Voorwoord

Het is niet eenvoudig om voor een wiskundige discipline een naam te geven die de volledige inhoud van deze discipline dekt. Dit geldt eveneens voor *computeralgebra*. Gaat het in dit geval over een soort algebra? En waarop slaat de aanduiding "computer" precies?

In het vakgebied dat we omschrijven als computeralgebra, staat zonder enige twijfel één ding centraal: de studie van algoritmen die toelaten om symbolische en dus *exacte* berekeningen te realiseren met behulp van een computer. Dit maakt meteen het verschil duidelijk met een vakgebied zoals "numerieke analyse", waarbij de studie van *benaderende* methoden behandeld wordt. Het feit dat "algebra" voorkomt in de benaming wijst erop dat de wiskundige grondslagen en de behandelde vraagstellingen algebraïsch van aard zijn.

In de studie van algoritmen zal blijken dat de efficiëntie sterk afhangt van de gebruikte wiskunde. Hoe meer en hoe ingenieuzer men gebruik maakt van onderliggende abstracte structuren en hun eigenschappen, hoe beter men in staat is om efficiënte algoritmen te ontwerpen. Dit principe loopt als een rode draad door deze cursus, en het toont ook aan dat dit vakgebied, of toch datgene uit het vakgebied dat in deze cursus behandeld wordt, zuiver wiskundig van aard is, en dat de aanduiding "computer" geenszins impliceert dat "er niet moet nagedacht worden", of dat dit een cursus is "over computers" die argeloze wiskundigen in staat moet stellen beter met dergelijke duivelstuigen te werken.

Het dient opgemerkt te worden dat deze cursus slechts een inleiding vormt tot dit vakgebied, en we hebben gepoogd om verschillende aspecten ervan te belichten. Bij elk algoritme dat aan bod komt, besteden we aandacht aan het bewijs van de correctheid en aan een korte analyse van de complexiteit van de bestudeerde algoritmen.

Hoewel het vak computeralgebra aan de universiteit Gent ook vroeger reeds bestond, is deze cursus in 2007 *from scratch* herschreven. Een groot aantal onnauwkeurigheden was dan ook moeilijk te vermijden, en we hopen dat het grootste deel daarvan in de versie van dit jaar is weggewerkt. We zijn de studenten van de voorbije jaren zeer erkentelijk voor het opmerken van talloze fouten en onnauwkeurigheden, en we stellen het nog steeds ten

zeerste op prijs om daarvan op de hoogte gebracht te worden.

Deze cursus is hoofdzakelijk gebaseerd op het boek “Modern computer algebra” van Joachim von zur Gathen en Jürgen Gerhard [vzGG03].

Inhoudsopgave

Lijst van algoritmen

Hoofdstuk 1

Fundamentele algoritmen

1.1 Inleiding

We beginnen met een discussie over de representatie van en fundamentele algoritmen voor gehele getallen en veeltermen. We waarschuwen de lezer dat moderne processoren getallen *niet* voorstellen zoals wij hier zullen beschrijven, maar de trucs weergeven die deze gebruiken, zou ons afleiden van ons huidige doel, met name het geven van een eenvoudige beschrijving over hoe men in principe de basisbewerkingen kan uitvoeren. De algoritmen in dit hoofdstuk die de basisbewerkingen kunnen uitvoeren, noemen we klassiek, omdat ze in feite de basisbewerkingen uitvoeren op een manier zoals we ze ook handmatig zouden uitvoeren.

Het berekenen van de grootste gemene deler, inclusief Bézoutcoëfficiënten, en de Chinese reststelling zijn als fundamenteel te beschouwen. Heel veel andere algoritmen steunen hierop. Het uitgebreid algoritme van Euclides zoals in dit hoofdstuk besproken, is eveneens een klassiek algoritme, en het algoritme voor de Chinese reststelling beschrijven we als een toepassing ervan

1.1.1 Representatie en optelling van getallen

Algebra begint met de studie van getallen, en computers werken met data, dus het eerste probleem in computeralgebra is het ingeven van getallen in de computer. Data wordt opgeslagen in de vorm van *woorden*, en we gaan er even van uit dat we een processor beschouwen die gebruik maakt van woorden van 64 bits. Dan kunnen we met één woord de natuurlijke getallen weergeven tussen 0 en $2^{64}-1$, of indien we dat wensen, gehele getallen tussen -2^{63} en $2^{63}-1$; dergelijke getallen worden *single precision integers* genoemd.

Dit is echter dikwijls niet voldoende, en daarom wordt vaak gebruik gemaakt van *multiprecision integers*. Een dergelijk getal wordt voorgesteld door een array van 64-bit woorden, waarbij het eerste woord de lengte van de array en het teken van het getal codeert. Meer bepaald beschouwen we

de voorstelling van een getal a in *radix* 2^{64}, d.w.z.

$$a = s \cdot \sum_{i=0}^{n-1} a_i \cdot (2^{64})^i \,, \tag{1.1}$$

waarbij $s \in \{-1, 1\}$, $0 \leq n \leq 2^{63} - 1$, en $0 \leq a_i \leq 2^{64} - 1$ voor alle i. Deze informatie kunnen we coderen in een array

$$[s \cdot n, a_0, \ldots, a_{n-1}]$$

van 64-bit woorden. Deze representatie kan uniek gemaakt worden door te eisen dat de eerste digit a_{n-1} verschillend is van nul (behalve voor het getal $a = 0$ uiteraard, dan is $n = 0$ en bestaat a_0 niet). We noemen dit de *standaard representatie* voor het getal a.

Met deze voorstelling kunnen we alle gehele getallen weergeven tussen $-2^{2^{69}-64}+1$ en $2^{2^{69}-64}-1$. Dit is ruim voldoende voor praktische doeleinden: het opslaan van het grootste getal neemt 2^{63} woorden in beslag, en dit is ongeveer 68 miljard gigabyte.

Voor we verder gaan, voeren we eerst een belangrijke notatie in, die we verder dikwijls zullen gebruiken om de complexiteit van een algoritme te meten.

Definitie 1.1.1 (Big-O notatie)**.** Zij $f : \mathbb{R} \to \mathbb{R}$. Dan is $\mathcal{O}(f)$ de klasse van functies $g : \mathbb{R} \to \mathbb{R}$ die een groei hebben die gelimiteerd is door die van f. Wiskundig preciezer geformuleerd:

$$g \in \mathcal{O}(f) \iff \limsup_{x \to \infty} \left| \frac{g(x)}{f(x)} \right| < \infty \,.$$

Deze notatie wordt ook soms de *Landau notatie* of de *asymptotische notatie* genoemd, en wordt gelezen als "g is van de orde f".

Opmerking 1.1.2. Meestal zullen de functies f en g weergegeven worden door hun voorschrift op een gekozen variabele (b.v. x); we schrijven bijgevolg $2x^2 + 3 \in \mathcal{O}(x^2)$.

In de literatuur vindt men ook dikwijls de schrijfwijze $g(x) = \mathcal{O}(f(x))$ terug. Deze dient met de nodige voorzichtigheid behandeld te worden, want $g(x) = \mathcal{O}(f(x))$ en $h(x) = \mathcal{O}(f(x))$ impliceert geenszins $g(x) = h(x)$.

We vermelden alvast even de meest gebruikte asymptotische klassen, in stijgende volgorde van complexiteit; zie Tabel 1.1. Hierbij is c steeds een constante, en beschouwen we functies over de natuurlijke getallen met variabele n.

$\mathcal{O}(1)$	constant
$\mathcal{O}(\log^* n)$	geïtereerd logaritmisch
$\mathcal{O}(\log n)$	logaritmisch
$\mathcal{O}((\log n)^c)\,,\ c>1$	polylogaritmisch
$\mathcal{O}(n^c)\,,\ 0<c<1$	fractionele macht
$\mathcal{O}(n)$	lineair
$\mathcal{O}(n\log n)$	lineairitmisch, log-lineair, quasi-lineair
$\mathcal{O}(n^c)\,,\ c>1$	polynomiaal, algebraïsch, voor $c=2$ kwadratisch
$\mathcal{O}(c^n)\,,\ c>1$	exponentieel, geometrisch
$\mathcal{O}(n!)$	factorieel, combinatorisch
$\mathcal{O}(c_1^{c_2^n})\,,\ c_1,c_2>1$	dubbel-exponentieel

Tabel 1.1: Meest gebruikte asymptotische klassen

Notatie 1.1.3. Wanneer we $\log a$ schrijven zonder grondtal, zullen we steeds het *binaire logaritme* bedoelen, i.e. het algoritme met grondtal 2 (en dus *niet* met grondtal 10 zoals soms gebruikelijk is). Dus $\log a = \ln a / \ln 2$.

Definitie 1.1.4. Zij a een geheel getal. We definiëren de *lengte van* a als

$$\begin{cases} \lambda(0) := 0 \\ \lambda(a) := \lfloor \log_{2^{64}} |a| \rfloor + 1 \quad (a>0)\,. \end{cases}$$

Dan wordt een getal a tussen $-2^{64\cdot 2^{63}}+1$ en $2^{64\cdot 2^{63}}-1$ in de standaard representatie weergegeven als een array van lengte $\lambda(a)+1$ (1 voor het codewoord $s\cdot n$). Merk op dat $\lambda(a)\in\mathcal{O}(\log|a|)$.

We gaan er nu van uit dat onze processor een instructie bezit om twee single precision getallen op te tellen. Merk op dat de som van twee dergelijke getallen de maximale toegestane waarde kan overschrijden, en dus zal de processor een *carry vlag* aanzetten (i.e. waarde 1 geven) indien dit maximum wordt overschreden. Het is niet moeilijk om in te zien dat het volgend algoritme de optelling van twee positieve multiprecision getallen realiseert.

Algoritme 1.1 Optelling van twee positieve multiprecision integers

input: multiprecision getallen $a = \sum_{i=0}^{n-1} a_i \cdot 2^{64i}$, $b = \sum_{i=0}^{n-1} b_i \cdot 2^{64i}$
output: het multiprecision getal $a + b = \sum_{i=0}^{n} c_i \cdot 2^{64i}$

```
1  γ_0 ← 0
2  for i = 0, ..., n − 1
3        do c_i ← a_i + b_i + γ_i,    γ_{i+1} ← 0
4           if c_i ≥ 2^64
5              then c_i ← c_i − 2^64,    γ_{i+1} ← 1
6  c_n ← γ_n
7  return Σ_{i=0}^{n} c_i · 2^{64i}
```

Hoeveel tijd neemt dit algoritme nu in beslag? De basis subroutine, met name de optelling van twee single precision getallen, neemt een zeker aantal CPU-cycles in beslag, stel k. Dit getal k hangt af van de processor in kwestie, en de effectieve tijd hangt af van de snelheid van de CPU. Daarbij komen dan nog een aantal cycles voor controlestructuren, lussen, geheugentoegang, enzovoort. We zullen er echter (terecht) van uitgaan dat het aantal cycles voor dergelijke operaties van dezelfde grootte-orde is als het aantal opteloperaties dat we tellen.

Het is uitermate belangrijk om in het vervolg afstand te nemen van dergelijke architectuur-afhankelijke details als in de vorige paragraaf. We zullen eenvoudigweg zeggen dat het optellen van twee woorden van lengte n kan uitgevoerd worden in $\mathcal{O}(n)$ tijd, dus in lineaire tijd. (Merk op dat de complexiteit berekend wordt in functie van de *lengte* van de getallen, en dus *niet* in functie van de getallen zelf!) Deze abstractie is verantwoord en zelfs noodzakelijk, want door de moderne optimalisatietechnieken, zowel op compiler-niveau als op processor-niveau, is het quasi onmogelijk om de precieze uitvoertijd te berekenen[1].

We zullen bij de analyse van de algoritmen geregeld een elementaire complexiteitsanalyse uitvoeren, omdat het belangrijk is om in te zien hoe efficiënt een gegeven algoritme werkt; anderzijds willen hier zeker niet de nadruk op leggen. De lezer die geïnteresseerd is in een diepere studie, verwijzen we naar het boek [BCS97].

[1] [vzGG03] vermeldt dat het "echter experimenteel gebleken dat de "big-O" uitspraken verrassend goed aansluiten bij de werkelijkheid, in die zin dat het optellen van twee woorden van dubbele lengte bijna exact dubbel zoveel uitvoertijd vereist"

1.1.2 Representatie en optelling van polynomen

De twee belangrijkste datatypes waarop onze algoritmen zullen toegepast worden, zijn de gehele getallen zoals in paragraaf 1.1.1, en polynomen over een commutatieve ring R. (We beschouwen enkel ringen met eenheid $1 \in R$.)

Een polynoom $a \in R[x]$ in x over R kunnen we weergeven als een eindige rij $(a_0, \ldots, a_n)$ bestaande uit elementen van R, de *coëfficiënten* van a, zodat

$$a = \sum_{i=0}^{n} a_i x^i \,.$$

Als $a_n \neq 0$, dan is n de *graad* van a, en a_n is de *leidende coëfficiënt*, genoteerd als $\mathrm{lc}(a)$. Als $\mathrm{lc}(a) = 1$, dan wordt a een *monisch polynoom* genoemd. Het is dikwijls handig om de graad van het nulpolynoom per conventie gelijk aan $-\infty$ te stellen.

Wanneer we een algoritme uitwerken voor polynomen over een ring R, beschouwen we de ring R steeds als een basisgegeven, en we gaan er dus van uit dat we de standaardbewerkingen kunnen uitvoeren in R. We zullen de complexiteit van een algoritme dan uitdrukken in functie van het aantal nodige bewerkingen in R.

Merk de sterke analogie op tussen de standaard voorstelling van gehele getallen in radix 2^{64} zoals in vergelijking (1.1), en de voorstelling van een polynoom in $R[x]$, in het bijzonder voor $R = \mathbb{Z}/2^{64}\mathbb{Z}$, de ring van de gehele getallen modulo 2^{64}. Het is precies deze analogie die heel belangrijk is voor computeralgebra; heel wat van de algoritmen zijn, eventueel mits kleine aanpassingen, zowel voor gehele getallen als voor polynomen bruikbaar: vermenigvuldiging, deling met rest, grootste gemene deler, Chinese reststelling, ... Anderzijds zijn er een aantal heel fundamentele problemen waar de analogie niet kan doorgetrokken worden, zoals de theorie van de resultanten, en belangrijker, het factorisatieprobleem. Aan de basis van dit onderscheid ligt de op het eerste gezicht zeer onschuldige *carry overdracht*. Deze zorgt er namelijk voor dat lage digits invloed hebben op de hogere digits bij de optelling van gehele getallen, en brengt de mooi gescheiden regels bij de optelling van polynomen grondig in de war.

Het algoritme om twee polynomen in $R[x]$ met elkaar op te tellen is behoorlijk triviaal.

Algoritme 1.2 Optelling van twee polynomen in $R[x]$

input: polynomen $a = \sum_{i=0}^{n} a_i x^i$, $b = \sum_{i=0}^{n} b_i x^i$
output: het polynoom $a + b = \sum_{i=0}^{n} c_i x^i$
1 **for** $i = 0, \ldots, n$
2 **do** $c_i \leftarrow a_i + b_i$
3 **return** $\sum_{i=0}^{n} c_i x^i$

Het algoritme om polynomen op te tellen is eenvoudiger dan dat voor gehele getallen. Deze vaststelling zet zich verder voor de gecompliceerdere algoritmen. Hoewel we de gehele getallen als meer intuïtief ervaren, zijn hun algoritmen telkens iets ingewikkelder, zoals we ook zullen zien in het volgende hoofdstuk.

1.2 Klassieke algoritmen

1.2.1 Vermenigvuldiging in $R[x]$

We beschouwen het product $c = \sum_{i=0}^{m+n} c_i x^i$ van twee polynomen $a = \sum_{i=0}^{n} a_i x^i$ en $b = \sum_{i=0}^{m} b_i x^i$. De coëfficiënten van c worden gegeven door

$$c_k = \sum_{\substack{0 \le i \le n,\ 0 \le j \le m \\ i+j=k}} a_i b_j$$

voor elke $0 \le k \le n + m$. Uiteraard zouden we bovenstaande formule rechtstreeks in een algoritme kunnen gieten, maar het volgende algoritme is inzichtelijker en is in essentie hetzelfde op een andere volgorde van de lussen na.

Algoritme 1.3 Vermenigvuldiging van twee polynomen in $R[x]$

input: polynomen $a = \sum_{i=0}^{n} a_i x^i$, $b = \sum_{i=0}^{m} b_i x^i$
output: het polynoom $a \cdot b = \sum_{i=0}^{n+m} c_i x^i$
1 **for** $i = 0, \ldots, n$
2 **do** $d_i \leftarrow a_i x^i \cdot b = \sum_{j=0}^{m} (a_i b_j) x^{i+j}$
3 **return** $c = \sum_{i=0}^{n} d_i$

Hoeveel tijd neemt dergelijk algoritme nu in beslag? Met andere woorden, hoeveel operaties in de grondring R hebben we nodig?

Stelling 1.2.1. *Het vermenigvuldigen van twee polynomen van graad $\leq n$ met behulp van Algoritme* 1.3 *heeft complexiteit $\mathcal{O}(n^2)$.*

Bewijs. We zullen iets algemener een polynoom a van graad n en een polynoom b van graad m beschouwen. Elk van de $n+1$ coëfficiënten van a wordt vermenigvuldigd met elk van de $m+1$ coëfficiënten van b, samen goed voor $(m+1)(n+1) \in \mathcal{O}(mn)$ operaties in R. Deze producten worden opgeteld in verschillende sommen $c_0, \ldots, c_{n+m}$, samen goed voor $\mathcal{O}(mn)$ optellingen. Alles samen hebben we dus $\mathcal{O}(mn)$ operaties in R nodig. □

Een algoritme zoals het bovenstaande, dat de definitie van een bewerking vrij letterlijk implementeert, zullen we een *klassiek algoritme* noemen. Op het eerste gezicht is men geneigd te denken dat bovenstaand algoritme de enige manier is om de vermenigvuldiging te implementeren, maar verrassend genoeg (en gelukkig maar) is dit niet het geval. We zullen zien dat het zelfs mogelijk is om een vermenigvuldiging in bijna lineaire tijd uit te voeren!

Voor de optelling is er geen verbetering meer mogelijk, maar dit is ook niet nodig: dit algoritme gebruikt slechts lineaire tijd.

Tenslotte merken we nog op dat het klassieke algoritme voor de vermenigvuldiging van twee multiprecision getallen identiek is aan Algoritme 1.3, en we laten de details dan ook achterwege.

In heel wat toepassingen spelen berekeningen "modulo een geheel getal" of "modulo een polynoom" een belangrijke rol. Een basishulpmiddel hierbij is de berekening van een deling met rest: gegeven twee gehele getallen a, b met $b \neq 0$, willen we gehele getallen q, r vinden zodanig dat $a = qb + r$, met $|r| < |b|$. We noemen q het *quotiënt* en r de *rest* van de deling.

We behandelen eerst het computationele aspect van dit probleem voor polynomen. We starten dus met twee polynomen $a, b \in R[x]$, $b \neq 0$, en we gaan op zoek naar $q, r \in R[x]$ zodat $a = qb + r$, met $\deg r < \deg b$. Een eerste probleem is dat q en r niet altijd bestaan: het is bijvoorbeeld onmogelijk om x^2 door $2x + 1$ te delen in $\mathbb{Z}[x]$. Dit probleem kan opgelost worden met behulp van de zogenaamde *pseudo-deling* die we later (zie paragraaf 3.1) zullen behandelen. Op dit moment zullen we het probleem vereenvoudigen door te veronderstellen dat de leidende coëfficiënt $\mathrm{lc}(b)$ een eenheid is in R, m.a.w. dat er een $\nu \in R$ bestaat met $\mathrm{lc}(b)\nu = 1$. Voor $R = \mathbb{Z}$ laat dit enkel nog $\mathrm{lc}(b) = \pm 1$ toe, maar wanneer R een veld is is dit helemaal geen beperking. We merken op dat uit deze veronderstelling volgt dat het quotiënt en de rest uniek bepaald zijn. Inderdaad, stel dat $a = qb + r = q'b + r'$, dan

zou $(q' - q)b = r - r'$; het rechterlid heeft graad kleiner dan $\deg b$, maar het linkerlid heeft graad ten minste $\deg b$ (want $\mathrm{lc}(b)$ is een eenheid!), tenzij $q' = q$ (en dan ook $r' = r$). We noteren $q = a \operatorname{quo} b$ en $r = a \operatorname{rem} b$.

1.2.2 Deling met rest in $R[x]$

Het volgend algoritme formaliseert de gekende klassieke methode om een deling met rest uit te voeren.

Algoritme 1.4 Polynomiale deling met rest

input: polynomen $a = \sum_{i=0}^{n} a_i x^i$, $b = \sum_{i=0}^{m} b_i x^i$ in $R[x]$
met b_m een eenheid, $n \geq m$.
output: polynomen $q, r \in R[x]$ met $a = qb + r$ en $\deg r < m$.

```
1  r ← a
2  for i = n − m, ..., 0
3        do if deg r = m + i
4              then q_i ← lc(r)/b_m,  r ← r − q_i x^i b
5              else  q_i ← 0
6  return q = Σ_{i=0}^{n−m} q_i x^i en r
```

Stelling 1.2.2. *De deling van een polynoom van graad n door een polynoom van graad $m \leq n$ met behulp van Algoritme 1.4 vereist $\mathcal{O}(m(n-m))$ operaties in R.*

Bewijs. Alle operaties in R gebeuren in regel 4 van het algoritme. Per uitvoering van de lus zijn er $\mathcal{O}(m)$ berekeningen nodig. Aangezien de lus $n-m+1$ keer doorlopen wordt bewijst dit het gestelde. □

Opmerking 1.2.3. Aangezien $m \leq n$ and $n-m \leq n$, kunnen we ook zeggen dat dit algoritme complexiteit $\mathcal{O}(n^2)$ heeft.

