithms and Orders for Finding Noncummutative Grobner Bases*. PhD-avhandling, Virginia Polytechnic Institute and State University, 1996.

[19] Koblitz, N.: *Algebraic aspects of cryptography. Algorithms and computation in mathematics vol. 3*. Springer, 1998.

[20] Levy-dit-Vehel, F., M.G. Marinari, L. Perret og C. Traverso: *A Survey on Polly Cracker Systems*. Springer, 2009. Fra boken *Gröbner bases, Coding, and Cryptography*, red. M. Sala, T. Mora, L. Perret, S. Sakata og C. Traverso.

[21] Mora, T.: *An introduction to commutative and non-commutative Groebner bases*. Theoretical Computer Science, 134, 1994.

[22] Mora, T.: *Solving Polynomial Equation Systems II*. Cambridge University Press, 2005.

[23] Rai, T.: *Infinite Gröbner Bases and Noncommutative Polly Cracker Cryptosystems*. PhD-avhandling, Virginia Polytechnic Institute and State University, 2004.

[24] Sipser, M.: *Introduction to the theory of computation*. Thomson Course Technology, Boston, 2006.

[25] Stegers, T.: *Faugere's F5 Algorithm Revisited*, 2006. http://eprint.iacr.org/.

[26] Traverso, C. og L. Donati: *Experimenting the Gröbner basis algorithm with the AIPI system*. Proc. ISSAC '89, ACM, 1989.

Vedlegg

A Ordliste

Denne lille og noe uformelle ordlista tar for seg noen av de mest «uvanlige» (derav noen egenproduserte) oversettelser av begreper som dukker opp i oppgaven. Noen av begrepene har flere engelske oversettelser, siden de ulike kildetekstene som er brukt i arbeidet, ikke kan sies å bruke samme notasjon. Oppslagene er sortert på kapittel. Med noen få unntak har vi ikke her henvist til de respektive kildene, da dette uansett ville blitt ufullstendig.

Kapittel 2

«er f i I» Ideal membership problem

monom monomial, term, power product

ledende ledd leading/head/maximal term/monomial

ledende monom leading/head monomial/term

ledende koeffisient leading/head coefficient

reduksjon til null (i tillegg til «reduction to zero») Gröbner representation

Kapittel 3

tillatelig ordning admissible order, reduction ordering (sistnevnte er fra [6] og er litt svakere, da en reduction ordering ikke trenger å være en velordning)

tupp tip, largest monomial, leading term

Kapittel 4

svak reduksjon til null weak Gröbner representation

spredt sparse

Kapittel 5

hovedsyzygy principal/trivial syzygy (trivial syzygy kan også betegne syzygyer på formen $\epsilon_k - \epsilon_k$)

R^m-monom module term (Faugère bruker ikke noe navn på disse, bare symboler, som for eksempel **T** for mengden av alle R^m-monomer)

veldefinert signert polynom admissible signed polynomial

omskrivbar rewritable

svak reduksjon til null se kapittel 5 (Faugère og hans etterfølgere skriver at et S-polynom $S = u_i p_i - u_j p_j$ er $\mathcal{O}(LM(S))$ for reduksjon til null og $o(LM(S)) \iff \mathcal{O}(u_i LM(p_i))$ for svak reduksjon til null)

Kapittel 6

blokkering obstruction

grunnmengde en variant av Cohens «basic sets»

svak reduksjon til null weak Gröbner representation/weak with respect to G

tett dense

Kapittel 7

spredt sparse

tett dense

binomer binomials

Kapittel 8

strengomskrivingsystemer String Rewrite Systems (SRS)

B Poly-pakken for ikke-kommutative polynomringer

B.1 Innledning

Programmet Poly ble laget fordi jeg ikke hadde et enkelt program for å jobbe med polynomer over ikke-kommutative polynomringer. Jeg trengte et program der man enkelt kunne lage polynomer og arbeide med dem, uten å ta veien om mer avanserte veialgebraer enn strengt tatt nødvendig, og der man kan velge mellom flere monomsorteringer og raskt skifte mellom dem. I mangel av et slikt ferdiglaget program (det finnes flere slike programmer fra før, også enkelte jeg hadde tilgang til, men jeg var ikke ute etter noe komplisert som tok tid å sette seg inn i), laget jeg et selv. En annen fordel med å skrive sitt eget program, er at man da får med nøyaktig den funksjonaliteten man ønsker seg, og man vet bestandig hvordan ting fungerer.

En ting som er vektlagt i programmet er muligheten til å beregne delvise Gröbnerbasiser. Siden idealer i ikke-kommutative polynomringer ofte har uendelige Gröbnerbasiser, vil ikke en «automatisk» implementasjon av Buchbergers algoritme være garantert å stoppe. Dersom det eneste man er ute etter, er å beregne de første elementene i basisen, er et slikt program til liten nytte. Poly er utstyrt med en «automatisk» og en «manuell» variant av Buchbergers algoritme. Den automatiske er ikke garantert å stoppe (den stopper hvis den finner en korrekt, endelig Gröbnerbasis), mens den manuelle versjonen beregner ett nytt element i gangen, og brukeren har hele tiden full oversikt over blokkeringer og hva de reduserer til.

Programmet er skrevet i Python. For en lettfattelig innføring i Python anbefales boka *Learning Python* av Mark Lutz (http://oreilly.com/catalog/9780596513986/). Python er et relativt enkelt, objektorientert språk som minner om Java, bare med mye simplere og mindre rigid syntaks. Å forstå Python-kode faller naturlig dersom man er vant med å lese pseudokoder. For å bruke Poly-pakken trenger man imidlertid ikke kunne annet enn det som oppgis i den nedenstående teksten.

B.2 Komme i gang

For å kjøre Python, må det innstalleres på maskinen. Last ned gratis fra www.python.org/ og følg instruksjonene. Start deretter kommandolinjevinduet, gå til filområdet der filen poly.py er lagret, og skriv `python`. Poly-modulen importeres som vist under.

```
C:\Skole\Matematikk\Master\python>python
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit
   (Intel)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import poly
>>>
```

Hver gang man kaller en metode fra modulen, må den ha forstavelsen *poly*, som f.eks.

```
>>> f = poly.pol('3*yx + 1*z')
```

Det er naturlig å starte med å definere kroppen koeffisientene er i og den tillatelige ordningen (ved å sette denne, definerer man samtidig variablene i ringen). Deretter kan man lage polynomer, og arbeide videre med dem. Merk at det finnes default-variabler for både kroppen og ordningen – se de to følgende avsnittene.

B.3 Hvor foregår beregningene?

Alle beregninger skjer i ringen $\mathbb{Z}_p \langle x_1, \ldots, x_n \rangle$, der brukeren selv må definere antall variabler (og hva de heter). Dersom man ikke aktivt setter disse selv, brukes ringen $\mathbb{Z}_{31} \langle x, y, z \rangle$ (variablene «hører til» den tillatelige ordningen). Kroppen kan man endre når som helst med metoden `setKropp(val)`, som tar inn et primtall. Andre kropper enn $\mathbb{Z}_p$ støttes ikke.

Koeffisienter skrives ut med minst mulig absoluttverdi. Hvis man for eksempel jobber i $\mathbb{Z}_5$, skriver programmet $x - 1$ istedenfor $x + 4$.

B.4 Ordninger

Programmet støtter tre ordninger: *Lengde-leksikografisk*, *vekt-leksikografisk* og *total* ordning. Ordningen er *global*, det vil si at når man har definert en ordning, kommer alle beregninger til å utføres med med hensyn til denne. Man setter den globale ordningen ved å kalle

```
>>> poly.setOrd(O)
```

Der `O` er den ønskede ordningen. Merk at dersom man skifter ordning, og deretter skriver ut et polynom man lagde *før* ordningskiftet, vil ikke polynomet ha endret seg etter den nye ordningen. Se mer om dette i avsnittet om polynomer. Dersom man *ikke* setter ordning manuelt, brukes lengde-leksikografisk ordning med $x > y > z$ (hvilket innebærer at polynomer med andre variabler enn x, y og z vil gi feilmelding når polynomet initialiseres).

Lengde-leksikografisk ordning. Klassen har ett attributt, nemlig variabellista som er en liste på formen `['x1',...,'xn']` slik at $x_1 > \ldots > x_n$. Følgende gjelder for denne lista:

- Variablene må ha et navn bestående av *ett* symbol, f.eks. `x`, `y`, `a`, `X`, ...
- Det er ikke nødvendig å inkludere (monomet) 1 i lista, dette legges til automatisk og slik at $1 < x_n$.

Initialisering: `O = LenLex(variabelliste)`

Vekt-leksikografisk ordning. Klassen har to attributter. Variabellista fungerer som over, mens vektlista er en *dictionary* på formen `{'x1': w(x1), ..., 'xn': w(xn)}`. Både variabellista og vektlista må deklareres spesifikt – variabellista genereres ikke automatisk på grunnlag av vektlista (hvis to variabler har lik vekt, må de fortsatt ha en innbyrdes ordning for lengde-leksikografisk sortering). For vektlista gjelder samme regler for variabelnavnene som over. Det er underforstått at de samme variablene skal finnes i begge listene.
Initialisering: `O = VektLex(vektliste,variabelliste)`

Total ordning. Klassen har ett attributt; variabellista. Se lengde-leksikografisk ordning.
Initialisering: `O = Total(variabelliste)`

Under ser vi demonstert print-funksjonen for ordninger. Vi ser at variabelen 1 har blitt med automatisk. Grunnen til at print-funksjonen printer vektlista i en annen rekkefølge enn de ble skrevet inn er at Python lagrer dictionaries randomisert.

```
>>> O = poly.VektLex({'x':4,'y':3,'z':2,'w':1},['x','y','z','w'])
>>> print O
Vekt-leksikografisk ordning med x > y > z > w > 1 og vekter
    {'y': 3, 'x': 4, '1': 0, 'z': 2, 'w': 1}
```

For enkelthets skyld er det i eksemplene fra nå benyttet lengde-leksikografisk ordning med $x > y > z$, med mindre annet er oppgitt.

B.5 Polynomer

Polynomer er implementert som en egen klasse, med to attributter:
- `mon`: en liste over alle monomer som er i polynomet
- `koeff`: en liste over koeffisientene – der `koeff[i]` er koeffisientene til `mon[i]`

Det går an å lage et polynom-objekt direkte:

```
>>> h = poly.Polynom(['xyx','xxy','y'],[4,3,-1])
>>> print h
3*xxy + 4*xyx - 1*y
```

Som det framgår av eksempelet, blir polynomet automatisk sortert med hensyn til den gjeldende ordningen. Husk at man ikke må forsøke å initialisere polynomer der man bruker variabler som ikke finnes i variabel-/vektlista til den angitte ordningen.

Det finnes også en mer «brukervennlig» måte å initialisere polynomer på, ved hjelp av `pol`-funksjonen:

```
>>> f = poly.pol('3*xx + 2*yxy + -2*z')
>>> print f
2*yxy + 3*xx - 2*z
```

Her skrives hele polynomet som én streng, med følgende kriterier:
– ledd skilles med `+`
– koeffisient og monom skilles med `*`

– selv om en koeffisient har negativt fortegn, må pluss-tegnet med som skille mellom ledd, som vist i eksempelet over.

Tuppen av et polynom hentes ved å kalle funksjonen `tip()`:

```
>>> f.tip()
'yxy'
```

Polynom-operasjoner. Python har praktiske, innebygde funksjoner som gjør at man lett kan overskrive standard-operasjonene `+`, `-`, `*` etc. Følgende operasjoner er lagt til for polynomer: Addisjon (`+`), subtraksjon (`-`) og multiplikasjon (`*`). Samtlige av disse returnerer et nytt polynom. Python sørger automatisk for at multiplikasjon blir utført før addisjon, slik det skal være.

```
>>> f = poly.pol('2*x+1*1')
>>> g = poly.pol('3*xy+-2*z')
>>> h = poly.pol('1*zy+1*x+2*x')
>>> p = f+h*g
>>> print p
3*zyxy + 9*xxy - 2*zyz - 6*xz + 2*x + 1*1
```

Det er også mulig å benytte Python-syntaks som `f += f`, som setter f til å være $2f$. Videre finnes det et par polynom-metoder som ikke returnerer et nytt polynom, men endrer selve Polynom-objektet metoden kalles fra:

`mult_koeff(`*val*`)` multipliserer polynomet med et element *val* i den gitte kroppen.

```
>>> print g
3*xy - 2*z
>>> g.mult_koeff(6)
>>> print g
18*xy - 12*z
```

`monisk()` multipliserer polynomet med en passelig konstant slik at det ledende leddet (i henhold til den valgte ordningen) får koeffisient 1.

```
>>> poly.setKropp(59)
kroppen har nå 59 elementer.
>>> print g
18*xy - 12*z
>>> g.monisk()
>>> print g
1*xy + 19*z
```

Bytte av ordning. Når man endrer monomordningen, vil ikke polynomene man har laget så langt endres. Hvis man derimot utfører en polynomoperasjon som får et nytt polynom som resultat, vil det nye polynomet sorteres med hensyn til den nye ordningen. Ønsker man å sortere et polynom uten å lage et nytt (som man gjør hvis man ganger med 1 eller adderer 0), kan man kalle metoden `sorter()`.