In essentie is het klassieke algoritme voor deling van gehele getallen met rest het zelfde. De details zijn iets ingewikkelder omwille van carry-overdrachten, en we laten deze discussie dan ook achterwege. In elk geval is het resultaat ook hier dat deling van een getal a van lengte n door een getal b van lengte m complexiteit $\mathcal{O}(m(n-m))$ heeft. Merk ook op dat $q = a \operatorname{quo} b$ en $r = a \operatorname{rem} b$ niet uniek bepaald zijn door de voorwaarde $|r| < |b|$ alleen; we zullen de gebruikelijke conventie aannemen dat $r \geq 0$ (ook als a of b zelf negatief is), waarna we opnieuw uniciteit van q en r hebben.

1.2.3 Machtsverheffing in een ring

Tenslotte geven we nog een eenvoudig algoritme om efficiënt een element a in een associatieve ring tot een macht $n \in \mathbb{N}$ te verheffen; dit is het zogenaamde "repeated squaring" algoritme.

Algoritme 1.5 Repeated squaring

input: $a \in R$, met R een associatieve ring met eenheid, en $1 \leq n \in \mathbb{N}$.
output: $a^n \in R$.
1 Schrijf $n = 2^k + n_{k-1} \cdot 2^{k-1} + \cdots + n_1 \cdot 2 + n_0$ met $n_i \in \{0, 1\}$.
{ binaire representatie van n }
2 $b_k \leftarrow a$
3 **for** $i = k - 1, \ldots, 0$
4 **do if** $n_i = 0$
5 **then** $b_i \leftarrow b_{i+1}^2$
6 **else** $b_i \leftarrow b_{i+1}^2 \cdot a$
7 **return** b_0

Stelling 1.2.4. *Zij $a \in R$ en $1 \leq n \in \mathbb{N}$. De berekening van a^n met behulp van Algoritme 1.5 vereist ten hoogste $2 \log n$ operaties in R.*

Bewijs. De correctheid van het algoritme volgt uit het feit dat $b_i = a^{\lfloor n/2^i \rfloor}$ voor alle $i \in \{k, \ldots, 0\}$. Het gebruikt $\lfloor \log n \rfloor$ kwadrateringen, en $w(n) - 1$ vermenigvuldigingen in R, waarbij $w(n)$ het *Hamming gewicht* van n is, i.e. het aantal enen in de binaire representatie van n. In elk geval is $w(n) \leq \lfloor \log(n) \rfloor + 1$ (vergelijk met Definitie 1.1.4), en bijgevolg zijn er ten hoogste $2\lfloor \log(n) \rfloor$ bewerkingen in R nodig. □

Dit eenvoudige algoritme was reeds door Euler gekend, die het gebruikte om $7^{160} \pmod{641}$ te berekenen door achtereenvolgens 7^2, 7^4, 7^8, 7^{16}, 7^{32}, 7^{64}, 7^{128}, $7^{160} = 7^{128} \cdot 7^{32}$ modulo 641 te reduceren.

1.3 Grootste gemene deler

Definitie 1.3.1. Beschouw een ring R en twee elementen $a, b \in R$. Een element $c \in R$ wordt een *grootste gemene deler van a en b*, kortweg een *gcd* genoemd, als

(i) $c \mid a$ en $c \mid b$,

(ii) als $d \mid a$ en $d \mid b$ dan $d \mid c$ voor alle $d \in R$.

Een element $c \in R$ wordt een *kleinste gemeen veelvoud van a en b*, kortweg een *lcm* genoemd, als

(i) $a \mid c$ en $b \mid c$,

(ii) als $a \mid d$ en $b \mid d$, dan $c \mid d$ voor alle $d \in R$.

Een element $u \in R$ wordt een *eenheid* genoemd als er een multiplicatief invers bestaat, m.a.w. als er een element $v \in R$ bestaat zodat $uv = 1$. Twee elementen $a, b \in R$ worden *geassocieerd* genoemd, genoteerd $a \sim b$, als $a = ub$ voor een eenheid $u \in R$. Twee elementen $a, b \in R$ worden *relatief priem* of *copriem* genoemd als ze een gcd hebben die een eenheid is.

In een willekeurige ring is het niet noodzakelijk dat een gcd en een lcm van twee elementen altijd bestaan. In de ring $\mathbb{Z}$ bestaan er voor elke twee elementen een gcd en een lcm, maar deze zijn niet uniek. Men toont eenvoudig aan dat twee verschillende gcds van twee elementen uit een domein altijd geassocieerd zijn.

Stel dat R een domein is. Een element $p \in R \setminus \{0\}$, verschillend van een eenheid, is *reduciebel* als er elementen $a, b \in R \setminus \{0\}$ bestaan, beide verschillend van een eenheid, zodanig dat $p = ab$. Een element $p \in R \setminus \{0\}$, verschillend van een eenheid is *irreduciebel* als het niet reduciebel is. Eenheden van het domein zijn dus noch reduciebel, noch irreduciebel. Een element $p \in R \setminus \{0\}$ verschillend van een eenheid is *priem* als $p \mid ab$ impliceert dat $p \mid a$ of $p \mid b$ voor alle $a, b \in R$. Een domein R is een *uniek factorisatiedomein (UFD)* als elk element $a \in R \setminus \{0\}$ verschillend van een eenheid op een unieke wijze, op herordening en vermenigvuldiging met eenheden na, kan geschreven worden als het product van irreduciebele elementen van R. Als R een uniek factorisatiedomein is, dan volgt uit de uniciteit van de ontbinding in irreduciebele elementen echter dat twee elementen $a, b \in R$ wel degelijk een grootste gemene deler hebben. Twee typische voorbeelden van UFDs zijn de ringen $\mathbb{Z}$ en $F[X]$, F een veld.

Vanuit wiskundig standpunt is het al dan niet uniek zijn van een gcd van twee elementen a en b een onschadelijk neveneffect van de structuur van de ring waartoe a en b behoren. Het is echter wenselijk dat een computeralgebrasysteem steeds dezelfde gcd berekent voor twee gegeven elementen.

Stel dat R een domein is waarvoor geldt dat elke twee elementen een gcd hebben. Het probleem is opgelost als we voor een willekeurig element $a \in R$ een representant gekozen hebben voor de verzameling $\{a' \in R \mid a' \sim a\}$. We noteren dit element a' als normal(a) en noemen dit ook de *normale vorm*

van a. Aangezien $a \sim \text{normal}(a)$, bestaat er een eenheid u zodanig dat $a = u \cdot \text{normal}(a)$. Het element u wordt de leidende eenheid van a genoemd, genoteerd $\text{lu}(a)$. Als conventie stellen we dat $\text{normal}(0) = 0$ en $\text{lu}(0) = 1$. De volgende twee eigenschappen zijn vereist voor de functie normal:

(i) twee elementen van R hebben dezelfde normale vorm als en slechts als ze geassocieerd zijn, en,

(ii) de normale vorm van een product is gelijk aan het product van de normale vormen.

Deze eigenschappen impliceren dat $\text{normal}(u) = 1$ voor elke eenheid $u \in R$. We noemen een element a dat in normale vorm is, m.a.w. $a = \text{normal}(a)$ of $\text{lu}(a) = 1$, *genormaliseerd*.

We veronderstellen nu dat een functie normal beschikbaar is over het domein R. De notatie $\gcd(a, b)$, met $a, b \in R$ stelt dan de unieke, genormaliseerde gcd voor van de elementen a en b. Het volgend lemma beschrijft een aantal eigenschappen van de gcd in het domein R.

Lemma 1.3.2. (i) $\gcd(a, b) = a \iff a \mid b$,

(ii) $\gcd(a, a) = \gcd(a, 0) = a$ *en* $\gcd(a, 1) = 1$,

(iii) $\gcd(a, b) = \gcd(b, a)$,

(iv) $\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c))$,

(v) $\gcd(c \cdot a, c \cdot b) = c \cdot \gcd(a, b)$

(vi) $a \sim b \Longrightarrow \gcd(a, c) = \gcd(b, c)$.

(vii) $\gcd(a, b) = \gcd(b, r)$ *als* $a = b \cdot q + r$.

Bewijs. Als oefening. □

Stel nu dat $R = F[X]$, met F een veld. Elk element van F is een eenheid in de ring R. Voor een willekeurig element $p \in R \setminus \{0\}$, noemen we de coëfficiënt van de hoogste graadsterm de *leidende coëfficiënt*, genoteerd als $\text{lc}(p)$. Aangezien dit een eenheid is, kunnen we $\text{lu}(p) = \text{lc}(p)$ definiëren. Met de conventie dat $\text{lu}(0) := 1$, kunnen we dan de functie normal definiëren als $\text{normal}(p) = p/\text{lc}(p)$. Een polynoom verschillend van 0 zal genormaliseerd zijn als en slechts als het monisch is.

Stel nu dat $R = \mathbb{Z}$. Stellen we $\text{lu}(a) = \text{sign}(a)$, voor elk element $a \in R$. Dan definiëren we $\text{normal}(a) = a/\text{lu}(a) = |a|$, als $a \neq 0$ (met de conventie dat $\text{normal}(0) = 0$). Een geheel getal is dus genormaliseerd als en slechts als het niet negatief is.

1.3.1 Het uitgebreid algoritme van Euclides

Definitie 1.3.3. Een domein R, samen met een afbeelding $d : R \to \mathbb{N} \cup \{-\infty\}$ is een *Euclidisch domein* als er voor alle $a, b \in R$, $b \neq 0$ twee elementen $q, r \in R$ bestaan waarvoor $a = qb + r$ en $d(r) < d(b)$. De elementen q, respectievelijk r, worden het *quotiënt*, respectievelijk de *rest*, van de deling van a door b genoemd. De afbeelding d wordt een *Euclidische functie* genoemd. Als de Euclidische functie d duidelijk is uit de context, zullen we deze niet expliciet vermelden.

Opmerking 1.3.4. Het quotiënt en de rest van de deling van a door b van twee elementen a, b van een Euclidisch domein (R, d) zijn niet noodzakelijk uniek.

Opmerking 1.3.5. Het is eenvoudig in te zien dat elk Euclidisch domein R een hoofdideaaldomein (PID) is. Inderdaad, beschouw een niet-triviaal ideaal $I \trianglelefteq R$, en kies een $a \in I \setminus \{0\}$ waarvoor $d(a) \in \mathbb{N}$ minimaal is. Dan zal $I = aR$.

Voorbeeld 1.3.6.

(i) Stel $R = \mathbb{Z}$ en $d(a) := |a|$ voor alle $a \in \mathbb{Z}$, dan is (R, d) een Euclidisch domein. Het quotiënt en de rest zijn uniek als we eisen dat $r \geq 0$.

(ii) Beschouw een veld F, stel $R = F[X]$ en $d(a) := \deg a$ voor alle $a \in F[X] \setminus \{0\}$ en $d(0) := -\infty$, dan is (R, d) een Euclidisch domein en bovendien zijn het quotiënt en de rest uniek.

(iii) Stel $R = \mathbb{Z}[i]$, de *ring der gehelen van Gauß*, $\mathbb{Z}[i] = \{a + bi \mid a, b \in \mathbb{Z}\}$, $i^2 = -1$. Toon aan dat (R, d) een Euclidisch domein is, met $d(a+bi) := a^2 + b^2$, voor alle $a + bi \in \mathbb{Z}[i]$.

(iv) Stel dat R een veld is, en definieer $d(a) := 1$ voor alle $a \in R^*$, en $d(0) = -\infty$. Dan zal (R, d) een Euclidisch domein zijn.

(v) Niet elk hoofdideaaldomein is een Euclidisch domein. Een voorbeeld hiervan wordt gegeven door $\mathbb{Z}\left[\frac{1+\sqrt{-19}}{2}\right]$.

Een Euclidisch domein is steeds een UFD. De volgende stelling is immers bekend.

Stelling 1.3.7. *Een hoofdideaaldomein is een uniek factorisatiedomein.*

Zonder bewijs. □

Gevolg 1.3.8. *Een Euclidisch domein is een uniek factorisatiedomein.*

Lemma 1.3.2, Eigenschap (vii), geeft onmiddellijk aanleiding tot een algoritme om in een Euclidisch domein R een grootste gemene deler van twee elementen te berekenen. Gegeven een Euclidisch domein R met bijhorende Euclidische functie d en twee elementen $a, b \in R$, dan geldt dat $a = bq + r$, met $d(r) < d(b)$. Passen we dit recursief toe, met $r_0 := a$, $r_1 := b$, $r_{i+1} = \text{rem}(r_{i-1}, r_i)$, dan zal de laatste niet nul rest een grootste gemene deler zijn van a en b. Immers, de waarden $d(r_i)$ vormen een strikt dalende rij en gelet op eigenschap (ii) en (vii), is het duidelijk dat dit algoritme altijd eindigt en een gcd bepaalt. Een eenvoudig voorbeeld in $\mathbb{Z}$ illustreert de werking van dit algoritme. We berekenen een grootste gemene deler van 126 en 35:

$$\begin{aligned} 126 &= 3 \cdot 35 + 21 \\ 35 &= 1 \cdot 21 + 14 \\ 21 &= 1 \cdot 14 + 7 \\ 14 &= 2 \cdot 7 \end{aligned}$$

We besluiten dat 7 een gcd van 126 en 35 is. In dit geval is het ook onmiddellijk duidelijk dat -7 een gcd is.

Bovenstaand voorbeeld toont aan dat de bekomen gcd kan geschreven worden als een lineaire combinatie van de elementen 126 en 35:

$$7 = 21 - 1 \cdot 14 = 21 - (35 - 1 \cdot 21) = 2 \cdot (126 - 3 \cdot 35) - 35 = 2 \cdot 126 - 7 \cdot 35$$

Aan de hand van deze redenering kunnen we ons algoritme uitbreiden, zodanig dat ook deze lineaire combinatie bepaald wordt. Algemeen worden de elementen s en t waarvoor $sa + tb = \gcd(a, b)$ de Bézoutcoëfficiënten genoemd. Algoritme 1.6 wordt het Uitgebreid algoritme van Euclides genoemd. In de vorm waarin het hier besproken wordt, bepaalt het steeds de unieke genormaliseerde gcd op voorwaarde dat over het Euclidisch domein R een functie normal beschikbaar is.

Algoritme 1.6 Uitgebreid algoritme van Euclides

input: $f, g \in R$, met R een Euclidisch domein.
output: r, s, t, met $r = \gcd(f, g) = sf + tg$.

```
1  ρ_0 ← lu(f), r_0 ← normal(f); s_0 ← ρ_0^{-1}; t_0 ← 0
2  ρ_1 ← lu(g), r_1 ← normal(g); s_1 ← 0; t_1 ← ρ_1^{-1}
3  i ← 1
4  while r_i ≠ 0
5        do q_i ← quo(r_{i-1}, r_i)
6           ρ_{i+1} ← lu(r_{i-1} − q_i r_i)
7           r_{i+1} ← (r_{i-1} − q_i r_i)/ρ_{i+1}
8           s_{i+1} ← (s_{i-1} − q_i s_i)/ρ_{i+1}
9           t_{i+1} ← (t_{i-1} − q_i t_i)/ρ_{i+1}
10          i ← i + 1
11 l ← i − 1
12 return r_l, s_l, t_l
```

In het volgende lemma bewijzen we een aantal relaties tussen de elementen r_i, s_i, t_i, q_i en ρ_i. Eén van de te bewijzen relaties zal onmiddellijk de correctheid van Algoritme 1.6 opleveren. Andere relaties zullen nuttig zijn in de afschatting van de complexiteit.

Lemma 1.3.9. *Voor alle $i \in \{0, \ldots, l\}$ geldt:*

(i) $R_i \cdot \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$,

(ii) $R_i = \begin{pmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{pmatrix}$,

(iii) $\gcd(f, g) = \gcd(r_i, r_{i+1}) = r_l$,

(iv) $s_i f + t_i g = r_i$, *dit geldt ook voor* $i = l + 1$,

(v) $s_i t_{i+1} - t_i s_{i+1} = (-1)^i (\rho_0 \cdots \rho_{i+1})^{-1}$,

(vi) $\gcd(r_i, t_i) = \gcd(f, t_i)$,

(vii) $f = (-1)^i \rho_0 \cdots \rho_{i+1}(t_{i+1} r_i - t_i r_{i+1})$,

$$g = (-1)^{i+1} \rho_0 \cdots \rho_{i+1}(s_{i+1} r_i - s_i r_{i+1}).$$

Bewijs. Het is duidelijk dat de elementen r_i, $0 \leq i \leq l+1$ de genormaliseerde resten zijn en de elementen q_i de quotiënten die optreden bij de opeenvolgende

uitvoering van de delingen met rest. De belangrijkste eigenschap is dat $s_i f + t_i g = r_i$, voor alle i. Meer bepaald geldt ook dat $s_l f + t_l g = r_l$ en dit is de gcd van f en g. We definiëren

$$R_0 := \begin{pmatrix} s_0 & t_0 \\ s_1 & t_1 \end{pmatrix}, \quad Q_i := \begin{pmatrix} 0 & 1 \\ \rho_{i+1}^{-1} & -q_i \rho_{i+1}^{-1} \end{pmatrix} \quad (1 \leq i \leq l),$$

dit zijn 2×2 matrices over de ring R, en $R_i := Q_i \cdots Q_1 R_0$, $0 \leq i \leq l$.

We bewijzen (i) en (ii) door middel van inductie. Het geval $i = 0$ volgt onmiddellijk uit de initialisaties (lijnen 1 en 2). Stel $i \geq 1$ dan geldt

$$Q_i \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \rho_{i+1}^{-1} & -q_i \rho_{i+1}^{-1} \end{pmatrix} \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} = \begin{pmatrix} r_i \\ (r_{i-1} - q_i r_i)\rho_{i+1}^{-1} \end{pmatrix} = \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$$

en (i) volgt uit $R_i = Q_i R_{i-1}$ en de inductiehypothese.

Uit de definitie van R_0 en de initialisaties volgt onmiddellijk (ii) voor $i = 0$. Als inductiehypothese veronderstellen we dat $i \geq 1$ en dat (ii) geldig is voor $i - 1$. Dan volgt, uit lijnen 8 en 9 van het algoritme

$$Q_i \begin{pmatrix} s_{i-1} & t_{i-1} \\ s_i & t_i \end{pmatrix} = \begin{pmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{pmatrix} = Q_i R_{i-1} = R_i,$$

waaruit (ii).

Stel $i \in \{0, \ldots, l\}$. Uit (i) volgt dat

$$\begin{pmatrix} r_l \\ 0 \end{pmatrix} = R_l \begin{pmatrix} f \\ g \end{pmatrix} = Q_l \cdots Q_{i+1} R_i \begin{pmatrix} f \\ g \end{pmatrix} = Q_l \cdots Q_{i+1} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix},$$

hieruit volgt dat r_l een lineaire combinatie is van r_i en r_{i+1}, dus elke gemeenschappelijke deler van r_i en r_{i+1} is een deler van r_l. Men gaat ook gemakkelijk na dat elke Q_i een inverse heeft over R, namelijk

$$Q_i^{-1} = \begin{pmatrix} q_i & \rho_{i+1} \\ 1 & 0 \end{pmatrix},$$

waaruit

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q_{i+1}^{-1} \cdots Q_l^{-1} \begin{pmatrix} r_l \\ 0 \end{pmatrix}.$$

Hieruit volgt dat r_l een deler is van r_i en r_{i+1}, en omdat r_l genormaliseerd is, geldt $r_l = \gcd(r_i, r_{i+1})$. Voor $i = 0$ vinden we dus dat $\gcd(f, g) = \gcd(r_0, r_1) = r_l$, waarmee (iii) aangetoond is.

Eigenschap (iv) volgt onmiddellijk uit (i) en (ii). Uit (ii) en de definitie van R_i volgt ook dat

$$\det R_i = \det Q_i \cdots \det Q_1 \det \begin{pmatrix} s_0 & t_0 \\ s_1 & t_1 \end{pmatrix} = (-1)^i (\rho_0 \cdots \rho_{i+1})^{-1}$$

waaruit dus (v) volgt.

Uit eigenschap (v) volgt ook dat $\gcd(s_i, t_i) = 1$ en dat R_i een inverse heeft. Stel nu dat $p \in R$ een deler is van t_i. Als $p \mid f$, dan $p \mid s_i f + t_i g = r_i$. Als $p \mid r_i$ dan is p ook een deler van $s_i f = r_i - t_i g$ en dus $p \mid f$ omdat s_i en t_i copriem zijn. Daaruit volgt (vi).

We vermenigvuldigen nu beide leden van (i) met R_i^{-1} en door (ii) en (v) vinden we

$$\begin{pmatrix} f \\ g \end{pmatrix} = R_i^{-1} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = (-1)^i (\rho_0 \cdots \rho_{i+1}) \begin{pmatrix} t_{i+1} & -t_i \\ -s_{i+1} & s_i \end{pmatrix} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$$

waaruit (vii) volgt door deze vergelijking te schrijven als een stelsel lineaire vergelijkingen. Merk tenslotte ook op dat $r_{l+1} = 0$ na uitvoering van het algoritme. □

Gevolg 1.3.10. *In een Euclidisch domein R hebben elke twee elementen f en g een gcd $h \in R$, en deze kan uitgedrukt worden als een lineaire combinatie $h = sf + tg$, met $s, t \in R$.*

Gevolg 1.3.11. *Algoritme 1.6 bepaalt de genormaliseerde gcd van twee elementen $f, g \in R$ en de bijhorende elementen s_i, t_i en r_i, $0 \leq i \leq l$.*

Normalisatie in $\mathbb{Z}$ of $F[X]$ is een goede oplossing om een unieke gcd in deze ringen te bepalen. Voor een computeralgebrasysteem is dit echter niet voldoende. Stel $f = -10x$ en $g = 5x^2$. Zowel f als g behoren tot $\mathbb{Z}[x]$ als $\mathbb{Q}[x]$. In $\mathbb{Q}[x]$ is het duidelijk dat $\gcd(f, g) = x$. In $\mathbb{Z}[x]$ zijn er twee kandidaten voor een gcd: $-5x$ en $5x$. Enerzijds moeten we dus voor $\mathbb{Z}[x]$ een geschikte normaliserende functie definiëren, anderzijds illustreert dit dat de gebruiker aan het computeralgebrasysteem gaat moeten duidelijk maken in welke ring de grootste gemene deler uitgerekend moet worden.