B.6 Divisjonsalgoritme

Programmet har implementert en rett-fram-versjon av divisjonsalgoritmen, omtrent identisk med pseudokoden gitt i algoritme 2.1. Slik kalles den:

```
>>> poly.setKropp(29)
kroppen har nå 29 elementer.
>>> print f
5*xxyyx - 1*xy + 3*x + 1*1
>>> print f1
1*x - 3*1
>>> print f2
2*yy + 1*x + 1*1
>>> F = [f1,f2]
>>> Q = poly.divalg(f,F)
>>> print Q[0]
-3*y + 1*1
```

Som vi ser, returnerer metoden en liste, der det første elementet er resten ved polynomdivisjonen. Vanligvis er det bare denne vi er interessert i, med den returnerte listen inneholder også monomene som har blitt brukt i reduksjonen. Det er litt krevende å få tak i dem, da de lagres som tupler innad i lister, og vi vet ikke på forhånd hvor mange tupler det blir. `Q[1]` består av én liste, som igjen inneholder én liste for hvert element i F. Disse listene inneholder tupler (a, b). Vi kan skrive ut listene slik:

```
>>> def printQ(list):
...     for x in range(len(list)):
...             print "tupler for element " + str(x+1)
...             for y in list[x]:
...                     print y[0],y[1]
...
>>> printQ(Q[1])
tupler for element 1
5*1 1*xyyx
15*1 1*yyx
16*yy 1*1
28*1 1*y
8*1 1*1
tupler for element 2
24*1 1*1
>>>
```

Dette betyr at:

$$f = 5f_1xyyx + 15f_1yyx + 16yyf_1 + 28f_1y + 8f_1 + 24f_2 + (-3y + 1)$$

For moro skyld kan man gjøre forsøket en gang til, men denne gangen med `F = [f2,f1]`. Resten blir den samme, men tuplene i `Q[1]` er annerledes.

B.7 Buchbergers algoritme

Med Poly-pakken er det to måter å kjøre Buchbergers algoritme på. Den ene er å kalle metoden `buchberger(F)`, der F er en endelig liste med polynom-objekter. Dersom idealet $\langle F \rangle$ har en endelig Gröbnerbasis, vil programmet returnere denne. Hvis ikke (eller hvis Gröbnerbasisen er endelig, men for stor), vil programmet kjøre til ressursene er brukt opp, og det tvinges til avbrudd. Denne metoden skriver ikke ut noe. Polynomene under er hentet fra eksempel 3.5.5, og bruker lengde-leksikografisk ordning der $y > x$.

```
>>> f1 = poly.pol('1*xyx + -1*x')
>>> f2 = poly.pol('1*yx + -1*x')
>>> f3 = poly.pol('1*xx + -1*y')
>>> F = [f1,f2,f3]
>>> G = poly.buchberger(F)
>>> for g in G:
...     print g
...
1*xyx - 1*x
1*yx - 1*x
1*xx - 1*y
1*y - 1*x
```

Den andre måten å kjøre Buchbergers algoritme på, er å beregne én og én overlappsrelasjon. For hver gang man gjør en iterasjon, reduseres én overlappsrelasjon modulo den foreløpige Gröbnerbasisen, og man får alltid vite:
– Hva overlappsrelasjonen reduserer til, og hvilket element som evt. legges til Gröbnerbasisen
– Alle nye blokkeringer dette elementet danner med tidligere elementer
– Hvorvidt det nye elementet gjør tidligere elementer redundante, eller om det er redundante blokkeringer blant de nye blokkeringene

Lista der blokkeringene lagres, er en grunnmengde for F (se 6.2.1). Det benyttes flere kriterier for å holde kardinaliteten av Q minimal, disse er også beskrevet i avsnitt 6.2. Denne metoden er grei å bruke dersom man tror (eller vet) at den aktuelle Gröbnerbasisen er uendelig/svært tung å beregne, og bare vil finne de første elementene.

Det første man må gjøre er å initialisere Q. Vi starter med å finne samtlige blokkeringer innad i mengden F. Merk at dette er et endelig antall. I eksempelet bruker vi samme F som i forrige eksempel. Initialiseringen gjøres slik:

```
>>> Q = poly.init_buchberger(F)
Kjører Buchbergers algoritme. Initialiserer G ...
lagt til som element nr. 0 i G:
1*xyx - 1*x
blokkeringene for dette elementet er:
('xy', 0, '1', '1', 0, 'yx')

lagt til som element nr. 1 i G:
1*yx - 1*x
vi tar da ut følgende elementer fra Q:
```

```
('xy', 0, '1', '1', 0, 'yx')
blokkeringene for dette elementet er:
('x', 1, '1', '1', 0, '1')
('1', 1, 'yx', 'y', 0, '1')

lagt til som element nr. 2 i G:
1*xx - 1*y
blokkeringene for dette elementet er:
('x', 2, '1', '1', 2, 'x')
('xy', 2, '1', '1', 0, 'x')
('1', 2, 'yx', 'x', 0, '1')
('y', 2, '1', '1', 1, 'x')
('xy', 2, '1', '1', 0, 'x')
redundante blant disse:
('xy', 2, '1', '1', 0, 'x')
```

Vi ser at `init`-metoden skriver ut elementene som legges til i G, og blokkeringene som legges til i Q. Noen blokkeringer forsvinner grunnet redundans, dette kommer også fram. Merk at siden python bruker 0-indekserte lister, blir også Gröbnerbasisen 0-indeksert (dvs. at vi ikke skriver $G = \{g_1, \ldots, g_t\}$, men $G = \{g_0, \ldots, g_{t-1}\}$).