Het eerste probleem kunnen we als volgt oplossen. Stel dat R een domein is met een normaliserende functie normal_R. Dan definiëren we

$$\text{normal}_{R[X]}(f) := \frac{\text{normal}_R(\text{lc}(f))}{\text{lc}(f)} \cdot f \tag{1.2}$$

Op deze manier wordt op inductieve wijze een normaliserende functie gedefinieerd en kunnen we een gcd op unieke wijze definiëren voor polynomen over $\mathbb{Z}$ of een veld F in meerdere variabelen. Merk echter wel op dat het (uitgebreid) algoritme van Euclides enkel toepasbaar is in Euclidische domeinen. De ring $\mathbb{Z}[x]$ is echter geen Euclidisch domein, hoewel elke twee elementen er een gcd hebben, dit omdat $\mathbb{Z}[x]$ wel een *uniek factorisatie domein* is. De bepaling van een gcd in een uniek factorisatie domein dat geen Euclidisch domein is, komt aan bod vanaf Sectie 3.1

1.3.2 Complexiteit in $\mathbb{Z}$ en $F[X]$

In deze sectie bestuderen we de complexiteit van het uitgebreid algoritme van Euclides, Algoritme 1.6, uitgaande van de klassieke algoritmen om vermenigvuldiging en deling met rest uit te voeren

Gegeven een Euclidisch domein (R, d) en twee elementen $f, g \in R$, met $n = d(f) \geq d(g) = m \geq 0$. Het aantal delingen dat uitgevoerd wordt in Algoritme 1.6 is begrensd door $l \leq d(g) + 1$. We starten de analyse met het geval $R = F[X]$, dan is $d = \deg$. We definiëren $n_i := \deg r_i$, $0 \leq i \leq l+1$, met $r_{l+1} = 0$. Dan geldt $n_0 = n \geq n_1 = m > n_2 > \cdots > n_l$, en $\deg q_i = n_{i-1} - n_i$, $1 \leq i \leq l$. Volgend lemma geeft informatie over de graad van de elementen s_i en t_i.

Lemma 1.3.12.

$$\deg s_i = \sum_{j=2}^{i-1} \deg q_j = n_1 - n_{i-1},\ 3 \leq i \leq l+1, \tag{1.3}$$

$$\deg t_i = \sum_{j=1}^{i-1} \deg q_j = n_0 - n_{i-1},\ 2 \leq i \leq l+1. \tag{1.4}$$

Bewijs. Per inductie op i tonen we

$$\deg s_{i-1} < \deg s_i, 2 \leq i \leq l+1 \tag{1.5}$$

en uitdrukking (1.3) aan. Voor $i = 2$ vinden we $s_2 = (s_0 - q_1 s_1)\rho_2^{-1} = (\rho_0^{-1} - q_1 \cdot 0)\rho_2^{-1}$, en $\deg s_1 = -\infty < 0 = \deg s_2$. Voor $i = 3$ vinden we $s_3 = (s_1 - q_2 s_2)\rho_3^{-1}$, dus $\deg s_3 = \deg q_2 + \deg s_2 = \deg q_2$ omdat $\deg s_1 < \deg s_2$ en $\deg s_2 = 0$.

Stel nu dat $i \geq 3$ en dat uitdrukking 1.3 geldig is voor $3 \leq i \leq k$. Dan volgt

$$\deg s_{i-1} < \deg s_i < n_{i-1} - n_i + \deg s_i = \deg(q_i s_i),$$

wat impliceert dat

$$\deg s_{i+1} = \deg(s_{i-1} - q_i s_i) = \deg q_i + \deg s_i > \deg s_i,$$

en

$$\deg s_{i+1} = \deg q_i + \deg s_i = \sum_{j=2}^{i-1} \deg q_j + \deg q_i = \sum_{j=2}^{i} \deg q_j\,.$$

Ook uitdrukking (1.4) tonen we op een analoge wijze aan. De beginsituatie is echter licht gewijzigd omdat $t_1 \neq 0$. Voor $i = 2$ vinden we

$t_2 = (t_0 - q_1 t_1)\rho_2^{-1}$, dus $\deg t_2 = \deg q_1 = n - m = n_0 - n_1$, hetgeen mogelijk 0 is. Aangezien $\deg t_1 = 0$, kunnen we de inductie maar laten starten vanaf $i = 3$, als we een analoge uitdrukking willen gebruiken als uitdrukking (1.5) voor t_i. Voor $i = 3$ vinden we $t_3 = (t_1 - q_2 t_2)\rho_3^{-1}$, dus $\deg t_3 = \deg q_2 + \deg t_2 = \deg q_2 + \deg q_1 = n_0 - n_1 + n_1 - n_2 = n_0 - n_2$. We tonen nu door inductie op i uitdrukking (1.4) en

$$\deg t_{i-1} < \deg t_i, 3 \leq i \leq l+1 \tag{1.6}$$

aan.

Stel $i \geq 3$ en stel dat uitdrukking 1.5 juist is voor $3 \leq i \leq k$. Volkomen analoog volgt nu dat $\deg t_{i+1} > \deg t_i$ en $\deg t_{i+1} = \sum_{j=1}^{i} \deg q_j$. Uitdrukking (1.4) voor $i = 1$ volgt (behalve de sommatie) onmiddellijk uit $t_1 = \rho_1^{-1}$. □

Stelling 1.3.13. *Om het uitgebreid algoritme van Euclides, Algoritme 1.6, uit te voeren voor polynomen $f, g \in F[X]$, waarbij $\deg f = n \geq \deg g = m$, zijn $\mathcal{O}(nm)$ operaties nodig in F.*

Bewijs. De deling van r_{i-1} door r_i kan uitgevoerd worden door hoogstens $Cn_i(n_{i-1} - n_i)$ operaties in F, met C een wel bepaalde constante. (Stelling 1.2.2). Om alle q_i en r_{i+1} te berekenen voor $1 \leq i \leq l$ zijn dus $l - 1 \leq m$ inversies en $\sum_{i=1}^{l} (Cn_i(n_{i-1} - n_i))$ optellingen en vermenigvuldigingen nodig. Verder is het duidelijk dat

$$\mathcal{O}\big(\sum_{i=1}^{l} (Cn_i(n_{i-1} - n_i))\big) \subseteq \mathcal{O}\big(Cm \sum_{i=1}^{l} (n_{i-1} - n_i)\big) \subseteq \mathcal{O}(nm),$$

waardoor we besluiten dat er $\mathcal{O}(nm)$ operaties nodig zijn om r_i, q_i en ρ_i te bepalen.

De berekening van $q_i t_i$ vergt hoogstens $C'(n_{i-1} - n_i)(n - n_{i-1})$ operaties (Stelling 1.2.1 en Lemma 1.3.12). Dit product aftrekken van t_{i-1} en dit dan vermenigvuldigen met ρ_{i+1}^{-1} levert t_{i+1} en vergt hoogstens $\mathcal{O}(n - n_i)$ operaties. Bijgevolg zijn er, om alle t_i te berekenen

$$\sum_{i=1}^{l} \big(C'(n_{i-1} - n_i)(n - n_{i-1})\big) \in \mathcal{O}(nm)$$

operaties in F nodig. Een analoge redenering toont aan dat de berekening van $q_i s_i$ hoogstens $\mathcal{O}(m^2)$ operaties vergt. Omdat $m \leq n$ is $\mathcal{O}(m^2) \subseteq \mathcal{O}(nm)$ waarmee de stelling volledig aangetoond is. □

We bekijken nu het geval $R = \mathbb{Z}$. Om de kost van Algoritme 1.6 te bepalen, hebben we essentieel gesteund op de kost van de vermenigvuldiging en deling met rest, en op Lemma 1.3.12. Van dit laatste kan de volgende versie geformuleerd worden voor $R = \mathbb{Z}$.

Lemma 1.3.14. $|s_i| \leq \frac{g}{r_{i-1}}$ *en* $|t_i| \leq \frac{f}{r_{i-1}}$, *voor* $1 \leq i \leq l+1$.

Zonder bewijs. □

Hiermee kan in principe de redenering om Stelling 1.3.13 aan te tonen, herhaald worden voor het geval $R = \mathbb{Z}$. We vermelden het resultaat zonder bewijs.

Stelling 1.3.15. *Om het algoritme van Euclides, Algoritme 1.6, uit te voeren voor twee gehele getallen f en g, met $\lambda(f) = n \geq \lambda(g) = m$, moeten er hoogstens $\mathcal{O}(nm)$ woordoperaties uitgevoerd worden.*

Zonder bewijs. □

1.3.3 Modulaire inversen

Het uitvoeren van optellingen en vermenigvuldigingen in een quotiëntring R/I is computationeel hetzelfde als het uitvoeren van deze bewerkingen in de originele ring R. Het verschil is dat het resultaat moet uitgedeeld worden ten opzichte van het ideaal I. Het berekenen van een inverse in de quotiëntring R/I is wiskundig gezien interessanter. De volgende stelling biedt een constructief antwoord op de vraag of een inverse bestaat en hoe ze te berekenen.

Stelling 1.3.16. *Stel dat R een Euclidisch domein is, $a, m \in R$ en $S = R/mR$. Dan is $a \pmod m \in S$ een eenheid als en slechts als* $\gcd(a, m) = 1$. *In dit geval kan $a^{-1} \pmod m$ berekend worden met behulp van de Bézout coëfficiënten en bijgevolg met behulp van het uitgebreid algoritme van Euclides.*

Bewijs. Stel dat $a \in S$ een inverse $b \in S$ heeft, wat gelijkwaardig is met

$$ba \equiv 1 \bmod m \iff \exists s, t \in R \quad sa + tm = 1 \implies \gcd(a, m) = 1.$$

Anderzijds, als $\gcd(a, m) = 1$, dan bestaan er twee elementen $s, t \in R$ waarvoor $sa + tm = 1$, waaruit dus volgt dat a een inverse heeft in S. Het uitgebreid algoritme van Euclides stelt ons in staat de elementen $s, t \in R$ te bepalen. □

Voorbeeld 1.3.17. Stel $R = \mathbb{Z}$, $m = 29$ en $a = 12$. Men kan nagaan dat $\gcd(a, m) = 1$ en dat $5 \cdot 29 + (-12) \cdot 12 = 1$, dus 17 is de inverse van 12 modulo 29.

1.3.4 Lineaire Diophantische vergelijkingen

Stel $a, f, g \in \mathbb{Z}$. Beschouw de vergelijking

$$sf + tg = a. \tag{1.7}$$

De verzameling van alle reële oplossingen van vergelijking (1.7) is een rechte in $\mathbb{R}^2$. Deze ééndimensionale oplossingenverzameling kan geschreven worden als $v + U$, waarbij v een oplossing is van vergelijking (1.7) en waarbij U de verzameling is van alle oplossingen van de vergelijking

$$sf + tg = 0. \tag{1.8}$$

We bewijzen een vergelijkbare eigenschap voor oplossingen over de gehele getallen. Het uitgebreid Euclidisch algoritme stelt ons daarenboven in staat de oplossingen te bepalen, indien ze bestaan.

Lemma 1.3.18. *Vergelijking* (1.7) *heeft een oplossing* $(s, t) \in R^2$ *als en slechts als* $\gcd(f, g) \mid a$.

Bewijs. Stel dat $s, t \in R$ een oplossing zijn van Vergelijking (1.7), dan $\gcd(f, g) \mid sf + tg$ en bijgevolg $\gcd(f, g) \mid a$. Omgekeerd, veronderstel dat $h = \gcd(f, g) \mid a$. Als $h = 0$ dan is de uitspraak triviaal. Stel dat $h \neq 0$, dan kunnen we $s^*, t^* \in R$ berekenen zodat $s^* f + t^* g = h$, met behulp van het uitgebreid Euclidisch algoritme, en

$$(s, t) = (\frac{s^* a}{h}, \frac{t^* a}{h})$$

is een oplossing van Vergelijking (1.7). □

Stelling 1.3.19. *Stel dat R een Euclidisch domein is, en stel $a, f, g \in R$ en $h = \gcd(g, f)$, met $h \mid a$.*

(i) *Als $h \neq 0$ en $(s^*, t^*) \in R^2$ een oplossing is van Vergelijking* (1.7)*, dan wordt de oplossingenverzameling gegeven door $(s^*, t^*) + U$, met*

$$U = R \cdot \left(\frac{g}{h}, -\frac{f}{h}\right) \subseteq R^2$$

de verzameling van alle oplossingen van de homogene vergelijking (1.8)*.*

(ii) *Stel $R = F[X]$, F een veld, $h \neq 0$. Als $\deg f + \deg g - \deg h > \deg a$, dan heeft Vergelijking* (1.7) *een unieke oplossing $(s, t) \in R^2$ waarvoor $\deg s < \deg g - \deg h$ en $\deg t < \deg f - \deg h$.*

Bewijs. (i) Stel dat $(s,t) \in R^2$. Omdat $h \neq 0$ en f/h en g/h copriem zijn, geldt

$$(1.8) \iff \frac{f}{h}s = -\frac{g}{h}t \iff \exists k \in R : \; s = k\frac{g}{h} \text{ en } t = k\frac{-f}{h} \iff (s,t) \in U.$$

Dan geldt ook dat

$$(1.7) \iff f \cdot (s - s^*) + g \cdot (t - t^*) = 0 \iff (s - s^*, t - t^*) \in U.$$

(ii) Stel dat $(s^*, t^*) \in R^2$ een oplossing is van Vergelijking (1.7). Deling van t^* door f/h levert

$$t^* = q\frac{f}{h} + t$$

en $\deg t < \deg f - \deg h$. Definiëren we

$$s = s^* + q\frac{g}{h},$$

dan impliceert (i) dat $(s,t) = (s^*, t^*) + q(g/h, -f/h)$ een oplossing is van Vergelijking (1.7). Nu zijn zowel $\deg(tg)$ als $\deg a$ kleiner dan $\deg f + \deg g - \deg h$, en bijgevolg is $\deg s + \deg f = \deg(sf) = \deg(a - tg)$. Dit bewijst het bestaan.

De uniciteit volgt uit $\deg(s + kg/h) = \deg(kg/h) \geq \deg g - \deg h$ voor alle $k \in R \setminus \{0\}$ en bijgevolg heeft de eerste component van elke oplossing van Vergelijking (1.7) verschillend van (s,t) graad ten minste $\deg g - \deg h$. □

1.4 De Chinese reststelling

De *Chinese reststelling* beschrijft een homomorfisme tussen bepaalde ringen en heeft vele en belangrijke toepassingen in de algebra. Vanuit algoritmisch standpunt is het homomorfisme eenvoudig realiseerbaar indien we beschikken over een algoritme om de grootste gemene deler te bepalen. Daarom wordt de Chinese reststelling hier als toepassing van het algoritme van Euclides behandeld.

Stel R een Euclidisch domein. Voor deze sectie definiëren we de elementen $m_i \in R$ als volgt.

$$\text{Stel } m_0, \ldots, m_{r-1} \in R \text{ zijn paarsgewijze copriem, i.e. } \gcd(m_i, m_j) = 1 \text{ voor } 0 \leq i < j < r, \text{ en stel } m = m_0 \cdot m_1 \cdots m_{r-1}, \quad (1.9)$$

m.a.w. $m = \text{lcm}(m_0, \ldots, m_{r-1})$. We noemen de afbeeldingen π_i de canonische ringhomomorfismen tussen R en $R/(m_i)$:

$$\begin{aligned} \pi_i : R &\to R/(m_i), \\ f &\mapsto f \bmod m_i. \end{aligned}$$

Met behulp van deze homomorfismen definiëren we een afbeelding tussen R en het cartesisch product van de ringen $R/(m_i)$:

$$\begin{aligned} \chi = \pi_0 \times \ldots \times \pi_{r-1} : R &\to R/(m_0) \times \ldots \times R/(m_{r-1}), \\ f &\mapsto (f \bmod m_0, \ldots, f \bmod m_{r-1}). \end{aligned}$$

Stelling 1.4.1. *De afbeelding χ is surjectief met kern (m).*

Bewijs. Stel $f \in R$. Dan geldt

$$\begin{aligned} f \in \ker \chi &\iff \chi(f) = (f \bmod m_0, \ldots, f \bmod m_{r-1}) = (0, \ldots, 0) \\ &\iff m_i \mid f,\ 0 \le i < r \iff \text{lcm}(m_0, \ldots, m_{r-1}) \mid f \iff m \mid f\,, \end{aligned}$$

dus $\ker \chi = (m)$. Definieer $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ (1 op de $i$de positie). We tonen aan dat er een $l_i \in R$ bestaat waarvoor $\chi(l_i) = e_i$, $0 \le i < r$. Stel $i = 0$. Aangezien $\gcd(m/m_0, m_0) = 1$, vinden we een $s, t \in R$, waarvoor $sm/m_0 + tm_0 = 1$. Stellen we $l_0 = sm/m_0$, dan geldt $l_0 \equiv 0 \bmod m_j$, $1 \le j < r$, en

$$l_0 = s\frac{m}{m_0} \equiv s\frac{m}{m_0} + tm_0 = 1 \bmod m_0\,,$$

dus $\chi(l_i) = e_i$ en deze redenering geldt voor alle i. We veronderstellen nu dat $v \in R/(m_0) \times \ldots \times R/(m_{r-1})$, dus $v = (v_0, v_1, \ldots, v_{r-1})$. Dan geldt

$$\begin{aligned} \chi(\sum v_i l_i) = \sum_{i=0}^{r-1} \chi(v_i)\chi(l_i) &= \sum_{i=0}^{r-1} (v_i \bmod m_0, \ldots, v_i \bmod m_{r-1})\chi(l_i) \\ &= \sum_{i=0}^{r-1} (0, \ldots, 0, v_i \bmod m_i, 0, \ldots, 0) = v\,, \end{aligned}$$

waaruit de stelling volgt. □

Op basis van deze stelling kunnen we nu de Chinese reststelling formuleren en bewijzen.

Stelling 1.4.2. *Er bestaat een isomorfisme tussen de volgende ringen:*

$$R/(m) \cong R/(m_0) \times \cdots \times R/(m_{r-1})\,,$$

en een groepsisomorfisme tussen de multiplicatieve groepen

$$(R/(m))^\times \cong (R/(m_0))^\times \times \cdots \times (R/(m_{r-1}))^\times\,.$$

Bewijs. Stelling 1.4.1 impliceert dat beide ringen isomorf zijn. Voor een willekeurig element $f \in R$ geldt: f is inverteerbaar modulo $m \iff \gcd(f, m) = 1 \iff \gcd(f, m_i) = 1$ voor $0 \leq i < r \iff f$ is inverteerbaar modulo m_i voor $0 \leq i < r$, waarmee het tweede gedeelte van de stelling aangetoond is. □

Op basis van het bewijs van Stelling 1.4.1 kan een algoritme opgesteld worden.

Algoritme 1.7 Algoritme voor Chinese reststelling

input: $m_0, m_1, \ldots, m_{r-1} \in R$, paarsgewijs co-priem,
met R een Euclidisch domein, $v_0, v_1, \ldots, v_{r-1} \in R$.
output: $f \in R$, zodat $f \equiv v_i \mod m_i$, $0 \leq i < r$.

1 $m \leftarrow m_0 \cdots m_{r-1}$
2 **for** $0 \leq i \leq r-1$
3 **do** bereken m/m_i
4 bepaal $s_i, t_i \in R$ zodat $s_i \dfrac{m}{m_i} + t_i m_i = 1$ (Algoritme 1.6)
5 $c_i \leftarrow \mathrm{rem}(v_i s_i, m_i)$
6 **return** $\displaystyle\sum_{i=0}^{r-1} c_i \frac{m}{m_i}$

We analyseren de complexiteit van het algoritme. We beschrijven eerst in detail de analyse voor het geval $R = F[X]$, F een veld.

Stelling 1.4.3. *Beschouw de ring $R = F[X]$, F een veld en veronderstel dat de elementen $m_0, \ldots, m_{r-1}, m \in R$ gedefinieerd zijn zoals in* (1.9)*, $d_i := \deg m_i \geq 1$ voor $0 \leq i < r, n = \deg m = \sum_{i=0}^{r-1} d_i$, en $v_i \in R$ met $\deg v_i < d_i$. Dan kan de unieke oplossing $f \in F[X]$ voor het stelsel*

$$f \equiv v_i \bmod m_i, 0 \leq i < r$$

bepaald worden door Algoritme 1.7 in $\mathcal{O}(n^2)$ operaties in F.

Bewijs. Het product $m_0 \cdots m_{r-1}$ kan bepaald worden door achtereenvolgens $m_0 m_1$, $m_0 m_1 m_2$, …, $m_0 m_1 \cdots m_{r-1}$ te bepalen. Dit kan in hoogstens

$$\sum_{i=0}^{r-2} C(d_{i+1} + \ldots + d_{r-1}) d_i \in \mathcal{O}(n^2)$$

operaties in F, met C een wel bepaalde constante (Stelling 1.2.1).