Deretter er det kun en metode som skal kalles: `pop_ko(G,Q)`. Denne metoden tar inn den foreløpige Gröbnerbasisen og grunnmengden Q, trekker ut det første elementet fra Q, og gjør det som er beskrevet over. Metoden returnerer et tuppel `(G,Q)`, som er den nye delvise Gröbnerbasisen og den oppdaterte grunnmengden. Grunnen til at metoden heter `pop_ko` er at Q er implementert som en kø (dette er *ikke* ideelt, da Q i dette tilfellet er usortert).

```
>>> (G,Q) = poly.pop_ko(F,Q)
vi velger ('x', 1, '1', '1', 0, '1') fra Q
overlappsrelasjonen reduserte til -1*y + 1*x
lagt til som element nr. 3 i G:
1*y - 1*x
vi tar da ut følgende elementer fra Q:
('1', 2, 'yx', 'x', 0, '1')
('1', 1, 'yx', 'y', 0, '1')
blokkeringene for dette elementet er:
('x', 3, 'x', '1', 0, '1')
('1', 3, 'x', '1', 1, '1')
('x', 3, 'x', '1', 0, '1')
redundante blant disse:
('x', 3, 'x', '1', 0, '1')
```

Slik kan vi fortsette til Q er tom (hvis den blir tom). Her ser vi at vi legger til elementet $y - x$ i G, og at dette elementet gjorde to tidligere elementer i Q redundante, og at det nye elementet videre genererte tre nye blokkeringer hvorav en er redundant.

```
>>> (G,Q) = poly.pop_ko(G,Q)
vi velger ('x', 2, '1', '1', 2, 'x') fra Q
overlappsrelasjonen reduserte til 0
```

Etter å ha kalt `pop_ko(G,Q)` noen ganger til, får vi:

```
>>> (G,Q) = poly.pop_ko(G,Q)
Q er tom
>>> for g in G:
...     print g
...
1*xyx - 1*x
1*yx - 1*x
1*xx - 1*y
1*y - 1*x
```

Resultatet er en korrekt Gröbnerbasis for $\langle G \rangle$.

C Implementasjon av Poly-pakken

De påfølgende sidene består av kildekoden til Poly-pakken. Filen poly.py, som stort sett er identisk med de neste sidene (eventuelle forskjeller er i så fall ikke i selve koden), er tilgjengelig som elektronisk vedlegg til oppgaven. Symbolet → dukker opp flere ganger på starten av linjene i koden, dette betyr at en linje i koden er så lang at den har blitt delt for å passe inn i formatet (dette er ikke en del av selve programkoden).

All koden er egenprodusert, med unntak av funksjonen `finn_invers()` som er hentet fra http://cobweb.ecn.purdue.edu/ kak/compsec/NewLectures/Lecture5.pdf og tilpasset formålet. Koden er relativt grundig dokumentert underveis, og dersom man har lest dokumentasjonen fra B, bør strukturen i koden være nogenlunde selvforklarende.

Som nevnt tidligere er Python et av de mer «banale» programmeringsspråkene, i den forstand at det sløyfer mye av den til tider byråkratiske notasjonen vi finner i for eksempel Java. Dette betyr ikke at Python er et «dårligere» alternativ, tvert imot har Python en svært god (dynamisk) behandling av lister, og dette er noe vi har benyttet oss av i denne implementasjonen. Igjen henviser vi til boka *Learning Python* av Mark Lutz:

http://oreilly.com/catalog/9780596513986/

Dersom man hopper direkte fra kapitlet om ikke-kommutative Gröbnerbasisalgoritmer til kildekoden, vil man sannsynligvis bli forvirret, for den største delen av programmet er *ikke* selve Buchberger-algoritmen, men snarere den grunnleggende behandlingen av polynomer og ordninger. Noen av funksjonene er mer «kronglete» i implementert utgave enn de er i den matematiske beskrivelsen fra tidligere kapitler, som for eksempel listemanipulasjonen i `fjern_redundante`.

```
# ========================== #
# ENDELIG KROPP              #
# ========================== #
# Polynomringen vi befinner  #
# oss i er over en endelig   #
# kropp, med P antall elemen- #
# ter der P er et primtall.  #
# ========================== #

# ENDELIG KROPP
# P: global variabel som kalles av flere metoder
P = 31          #default antall elementer i kroppen


# ENDRE KROPP
# input: et PRIMTALL
def setKropp(val):
     global P
     P = val
     print 'kroppen har nå ' + str(P) + ' elementer.'


# INVERSER I KROPPEN
# input: et element i kroppen (ikke 0)
# output: invers av input'en modulo P
def finn_inverse(a):
     global P
     if a < 0:
          a = a + P
     a1, a2 = 0, P
     b1, b2 = 1, a
     while b2 != 0:
          q = a2 // b2
          t = a1 - b1 * q, a2 - b2 * q
          a1, a2 = b1, b2
          b1, b2 = t
     if a2 != 1:
          print "Feil! Kroppen har ikke et gyldig antall
elementer."
          return 0
     else:
          if a1 < 0:
               a1 = a1 + P
          return a1


# ========================== #
# MONOMORDNING-KLASSER       #
# ========================== #
# Følgende ordninger er med: #
#  - lengde-leksikografisk   #
#  - vekt-leksikografisk     #
#  - total                   #
# Superklasse: Ordning       #
# Vekt-leks. og total er sub- #
# klasser av lengde-leks.    #
#    I Ordning-klassen må det #
# deklareres en variabelliste #
# på formen ['x1','...','xn'] #
# der x1 > ... > xn > 1.     #
# ========================== #

# SUPERKLASSE FOR ORDNINGER
class Ordning:

     def __init__(self,value):
          self.liste = value
     def setliste(self, value):
          self.liste = value



     # sorterer et polynom - bruker smlikn-metoden
     # MERK: "polynom" er ikke definert som klasse enda!
     # input: et tuppel ("polynom") på formen ([monom-
→ liste],[koeffisient-liste])
     # output: et tilsvarende tuppel der listene er sortert etter
→ valgt ordning (jf. subklassen)
     def sort(self,f):
          lengde = len(f[0])
          liste = ([],[])
          if lengde != 0 and lengde != 1:
               liste = self.merge(self.sort((f[0][(lengde//2):],
→ f[1][(lengde//2):])), self.sort((f[0][:(lengde//2)],
→ f[1][:(lengde//2)])))
               return liste
          else:
               return f

# Brukes til mergesort
     def merge(self,f1,f2):
          global P
          len1 = len(f1[0])
          len2 = len(f2[0])
          i, j = 0,0
          R = []
          S = []
          while len1 > i and len2 > j:
               test = self.smlikn(f1[0][i],f2[0][j])
               if f1[0][i] == f2[0][j]:   #hvis monomene er like
                    nykoeff = (f1[1][i]+f2[1][j]) % P
                    if nykoeff != 0:      #hvis koeff ikke er null
                         R.append(f1[0][i])
                         if nykoeff <= abs(nykoeff-P):
                              S.append(nykoeff)
                         else:
                              S.append(nykoeff-P)
                    i += 1
                    j += 1
               elif test:            #hvis det første er størst
                    if f1[1][i] != 0:     #hvis koeff ikke er null
                         R.append(f1[0][i])
                         if f1[1][i] <= abs(f1[1][i]-P):
```

```
                              S.append(f1[1][i])
                        else:
                              S.append(f1[1][i]-P)
                  i += 1
            else:
                  if f2[1][j] != 0:     #hvis koeff ikke er null
                        R.append(f2[0][j])
                        if f2[1][j] <= abs(f2[1][j]-P):
                              S.append(f2[1][j])
                        else:
                              S.append(f2[1][j]-P)
                  j += 1
      while len1 > i:
            if f1[1][i] != 0:
                  R.append(f1[0][i])
                  if f1[1][i] <= abs(f1[1][i]-P):
                        S.append(f1[1][i])
                  else:
                        S.append(f1[1][i]-P)
            i += 1
      while len2 > j:
            if f2[1][j] != 0:          #hvis koeff ikke er null
                  R.append(f2[0][j])
                  if f2[1][j] <= abs(f2[1][j]-P):
                        S.append(f2[1][j])
                  else:
                        S.append(f2[1][j]-P)
            j += 1
      return R,S


# LENGDE-LEKSIKOGRAFISK ORDNING
class LenLex(Ordning):
      # lengde-lex med variabelrekkefølge lagret i LISTE i
superklassen
      # input: variabellista ['x1',...,'xn'] der x1 > ... > xn
      def __init__(self,value):
            Ordning.__init__(self,value)
            if self.liste[-1] != '1':
                  self.liste.append('1')

      # skriver ut ordning og variabelliste
      def __repr__(self):
            val = ''
            for i in self.liste[:-1]:
                  val = val + i + ' > '
            return 'Lengde-leksikografisk ordning med ' + val +
→ self.liste[-1]

      # sammenlikn-metode for monomer
      # input: to monomer a og b
      # output: 1 hvis a er størst (eller ved likhet), 0 ellers
      def smlikn(self,a,b):
            if a == '1' and b != '1':
                  return 0
            elif b == '1' and a != '1':
                  return 1
            elif len(a)>len(b):
                  return 1
            elif len(b)>len(a):
                  return 0
            else:
                  for i in range(len(a)):
                        if self.liste.index(a[i]) <
→ self.liste.index(b[i]):
                              return 1
                        elif self.liste.index(a[i]) >
→ self.liste.index(b[i]):
                              return 0
                  return 0


# VEKT-LEKSIKOGRAFISK ORDNING
class VektLex(LenLex):
      # vektene er lagret som en DICTIONARY på formen { 'x1': 3,
→ 'x2': 4, ... } - trenger IKKE inneholde variabelen 1
      vektliste = None#ingen default vektliste

      # input: vlist - vektlista
      #     list - variabellista
      def __init__(self,vlist,list):
            LenLex.__init__(self,list)
            self.vektliste = vlist
            #forsikre at 1 er med og har vekt 0:
            self.vektliste['1'] = 0

      # skriver ut ordning, variabelliste og vektliste
      def __repr__(self):
            val = ''
            for i in self.liste[:-1]:
                  val = val + i + ' > '
            return 'Vekt-leksikografisk ordning med ' + val +
→ self.liste[-1] + ' og vekter ' + str(self.vektliste)

      # tverrsum-beregning