De berekening van m/m_i kan in hoogstens $C'd_i(n - d_i)$ operaties in F, met C' een wel bepaalde constante (Stelling 1.2.2). Lijn 3 wordt r maal uitgevoerd, dit levert dus $\sum_{i=0}^{r-1} C'd_i(n - d_i) \in \mathcal{O}(n^2)$ operaties in F. Elke uitvoering van Lijn 4 kost $\mathcal{O}(d_i(n - d_i))$ operaties (Stelling 1.3.13). Lemma 1.3.12 (1.3) garandeert dat $\deg s_i < \deg m_i = d_i$. Bijgevolg zijn er $\mathcal{O}(d_i^2)$ operaties nodig om v_i met s_i te vermenigvuldigen en het resultaat te delen door m_i. Elke stap van de lus vergt dus $\mathcal{O}(d_i n)$ operaties. De totale kost van de lus is $\mathcal{O}(n^2)$ operaties.

Tenslotte zijn er $\mathcal{O}(d_i(n-d_i))$ operaties nodig om het product van m/m_i en c_i te berekenen, en $\mathcal{O}(nr)$ om alle producten op te tellen. Dit geeft ook voor Lijn 6 een totale kost van $\mathcal{O}(n^2)$ operaties. Hieruit mogen we besluiten dat het volledige algoritme in $\mathcal{O}(n^2)$ operaties kan uitgevoerd worden. □

Het geval $R = \mathbb{Z}$ is analoog aan het geval $R = F[X]$. Zonder in te gaan op de details, vermelden we de volgende stelling zonder bewijs.

Stelling 1.4.4. *Beschouw de ring $R = \mathbb{Z}$ en veronderstel dat de elementen $m_0, \ldots, m_{r-1}, m \in \mathbb{N}$ gedefinieerd zijn zoals in (1.9), $n := \lambda(m)$ en $v_i \in \mathbb{N}$ met $0 \leq v_i < m_i$. Dan kan de unieke oplossing $f \in \mathbb{Z}$, $0 \leq f < m$ voor het stelsel*

$$f \equiv v_i \bmod m_i, 0 \leq i < r$$

bepaald worden door Algoritme 1.7 in $\mathcal{O}(n^2)$ woordoperaties.

Zonder bewijs. □

Hoofdstuk 2

Snellere algoritmen voor vermenigvuldiging

In dit hoofdstuk beschrijven we een aantal snelle(re) algoritmen om de vermenigvuldiging uit te voeren, zowel voor gehele getallen als voor polynomen over bepaalde ringen. We zullen zien dat deze snelle(re) algoritmen beter presteren dan de klassieke algoritmen, maar dat betekent niet dat de klassieke algoritmen daarmee overbodig worden. Dit hoofdstuk bevat ook een kleine sectie over deling met rest, waarin we een algoritme om deling met rest uit te voeren beschrijven dat een complexiteit heeft van dezelfde orde als vermenigvuldiging. Daarmee kunnen we het gedeelte over de basisbewerkingen in deze cursus afsluiten.

2.1 Vermenigvuldiging

2.1.1 Het algoritme van Karatsuba

Het algoritme van Karatsuba is een algoritme dat ons in staat zal stellen om twee polynomen te vermenigvuldigen in $\mathcal{O}(n^{\log 3})$ tijd. Aangezien $\log 3 = \ln 3/\ln 2 < 1.59$, is dit een aanzienlijke verbetering in vergelijking met het klassieke algoritme. De kern van Karatsuba's algoritme [KO63] is de volgende elementaire vaststelling.

Wanneer we de polynomen $ax+b$ en $cx+d$ willen vermenigvuldigen, doen we dit op de klassieke wijze als

$$(ax + b)(cx + d) = (ac)x^2 + (ad + bc)x + bd\,.$$

Hiervoor hebben we 4 vermenigvuldigingen en 1 optelling nodig. Verrassend genoeg kunnen we op eenvoudige wijze het aantal vermenigvuldigingen reduceren, waarbij we weliswaar het aantal optellingen zullen moeten opdrijven. Inderdaad, we herschrijven $ad + bc$ als

$$ad + bc = (a + b)(c + d) - ac - bd\,.$$

Aangezien we toch reeds ac en bd moeten berekenen als coëfficiënten van x^2 en x^0, hebben we slechts één nieuwe vermenigvuldiging nodig, met name $(a + b)(c + d)$. We hebben nu wel 4 optellingen nodig in plaats van 1.

Het aantal basisbewerkingen voor de berekening van $(ax+b)(cx+d)$ is hierbij verhoogd van 5 naar 7, maar het voordeel van deze methode blijkt als we dit principe recursief toepassen, aangezien optellen van *polynomen* een "goedkopere" operatie is dan het vermenigvuldigen van polynomen.

Algoritme 2.1 Het algoritme van Karatsuba voor polynomen

input: polynomen $f, g \in R[x]$ van graad kleiner dan $n = 2^\ell,\ \ell \in \mathbb{N}$
output: het polynoom $f \cdot g$
1 **if** $n = 1$
2 **then return** $f \cdot g \in R$
3 Stel $f = F_1 x^{n/2} + F_0$, $g = G_1 x^{n/2} + G_0$,
met $F_0, F_1, G_0, G_1 \in R[x]$ van graad kleiner dan $n/2$
4 $f \leftarrow F_1 G_1$ (recursief)
5 $g \leftarrow F_0 G_0$ (recursief)
6 $h \leftarrow (F_0 + F_1)(G_0 + G_1)$ (recursief)
7 **return** $f \cdot x^n + (h - f - g) \cdot x^{n/2} + g$

Om de complexiteit van Karatsuba's algoritme te berekenen, bewijzen we eerst een nogal technisch lemma, dat nuttig blijkt te zijn voor het berekenen van de complexiteit van recursieve algoritmen. We noteren de vereiste tijd voor uitvoering van het algoritme bij een input van grootte n als $T(n)$, en veronderstellen dat die uitvoering bestaat uit b recursieve calls van het algoritme voor input grootte $n/2$, plus een zekere tijd $S(n)$, die superlineair is.

Definitie 2.1.1. Een functie $S\colon \mathbb{N} \to \mathbb{R}^+$ wordt *superlineair* genoemd, als $S(n) \geq n$ voor alle $n \in \mathbb{N}$, en bovendien $S(m+n) \geq S(m) + S(n)$ voor alle $m, n \in \mathbb{N}$. In het bijzonder is $S(2n) \geq 2S(n)$ voor alle $n \in \mathbb{N}$.

Lemma 2.1.2. *Stel $b \in \mathbb{N}$, en veronderstel dat S en T functies zijn van $\mathbb{N}$ naar $\mathbb{R}^+$, waarbij S superlineair is, en zodat*

$$T(n) \leq bT(n/2) + S(n) \text{ voor alle } n = 2^i \text{ met } 1 \leq i \in \mathbb{N}\,.$$

Dan geldt, voor elke $n = 2^i$ met $1 \leq i \in \mathbb{N}$:

$$T(n) \leq \begin{cases} 2S(n)(1 - n^{-1}) + T(1) \in \mathcal{O}(S(n)) & \text{als } b = 1\,; \\ S(n)\log n + T(1) \cdot n \in \mathcal{O}(S(n)\log n) & \text{als } b = 2\,; \\ \frac{2}{b-2}\left(n^{\log b - 1} - 1\right) \cdot S(n) + T(1) \cdot n^{\log b} \in \mathcal{O}(S(n) n^{\log b - 1}) & \text{als } b > 2\,, \end{cases}$$

Bewijs. Door middel van inductie op i zien we dat

$$T(2^i) \leq b^i \cdot T(1) + \sum_{0 \leq j < i} b^j S(2^{i-j}) \tag{2.1}$$

voor alle $1 \leq i \in \mathbb{N}$. Aangezien $b^i = 2^{i \log b}$ en $S(2^{i-j}) \leq 2^{-j} S(2^i)$ voor alle $i, j \in \mathbb{N}$, verkrijgen we uit vergelijking (2.1) dat

$$T(2^i) \leq T(1) \cdot 2^{i \log b} + S(2^i) \cdot \underbrace{\sum_{0 \leq j < i} (b/2)^j}_{s} .$$

Als $b = 2$, dan hebben we $s = i$; als $b \neq 2$, dan is

$$s = \frac{(b/2)^i - 1}{(b/2) - 1} = \frac{2}{b-2}(2^{i(\log b - 1)} - 1) .$$

Door $n = 2^i$ te stellen bekomen we

$$T(n) \leq \begin{cases} 2S(n)(1 - n^{-1}) + T(1) & \text{als } b = 1 ; \\ S(n) \log n + T(1) \cdot n & \text{als } b = 2 ; \\ \frac{2}{b-2}\left(n^{\log b - 1} - 1\right) \cdot S(n) + T(1) \cdot n^{\log b} & \text{als } b > 2 , \end{cases}$$

en door rekening te houden met $n \leq S(n)$ bekomen we de de gezochte complexiteit. □

Opmerking 2.1.3. Indien S lineair is in n, dan vereenvoudigen voorgaande formules tot

$$T(n) \in \begin{cases} \mathcal{O}(n) & \text{als } b = 1 ; \\ \mathcal{O}(n \log n) & \text{als } b = 2 ; \\ \mathcal{O}(n^{\log b}) & \text{als } b > 2 . \end{cases}$$

Stelling 2.1.4. *Het vermenigvuldigen van twee polynomen van graad $\leq n$ met behulp van Karatsuba's Algoritme* 2.1 *heeft complexiteit $\mathcal{O}(n^c)$, waarbij $c = \ln 3 / \ln 2 \approx 1.5849625$.*

Bewijs. Het berekenen van de polynomen $F_0 + F_1$ en $G_0 + G_1$ in regel 6 van het algoritme vereist 2 keer $n/2$ optellingen. Het berekenen van $f \cdot x^n + g$ in regel 7 vereist geen bewerkingen, aangezien de machten van x die voorkomen bij deze twee termen disjunct zijn aan elkaar. De uitwerking van $h - f - g$ in regel 7 heeft 2 keer n bewerkingen nodig, en tenslotte zijn er nog n optellingen nodig om $(h - f - g) \cdot x^{n/2}$ op te tellen bij $f \cdot x^n + g$. Daarnaast worden er in regels 4–6 samen 3 recursieve calls opgeroepen met input grootte $n/2$.

We kunnen dus Opmerking 2.1.3 toepassen met $b = 3$ en $S(n) = 4n$, en we bekomen

$$T(n) \in \mathcal{O}(n^{\log 3}) . \qquad \square$$

Opmerking 2.1.5. Als n geen macht van 2 is, kunnen we het algoritme toepassen voor de kleinste macht van 2 die groter is dan n, i.e. voor $2^{\lceil \log n \rceil}$. Echter, indien n slechts een beetje groter is dan een macht van 2, gebruikt dit algoritme ongeveer een factor 3 maal te veel tijd.

Het is efficiënter om de polynomen telkens op te splitsen in twee blokken die elk ongeveer de helft van de graad van het startpolynoom zijn.

Opmerking 2.1.6. Het algoritme van Karatsuba kan zonder essentiële aanpassingen ook worden toegepast om twee gehele getallen te vermenigvuldigen. Uiteraard zal ook hier gelden dat het vermenigvuldigen van twee getallen van lengte n met dit algoritme $\mathcal{O}(n^{\log 3})$ woordoperaties vereist.

Opmerking 2.1.7. Er bestaat een sterke veralgemening van het algoritme van Karatsuba, het zogenaamde Toom–Cook algoritme, voor het eerst beschreven door Andrei Toom [Too63] en nadien verbeterd door Stephen Cook [Coo66]. Hierbij wordt het polynoom (of getal) in k gelijke delen verdeeld; dit algoritme wordt vooral toegepast met $k = 3$ (en wordt dan ook het Toom-3 algoritme genoemd), en heeft in dat geval complexiteit $\mathcal{O}(n^c)$ met $c = \ln 5 / \ln 3 \approx 1.4649735$. Voor $k = 2$ vinden we precies het algoritme van Karatsuba terug, terwijl we voor $k = 1$ gewoon het klassieke algoritme verkrijgen.

2.1.2 Discrete Fourier transformatie (DFT) en Fast Fourier transformatie (FFT)

In deze paragraaf zullen we een algoritme beschrijven om twee polynomen te vermenigvuldigen in quasi-lineaire tijd. We zullen moeten aannemen dat de coëfficiëntenring R bepaalde eenheidswortels bevat.

Definitie 2.1.8. Zij R een commutatieve ring (met $1 \in R$), $1 \leq n \in \mathbb{N}$, en $\omega \in R$.

(i) $a \in R$ is een *nuldeler* als er een $b \in R^* := R \setminus \{0\}$ bestaat zodat $ab = 0$.

(ii) $a \in R$ is een *eenheid* als er een $b \in R$ bestaat zodat $ab = 1$.

(iii) ω is een n^{de} *eenheidswortel* als $\omega^n = 1$.

(iv) ω is een *primitieve* n^{de} *eenheidswortel* als $\omega^n = 1$, $n \cdot 1$ is een eenheid in R, en $\omega^{n/p} - 1$ is geen nuldeler, voor elke priemdeler p van n.

Het volgend lemma veralgemeent de definiërende eigenschap in Definitie 2.1.8(iii).

Lemma 2.1.9. *Zij R een ring, $\ell, n \in \mathbb{N}$ met $1 \leq \ell < n$, en $\omega \in R$ een primitieve n^{de} eenheidswortel. Dan geldt:*

(i) *$\omega^\ell - 1$ is geen nuldeler in R;*

(ii) *$\sum_{j=0}^{n-1} \omega^{\ell j} = 0$.*

Bewijs. We zullen meerdere malen gebruik maken van de identiteit

$$(c-1)\sum_{j=0}^{m-1} c^j = c^m - 1 \tag{2.2}$$

voor alle $m \in \mathbb{N}$ en alle $c \in R$.

(i) Stel $g := \gcd(\ell, n)$ en $s, t \in \mathbb{Z}$ zodanig dat $s\ell + tn = g$. Aangezien $1 \leq g < n$ bestaat er een priemdeler p van n/g. Passen we nu (2.2) toe met $c = \omega^g$ en $m = n/pg$, dan bekomen we $a \cdot (\omega^g - 1) = \omega^{n/p} - 1$ voor een zekere $a \in R$. Aangezien $\omega^{n/p} - 1$ geen nuldeler is in R, volgt hieruit dat ook $\omega^g - 1$ geen nuldeler is.
We passen nu nogmaals vergelijking (2.2) toe, met $c = \omega^\ell$ en $m = s$; we bekomen dat $b \cdot (\omega^\ell - 1) = \omega^{s\ell} - 1$ voor een zekere $b \in R$. Merk op dat $\omega^{s\ell} - 1 = \omega^{s\ell}\omega^{tn} - 1 = \omega^g - 1$. Aangezien $\omega^g - 1$ geen nuldeler is, is ook $\omega^\ell - 1$ geen nuldeler, wat we wilden bewijzen.

(ii) We passen opnieuw (2.2) toe, met $c = \omega^\ell$ en $m = n$, en we krijgen

$$(\omega^\ell - 1) \cdot \sum_{j=0}^{n-1} \omega^{\ell j} = \omega^{ln} - 1 = 0\,,$$

en aangezien $\omega^\ell - 1$ geen nuldeler is wegens (i) volgt het gestelde. □

Opmerking 2.1.10. Als R een veld is, dan is ω een primitieve n^{de} eenheidswortel a.s.a. $\omega^n = 1$, $\omega^{n/p} \neq 1$ voor elke priemdeler p van n, en n is geen veelvoud van $\text{char}(R)$.

Lemma 2.1.11. *Zij $q = p^h$ een priemmacht. Het eindig veld $\mathsf{GF}(q)$ heeft een primitieve n^{de} eenheidswortel a.s.a. n een deler is van $q - 1$.*

Bewijs. Dit volgt onmiddellijk uit Opmerking 2.1.10 en het feit dat de multiplicatieve groep van $\mathsf{GF}(q)$ een cyclische groep is van de orde $q-1$. (Merk ook op dat $n \mid q - 1$ impliceert dat n geen veelvoud is van $p = \text{char}(\mathsf{GF}(q))$.) □

Zij nu R een ring, $1 \leq n \in \mathbb{N}$, en $\omega \in R$ een primitieve n^{de} eenheidswortel. In de volgende definitie zullen we een polynoom $f = \sum_{i=0}^{n-1} f_i x^i \in R[x]$ van graad $< n$ identificeren met zijn coëfficiëntenvector $(f_0, \ldots, f_{n-1}) \in R^n$.

Definitie 2.1.12. (i) De R-lineaire afbeelding

$$\mathrm{DFT}_\omega : R^n \to R^n : f \mapsto \big(f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})\big)$$

wordt de *discrete Fourier transformatie (DFT)* genoemd.

(ii) De *(cyclische) convolutie* van twee vectoren $a = (a_0, \ldots, a_{n-1})$ en $b = (b_0, \ldots, b_{n-1})$ in R^n is de vector $c = a *_n b = (c_0, \ldots, c_{n-1})$ gegeven door

$$c_k = \sum_{i+j \equiv k \pmod n} a_i b_j = \sum_{i=0}^{n-1} a_i b_{k-i}\,,$$

waarbij de indices modulo n beschouwd moeten worden.

(iii) De *(cyclische) convolutie* van twee polynomen $f = \sum_{i=0}^{n-1} f_i x^i$ en $g = \sum_{i=0}^{n-1} g_i x^i$ in $R[x]$ van graad $< n$ is het polynoom $f *_n g$ bekomen door hun coëfficiëntenvectoren cyclisch te convoluteren.

Merk op dat de convolutie van twee polynomen in feite wordt bekomen door de polynomen te vermenigvuldigen in de ring $R[x]/(x^n - 1)$, aangezien $f *_n g \equiv fg \pmod{x^n - 1}$. Indien de waarde van n duidelijk is uit de context, zullen we eenvoudigweg $f * g$ schrijven in plaats van $f *_n g$.

Lemma 2.1.13. *Zij $f, g \in R[x]$ twee polynomen van graad $< n$. Dan geldt $\mathrm{DFT}_\omega(f * g) = \mathrm{DFT}_\omega(f) \bullet \mathrm{DFT}_\omega(g)$, waarbij $\bullet$ de puntsgewijze vermenigvuldiging van vectoren is.*

Bewijs. We hebben $f * g = fg + q \cdot (x^n - 1)$ voor een zekere $q \in R[x]$, en dus

$$(f * g)(\omega^i) = f(\omega^i)g(\omega^i) + q(\omega^i)(\omega^{in} - 1) = f(\omega^i)g(\omega^i)$$

voor alle $i \in \{0, \ldots, n-1\}$. □

Stelling 2.1.14. *De afbeelding $\mathrm{DFT}_\omega : R^n \to R^n$ is een isomorfisme; meer bepaald hebben we $\mathrm{DFT}_\omega^{-1} = n^{-1} \cdot \mathrm{DFT}_{\omega^{-1}}$.*

Bewijs. Uit de definitie van DFT_ω volgt dat

$$\mathrm{DFT}_\omega(f_0, \ldots, f_{n-1}) = (f_0, \ldots, f_{n-1}) \cdot V_\omega\,,$$

waarbij

$$V_\omega := \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{pmatrix}$$

een Vandermonde matrix is. We berekenen nu de matrix $W = V_\omega \cdot V_{\omega^{-1}}$, en we bekomen $W_{ij} = \sum_{k=0}^{n-1}(\omega^{i-j})^k$ voor alle $i, j \in \{0, \ldots, n-1\}$. Als $i = j$, dan bekomen we $W_{ii} = n$; als $i \neq j$, dan volgt uit Lemma 2.1.9(ii) dat $W_{ij} = 0$. Bijgevolg is $W = n \cdot I_n$, en hieruit volgt het gestelde. □

We bespreken nu eerst een belangrijk algoritme om de DFT snel te berekenen, met name de Fast Fourier transformatie (FFT). Dit algoritme werd (her)ontdekt door Cooley en Tukey [CT65] in 1965; in feite was dit algoritme al gekend door Gauß in 1805, maar werd pas na zijn dood gepubliceerd in 1866. (Het is ook Gauß die de gewone Fourier transformatie ontdekte, voor Fourier dat deed in 1807.) Het FFT algoritme is in de loop van de 19e en 20e eeuw meerdere malen herontdekt, maar het is pas toen Cooley en Tukey de aandacht erop vestigden en een expliciete computerimplementatie voorstelden, dat dit algoritme een ware revolutie ontketende.

Algoritme 2.2 Fast Fourier transformatie (FFT)

input: $n = 2^k$ met $k \in \mathbb{N}$; $f = \sum_{i=0}^{n-1} f_i x^i$; de machten ω, ω^2, …, ω^{n-1} van een primitieve n^{de} eenheidswortel $\omega \in R$.
output: $\text{DFT}_\omega(f) \in R^n$.
1 **if** $n = 1$ **then return** f_0
2 $r \leftarrow \sum_{j=0}^{n/2-1}(f_j + f_{j+n/2})x^j$, $s \leftarrow \sum_{j=0}^{n/2-1}(f_j - f_{j+n/2})\omega^j x^j$
3 Evalueer **recursief** de polynomen r en s in de machten van ω^2
4 **return** $\big(r(1), s(1), r(\omega^2), s(\omega^2), \ldots, r(\omega^{n-2}), s(\omega^{n-2})\big)$

Stelling 2.1.15. *Zij $n = 2^k$ met $k \in \mathbb{N}$, en $\omega \in R$ een primitieve n^{de} eenheidswortel in R, $f \in R[x]$, $\deg f < n$. Algoritme 2.2 berekent $\text{DFT}_\omega(f)$ correct, en gebruikt daartoe hoogstens $n \log n$ optellingen en hoogstens $\frac{n}{2}\log n$ vermenigvuldigingen met machten van ω.*

Bewijs. We bewijzen de correctheid door middel van inductie op k. Als $k = 0$, dan is $f = f_0$ een constante veelterm, en dan is $\text{DFT}_\omega(f) = f_0$. Stel dus $k \geq 1$. We zullen aantonen dat $r(\omega^{2i}) = f(\omega^{2i})$ en $s(\omega^{2i}) = f(\omega^{2i+1})$ voor alle $i \in \{0, \ldots, n/2-1\}$; de correctheid zal dan volgen uit de inductiehypothese.