      # input: et monom a
      # output: tverrsummen (jf. vektlista)
      def tverrsum(self,a):
            sumA = 0
            for x in a:
                  sumA = sumA + self.vektliste[x]
            return sumA

      # sammenlikn-metode for monomer
      # input: to monomer a og b
      # output: 1 hvis a er størst (eller ved likhet), 0 ellers
      def smlikn(self,a,b):
            sumA = self.tverrsum(a)
            sumB = self.tverrsum(b)
            if sumA > sumB:
                  return 1
            elif sumB > sumA:
```

```
            return 0
        #hvis de har lik tverrsum, kjør lengde-lex:
        print "vi må bruke LenLex!"
        k = LenLex.smlikn(self,a,b)
        if k:
            return 1
        else:
            return 0


# TOTAL ORDNING
class Total(LenLex):
    # har kun variabelliste, som superklassen.
    # input: variabellista ['x1','...','xn'] der x1 > ... > xn
    def __init__(self,value):
        LenLex.__init__(self,value)

    # skriver ut ordning og variabelliste
    def __repr__(self):
        val = ''
        for i in self.liste[:-1]:
            val = val + i + ' > '
        return 'Total ordning med ' + val + self.liste[-1]

    # sammenlikn-metode for monomer
    # input: to monomer a og b
    # output: 1 hvis a er størst (eller ved likhet), 0 ellers
    def smlikn(self,a,b):
        #lage vektor basert på kommutativ versjon
        A = self.vektor(a)
        B = self.vektor(b)
        #sammenlikne
        for i in range(len(A)):
            if A[i] > B[i]:
                return 1
            elif B[i] > A[i]:
                return 0
        #hvis vektorene er like:
        print "vi må bruke LenLex!"
        k = LenLex.smlikn(self,a,b)
        if k:
            return 1
        else:
            return 0

    # vektor-metode som teller antall av hvert symbol (jf.
→ kommutativ gradert leksikografisk ordning)
    # input: et monom a
    # output: en vektor [q1,q2,...,qn] der qi er antall av symbolet
→ liste[i] som finnes i a
    def vektor(self,a):
        res = []
        for i in self.liste:
            res.append(a.count(i))
        return res
```

```
# SETTER EN ORDNING
ORD = LenLex(['x','y','z'])     #default ordning er lengde-
leksikografisk med x > y > z

def setOrd(ordning):
    global ORD
    ORD = ordning


# ========================== #
# POLYNOMER                  #
# ========================== #
# Polynomer er implementert  #
# som klasser, der +,- og *  #
# tolkes direkte vha pythons #
# innebygde egenskaper.      #
# ========================== #

# POLYNOMKLASSE
class Polynom:
    mon = []
    koeff = []

    # input: val1 - monomlista
    #     val2 - koeffisientlista
    def __init__(self,val1,val2):
        global P   #koeffisientkroppen
        global ORD #den tillatelige ordningen

        mon = []
        koff = []
        for i in range(len(val1)):
            x = val2[i] % P
            if x != 0:
                mon.append(val1[i])
                koff.append(val2[i])
        val1,val2 = ORD.sort((mon,koff))
        self.mon = val1
        self.koeff = val2

    # printer polynom
    # kalles av "print f" der f er Polynom
    # output: en pen og naturlig representasjon av polynomet
    def __str__(self):
        if not self.koeff:
            return '0'
        elif self.koeff[0] > 0:
            val = ''
        else:
            val = '-'
        for i in range(len(self.mon)-1):
            if self.koeff[i+1] > 0:
                val = val + str(abs(self.koeff[i])) + '*' +
→ self.mon[i] + ' + '
```

```
                else:
                    val = val + str(abs(self.koeff[i])) + '*' +
→ self.mon[i] + ' - '
          val = val + str(abs(self.koeff[-1])) + '*' + self.mon[-1]
          return val

     # set-metoder
     def setmon(self,val):
          self.mon = val
     def setkoeff(self,val):
          self.koeff = val

     # resortere polynomet (hvis man skifter ordning)
     def sorter(self):
          global ORD
          (a,b) = ORD.sort((self.mon,self.koeff))
          self.setmon(a)
          self.setkoeff(b)

     # tuppen av polynomet (som streng)
     def tip(self):
          return self.mon[0]

     # addisjon av polynomer
     # input: polynomet som skal legges til dette
     # output: et nytt Polynom-objekt som er summen
     def __add__(self,other):
          hm = (self.mon+other.mon,self.koeff+other.koeff)
          return Polynom(hm[0],hm[1])

     # subtraksjon av polynomer
     # input: polynomet som skal trekkes fra dette
     # output: et nytt Polynom-objekt som er differansen
     def __sub__(self,other):
          hm = []
          for i in other.koeff:
               hm.append(-1*i)
          h = Polynom(other.mon,hm)
          return self.__add__(h)

     # multiplikasjon av polynomer
     # input: polynomet som skal ganges med dette
     # output: et nyt Polynom-objekt som er produktet
     def __mul__(self,other):
          h = ([],[])
          for i in range(len(self.mon)):
               for j in range(len(other.mon)):
                    m = self.mon[i]
                    n = other.mon[j]
                    if m != '1' and n != '1':
                         h[0].append(m+n)
                    elif m != '1' and n == '1':
                         h[0].append(m)
                    else:
                         h[0].append(n)
                    h[1].append(self.koeff[i]*other.koeff[j])
          return Polynom(h[0],h[1])

     # multiplisere med et tall
     # input: et element i P
     # gir ikke output, men endrer selve polynomet
     def mult_koeff(self,val):
          global P
          for i in range(len(self.koeff)):
               t = self.koeff[i]*val % P
               if t <= abs(t-P):
                    self.koeff[i] = t
               else:
                    self.koeff[i] = t - P

     # gjøre polynomet monisk
     # gir ikke output, men endrer selve polynomet
     def monisk(self):
          global P
          a = finn_inverse(self.koeff[0])
          self.mult_koeff(a)


# BRUKERVENNLIG KONSTRUKSJON AV POLYNOMER
# input: en streng som '3*xy + 2*xx + -1*1', der
#     - ledd skilles med +
#     - koeffisienter og monomer skilles med *
#     - negative koeffisienter skrives med - foran, men ikke glem +
→ mellom leddene
# output: et Polynom-objekt
def pol(str):
     mono = []
     koeffi = []
     for term in str.split('+'):
          mono.append(term.split('*')[1].strip())
          koeffi.append(int(term.split('*')[0]))
     return Polynom(mono,koeffi)


# DIVISJONSALGORITME
# input: f - et polynom
#      F - en mengde av polynomer
# output: r = f modulo F og tupler (a,b) i en nøstet liste
def divalg(f,F):
     h = Polynom(f.mon,f.koeff) #kopierer f
     s = len(F)            #antall elementer i F
     AB = []               #initialiserer (a,b)-tuplene:
     for i in range(s):
          AB.append([])
     r = ([],[])           #resten er foreløpig to tomme lister
     i = 0                 #indeks i F
     while h.mon:
          t = h.tip().find(F[i].tip())