Merk vooraf op dat $\omega^{n/2} = -1$. Inderdaad,

$$0 = \omega^n - 1 = (\omega^{n/2} + 1)(\omega^{n/2} - 1)\,;$$

aangezien $\omega^{n/2}-1$ geen nuldeler kan zijn, moet wel $\omega^{n/2}+1=0$. We hebben nu

$$\begin{aligned} r(\omega^{2i}) &= \sum_{j=0}^{n/2-1} (f_j + f_{j+n/2})\,\omega^{2ij} \\ &= \sum_{j=0}^{n/2-1} f_j\,\omega^{2ij} + \sum_{j=0}^{n/2-1} f_{j+n/2}\,\omega^{2ij+in} \\ &= \sum_{j=0}^{n-1} f_j\,\omega^{2ij} = f(\omega^{2i}) \end{aligned}$$

en

$$\begin{aligned} s(\omega^{2i}) &= \sum_{j=0}^{n/2-1} (f_j - f_{j+n/2})\,\omega^{j}\omega^{2ij} \\ &= \sum_{j=0}^{n/2-1} f_j\,\omega^{(2i+1)j} + \sum_{j=0}^{n/2-1} f_{j+n/2}\,\omega^{(2i+1)j+in+n/2} \\ &= \sum_{j=0}^{n-1} f_j\,\omega^{(2i+1)j} = f(\omega^{2i+1})\,. \end{aligned}$$

We analyseren vervolgens de complexiteit van dit algoritme. We noteren het aantal optellingen, respectievelijk vermenigvuldigingen, bij invoergrootte n als $A(n)$, respectievelijk $B(n)$. Als $n=1$, dan is duidelijk $A(1)=B(1)=0$. De uitvoering van lijn (2) vergt n optellingen (voor r en s samen) en $\frac{n}{2}$ vermenigvuldigingen (voor s). De recursieve oproep vergt $2A(n/2)$ optellingen en $2B(n/2)$ vermenigvuldigingen. Uit Lemma 2.1.2 en het feit dat er twee recursieve oproepen zijn, volgt onmiddellijk

$$A(n) \leq n\log n\,, \quad B(n) \leq \frac{1}{2} n\log n\,.$$

□

Uit Stelling 2.1.15 volgt dat er $\mathcal{O}(n\log n)$ operaties in R vereist zijn, hetgeen geformuleerd is in het onderstaande gevolg. We kunnen deze afschatting iets sneller maken, maar met het oog op verdere toepassingen hebben we in bovenstaande stelling een afschatting gemaakt van het aantal optellingen en vermenigvuldigingen afzonderlijk. In het onderstaande gevolg geven we de korte versie.

Gevolg 2.1.16. *Onder de voorwaarden van Stelling 2.1.15 berekent Algoritme 2.2 $DFT_\omega(f)$ in $\mathcal{O}(n \log n)$ operties in R.*

Bewijs. Het is duidelijk dat in lijnen (2) en (3) een lineair aantal bewerkingen vereist is (namelijk n optellingen en $n/2$ vermenigvuldigingen); nadien roept het algoritme zichzelf 2 maal recursief op met halve input grootte, met name voor de berekening van de polynomen r en s. We kunnen dus Opmerking 2.1.3 toepassen met $b = 2$, en we bekomen de complexiteit $\mathcal{O}(n \log n)$. □

Het eenvoudige Lemma 2.1.13 zal ons nu in staat stellen om op zeer efficiënte wijze het convolutieproduct van twee polynomen te berekenen.

Algoritme 2.3 Fast convolution

input: $n = 2^k$ met $k \in \mathbb{N}$; $f, g \in R[x]$ van graad $< n$,
$\omega \in R$ een primitieve n^{de} eenheidswortel.
output: $f * g \in R[x]$.
1 Bereken $\omega^2, \ldots, \omega^{n-1}$
2 $\alpha \leftarrow \mathrm{DFT}_\omega(f)$
3 $\beta \leftarrow \mathrm{DFT}_\omega(g)$
4 $\gamma \leftarrow \alpha \bullet \beta$
5 **return** $\mathrm{DFT}_\omega^{-1}(\gamma) = n^{-1}\,\mathrm{DFT}_{\omega^{-1}}(\gamma)$

Definitie 2.1.17. We zeggen dat een commutatieve ring R (met eenheid) *FFT ondersteunt* als R voor elke $k \in \mathbb{N}$ een primitieve $(2^k)^{\text{de}}$ eenheidswortel heeft.

Een voorbeeld van een ring die FFT ondersteunt is $\mathbb{C}$; een eindig veld daarentegen ondersteunt nooit FFT.

Stelling 2.1.18. *Zij $n = 2^k$ met $k \in \mathbb{N}$, en $\omega \in R$ een primitieve n^{de} eenheidswortel in R, $f, g \in R[x]$, $\deg f, \deg g < n$. Algoritme 2.3 berekent $f * g$ correct, en gebruikt daartoe $3n \log n$ optellingen, $\frac{3}{2} n \log n + n$ vermenigvuldigingen met machten van ω, n vermenigvuldigingen in R, n delingen door n in R, dus in totaal in $\frac{9}{2} n \log n + \mathcal{O}(n)$ aritmetische operaties in R*

Bewijs. Uit Lemma 2.1.13 volgt dat Algoritme 2.3 correct de convolutie van f en g berekent. De kost voor het algoritme per regel is als volgt.

1. $n - 2$ vermenigvuldigingen met ω;

2–3. $2n \log n$ optellingen en $n \log n$ vermenigvuldigingen met machten van ω.;

4. n vermenigvuldigingen in R;

5. $n \log n$ optellingen, $\frac{1}{2} n \log n$ vermenigvuldigingen met machten van ω, en n delingen door n in R. □

Gevolg 2.1.19. *Zij R een ring die FFT ondersteunt, en zij $n = 2^k$ met $k \in \mathbb{N}$. De vermenigvuldiging van twee polynomen $f, g \in R[x]$, $\deg f, \deg g < n$ met behulp van Algoritme 2.3 vergt $\mathcal{O}(n \log n)$ operaties in R.*

Bewijs. Aangezien $\deg(fg) < 2n$ volgt dit onmiddellijk uit Stelling 2.1.18 met n vervangen door $2n$. □

Ook hier kan men met een kortere afschatting aantonen dat Algoritme 2.3, onder de voorwaarden van Gevolg 2.1.18, $\mathcal{O}(n \log n)$ aritmetische operaties vergt in R, zonder dat we de gedetailleerde afschatting hoeven te maken in het bewijs van de stelling. Maar ook hier hebben we de details gegeven met het oog op latere toepassing.

2.1.3 Het algoritme van Schönhage en Strassen

Het feit dat een ring FFT moet ondersteunen, is een zware beperking. In het bijzonder kan Algoritme 2.3 niet zonder meer "vertaald" worden naar een algoritme om gehele getallen te vermenigvuldigen. Toch ligt het principe van DFT en de daarmee samenhangende fast convolution aan de basis van snelle algoritmen om zowel polynomen over ringen die geen FFT ondersteunen, als gehele getallen, te vermenigvuldigen. Schönhage en Strassen hebben in 1971 een algoritme ontwikkeld om twee gehele getallen met n cijfers te vermenigvuldigen in $\mathcal{O}(n \log n \log \log n)$ woordoperaties. Hoewel het oorspronkelijke artikel een algoritme beschrijft om gehele getallen te vermenigvuldigen, starten hier met de afgeleide versie voor polynomen over een ring die aan bepaalde voorwaarden voldoet, omdat dit rechtstreeks aansluit bij de vorige sectie. Aan de basis ligt het idee om de ring R uit te breiden met de nodige algebraïsche elementen, en zodoende het FFT principe opnieuw toepasbaar te maken.

Stel dat R een domein is waarin 2 een eenheid is. Voor een willekeurige $k \in \mathbb{N}$, stel $n = 2^k$, en definieer $D := R[x]/(x^n + 1)$. Noem $\omega = x \mod x^n + 1 \in D$. Dan geldt in D dat $\omega^n = -1$, dat $\omega^{2n} = 1$, en dat $\omega^n - 1 = -2$, door de veronderstelling een eenheid in R. In de ring D is ω dus een primitieve $2n^{\text{de}}$ eenheidswortel.

Beschouw nu twee polynomen $f, g \in R[x]$, met $\deg f, \deg g < n = 2^k$, voor een zekere $k \in \mathbb{N}$. Indien ook $\deg(fg) < n$ dan kennen we fg als we $fg \mod x^n + 1$ kennen. Dit laatste wordt de *negative wrapped convolution* van f en g genoemd. De essentie van het algoritme van Schönhage en Strassen is het opstellen van een efficiënt algoritme om deze negative wrapped convolution te berekenen, en dit algoritme is FFT gebaseerd.

Met $m = 2^{\lfloor k/2 \rfloor}$, $t = n/m = 2^{\lceil k/2 \rceil}$ "verdelen" we de coëfficiënten van f en g in t blokken van grootte m:

$$f = \sum_{0 \le j < t} f_j x^{mj}\,, \qquad g = \sum_{0 \le j < t} g_j x^{mj}\,,$$

met $f_j, g_j \in R[x]$, $\deg f_j, g_j < m$ $0 \le j < t$. Definiëren we

$$f' := \sum_{0 \le j < t} f_j y^j\,, \qquad g' := \sum_{0 \le j < t} g_j y^j\,, \tag{2.3}$$

dan zijn $f', g' \in R[x, y]$ en $f = f'(x, x^m)$ en $g = g'(x, x^m)$. Nu berekenen we $f'g' \mod y^t + 1$, stel $f'g' \equiv h' \mod y^t + 1$, of, m.a.w.

$$f'g' = h' + q'(y^t + 1) \tag{2.4}$$

voor een zekere $h', q' \in R[x, y]$. Dit impliceert

$$fg = h'(x, x^m) + q'(x, x^m)(x^{tm} + 1) \equiv h'(x, x^m) \mod (x^n + 1)\,. \tag{2.5}$$

Beschouw nu de $4m^{\text{de}}$ eenheidswortel

$$\xi = x \mod (x^{2m} + 1) \in D = R[x]/(x^{2m} + 1) \tag{2.6}$$

Met het klassieke algoritme voor deling met rest kan $h' \in R[x][y]$, $\deg_y h' < t$ berekend worden en deze h' is uniek. Vergelijken we de coëfficiënten van y^j met $j \ge t$, dan zien we dat $\deg_x q' \le \deg_x(f'g') < 2m$ en we besluiten dat

$$\deg_x h' \le \max\{\deg_x(f'g'), \deg_x q'\} < 2m \tag{2.7}$$

Definiëren we $f^* := f' \mod (x^{2m} + 1)$, $g^* := g' \mod (x^{2m} + 1)$, en $h^* := h' \mod (x^{2m} + 1)$ in $D[y]$, dan impliceert (2.4)

$$f^* g^* \equiv h^* \mod (y^t + 1) \in D[y]\,. \tag{2.8}$$

Merk op dat de definitie van f^*, g^* en h^* een reductie modulo $x^{2m}+1$ inhoudt, maar aangezien deze drie polynomen graad in x kleiner dan $2m$ hebben (door

(2.7)), vergt dit geen enkele berekening. Wiskundig gezien houdt deze reductie enkel een andere algebraïsche interpretatie in van de coëfficiëntenlijst. Meer bepaald kunnen bijvoorbeeld de coëfficiënten van $h' \in R[x][y]$ afgelezen worden uit de coëfficiënten van $h^* \in D[y]$.

We illustreren dit met een klein voorbeeld. Stel $h = 4x^3 + 3x^2 + 2x + 1 \in \mathbb{Q}[x]$, $m = 2$, en $\xi = x \mod (x^4 + 1) \in D = \mathbb{Q}[x]/(x^4 + 1)$. Dan is $h' = (4x + 3)y + (2x + 1) \in \mathbb{Q}[x, y]$ en $h^* = (4\xi + 3)y + (2\xi + 1) \in D[y]$.

Er is geen berekening vereist om h^* uit h' te bepalen, maar opeens belanden we in een situatie waarin we FFT kunnen gebruiken om (2.8) te berekenen. Omdat $t \in \{m, 2m\}$, bevat D een primitieve $2t^{\text{de}}$ eenheidswortel η, namelijk $\eta = \xi$ als $t = 2m$ en $\eta = \xi^2$ als $t = m$. Uit (2.8) halen we dat

$$f^*(\eta y)g^*(\eta y) \equiv h^*(\eta y) \mod \left((\eta y)^t + 1\right),$$

of nog

$$f^*(\eta y)g^*(\eta y) \equiv h^*(\eta y) \mod \left(y^t - 1\right), \tag{2.9}$$

doordat $\eta^t = -1$. De polynomen $f^*(\eta y), g^*(\eta y) \in D[y]$, en met Algoritme 2.3 kunnen we $h^*(\eta y)$ berekenen in $\mathcal{O}(t \log t)$ operaties in D. Merk op dat we hiervoor een primitieve t^{de} eenheidswortel nodig hebben in D, maar daarvoor kiezen we $\omega = \eta^2$. Een vermenigvuldiging van twee elementen in D komt opnieuw neer op een negative wrapped convolution over R, dewelke nu recursief afgehandeld kan worden. Dit leidt ons tot het volgende algoritme.

Algoritme 2.4 Fast negative wrapped convolution

input: $k \in \mathbb{N}$, $f, g \in R[x]$, $\deg f, \deg g < n = 2^k$, $k \in \mathbb{N}$,
R een domein, 2 een eenheid in R.
output: $h \in R[x]$, met $fg \equiv h \mod (x^n + 1)$ en $\deg h < n$

1 **if** $k \leq 2$
then Bereken fg (met klassiek algoritme (of met Karatsuba))
2 **return** $fg \mod x^n + 1$
3 $m \leftarrow 2^{\lfloor k/2 \rfloor}$, $t \leftarrow n/m$
4 initialiseer $f', g' \in R[x, y]$, met $\deg_x f', \deg_x g' < m$
en $f = f'(x, x^m)$ en $g = g'(x, x^m)$.
5 initialiseer $D = R[x]/\langle x^{2m} + 1 \rangle$
6 **if** $t = 2m$
7 **then** $\eta \leftarrow x \mod (x^{2m} + 1)$
8 **else** $\eta \leftarrow x^2 \mod (x^{2m} + 1)$
9 $f^* \leftarrow f' \mod (x^{2m} + 1)$, $g^* \leftarrow g' \mod (x^{2m} + 1)$
10 Bereken $h^* \in D[y]$, $\deg h < t$ met Algoritme 2.3, met $\omega = \eta^2$, zodat
$f^*(\eta y)g^*(\eta y) \equiv h^*(\eta y) \mod (y^t - 1)$;
gebruik Algoritme 2.4 recursief voor de vermenigvuldigingen in D.
11 Bepaal $h^* = h^*(\eta(\eta^{-1}y))$.
12 Bereken $h' \in R[x, y]$, $\deg_x h' < 2m$, zodat $h^* = h' \mod (x^{2m} + 1)$.
13 $h \leftarrow h'(x, x^m) \mod (x^n + 1)$
14 **return** h

Voorbeeld 2.1.20. Stel $R = \mathsf{GF}(5)$ (merk op dat R dus geen FFT ondersteunt), $f = x^4 + 2x + 3$, $g = 2x^3 + x^2 + 4x + 2$. De berekening van $fg \in R[x]$ via Algoritme 2.4 ziet er schematisch als volgt uit. We kunnen aanvangen met $k = 3$.

Bij de eerste oproep van het algoritme vinden we $m = 2$, $t = 4$, $f' = y^2 + 2x + 3$ en $g' = (2x + 1)y + 4x + 2$ (lijnen 3 en 4). Hiermee wordt $\eta = x \mod (x^4 + 1)$ (lijnen 6–8). Dus

$$f^*(y) = y^2 + 2\eta + 3\,, \qquad g^*(y) = (2\eta + 1)y + 4\eta + 2\,,$$
$$f^*(\eta y) = \eta^2 y^2 + 2\eta + 3\,, \qquad g^*(\eta y) = (2\eta^2 + \eta)y + 4\eta + 2\,.$$

In principe wordt nu Algoritme 2.3 gebruikt om $f^*(\eta y)g^*(\eta y)$ te bepalen.

In dit overzicht vinden we door directe berekening

$$\begin{aligned} h^*(\eta y) &= (2\eta^4+\eta^3)y^3 + (4\eta^3+2\eta^2)y^2 + (4\eta^3+3\eta^2+3\eta)y + 3\eta^2+\eta+1\,, \\ h^* &= h^*(\eta(\eta^{-1}y)) \\ &= (2\eta+1)y^3 + (4\eta+2)y^2 + (4\eta^2+3\eta+3)y + 3\eta^2+\eta+1\,. \end{aligned}$$

Nu kunnen we het polynoom h' "aflezen" uit h^* (Lijn 11)

$$\begin{aligned} h'(x,y) &= (2x+1)y^3 + (4x+2)y^2 + (4x^2+3x+3)y + 3x^2+x+1\,, \\ h'(x,x^2) &= 2x^7+x^6+4x^5+x^4+3x^3+x^2+x+1 \equiv fg \mod (x^8+1) \end{aligned}$$

Uiteraard is de laatste equivalentie een gelijkheid, waardoor we fg bepaald hebben.

De discussie tot nu toe toont de correctheid aan van Algoritme 2.4. De volgende stelling verschaft informatie over de complexiteit.

Stelling 2.1.21. *Stel R een domein en $2 \in R$ een eenheid. Twee polynomen $f, g \in R[x]$, $\deg f, \deg g < n$ kunnen met Algoritme 2.4 vermenigvuldigd worden in $\mathcal{O}(n \log n \log\log n)$ operaties in R.*

Bewijs. Het aantal operaties in R om het algoritme uit te voeren voor twee polynomen van graad strikt kleiner dan $n = 2^k$, $k \in \mathbb{N}$ noteren we als $T(k)$. Wanneer we ons in de laatste fase bevinden, d.i. als $k \leq 2$, dan worden de eerste drie lijnen uitgevoerd in een constant aantal operaties. De initialisaties (lijnen (3), (4) en (5)) vergen geen aritmetische operaties in R.

De berekeningen in lijn (10) gebeuren met Algoritme 2.3. Uit Stelling 2.1.18 weten we dat dit $3t \log t$ optellingen in D, $\frac{3}{2}t \log t$ vermenigvuldigingen met machten van $\omega = \eta^2$ in de uitvoeringen van de FFT stappen, plus t delingen door $n \in R$, en t vermenigvuldigingen van "willekeurige" elementen in D. Merk trouwens op dat de machten van ω die nodig zijn in de eerste stap van Algoritme 2.3, niet "berekend" moeten worden.

Een optelling in D vergt $2m$ optellingen in R, een deling door n vergt $2m$ delingen in R, en een vermenigvuldiging van een element $a = \sum_{j=0}^{2m-1} a_j x^j$ mod $(x^{2m}+1)$ met een macht van η correspondeert met een cyclische shift van de coëfficiënten a_j en een tekenwissel van de coëfficiënten van machten van x groter van $2m-1$. Dit vergt hoogstens $2m$ operaties in R. Een essentiële vermenigvuldiging in D, dus van twee arbitraire elementen, vergt $T(\lfloor k/2 \rfloor + 1)$ operaties in R. De berekening van $f^*(\eta y)$, $g^*(\eta y)$ uit f^* en g^* en van $h^*(y) = h^*(\eta(\eta^{-}1y))$ uit $h^*(\eta y)$ vergt $3t$ vermenigvuldigingen met machten van η. De totale kost voor de uitvoering van Lijn (10) en de berekening van h^* is dus hoogstens

$$9mt \log t + 8mt + tT\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right)\,.$$

De uitvoering van Lijn (11) vergt ten hoogste $n = mt$ optellingen. Voor $T(k)$ vinden we dus, als $k > 2$,

$$T(k) \leq 2^{\lceil \frac{k}{2} \rceil} T\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) + 9 \cdot 2^k \left(\left\lceil \frac{9}{2} \right\rceil + 1\right) .$$

Vervangen we k door $k+1$, $k > 1$, en gebruiken we $\lfloor (k+1)/2 \rfloor = \lceil k/2 \rceil$ en $\lceil (k+1)/2 \rceil = \lfloor k/2 \rfloor + 1$, dan vinden we met enige manipulaties,

$$\begin{aligned} 2^{-k} T(k+1) + 45 &\leq 2^{\lceil \frac{(k+1)}{2} \rceil - k} T\left(\left\lfloor \frac{k+1}{2} \right\rfloor + 1\right) + 90 + 18\left(\left\lceil \frac{k+1}{2} \right\rceil + 1\right) - 45 \\ &= 2\left(2^{-\lceil \frac{k}{2} \rceil} T\left(\left\lceil \frac{k}{2} \right\rceil + 1\right) + 45\right) + 18\left(\left\lfloor \frac{k}{2} \right\rfloor - \frac{1}{2}\right) \end{aligned}$$

Stel $S(k) = \left(2^{-k} T(k+1) + 45\right)/(k-1)$, dan vinden we voor $k > 1$ en door inductie,

$$\begin{aligned} S(k) &\leq \frac{2\left(\lceil \frac{k}{2} \rceil - 1\right)}{k-1} S\left(\left\lceil \frac{k}{2} \right\rceil\right) + 9\frac{2\left(\lfloor \frac{k}{2} \rfloor - \frac{1}{2}\right)}{k-1} \\ &\leq S\left(\left\lceil \frac{k}{2} \right\rceil\right) + 9 \leq \ldots \leq S(2) + 9\left(\lceil \log k \rceil - 1\right) . \end{aligned}$$

Daaruit volgt tenslotte

$$\begin{aligned} T(k) &= 2^{k-1}\left((k-2)S(k-1) - 45\right) \\ &\leq \frac{9}{2} 2^k (k-2)(\lceil \log(k-1) \rceil - 1) + \frac{S(2)}{2} 2^k (k-2) - \frac{45}{2} 2^k \\ &\in \frac{9}{2} 2^k k \log k \mathcal{O}(2^k k) = \frac{9}{2} n \log n \log\log n + \mathcal{O}(n \log n) . \end{aligned}$$ □

In Lijn (1) wordt in feite beslist om vanaf $k \leq 2$ een klassiek algoritme te gebruiken. In de praktijk moet een waarde k_0 bepaald worden, zodanig dat voor $k \leq k_0$ het klassieke algoritme sneller het product bepaald dan Algoritme 2.4.

Het algoritme van Schönhage en Strassen om gehele getallen te vermenigvuldigen, kan nu afgeleid worden uit Algoritme 2.4. Essentieel worden gehele getallen voorgesteld als polynomen over de ring $\mathbb{Z}/(2^N + 1)$. Het getal N is groot genoeg zodat we ab kennen als we $ab \mod 2^N + 1$ kennen. Uit de polynomen die a en b voorstellen worden ook nu polynomen a' en b' afgeleid zoals in Lijn (4) van het algoritme, waarna het product van a' en b' modulo $x^K + 1$ bepaald wordt. De ring $\mathbb{Z}/(2^N + 1)$ voldoet aan de voorwaarden van

Stelling 2.1.21, d.i. 2 is een eenheid. Tenslotte, nadat het product van de polynomen gekend is, dient een evaluatie gedaan te worden in een macht van 2. Dit levert extra rekenwerk, maar een vermenigvuldiging met een macht van 2 in een binaire representatie komt neer het op shiften van bits. De volgende stelling vermelden we nu zonder bewijs.

Stelling 2.1.22 (Schönhage en Strassen [SS71])**.** *Het vermenigvuldigen van gehele getallen van lengte n kan gebeuren met $\mathcal{O}(n \log n \log\log n)$ woordoperaties.*

Zonder bewijs. □

Als R een veld is, (dat bijvoorbeeld FFT niet ondersteunt), dan beschikken we in elk geval reeds over een $\mathcal{O}(n \log n \log\log n)$ algoritme om polynomen over R te vermenigvuldigen. De eis dat 2 een eenheid moet zijn in R, is echter voor willekeurige ringen een beperking, ook voor velden van karakteristiek 2 maakt deze voorwaarde het algoritme onbruikbaar. We gaan dieper in op de mogelijkheden om dit op te lossen. Deze aanpassing werd door Schönhage gegeven in [Sch77].

Dat 2 een eenheid moet zijn, is noodzakelijk om Algoritme 2.3 te kunnen gebruiken. Meer bepaald wordt er in Lijn (5) van het algoritme een deling door een macht van 2 uitgevoerd. Dit kan opgelost worden door een aanpassing van Algoritme 2.2 van een 2-adische variant naar een 3-adische variant.

Algoritme 2.5 3-adische Fast Fourier transformatie

input: $n = 3^k$ met $k \in \mathbb{N}$; $f = \sum_{i=0}^{n-1} f_i x^i$; de machten ω, ω^2, …, ω^{n-1} van een primitieve n^{de} eenheidswortel $\omega \in R$.
output: $\text{DFT}_\omega(f) \in R^n$.

1 **if** $n = 1$ **then return** f_0
2 $\xi \leftarrow \omega^{n/3}$
3 $r_0 \leftarrow \sum_{j=0}^{n/3-1}(f_j + f_{j+n/3} + f_{j+2n/3})x^j$
4 $r_1 \leftarrow \sum_{j=0}^{n/3-1}(f_j + f_{j+n/3}\xi + f_{j+2n/3}\xi^2)\omega^j x^j$
5 $r_2 \leftarrow \sum_{j=0}^{n/3-1}(f_j + f_{j+n/3}\xi^2 + f_{j+2n/3}\xi^4)\omega^{2j} x^j$
6 Evalueer **recursief** de polynomen r_0, r_1 en r_2 in de machten van ω^3
7 **return** $\big(r_0(1), r_1(1), r_2(1), r_0(\omega^3), r_1(\omega^3), r_2(\omega^2), \ldots, r_0(\omega^{n-3}), r_1(\omega^{n-3}), r_2(\omega^{n-3})\big)$

De volgende stelling kan men aantonen door het bewijs van Stelling 2.1.15 aan te passen.

Stelling 2.1.23. *Zij $n = 3^k$ met $k \in \mathbb{N}$, en $\omega \in R$ een primitieve n^{de} eenheidswortel in R, $f \in R[x]$, $\deg f < n$. Algoritme 2.2 berekent $\text{DFT}_\omega(f)$ correct, in $\mathcal{O}(n \log n)$ operaties in R.*

Oefening. □

Stel nu dat 3 een eenheid is in R, en dat voldaan is aan de voorwaarden van Stelling 2.1.23. Dan kunnen we Algoritme 2.3 gebruiken, gebaseerd op de 3-adische variant.

Om Algoritme 2.4 aan te passen aan grondtal 3, zijn er heel wat (kleine) wijzigingen nodig. Essentieel worden er opnieuw geschikte eenheidswortels toegevoegd aan de ring. We zullen echter zien dat grondtal 3 een extra complicatie meebrengt om het probleem te herleiden naar 3-adische FFTs.

We veronderstellen dat R een domein is, waarin nu $3 \in R$ een eenheid is. Stel $n = 3^k$, $k \in \mathbb{N}$, en definieer nu $D := R[x]/(x^{2n} + x^n + 1)$. Noem $\omega = x \mod x^{2n} + x + 1$, dan is het eenvoudig te controleren dat ω een primitieve $3n^{\text{de}}$ eenheidswortel is in D.

Op basis van deze observatie zullen we de negative wrapped convolution aanpassen. Beschouw twee polynomen $f, g \in R[x]$, $\deg(fg) < n = 2 \cdot 3^k, k \in \mathbb{N}$. Het volstaat om $fg \mod x^{2n} + x^n + 1$ te berekenen door de veronderstelling op de graad van fg.

We definiëren $m = 3^{\lceil k/2 \rceil}$, $t := n/m$. We "verdelen" opnieuw de coëfficiënten van de polynomen f en g in blokken van grootte m:

$$f = \sum_{0 \le j < 2t} f_j x^{mj}, \qquad g = \sum_{0 \le j < 2t} x^{mj},$$

met $f_j, g_j \in R[x]$, $\deg f_j, g_j < m$ $0 \le j < 2t$. Definiëren we

$$f' := \sum_{0 \le j < 2t} f_j y^j, \qquad g' := \sum_{0 \le j < 2t} g_j y^j, \tag{2.10}$$

dan zijn $f', g' \in R[x, y]$ en $f = f'(x, x^m)$ en $g = g'(x, x^m)$. Merk op dat $\deg(f'), \deg(g') < 2t$ (en niet $< t$ zoals bij de negative wrapped convolution). Een aantal observaties kunnen we nu wel overnemen. We berekenen nu $f'g' \mod y^{2t} + y^t + 1$, stel $f'g' \equiv h' \mod y^{2t} + y^t + 1$, of, m.a.w.

$$f'g' = h' + q'(y^{2t} + y^t + 1) \tag{2.11}$$

voor een zekere $h', q' \in R[x,y]$, dus nu is

$$fg = h'(x,x^m) + q'(x,x^m)(x^{2tm} + x^{tm} + 1) \equiv h'(x,x^m) \mod (x^{2n} + x^n + 1)\,. \tag{2.12}$$

Beschouw nu de $3m^{\text{de}}$ eenheidswortel

$$\xi = x \mod (x^{2m} + 1) \in D = R[x]/(x^{2m} + x^m + 1) \tag{2.13}$$

Met het klassieke algoritme voor deling met rest kan $h' \in R[x][y]$, $\deg_y h' < 2t$ berekend worden en deze h' is uniek. Vergelijken we de coëfficiënten van y^j met $j \geq 2t$, dan zien we dat $\deg_x q' \leq \deg_x(f'g') < 2m$ en we besluiten dat

$$\deg_x h' \leq \max\{\deg_x(f'g'), \deg_x q'\} < 2m \tag{2.14}$$

Definiëren we $f^* := f' \mod (x^{2m} + x^m + 1)$, $g^* := g' \mod (x^{2m} + x^m + 1)$, en $h^* := h' \mod (x^{2m} + x^m + 1)$ in $D[y]$, dan impliceert (2.11)

$$f^*g^* \equiv h^* \mod (y^{2t} + y^t + 1) \in D[y]\,. \tag{2.15}$$

Bij de negative wrapped convolution waren we nu op het punt gekomen om een convolutie uit te voeren op basis van FFT. De modulus is nu echter niet meer eenvoudig $y^t - 1$. De eenheidswortel die we impliciet gebruiken door modulo $x^{2m} + x^m + 1$ te werken, biedt ons echter de mogelijk om de modulus $y^{2t} + y^t + 1$ te reduceren naar geschikte moduli om een convolutieproduct te bepalen, ditmaal uiteraard op basis van de 3-adische FFT.

Nu is $t \in \{m, \frac{m}{3}\}$. Dus als $t = m$, dan stellen we $\eta = \xi$, als $t = \frac{m}{3}$, dan stellen we $\eta = \xi^3$, zodat we steeds beschikken over een primitieve $3t^{\text{de}}$ eenheidswortel in D. Merk op dat $y^{2t} + y^t + 1 = (y^t - \eta)(y^t - \eta^2)$. Definiëren we

$$\begin{aligned} f^{(1)} &= f^* \mod (y^t - \eta^t)\,, & f^{(2)} &= f^* \mod (y^t - \eta^{2t})\,, \\ g^{(1)} &= g^* \mod (y^t - \eta^t)\,, & g^{(2)} &= g^* \mod (y^t - \eta^{2t})\,, \\ h^{(1)} &= h^* \mod (y^t - \eta^t)\,, & h^{(2)} &= h^* \mod (y^t - \eta^{2t})\,, \end{aligned}$$

dan volgt uit Vergelijking 2.15,

$$h^{(1)}(y) = f^{(1)}(y)g^{(1)}(y) \mod (y^t - \eta^t) \tag{2.16}$$

$$h^{(2)}(y) = f^{(2)}(y)g^{(2)}(y) \mod (y^t - \eta^{2t}) \tag{2.17}$$

Omdat $\deg(h^*) < 2t$ kunnen we h^* bepalen met de Chinese reststelling als we $h^{(1)}$ en $h^{(2)}$ kennen. Men gaat eenvoudig na dat voor

$$h^* := \frac{1}{3}\left(y^t(h^{(2)} - h^{(1)}) + \eta^{2t}h^{(1)} - \eta^t h^{(2)}\right)(2\eta^t + 1)$$

geldt dat h^* mod $(y^t - \eta^t) = h^{(1)}$ en h^* mod $(y^t - \eta^{2t}) = h^{(2)}$. Dus deze h^* is de unieke oplossing met $\deg(h^*) < 2t$. De polynomen $f^{(1)}, f^{(2)}$, $g^{(1)}$ en $g^{(2)}$ zijn polynomen van graad kleiner dan t, dus kunnen we met behulp van Algoritme 2.3, gebaseerd op de 3-adische FFT, $h^{(1)}$ en $h^{(2)}$ bepalen. Dit levert ons $h^{(1)}$ mod $y^t - 1 = h^{(1)}$ mod $y^t - \eta^t$, en $h^{(2)}$ mod $y^t - 1 = h^{(2)}$ mod $y^t - \eta^{2t}$. Merk op dat $\omega = \eta^3$ de t^{de} eenheidswortel voor de 3-adische FFT.

Algoritme 2.6 Het algoritme van Schönhage

input: $k \in \mathbb{N}$, $f, g \in R[x]$, $\deg f, \deg g < 2n = 2 \cdot 3^k$, $k \in \mathbb{N}$,
R een domein, 3 een eenheid in R.
output: $h \in R[x]$, met $fg \equiv h$ mod $(x^{2n} + x^n + 1)$ en $\deg h < 2n$

1 **if** $k \leq 2$
then Bereken fg (met klassiek algoritme, of met Karatsuba)
2 **return** fg mod $x^{2n} + x^n + 1$
3 $m \leftarrow 3^{\lceil k/2 \rceil}$, $t \leftarrow n/m$
4 initialiseer $f', g' \in R[x, y]$, met $\deg_x f', \deg_x g' < m$
en $f = f'(x, x^m)$ en $g = g'(x, x^m)$.
5 initialiseer $D = R[x]/\langle x^{2m} + x^m + 1 \rangle$
6 **if** $t = m$
7 **then** $\eta \leftarrow x$ mod $(x^{2m} + x + 1)$
8 **else** $\eta \leftarrow x^3$ mod $(x^{2m} + x + 1)$
9 $f^* \leftarrow f'$ mod $(x^{2m} + x^m + 1)$, $g^* \leftarrow g'$ mod $(x^{2m} + x^m + 1)$
10 **for** $j = 1, 2$
11 **do** $f^{(j)} \leftarrow f^*$ mod $(y^t - \eta^{jt})$, $g^{(j)} \leftarrow g^*$ mod $(y^t - \eta^{jt})$
12 Bereken $h^{(j)} \in D[y]$, $\deg h^{(j)} < t$ met Algoritme 2.3 (**3-adische versie**),
met $\omega = \eta^3$, zodat $f^{(j)}(\eta^{(j)}y)g^{(j)}(\eta^{(j)}y) \equiv h^{(j)}(\eta^{(j)}y)$ mod $(y^t - 1)$;
gebruik Algoritme 2.6 recursief voor de vermenigvuldigingen in D.
13 Bepaal $h^{(j)} = h^{(j)}(\eta^{(j)}(\eta^{-j}y))$.
14 $h^* \leftarrow \frac{1}{3}(y^t(h_2 - h_1) + \eta^{2t}h_1 - \eta^t h_2)(2\eta^t + 1)$
15 Bepaal $h' \in R[x, y]$, $\deg_x h' < 2m$, zodat $h^* = h'$ mod $(x^{2m} + x^m + 1)$.
16 $h \leftarrow h'(x, x^m)$ mod $(x^{2n} + x^n + 1)$
17 **return** h

De correctheid van het algoritme volgt uit de voorafgaande discussie. De complexiteit van het algoritme kan op een analoge wijze afgeschat worden als de complexiteit van Algoritme 2.4. We kunnen nu de volgende stelling formuleren, wat betreft de complexiteit zonder bewijs.

Stelling 2.1.24. *Stel R een domein en $3 \in R$ een eenheid. Twee polynomen $f, g \in R[x]$, $\deg f, \deg g < n$ kunnen met Algoritme 2.4 vermenigvuldigd worden in $\mathcal{O}(n \log n \log\log n)$ operaties in R.*

Bewijs. De correctheid van het algoritme volgt uit de discussie voorafgaand aan de stelling. De complexiteit nemen we aan zonder bewijs. □

Een kleine wijziging kan nog aangebracht worden aan Algoritme 2.3, zodat de voorwaarde dat $n \in R$ eenheid is, wegvalt. Vervangen we Lijn (5) in dit algoritme door

return $n\,\mathrm{DFT}_{\omega}^{-1}(\gamma) = \mathrm{DFT}_{\omega^{-1}}(\gamma)$,

dan worden er geen delingen uitgevoerd in R, en wordt $n(a * b)$ bepaald in plaats van $a * b$. Dit geldt voor zowel de 2-adische als de 3-adische variant. Men kan aantonen dat, met deze wijziging, Algoritme 2.4 $2^{\kappa}h$, $\kappa \in \mathbb{N}$ en Algoritme 2.6 $3^{\lambda}h$, $\lambda \in \mathbb{N}$, berekent. Met het uitgebreid algoritme van Euclides kunnen we $s, t \in \mathbb{Z}$ bepalen waarvoor $s2^{\kappa} + t3^{\lambda} = 1$, waarmee we dan fg kunnen berekenen. We vermelden de volgende stelling zonder bewijs

Stelling 2.1.25. *Van polynomen f, g over een domein R, $\deg f, \deg g < n$, kan het product fg in $\mathcal{O}(n \log n \log\log n)$ aritmetische operaties over R bepaald worden.*

We beschrijven nu een alternatief algoritme om op een snelle wijze gehele getallen te vermenigvuldigen. Ditmaal wordt algoritme 2.3 gebruikt met als basisring een eindig veld (dat weliswaar geen FFT ondersteunt, maar toch geschikte eenheidswortels kan bevatten). Een beperking ontstaat op een ander vlak: het algoritme is enkel geschikt om gehele getallen met "beperkte" lengte met elkaar te vermenigvuldigen.

Beschouw twee niet-negatieve gehele getallen in de 2^{64}-aire representatie, stel

$$a = \sum_{j=0}^{n-1} a_j 2^{64j} \quad b = \sum_{k=0}^{n-1} b_k 2^{64k}\,.$$

Met de a_j's en b_k's definiëren we de polynomen

$$A(x) := \sum_{j=0}^{n-1} a_j x^j \quad B(x) := \sum_{k=0}^{n-1} b_k x^k\,,$$

zodat dus $a = A(2^{64})$, $b = B(2^{64})$. Met

$$C(x) = A(x)B(x) = \sum_{l=0}^{2n-1} c_l x^l \in \mathbb{Z}[x]\,,$$

volgt $ab = C(2^{64})$. Er geldt voor alle l dat

$$0 \leq c_l = \sum_{j+k=l} a_j b_k < \sum_{j+k=l} 2^{128} \leq n \cdot 2^{128}\,.$$

We veronderstellen nu dat $n < 2^{61}$ en we beschouwen drie single precision priemgetallen p_1, p_2, p_3 gelegen tussen 2^{63} en 2^{64}. Als we $A(x)B(x)$ modulo p_j kennen, dan kunnen we met de Chinese reststelling $A(x)B(x)$ berekenen.

We kennen $C(x)$ als we $C(x) \mod x^N + 1$ kennen, met $2n - 1 < N$. Stellen we $t = \lceil \log_2(2n-1) \rceil$, dan kunnen we FFT uitvoeren modulo $x^{2^t} - 1$, op voorwaarde dat we in $\mathsf{GF}(p_j)$ een primitieve $(2^t)^{\text{de}}$ eenheidswortel hebben. Dit is het geval als $2^t \mid p_j - 1$, $j = 1, 2, 3$ (Lemma 2.1.11). We gaan hier niet dieper in op het vinden van geschikte priemgetallen bij een bepaalde t, maar er bestaan goede algoritmen voor. Voor meer details hierover verwijzen we naar [vzGG03]. We geven enkele voorbeelden van paren (p, ω).

k	29	71	75	95	108	123
ω	21	287	149	55	64	493

Voor elk van de 6 paren (p, ω) geldt dat $p = k \cdot 2^{57} + 1$ priem is en dat ω de kleinste positieve primitieve $2^{57^{\text{de}}}$ eenheidswortel modulo p is. Dit zijn in feite alle priemgetallen kleiner dan 2^{64} (en alle behalve de eerste groter dan 2^{63}) waarvoor 2^{57} een deler is van $p_j - 1$. Het priemgetal voor $k = 108$ is deelbaar door 2^{59}, dit is echter het enigste in de reeks, en er is geen enkel ander priemgetal kleiner dan 2^{64} dat deelbaar is door een hogere macht van 2. Eens we in een implementatie drie geschikte paren berekend hebben, kunnen deze altijd opnieuw gebruikt worden in het onderstaande algoritme.

Algoritme 2.7 Three primes FFT integer vermenigvuldiging

input: Twee gehele getallen $a, b \in \mathbb{N}$ in 2^{64}-adische representatie, $a, b < 2^{64 \cdot 2^{s-1}}$, met $s \leq 62$.
output: $ab \in \mathbb{N}$.
(We veronderstellen te beschikken over drie voorafbepaalde paren (p_j, ω_j), p_j single precision gehele getallen zodat $2^{63} \leq p_j < 2^{64}$ priem is en ω_j is een primitieve $2^{s\text{de}}$ eenheidswortel in $\mathsf{GF}(p_j)$).

1 Bepaal de polynomen $A, B \in \mathbb{Z}[x]$ met niet-negatieve coëfficiënten zodat $a = A(2^{64})$ en $b = B(2^{64})$
2 $t \leftarrow \lceil \log_2(1 + \deg(AB)) \rceil$
3 **for** $j = 1, 2, 3$
4 **do** Bereken $C_j \equiv AB \mod p_j$ met $\omega \equiv \omega_j^{2^{s-t}}$ (Algoritme 2.3)
5 Bereken $C \in \mathbb{Z}[x]$, met niet-negatieve coëfficiënten kleiner dan $p_1 p_2 p_3$ zodat $C \equiv C_j \mod p_j$ voor $j = 1, 2, 3$
6 **return** $C(2^{64})$

Het getal s is in feite een implementatie afhankelijke beperking van de lengte van de getallen a en b. Gelet op de priemgetallen die nodig zijn, en het gegeven overzicht net voor het algoritme, is $s = 57$ als de priemgetallen kleiner zijn dan 2^{64}. Dit maakt dat de getallen die met dit algoritme vermenigvuldigd kunnen worden, getallen zijn van maximaal $64 \cdot 2^{56}$ bits, ofte 536870912 Gigabyte. Dit is een duidelijke beperking van het algoritme.