          if t > -1:
               a = h.tip()[:t] or '1'
               b = h.tip()[t+len(F[i].mon[0]):] or '1'
               koff = h.koeff[0]*finn_inverse(F[i].koeff[0]) % P
               AB[i].append((Polynom([a],[koff]),Polynom([b],[1])))
               h = h - AB[i][-1][0]*F[i]*AB[i][-1][1]
```

```
                i = 0
            elif i<(s-1):
                i += 1
            else:
                r[0].append(h.mon.pop(0))
                r[1].append(h.koeff.pop(0))
                i = 0
      return (Polynom(r[0],r[1]),AB)


# ========================== #
# BUCHBERGERS ALGORITME      #
# ========================== #
# Buchbergers algoritme og   #
# oppdater-metoden kommer til #
# slutt. Først en rekke meto- #
# der for å finne blokkeringer#
# og overlappsrelasjoner.    #
#    For "manuell" versjon av #
# Buchbergers, se neste del. #
# Den første delen ("blokke- #
# ringer og overlapper") er  #
# felles for begge versjonene.#
# ========================== #


# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #
# BLOKKERINGER OG OVERLAPPER #
# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #

# FINNE VENSTRE OVERLAPPER MELLOM TO MONOMER
# input: to monomer (dvs. strenger) u og v
# output: en liste med alle (l,r) som er slik at l*u = v*r er en
overlapp
def venstre_overlapper(u,v):
      RES = []
      s = min(len(u),len(v))
      for i in range(1,s+1):
            if u[:i] == v[-i:]:
                  l = u[i:] or '1'
                  r = v[:-i] or '1'
                  RES.append((l,r))
      return RES


# FINNE HØYRE OVERLAPPER MELLOM TO MONOMER
# input: to monomer (dvs. strenger) u og v
# output: en liste med alle (r,l) som er slik at u*r = l*v er en
overlapp
def hoyre_overlapper(u,v):
      return venstre_overlapper(v,u)


# FINNE MIDT-OVERLAPPER MELLOM TO MONOMER
# input: to monomer (dvs. strenger) u og v, der u er lengre enn v
# output: dersom vi har overlapp u = a*v*b returneres en liste med
→ alle (a,b)-tupler (både a,b ulik 1)
def midt_overlapp(u,v):
      RES = []
      u2 = u[1:-1]
      s = len(v)
      t = len(u2)
      for i in range(0,t-s+1):
            if u2[i:i+s] == v:
                  RES.append((u[0]+u2[:i], u2[i+s:]+u[-1]))
      return RES


# FINNE ALLE BLOKKERINGER MELLOM TO MONOMER
# input: to monomer u og v, der u har "høyere nummer" enn v; og to
→ nummer i og j der u = tip(fi) og v = tip(fj)
# output: en liste med blokkeringer (l,u,r;lambda,v,rho) som
→ inneholder alle mulige overlapper mellom u og v
def finn_blokkeringer(u,v,i,j):
      RES = []
      # blokkeringer på formen (l,i,1;1,j,rho)
      R2 = venstre_overlapper(u,v)
      for s in R2:
            RES.append((str(s[1]),i,'1','1',j,str(s[0])))
      # blokkeringer på formen (1,i,r;lambda,j,1)
      R1 = hoyre_overlapper(u,v)
      for r in R1:
            RES.append(('1',i,str(r[0]),str(r[1]),j,'1'))
      # midt-blokkeringer
      if len(u) > len(v):
            R3 = midt_overlapp(u,v)
            for t in R3:
                  RES.append(('1',i,'1',str(t[0]),j,str(t[1])))
      elif len(v) > len(u):
            R3 = midt_overlapp(v,u)
            for t in R3:
                  RES.append((str(t[0]),i,str(t[1]),'1',j,'1'))
      return RES


# FINNE SELV-OVERLAPPER
# input: et monom u og et nummer D
# output: en liste med blokkeringer (l,u,1;1,u,rho)
def finn_selvoverlapper(u,D):
      R = []
      s = min(len(u),len(u))
      for i in range(1,s):
            if u[:i] == u[-i:]:
                  r = u[i:] or '1'
                  l = u[:-i] or '1'
                  R.append((r,l))
      RES = []
      for r in R:
            RES.append((str(r[1]),D,'1','1',D,str(r[0])))
      return RES
```

```
# BEREGNE ALLE OVERLAPPER MELLOM ET POLYNOM OG EN ENDELIG MENGDE
# input: en endelig mengde F og et Polynom f
# output: en liste med blokkeringer på formen (l,D,r;lambda,j,rho)
→ der D er |F| og j <= D
# husk at selv-overlapper er med!
# husk at lister i Python er 0-indeksert!
def get_overlapper(F,f):
      D = len(F)
      O = []
      # selv-overlapper
      O = O + finn_selvoverlapper(f.tip(),D)
      # overlapper med tidligere polynomer
      for i in range(len(F)):
            O = O + finn_blokkeringer(f.tip(),F[i].tip(),D,i)
      return O


# KONVERTERE FRA BLOKKERING TIL POLYNOMER
# input: en blokkering (l,i,r;lambda,j,rho) i form av et 6-tuppel
→ der l,r,lambda,rho er strenger og i,j er tall
# output: et 4-tuppel (L,R,LAMBDA,RHO) der alle elementene er
→ Polynom-objekter
def blokkering_til_polynom(blokk):
      L = Polynom([blokk[0]],[1])
      R = Polynom([blokk[2]],[1])
      LAMBDA = Polynom([blokk[3]],[1])
      RHO = Polynom([blokk[5]],[1])
      return (L,R,LAMBDA,RHO)


# BEREGNE OVERLAPPSRELASJON
# input: to Polynom-objekter f og g med polynom-blokkering
→ blokk=(l,r;lambda,rho)
# output: overlappsrelasjonen, i form av et Polynom-objekt
def overlappsrelasjon(f,g,blokk):
      global P
      a = finn_inverse(f.koeff[0])
      blokk[0].setkoeff(map((lambda x: x*a),blokk[0].koeff))
      h1 = blokk[0]*f*blokk[1]
      d = finn_inverse(g.koeff[0])
      blokk[2].setkoeff(map((lambda x: x*d),blokk[2].koeff))
      h2 = blokk[2]*g*blokk[3]
      h=h1-h2
      return h


# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #
# REDUNDANS-KRITERIER         #
# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #

# SJEKKE OM EN BLOKKERING DELER EN ANNEN (KRITERIUM NR EN)
# input: to blokkeringer b1 = (l1,i,r1;lambda1,j,rho1) og b2 =
→ (l2,i,r2;lambda2,k,rho2) på listeform
# output: 1 dersom b1 deler b2
#       0 hvis ikke
def blokkering_deler(b1,b2):
      l1 = b1[0]
      r1 = b1[2]
      l2 = b2[0]
      r2 = b2[2]
      if l2[len(l2)-len(l1):] == l1 or l1 == '1':
            if r2[:len(r1)] == r1 or r1 == '1':
                  return 1
      return 0


# FJERNE REDUNTANTE ELEMENTER DEL 1 (del 2 gjøres direkte i
→ Oppdater)
# input: G - en foreløpig Gröbnerbasis
#      Q - en liste med blokkeringer som skal reduseres slik at alle
→ blokkeringer som deles av en annen blokkering i Q taes ut
# output: oppdatert versjon av Q
# merk: denne algoritmen er satt som egen siden den er "lite
→ intuitiv" å lese, da man må trikse litt for å klare å ta ut
→ elementer av en liste samtidig som  den traverseres flere ganger.
def fjern_redundante(G,Q):
      s = 0
      t1 = 1
      t2 = 1
      while s < len(Q):
            t1 = 1
            t2 = 1
            for i in Q[:s]:
                  if blokkering_deler(i,Q[s]):
                        Q.pop(s)
                        t1 = 0
                        break
            for j in Q[s+1:]:
                  if blokkering_deler(j,Q[s]):
                        Q.pop(s)
                        t2 = 0
                        break
            if t1 and t2:
                  s += 1
      return Q


# KRITERIUM NR TO
# input: en blokkeringer b = (l,i,r;lambda,j,rho), u = tip(fi) og et
→ monom v = tip(fk)
# output: 1 dersom l*tip(fi)*r = w1*tip(fk)*w2 for monomer w1, w2
→ ulik 1 og dersom minst en av {l,r} og minst en av {lambda,rho} er 1