Zonder in te gaan op de details, vermelden we nog dat het algoritme kan uitgevoerd worden in $\mathcal{O}(t2^t)$ operaties.

Het algoritme van Schönhage en Strassen uit 1971 (en enkele varianten nadien), waren tot nu toe de snelst gekende algoritmen om te vermenigvuldigen. Voor gehele getallen is de complexiteit $\mathcal{O}(n \log n \log\log n)$, maar Schönhage en Strassen zelf stellen als conjectuur dat $\mathcal{O}(n \log n)$ mogelijk is voor een op dat ogenblik nog onbekend algoritme. Zeer recent (in 2009) werd door Marin Fürer [FÖ09] een theoretische verbetering aan het algoritme van Schönhage en Strassen gepubliceerd. Het artikel beschrijft op zeer gedetailleerde wijze een in essentie op FFT gebaseerd algoritme om gehele getallen van lengte n in $n \log n 2^{\mathcal{O}(\log_2^* n)}$ woordoperaties te bepalen, met

$$\log_2^* n = \min\{i \geq 0 : \log_2^{(i)} n \leq 1\} \text{ met } \log_2^{(0)} n = n \text{ en } \log_2^{(i+1)} n = \log_2 \log_2^{(i)} n \,.$$

We hebben in deze sectie een aantal algoritmen besproken om gehele getallen en polynomen met elkaar te vermenigvuldigen. Het is duidelijk dat

de complexiteit dikwijls afhankelijk is van de basisstructuur waar de te vermenigvuldigen objecten toe behoren. Verderop zullen we dikwijls enkel elementaire veronderstellingen maken over de ring of het veld waartoe de basiselementen behoren. Daarom zullen we complexiteit afschatten in termen van de *multiplicatietijd*, d.i de complexiteit, of het aantal operaties over de basisring, om twee elementen uit die ring te vermenigvuldigen.

Notatie 2.1.26. We zullen de tijd die nodig is om twee objecten van grootte n met elkaar te vermenigvuldigen, noteren als $\mathsf{M}(n)$. We nemen steeds aan dat M superlineair is; zie Definitie 2.1.1.

Zoals we besproken hebben, hangt de multiplicatietijd af van het gekozen algoritme, maar ook van het soort objecten dat we beschouwen (b.v. het al dan niet ondersteunen van FFT). We geven een overzicht van de multiplicatietijden van de algoritmen voor vermenigvuldiging die we tot nu toe besproken hebben.

basis	algoritme	$\mathsf{M}(n)$
klassiek	1.3	$\mathcal{O}(n^2)$
Karatsuba	2.1	$\mathcal{O}(n^{\log 3})$
FFT (R ondersteunt FFT)	2.3	$\mathcal{O}(n \log n)$
Schönhage en Strassen (en variaties)	2.4	$\mathcal{O}(n \log n \log\log n)$
Fürer		$n \log n 2^{\mathcal{O}(\log_2^* n)}$

2.2 Deling met rest

2.2.1 Deling met rest door middel van Newton iteratie

Newton iteratie is een bekende methode om de wortels van reële vergelijkingen benaderd te berekenen. Wat heeft een dergelijke continue approximatiemethode nu te maken met computeralgebra? Verrassend genoeg kan dit idee ook worden toegepast om *discrete, exacte* berekeningen uit te voeren. De diepere reden hiervoor is het bestaan van *discrete valuaties* op de ringen die ons interesseren, met name $R[x]$ en $\mathbb{Q}$. Aan een dergelijke valuatie kan men een topologie en de structuur van een (niet-archimedische) metrische ruimte opleggen, waardoor het begrip continuïteit opnieuw betekenis krijgt. Twee polynomen in $R[x]$ zullen "dicht bij elkaar liggen" als ze slechts verschillen vanaf een zekere *hoge* graad van x; analoog zullen twee getallen in de p-adische topologie dicht bij elkaar liggen als hun p-adische ontwikkeling slechts verschilt vanaf een hoge macht van p.

We zullen ons opnieuw beperken tot het geval $R[x]$; voor de gehele getallen is er een gelijkaardig algoritme, maar opnieuw gaat dit gepaard met de nodige complicaties, die we hier niet zullen bespreken. Ook zullen we opnieuw aannemen dat de leidende coëfficënt $\mathrm{lc}(b)$ van b een eenheid is in R; we zullen voor het gemak onderstellen dat b monisch is, i.e. dat $\mathrm{lc}(b) = 1$.

Definitie 2.2.1. Zij $f = \sum_{i=0}^{n} f_i x^i$ een polynoom in $R[x]$ van graad n. De *k-reversal* van f (of kortweg *reversal*) is de rationale functie $\mathrm{rev}_k(f) := x^k f(1/x)$. Merk op dat $\mathrm{rev}_k(f)$ een polynoom is als en slechts als $k \geq \deg(f)$. Wanneer $k = n = \deg(f)$ is dit het polynoom bekomen door de coëfficiënten van f om te draaien, dus $\mathrm{rev}_n(f) = \sum_{i=0}^{n} f_{n-i} x^i$. We noteren $\mathrm{rev}_{\deg(f)}(f)$ kortweg als $\mathrm{rev}(f)$.

Lemma 2.2.2. *Gegeven $a, b \in R[x]$ met $\deg(a) = n$ en $\deg(b) = m \leq n$, en b monisch. Dan is $\mathrm{rev}_m(b)$ inverteerbaar modulo x^{n-m+1}. Zij d de unieke veelterm in $R[x]$ van graad $\leq n - m$ zodat*

$$d \equiv \mathrm{rev}_n(a) \cdot \mathrm{rev}_m(b)^{-1} \pmod{x^{n-m+1}}. \tag{2.18}$$

Stel dan $q := \mathrm{rev}_{n-m}(d)$ en $r := a - q \cdot b$; dan zijn q en r het quotiënt en de rest, respectievelijk, bij deling van a door b.

Bewijs. Merk op dat $\mathrm{rev}_m(b)$ een veelterm is van graad $\leq m$ met constante term 1; in het bijzonder is $\gcd(\mathrm{rev}_m(b), x^{n-m+1}) = 1$. In Stelling 1.3.16 zagen we dat hieruit volgt dat $\mathrm{rev}_m(b)$ inverteerbaar is modulo x^{n-m+1}; we zullen echter dadelijk in Stelling 2.2.3 deze inverse ook expliciet construeren.

Noem het quotiënt en de rest bij deling van a door b respectievelijk Q en R. Dan is $a = Qb + R$; wanneer we hierin $1/x$ substitueren voor de variabele x en vermenigvuldigen met x^n, bekomen we

$$x^n a(1/x) = x^{n-m} Q(1/x) \cdot x^m b(1/x) + x^{n-m+1} \cdot x^{m-1} R(1/x).$$

Merk op dat $\mathrm{rev}_n(a)$, $\mathrm{rev}_m(b)$, $\mathrm{rev}_{n-m}(Q)$ en $\mathrm{rev}_{m-1}(R)$ polynomen zijn. We verkrijgen dus

$$\mathrm{rev}_n(a) \equiv \mathrm{rev}_{n-m}(Q) \cdot \mathrm{rev}_m(b) \pmod{x^{n-m+1}}.$$

Aangezien $\mathrm{rev}_m(b)$ inverteerbaar is modulo x^{n-m+1}, volgt uit vergelijking (2.18) nu dat $\mathrm{rev}_{n-m}(Q) \equiv \mathrm{rev}_{n-m}(q) \pmod{x^{n-m+1}}$. Beide leden van deze congruentie hebben echter graad ten hoogste $n - m$, en dus is $\mathrm{rev}_{n-m}(Q) = \mathrm{rev}_{n-m}(q)$, en bijgevolg $Q = q$ en dan ook $R = r$. □

Dankzij Lemma 2.2.2 is het probleem dus herleid tot het berekenen van het inverse van een gegeven polynoom met constante term 1 (met name

$\mathrm{rev}_m(b)$) modulo een macht van x (namelijk x^{n-m+1}). Hiervoor zullen we Newton iteratie gebruiken.

We herinneren dat Newton iteratie de berekening inhoudt van opeenvolgende benaderingen van een vergelijking $\varphi(g) = 0$. Vertrekkende van een goed gekozen startwaarde g_0 worden de opeenvolgende benaderingen gevonden door

$$g_{i+1} = g_i - \frac{\varphi(g_i)}{\varphi'(g_i)}\,,$$

waarbij φ' de afgeleide is van φ naar g.

We willen dit nu toepassen op $R[x]$ in plaats van $\mathbb{R}$, met $\varphi(g) = f - 1/g$, waarbij f een *vast* element van $R[x]$ is (met name het polynoom waarvan we de inverse willen berekenen). De Newton iteratiestap is dan

$$g_{i+1} = g_i - \frac{f - 1/g_i}{1/g_i^2} = 2g_i - fg_i^2\,.$$

Niet alleen blijkt deze methode te werken, we zullen zien dat de keuze van een goede startwaarde g_0 hier geen enkel probleem is — hetgeen bij Newton iteratie voor reële functies geenszins het geval is!

Stelling 2.2.3. *Zij $f, g_0, g_1, \cdots \in R[x]$, met $f(0) = 1$, $g_0 = 1$, en $g_{i+1} \equiv 2g_i - fg_i^2 \pmod{x^{2^{i+1}}}$ voor alle i. Dan is $fg_i \equiv 1 \pmod{x^{2^i}}$ voor alle i.*

Bewijs. Het bewijs verloopt per inductie op i. Voor $i = 0$ is $fg_0 \equiv f \equiv f(0) \equiv 1 \pmod{x}$. Voor de inductiestap vinden we

$$1 - fg_{i+1} \equiv 1 - 2fg_i + f^2g_i^2 \equiv (1 - fg_i)^2 \equiv 0 \pmod{(x^{2^i})^2}\,. \qquad \square$$

We krijgen dus het volgende algoritme om de inverse van f modulo x^ℓ te berekenen.

Algoritme 2.8 Polynomiale inversie d.m.v. Newton iteratie

input: een polynoom $f \in R[x]$ met $f(0) = 1$, en $\ell \in \mathbb{N}$.
output: $g \in R[x]$ zodat $fg \equiv 1 \pmod{x^\ell}$.

```
1  g_0 ← 1,  r ← ⌈log l⌉
2  for i = 1, ..., r
3        do g_i ← 2g_{i-1} − f g_{i-1}^2 rem x^{2^i}
4  return g_r
```

Als we het inverse willen berekenen van een polynoom $f \in R[x]$ waarbij $f(0)$ een eenheid is in R verschillend van 1, dan stellen we eenvoudigweg

$g_0 = f(0)^{-1}$ in regel 1. Als $f(0)$ geen eenheid is, dan is f niet inverteerbaar, want $fg \equiv 1 \pmod{x^\ell}$ impliceert $f(0)g(0) = 1$.

Stelling 2.2.4. *Algoritme 2.8 berekent correct de inverse van f modulo x^ℓ, en vereist $\mathcal{O}(\mathsf{M}(\ell))$ operaties in R.*

Bewijs. De correctheid volgt uit Stelling 2.2.3 en het feit dat $x^\ell \mid x^{2^r}$. De berekeningen in regel 3 tijdens de i'de uitvoering van de lus vereisen minder dan $2\mathsf{M}(2^i)$ operaties in R. De totale kost is dus minder dan $2 \cdot \sum_{i=1}^{r} \mathsf{M}(2^i)$, en uit Lemma 2.1.2 met $S = \mathsf{M}$ en $b = 1$ volgt dat dit complexiteit $\mathcal{O}(\mathsf{M}(\ell))$ heeft. □

Samen met Lemma 2.2.2 bekomen we nu een efficiënt algoritme voor deling met rest in $R[x]$.

Algoritme 2.9 Deling met rest in $R[x]$ m.b.v. Newton iteratie

input: polynomen $a, b \in R[x]$, $b \neq 0$ monisch.
output: polynomen $q, r \in R[x]$ met $a = qb + r$ en $\deg r < \deg b$.
1 **if** $\deg a < \deg b$
2 **then return** $q = 0$ en $r = a$
3 $\ell \leftarrow \deg a - \deg b$
4 Bereken het inverse van $\mathrm{rev}(b)$ modulo $x^{\ell+1}$ m.b.v. Algoritme 2.8
5 $d \leftarrow \mathrm{rev}(a) \cdot \mathrm{rev}(b)^{-1} \bmod x^{\ell+1}$
6 **return** $q = \mathrm{rev}_\ell(d)$ en $r = a - bq$

Stelling 2.2.5. *Zij $a, b \in R[x]$ met $\deg(a) = n$ en $\deg(b) = m \leq n$. De deling met rest van a door b met behulp van Algoritme 2.9 vereist $\mathcal{O}(\mathsf{M}(n))$ operaties in R.*

Bewijs. Uit Stelling 2.2.4 volgt dat we $\mathcal{O}(\mathsf{M}(n-m))$ ringoperaties nodig hebben in regel 4. In regel 5 gebruiken we $\mathsf{M}(n-m)$ operaties in R (we rekenen immers modulo x^{n-m+1}); in regel 6 volstaat het de eerste m coëfficiënten te berekenen aangezien $\deg r < \deg b$, en hebben we bijgevolg ten hoogste $\mathsf{M}(m) + m \in \mathcal{O}(\mathsf{M}(m))$ operaties nodig. Het volstaat nu vast te stellen dat elk van deze complexiteiten in $\mathcal{O}(\mathsf{M}(n))$ zit. □

We besluiten dus dat we de deling met rest kunnen uitvoeren in een tijd die dezelfde orde heeft als de vermenigvuldiging.

Hoofdstuk 3

Algoritmen voor de grootste gemene deler

Met het (uitgebreide) algoritme van Euclides kan de grootste gemene deler van twee gehele getallen berekend worden. We hebben in Hoofdstuk 1 dit algoritme in een abstract kader besproken, zodat het niet alleen van toepassing is in de ring der gehele getallen, maar in elk Euclidisch domein, maar we zijn vrij snel op beperkingen gestoten wanneer we de grootste gemene deler willen berekenen in een uniek factorisatiedomein dat geen Euclidisch domein is. In dit hoofdstuk wordt de theorie verder uitgediept, met het oog op het ontwerp van een aantal algoritmen die de bepaling van de grootste gemene deler toelaten in een aantal specifieke uniek factorisatiedomeinen die geen Euclidisch domein zijn.

3.1 Grootste gemene deler van polynomen over een UFD

We hebben reeds vermeld dat $\mathbb{Z}$ een Euclidisch domein is, maar $\mathbb{Z}[x]$ niet. Deze laatste ring is echter wel een UFD (zelfs met een functie normal), en dus hebben elke twee polynomen over $\mathbb{Z}$ een (zelfs unieke genormaliseerde) gcd.

Voorbeeld 3.1.1. (i) De ring $\mathbb{Z}$ is een Euclidisch domein, maar $\mathbb{Z}[x]$ is dat niet. We tonen straks aan dat $\mathbb{Z}[x]$ een UFD is.

(ii) De ring $F[x]$, met F een veld, is een Euclidisch domein, maar $F[x][y]$ (of ook $F[x, y]$ genoteerd), is dat niet. Algemeen tonen we straks aan dat $R[x]$ een UFD is als R dat is.

Vanaf nu veronderstellen we steeds dat we in een ring R beschikken over een functie normal.

Definitie 3.1.2. Stel dat R een UFD is. Beschouw een polynoom f over R, i.e. $f = f_n x^n + \cdots f_1 x + f_0 \in R[x]$. De *content* van f is gedefinieerd als $\text{cont}(f) = \gcd(f_0, f_1, \cdots, f_n) \in R$. Als $n = 0$ dan is $\text{cont}(f) = \text{normal}(f_0)$. Het polynoom f is *primitief* als $\text{cont}(f) = 1$. We definiëren het *primitief*

gedeelte van het polynoom f, $\mathrm{pp}(f)$, als $f = \mathrm{cont}(f) \cdot \mathrm{pp}(f)$. De content en het primitief gedeelte van f zijn uniek omdat we een unieke gcd hebben.[1]

Lemma 3.1.3. *Stel R een UFD. Voor een polynoom $f \in R[x]$ en $c \in R$ geldt* $\mathrm{cont}(cf) = \mathrm{cont}(c) \cdot \mathrm{cont}(f)$ *en* $\mathrm{pp}(c \cdot f) = \mathrm{pp}(c) \cdot \mathrm{pp}(f)$.

Bewijs. Oefening. □

Het volgende lemma, toegeschreven aan Gauß, speelt een belangrijke rol om te bewijzen dat $R[x]$ een UFD is als R dat is.

Lemma 3.1.4 (Lemma van Gauß)**.** *Stel dat R een UFD is, dan is het product van twee primitieve polynomen over R een primitief polynoom.*

Bewijs. Stel dat $f, g \in R[x]$ twee primitieve polynomen zijn en kies een priemelement $p \in R$. Dan is $D = R/(p)$ een domein, en dus is $D[x]$ ook een domein, want $D[x]$ kan geen nuldelers bevatten als D er geen bevat. Omdat f en g primitief zijn, zijn $f \bmod p$ en $g \bmod p$ noodzakelijk verschillend van 0 in $D[x]$. Daaruit volgt dat $gf \bmod p$ verschillend is van 0 in $D[x]$, of, gelijkwaardig, $p \nmid \mathrm{cont}(fg)$, dus $\mathrm{cont}(fg) = 1$. □

Gevolg 3.1.5. *Beschouw twee elementen $f, g \in R[x]$, R een UFD. Dan geldt* $\mathrm{cont}(fg) = \mathrm{cont}(f)\,\mathrm{cont}(g)$ *en* $\mathrm{pp}(fg) = \mathrm{pp}(f)\,\mathrm{pp}(g)$.

Bewijs. Lemma 3.1.4 impliceert dat $h^* := \mathrm{pp}(f)\,\mathrm{pp}(g)$ primitief is. Verder geldt dat

$$fg = \mathrm{cont}(f)\,\mathrm{pp}(f)\,\mathrm{cont}(g)\,\mathrm{pp}(g) = \mathrm{cont}(f)\,\mathrm{cont}(g)h^*\,.$$

Door Lemma 3.1.3 vinden we nu het eerste deel van de stelling,

$$\mathrm{cont}(fg) = \mathrm{cont}(\mathrm{cont}(f)\,\mathrm{cont}(g)) \cdot \mathrm{cont}(h^*) = \mathrm{cont}(f)\,\mathrm{cont}(g)\,,$$

waardoor uit

$$\mathrm{cont}(f)\,\mathrm{cont}(g) \cdot h^* = fg = \mathrm{cont}(fg)\,\mathrm{pp}(fg)\,,$$

het tweede deel volgt. □

We veralgemenen de definitie van content en primitief deel van een polynoom over R naar een polynoom over het breukenveld K van R. Stel

$$f = \sum_{i=0}^{n} \frac{a_i}{b} x^i \in K[x]\,,$$

[1] omdat we veronderstellen dat we over een functie normal beschikken

met een gemeenschappelijke noemer $b \in R \setminus \{0\}$ en alle $a_i \in R$, dan definiëren we $\operatorname{cont}(f) = \gcd(a_0, \ldots, a_n)/\operatorname{cont}(b) \in K$ en $\operatorname{pp}(f) = f/\operatorname{cont}(f)$. Bewijs als oefening dat Lemma 3.1.3 en Gevolg 3.1.5 geldig zijn voor $c \in K$ en $f, g \in K[x]$.

We kunnen nu de volgende beroemde stelling van Gauß bewijzen.

Stelling 3.1.6 (Gauß)**.** *Als R een UFD is, dan is $R[x]$ een UFD.*

Bewijs. Voor elke twee niet nul polynomen $f, g \in R[x]$ geldt $\deg(fg) = \deg f + \deg g$ omdat R een domein is. Dit impliceert ook dat de verzameling van de eenheden van $R[x]$ juist de verzameling van de eenheden van R is, en dat een priem in R irreduciebel is in $R[x]$.

Stel $f \in R[x]$ is verschillend van een eenheid en verschillend van nul. Omdat R een UFD is kan $\operatorname{cont}(f)$ op een unieke wijze geschreven worden als het product van irreduciebele elementen van R. Noteer het breukenveld van R als K. De ring $K[x]$ is een Euclidisch domein en bijgevolg een UFD, en $\operatorname{pp}(f) = f_1 f_2 \cdots f_r$ in $K[x]$ met irreduciebele niet constante polynomen $f_1, \cdots, f_r$ over K. Wanneer we de contents wegdelen, dan levert Gevolg 3.1.5 de factorisatie

$$\operatorname{pp}(f) = \operatorname{pp}(f_1) \cdots \operatorname{pp}(f_r) \tag{3.1}$$

in irreduciebele primitieve polynomen in $R[x]$. Omdat elke factor $\operatorname{pp}(f_i)$ primitief is in $R[x]$ en irreduciebel in $K[x]$, is deze factor ook irreduciebel in $R[x]$. Dit bewijst alvast het bestaan van een factorisatie in irreduciebele elementen in $R[x]$.