→ og 0 hvis ikke
def kriterium_to(b,u,v):
      l = b[0]
      r = b[2]
      # sjekker at l,r,lambda,rho oppfyller kravene
      if '1' not in [l,r]:
            return 0
      if '1' not in [b[3],b[5]]:
            return 0
```

```
      # lager l*tip(fi)*r
      p = ''
      if v == '1':
            return 1
      if u == '1':
            return 0
      if l != '1':
            if r != '1':
                  p = l + u + r # dette skal ikke gå an
            else:
                  p = l + u
      else:
            if r != '1':
                  p = u + r
            else:
                  p = u #dette skal heller ikke gå an
      p2 = p[1:-1]
      s = len(v)
      t = len(p2)
      for i in range(0,t-s+1):
            if p2[i:i+s] == v:
                  return 1
      return 0


# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #
# OPPDATER OG BUCHBERGERS     #
# ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ #

# OPPDATER-ALGORITMEN
# input: G - en foreløpig Gröbnerbasis
#      Qgammel - en grunnmengde for G
#      g - et element som skal legges til i G
# output: Q, grunnmengde for G U {g}
def oppdater(G,Qgammel,g):
      Q = Qgammel
      # tester om g gjør tidligere elementer i Q redundante (sjekker
→ først hvilke elementer kriteriet
# holder for - fjerner dem i neste for-løkke for å unngå krøll med
→ indekser)
      R = []
      for i in range(len(Q)):
            if kriterium_to(Q[i],G[Q[i][1]].tip(),g.tip()):
                  R.append(i)
      R.reverse()
      for r in R:
            Q.pop(r)
      # lager alle nye blokkeringer
      O = get_overlapper(G,g) # er allerede fri for overflødige selv-
→ overlapper
      # ta ut redundante elementer fra O
      O = fjern_redundante(G,O)
      Q = Q + O
      return Q
```

```
# BUCHBERGERS ALGORITME
# input: F = [f1,...,fs] - en endelig mengde polynomer
# output: en endelig Gröbnerbasis G = [g1,...,gt]
# merk: algoritmen stopper ikke dersom idealet <F> har uendelig
→ Gröbnerbasis (den er heller ikke garantert å stoppe hvis <F> har
→ endelig Gröbnerbasis)
def buchberger(F):
      # initialiserer G og Q
      G = []
      Q = oppdater(G,[],F[0])
      G.append(F[0])
      # bygger Q fra F
      for i in range(1,len(F)):
            Q = oppdater(G,Q,F[i])
            G.append(F[i])
      # vurderer et og et element i Q
      while Q:
            sigma = Q.pop(0)
            b = blokkering_til_polynom(sigma)
            c = overlappsrelasjon(G[sigma[1]],G[sigma[4]],b)
            r = divalg(c,G)
            if r[0].mon:
                  r[0].monisk()
                  Q = oppdater(G,Q,r[0])
                  G.append(r[0])
      return G


# ========================== #
# MANUELL VERSJON            #
# ========================== #
# Der Buchbergers ikke kjører #
# fra start til "slutt", men  #
# man reduserer en og en blok-#
# kering "manuelt".           #
#   Merk at flere av metodene #
# som kalles er fra forrige   #
# del.                        #
# ========================== #

# FJERNE REDUNTANTE ELEMENTER DEL 1 (del 2 gjøres direkte i
→ Oppdater)
# input: G - en foreløpig Gröbnerbasis
#      Q - en liste med blokkeringer som skal reduseres slik at alle
→ blokkeringer som deles av en annen blokkering i Q taes ut
# output: oppdatert versjon av Q
# identisk med nonprint-versjonen, bortsett fra at den skriver ut de
→ redundante blokkeringene
def fjern_redundante_print(G,Q):
      T = [] #liste over redundante blokkeringer
      s = 0
      t1 = 1
      t2 = 1
      while s < len(Q):
            t1 = 1
            t2 = 1
```

```
            for i in Q[:s]:
                  if blokkering_deler(i,Q[s]):
                        print Q[s]
                        T.append(Q.pop(s))
                        t1 = 0
                        break
            for j in Q[s+1:]:
                  if blokkering_deler(j,Q[s]):
                        print Q[s]
                        T.append(Q.pop(s))
                        t2 = 0
                        break
            if t1 and t2:
                  s += 1
      if T:
            print 'redundante blant disse:'
            for t in T:
                  print t
      return Q

# OPPDATER-ALGORITMEN
# input: G - en foreløpig Gröbnerbasis
#      Qgammel - en grunnmengde for G
#      g - et element som skal legges til i G
# output: Q, grunnmengde for G U {g}
# identisk som forrige versjon, men skriver ut det som skjer
→ underveis
def oppdater_print(G,Qgammel,g):
      Q = Qgammel
      # tester om g gjør tidligere elementer i Q redundante (sjekker
→ først hvilke elementer kriteriet
      # holder for - fjerner dem i neste for-løkke for å unngå krøll
→ med indekser)
      R = []
      for i in range(len(Q)):
            if kriterium_to(Q[i],G[Q[i][1]].tip(),g.tip()):
                  R.append(i)
      R.reverse()
      if R:
            print 'vi tar da ut følgende elementer fra Q:'
      for r in R:
            print Q[r]
            Q.pop(r)
      # lager alle nye blokkeringer
      O = get_overlapper(G,g) # er allerede fri for overflødige selv-
→ overlapper
      print 'blokkeringene for dette elementet er: '
      if O:
            for o in O:
                  print o
      else:
            print 'ingen'
      # ta ut redundante elementer fra O
      O = fjern_redundante_print(G,O)
      Q = Q + O
      return Q
```

```
# INITIALISERING FOR BUCHBERGERS
# beregner alle overlapper i mengden F (reduserer dem ikke!)
# input: en tuppredusert mengde F
# output: en liste (kø) med overlappsrelasjonene fra mengden, Q
def init_buchberger(F):
      G = []
      print 'Kjører Buchbergers algoritme. Initialiserer G ...'
      print 'lagt til som element nr. ' + str(len(G)) + ' i G:'
      print F[0]
      Q = oppdater_print(G,[],F[0])
      G.append(F[0])
      # bygger Q fra F
      for i in range(1,len(F)):
            print ''
            print 'lagt til som element nr. ' + str(len(G)) + ' i G:'
            print F[i]
            Q = oppdater_print(G,Q,F[i])
            G.append(F[i])
      return Q


# 1-STEG-I-GANGEN-BUCHBERGER
# input: G - en foreløpig Gröbnerbasis
#      Q - en liste blokkeringer
# output: (G,Q) - (ny foreløpig Gröbnerbasis, oppdatert kø)
# tar ut første blokkering i køen, reduserer modulo G.
# dersom svaret er ulik 0, legges det til i G
# finner blokkeringer mellom det nye elementet og G, og legger til i
→ Q.
def pop_ko(G,Q):
      if Q:
            sigma = Q.pop(0)
            print 'vi velger ' + str(sigma) + ' fra Q'
            b = blokkering_til_polynom(sigma)
            c = overlappsrelasjon(G[sigma[1]],G[sigma[4]],b)
            r = divalg(c,G)
            print 'overlappsrelasjonen reduserte til ' + str(r[0])
            if r[0].mon:
                  r[0].monisk()
                  print 'lagt til som element nr. ' + str(len(G)) + '
→ i G:'
                  print r[0]
                  Q = oppdater_print(G,Q,r[0])
                  G.append(r[0])
      else:
            print "Q er tom"
      return (G,Q)
```