Beschouw een constant polynoom $f \in R[x]$. Omdat de functie deg additief is behoort noodzakelijk elke irreduciebele factor van f tot R, en de uniciteit van de factorisatie van f in $R[x]$ volgt uit de uniciteit van de factorisatie in R. Beschouw nu een niet constant polynoom $f \in R[x]$. Veronderstel dat

$$p_1 \cdots p_k \cdot f_1 \cdots f_r = f = q_1 \cdots q_l \cdot g_1 \cdots g_s$$

twee factorisaties zijn van f in $R[x]$ waarbij

$$p_1, \cdots, p_k, q_1, \cdots q_l \in R \,\text{en}\, f_1, \cdots, f_r, g_1, \cdots, g_s \in R[x]$$

primitief zijn en niet constant. Dan geldt $p_1 \cdots p_k = \operatorname{cont}(f) = q_1 \cdots q_l$ door Gevolg 3.1.5. Dus $k = l$ en $p_1 = q_1, \cdots, p_k = q_k$ na herordening, omdat R een UFD is. Daarenboven,

$$f_1 \cdots f_r = \operatorname{pp}(f) = g_1 \cdots g_s \tag{3.2}$$

in $R[x]$. Omdat elke factorisatie in $K[x]$ van een niet constant primitief polynoom een niet-triviale factorisatie oplevert in $R[x]$, zoals (3.1), zijn deze

polynomen irreduciebel in $K[x]$. Dus (3.2) bevat ook twee factorisaties van $\mathrm{pp}(f)$ in $K[x]$ in irreduciebele elementen, dus $r = s$ en, na geschikte herordening, $f_i = b_i g_i$, met $b_i \in K$, $1 \leq i \leq r$. Omdat f_i en g_i primitief zijn, hebben we

$$f_i = \mathrm{pp}(f_i) = \mathrm{pp}(b_i g_i) = \mathrm{pp}(b_i)\,\mathrm{pp}(g_i) = \mathrm{pp}(b_i) g_i$$

voor $1 \leq i \leq r$. Dit bewijst de stelling, omdat $\mathrm{pp}(b_i)$ een eenheid is in R. □

Omdat $R[x]$ een UFD is als R een UFD is, hebben elke twee elementen van $R[x]$ een gcd. Om de uniciteit van de gcd te verzekeren voor $R[x]$ definiëren we $\mathrm{lu}(f) = \mathrm{lu}(\mathrm{lc}(f))$ voor alle $f \in R[x]$. Bewijs als oefening dat dit een functie normal definieert op $R[x]$. Een element $f \in R[x]$ is aldus genormaliseerd als $\mathrm{lc}(f)$ genormaliseerd is en $\gcd(f, g)$ is het unieke genormaliseerde geassocieerde element in $R[x]$ van alle grootste gemene delers van f en g, zoals in een Euclidisch domein.

De stelling van Gauß impliceert onmiddellijk

Gevolg 3.1.7. *Stel dat R een veld is of de ring der gehele getallen $\mathbb{Z}$. Dan is voor alle $n \geq 0$ $R[X_1, \cdots, X_n]$ een UFD.*

Gevolg 3.1.8. *Stel dat R een UFD is met als breukenveld K. Stel dat $f, g \in R[x]$, en stel $h = \gcd(f, g) \in R[x]$.*

(i) *De priemen van $R[x]$ zijn de priemen van R samen met de primitieve polynomen in $R[x]$ die irreduciebel zijn in $K[x]$;*

(ii) $\mathrm{cont}(h) = \gcd(\mathrm{cont}(f), \mathrm{cont}(g))$ *in R en* $\mathrm{pp}(h) = \gcd(\mathrm{pp}(f), \mathrm{pp}(g))$ *in $R[x]$. Meer bepaald, $h = \gcd(\mathrm{cont}(f), \mathrm{cont}(g)) \cdot \gcd(\mathrm{pp}(f), \mathrm{pp}(g))$ en h is primitief van zodra f of g primitief is;*

(iii) $h/\mathrm{lc}(h) \in K[x]$ *is de monische* gcd *van f en g in $K[x]$.*

Bewijs. (i) Stel $p \in R[x]$. Veronderstel dat p priem is. Als p constant is, dan is p priem in R, anders is het een primitief polynoom en irreduciebel in $K[x]$, omdat een factorisatie in $K[x]$ een factorisatie in $R[x]$ geeft, zoals in Uitdrukking (3.1).

Anderzijds, als p niet priem is, dan bestaat er een factorisatie $p = uv$ met $u, v \in R$ verschillend van een eenheid. Dit toont aan dat p niet priem is in R, en als p primitief is en niet constant, dat p reduciebel is in $K[x]$.

(ii) Het polynoom h deelt f en bijgevolg is $\mathrm{cont}(h)$ een deler van $\mathrm{cont}(f)$. Hetzelfde geldt voor g, dus $\mathrm{cont}(h) \mid \gcd(\mathrm{cont}(f), \mathrm{cont}(g))$. Anderzijds, omdat $\gcd(\mathrm{cont}(f), \mathrm{cont}(g)) \in R$ een gemeenschappelijke factor is van f en g deelt deze gcd ook h en bijgevolg ook $\mathrm{cont}(h)$. Dit bewijst het eerste gedeelte van (ii). Het tweede gedeelte volgt analoog, gebruikmakend van het

feit dat $\mathrm{pp}(h) \mid \mathrm{pp}(f)$, door Gevolg 3.1.5 en het feit dat $\mathrm{pp}(h)$ genormaliseerd is omdat h genormaliseerd is.

(iii) Omdat $h/\operatorname{lc}(h)$ een deler is van f en g in $K[x]$, is het ook een deler van hun monische gcd h^*. Anderzijds $f = f^*h^*$ voor een zekere $f^* \in K[x]$. Van beide leden de content nemend, en samen met Gevolg 3.1.5, vinden we dat $\mathrm{pp}(h^*) \mid \mathrm{pp}(f) \mid f$ in $R[x]$, en analoog, $\mathrm{pp}(h^*)|g$. Dus $\mathrm{pp}(h^*)$ deelt $h = \gcd(f, g)$ in $R[x]$, wat impliceert dat h^* en $h/\operatorname{lc}(h)$ elkaar delen in $K[x]$. Omdat beide monisch zijn, zijn ze dan ook gelijk. □

Merk op dat (ii) niet geldt als h niet genormaliseerd is. Stel bijvoorbeeld $R = \mathbb{Z}$, $f = g = x$ en $h = -x$, dan $\mathrm{pp}(h) = -x \neq x = \gcd(\mathrm{pp}(f), \mathrm{pp}(g))$. We geven twee voorbeelden.

Voorbeeld 3.1.9. Stel $R = \mathbb{Z}$. Het breukenveld van R is dan $\mathbb{Q}$.

$$f = 18x^3 - 42x^2 + 30x - 6,\ g = -12x^2 + 10x - 2 \in \mathbb{Z}[x]$$

dan

$$\begin{aligned} \mathrm{cont}(f) = \gcd(18, -42, 30, -6) = 6,\ \mathrm{cont}(g) = \gcd(-12, 10, -2) = 2, \\ \mathrm{pp}(f) = 3x^3 - 7x^2 + 5x - 1,\ \mathrm{pp}(g) = -6x^2 + 5x - 1 \end{aligned}$$

Met het Euclidisch algoritme in $\mathbb{Q}[x]$ vinden we dat $\gcd(f, g) = \gcd(\mathrm{pp}(f), \mathrm{pp}(g)) = x - \frac{1}{3} \in \mathbb{Q}[x]$. Bijgevolg

$$\begin{aligned} \mathrm{pp}(\gcd(f, g)) = \gcd(\mathrm{pp}(f), \mathrm{pp}(g)) = 3x - 1 \text{ in } \mathbb{Z}[x]\,, \\ \mathrm{cont}(\gcd(f, g)) = \gcd(\mathrm{cont}(f), \mathrm{cont}(g)) = \gcd(6, 2) = 2\,, \\ \gcd(f, g) = \mathrm{cont}(\gcd(f, g)) \cdot \mathrm{pp}(\gcd(f, g)) = 6x - 2 \text{ in } \mathbb{Z}[x]. \end{aligned}$$

De polynomen f en $\mathrm{pp}(f)$ zijn genormaliseerd in $\mathbb{Z}[x]$ omdat hun leidende coëfficiënten positief zijn, maar g en $\mathrm{pp}(g)$ zijn niet genormaliseerd. Zowel $\gcd(f, g)$ als $\gcd(\mathrm{pp}(f), \mathrm{pp}(g))$ zijn genormaliseerd.

Voorbeeld 3.1.10. Beschouw de volgende polynomen over $\mathsf{GF}(5)[x, y]$:

$$\begin{aligned} f &= (y^3 + 3y^2 + 2y)x^3 + (y^2 + 3y + 2)x^2 + (y^3 + 3y^2 + 2y)x + (y^2 + 3y + 2)\,, \\ g &= (2y^3 + 3y^2 + y)x^2 + (3y^2 + 4y + 1)x + (y + 1)\,. \end{aligned}$$

Stel $R = \mathsf{GF}(5)[y]$, dan vinden we met het algoritme van Euclides

$$\begin{aligned} \mathrm{cont}(f) &= \gcd(y^3 + 3y^2 + 2y, y^2 + 3y + 2, y^3 + 3y^2 + 2y, y^2 + 3y + 2) \\ &= y^2 + 3y + 2\,, \\ \mathrm{cont}(g) &= \gcd(2y^3 + 3y^2 + y, 3y^2 + 4y + 1, y + 1) = y + 1\,, \\ \mathrm{pp}(f) &= yx^3 + x^2 + yx + 1\,, \\ \mathrm{pp}(g) &= (2y^2 + y)x^2 + (3y + 1)x + 1 \end{aligned}$$

Omdat R een Euclidisch domein is en $f, g \in R[x]$, kunnen we bovenstaande resultaten toepassen om de gcd van f en g te berekenen.

$$\begin{aligned}
\gcd(\mathrm{cont}(f), \mathrm{cont}(g)) &= y + 1\,, \\
\gcd(\mathrm{pp}(f), \mathrm{pp}(g)) &= yx + 1 \,(\text{in } \mathsf{GF}(5)[y][x] = \mathsf{GF}(5)[x, y])\,, \\
\gcd(f, g) &= \gcd(\mathrm{pp}(f), \mathrm{pp}(g)) = x + \frac{1}{y} \,(\text{in } \mathsf{GF}(5)(y)[x])\,, \\
\gcd(f, g) &= \gcd(\mathrm{cont}(f), \mathrm{cont}(g)) \cdot \gcd(\mathrm{pp}(f), \mathrm{pp}(g)) \\
&= (y^2 + y)x + (y + 1) \,(\text{in } \mathsf{GF}(5)[y][x])\,.
\end{aligned}$$

Dus f en $\mathrm{pp}(f)$ zijn genormaliseerd in $R[x]$, terwijl g en $\mathrm{pp}(g)$ dat niet zijn. Zowel $\gcd(f, g)$ als $\gcd(\mathrm{pp}(f), \mathrm{pp}(g))$ zijn genormaliseerd.

Indien we beschikken over een algoritme om de gcd te bepalen in een UFD R, dan geeft Gevolg 3.1.8 aanleiding tot het volgende algoritme om de gcd te bepalen van twee *polynomen* f en g over R. We mogen veronderstellen dat f en g primitief zijn.

Algoritme 3.1 gcd van primitieve polynomen over een UFD

input: $f, g \in R[x]$, met R een UFD en f en g primitief.
output: $h = \gcd(f, g)$.

1 Gebruik Algoritme 1.6 om de monische gcd v van f en g in $K[x]$ te bepalen, met K het breukenveld van R.
2 $b \leftarrow \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$
3 **return** $\mathrm{pp}(bv) \in R[x]$

Stelling 3.1.11. *Algoritme 3.1 berekent de gcd van twee primitieve polynomen f en g over een UFD R.*

Bewijs. Stel dat h de genormaliseerde gcd van f en g is in $R[x]$. Gevolg 3.1.8 impliceert dat v, berekend in lijn 1 van het algoritme, gelijk is aan $h/\,\mathrm{lc}(h)$. Omdat h een deler is van f en g in $R[x]$, is $\mathrm{lc}(h)$ een deler van $\mathrm{lc}(f)$ en $\mathrm{lc}(g)$ in R. Bijgevolg is $\mathrm{lc}(h)$ een deler van $\gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, en $bv \in R[x]$.

Omdat $b = \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, is b genormaliseerd. Verder is $\mathrm{lc}(bv) = b$. Dus uit de definitie van de functie $\mathrm{normal}_{R[x]}$ (zie uitdrukking 1.2 pagina 16) volgt dat $bv \in R[x]$ genormaliseerd is. Ook $\mathrm{cont}(bv)$ is genormaliseerd, dus $\mathrm{pp}(bv)$ is ook genormaliseerd. Per definitie is $\mathrm{pp}(bv)$ ook primitief. We besluiten dat $\mathrm{pp}(bv) = h$. □

Indien we over een algoritme beschikken om de gcd te bepalen in een UFD R, dan is Algoritme 3.1 geschikt om de gcd van twee primitieve polynomen f en g over R te bepalen. Indien f of g niet primitief zijn, komt daar nog de bepaling van $\gcd(\mathrm{cont}(f), \mathrm{cont}(g))$ bij. Het is duidelijk dat de complexiteit enkel essentieel afhangt van de eerste lijn. Algoritme 3.1 heeft dus dezelfde complexiteit als Algoritme 1.6 uitgedrukt in operaties over het breukenveld van R.

Stel dat men de gcd van twee elementen in $\mathbb{Z}[x]$ wil bepalen met Algoritme 3.1. Algoritme 1.6 wordt dan uitgevoerd over $\mathbb{Q}$. Het is duidelijk dat operaties in $\mathbb{Q}$ dan aanleiding geven tot woordoperaties over $\mathbb{Z}$. Voorbeelden tonen aan dat sprake is van snel stijgende getallengtes tijdens de uitvoering van Algoritme 1.6. Wel kan aangetoond worden dat deze getallengtes begrensd zijn, en dat de uitvoeringstijd polynomiaal blijft in functie van de graad van de polynomen.

Stelling 3.1.12. *Stel $R = \mathbb{Z}[x]$, respectievelijk $R = F[x, y]$, F een veld. Stel $n = \max\{\deg f, \deg g\}$, $f, g \in R$. Algoritme 3.1 bepaalt $\gcd(f, g)$ in $\mathcal{O}(n^6)$ operaties in $\mathbb{Z}$, respectievelijk F.*

Bewijs. Zonder bewijs. □

De snelstijgende getallengtes zijn echter een goede motivatie om op zoek te gaan naar *modulaire algoritmen*, waarvan dan inderdaad kan aangetoond worden dat ze minder woordoperaties vereisen.

Voor we de beschrijving van modulaire algoritmes starten, merken we nog op dat er nog een elementair algoritme bestaat om de gcd in $\mathbb{Z}[x]$ te bepalen. Het basisprincipe is de zogenaamde *pseudodeling* in $\mathbb{Z}[x]$. Gegeven twee polynomen $a, b \in \mathbb{Z}[x]$, met $b \neq 0$, dan bestaan er steeds polynomen $q, r \in \mathbb{Z}[x]$ zodanig dat

$$\mathrm{lc}(b)^{1+\deg a-\deg b} a = qb + r, \quad \deg r < \deg b$$

Deze pseudo deling is geldig in elk domein. We beschouwen een domein R, en we veronderstellen dat we voor het domein R een functie normal ter beschikking hebben, zodat we voor $R[x]$ een functie normal kunnen definiëren via $\mathrm{normal}(f) = \mathrm{normal}(\mathrm{lc}(f))f/\mathrm{lc}(f)$. We kunnen dan onmiddellijk het volgend algoritme afleiden.

Algoritme 3.2 Primitief algoritme van Euclides

input: Primitieve polynomen $f, g \in R[x]$, met R een UFD,
$\deg f = n$ en $\deg g = m$.
output: $h = \gcd(f, g) \in R[x]$.

```
1  r_0 ← f, r_1 ← g, n_0 ← n, n_1 ← m
2  i ← 1
3  while r_i ≠ 0
4        do a_{i-1} ← lc(r_i)^{1+n_{i-1}-n_i} r_{i-1},
5           q_i ← quo(a_{i-1}, r_i)
6           r_{i+1} ← pp(rem(a_{i-1}, r_i))
7           n_{i+1} ← deg r_{i+1}
8           i ← i + 1
9  l ← i − 1
10 return normal(r_l)
```

Stelling 3.1.13. *Stel $f, g \in R[x]$, R een domein waar een functie normal beschikbaar is. Algoritme 3.2 berekent de grootste gemene deler van f en g.*

Bewijs. Stel $\alpha_i = \mathrm{lc}(r_i) \in R$, voor alle i. Stel K het breukenveld van R. Men gaat na dat $r_i/\alpha_i \in K[x]$ de i-de rest is in het Algoritme van Euclides (Algoritme 1.6) voor $f/\mathrm{lc}(f), g/\mathrm{lc}(g) \in K[x]$. Meer bepaald is r_l primitief en op een factor $\rho \in R$ gelijk aan de monische gcd. Vandaar is $\mathrm{normal}(r_l) = h = \gcd(f, g) \in R[x]$. □

Men kan eveneens de complexiteit van het algoritme analyseren. Daartoe is geen uitgebreide redenering nodig, maar wel een aantal theoretische concepten die buiten het bestek van deze nota's vallen. Daarom vermelden we de volgende stelling zonder bewijs.

Stelling 3.1.14. *Beschouw Algoritme 3.2. Stel $\delta = \max\{n_{i-1} - n_i \| 1 \leq i \leq l\}$, de maximale graad van de optredende quotiënten.*

(i) *Stel $R = \mathbb{Z}$ en $\|f\|_\infty, \|g\|_\infty \leq A$, dan vergt de uitvoering van het algoritme $\mathcal{O}(n^3 m \delta^2 \log^2(nA))$ woordoperaties.*

(ii) *Stel $R = F[y]$, F een veld en $\deg_y f, \deg_y g \leq d$, dan vergt de uitvoering van het algoritme $\mathcal{O}(n^3 m \delta^2 d^2)$ operaties in F*

Zonder bewijs. □

3.2 De resultante

Het verband tussen de gcd van twee polynomen over een UFD R en over het breukenveld K van R wordt verder beschreven, en dit via de *resultante*.

Lemma 3.2.1. *Stel $f, g \in R[x]$ zijn twee niet nul polynomen over het UFD R. Dan geldt dat $\deg \gcd(f,g) > 0$ als en slechts als er elementen $s, t \in R[x] \setminus \{0\}$ bestaan zodat $sf + tg = 0$, $\deg s < \deg g$, en $\deg t < \deg f$.*

Bewijs. Stel $h = \gcd(f,g)$. Als $\deg h > 0$, dan zijn $s = -\frac{g}{h}$, $t = \frac{f}{h}$ de gezochte coëfficiënten. Omgekeerd, stel dat $s, t \in R[x]$ aan de veronderstellingen voldoen. Indien f en g geen gemeenschappelijke factor h bezitten met $\deg h > 0$, dan impliceert $sf = -tg$ dat $f \mid t$, hetgeen onmogelijk is aangezien $t \neq 0$ en $\deg f > \deg t$. Deze contradictie toont aan dat $\deg h > 0$. □

Voor $R = F$ een veld kan Lemma 3.2.1 op een andere wijze geformuleerd worden. Daartoe definiëren we een afbeelding, gebaseerd op de lineaire combinatie die we bestuderen.

Gegeven twee polynomen $f, g \in F[x] \setminus \{0\}$, van respectieve graad n en m. Definieer:

$$\varphi = \varphi_{f,g} : F[x] \times F[x] \longrightarrow F[x] : (s,t) \longmapsto sf + tg \tag{3.3}$$

Deze afbeelding is een lineaire afbeelding tussen oneindigdimensionale vectorruimten over F (en op natuurlijke wijze een $F[x]$-lineaire afbeelding van $F[x]$-modules). Voor elk element $d \in \mathbb{N}$ definiëren we $P_d = \{a \in F[x] : \deg a < d\}$, en $P_0 = \{0\}$. De restrictie van φ,

$$\varphi_0 : P_m \times P_n \longrightarrow P_{n+m}$$

is een F-lineaire afbeelding tussen vectorruimten van gelijke dimensie. Lemma 3.2.1 kan nu als volgt geformuleerd worden.

Stelling 3.2.2. *Stel dat $f, g \in F[x] \setminus \{0\}$ twee polynomen zijn van respectieve graad n en m.*

(i) $\gcd(f,g) = 1 \iff \varphi_0$ *is een isomorfisme*

(ii) *Als $\gcd(f,g) = 1$ en $n+m \geq 1$, dan zijn de Bézout coëfficiënten s_l en t_l berekend met het uitgebreid algoritme van Euclides de unieke oplossing in $P_m \times P_n$ van $\varphi_0(s_l, t_l) = 1$.*

Bewijs. Door Lemma 3.2.1 weten we dat $\deg \gcd(f,g) \geq 1 \iff$ er bestaat een koppel niet nul elementen $(s,t) \in P_m \times P_n$ zodat $\varphi_0(s,t) = 0 \iff \varphi_0$ is niet injectief. Voor de afbeelding φ_0 zijn de volgende drie eigenschappen gelijkwaardig, omdat $\dim(P_m \times P_n) = \dim(P_{m+n}) < \infty$:

(1) φ_0 is een isomorfisme

(2) φ_0 is injectief

(3) φ_0 is surjectief

Deel (i) van de stelling is aangetoond. Voor (ii) garandeert Lemma 1.3.12 dat $(s_l, t_l) \in P_m \times P_n$. Omdat φ_0 een isomorfisme is, is de oplossing voor $\varphi_0(s_l, t_l) = 1$ uniek. □

Om praktische berekeningen uit te voeren zullen we φ_0 voorstellen door een matrix.

$$f = \sum_{j=0}^{n} f_j x^j, g = \sum_{j=0}^{m} g_j x^j,$$

met alle $f_j, g_j \in F$. Een basis voor de vectorruimte $P_m \times P_n$ wordt gegeven door de vectoren $\{(x^i, 0) \mid 0 \leq i < m\} \cup \{(0, x^j) \mid 0 \leq j < n\}$, een natuurlijke basis voor P_{n+m} wordt gegeven door $\{x^k \mid 0 \leq k < n + m\}$. Ten opzichte van deze basissen wordt φ_0 voorgesteld door de volgende $(n + m) \times (n + m)$ matrix S

$$S = \begin{pmatrix} f_n & & & & g_m & & & & \\ f_{n-1} & f_n & & & g_{m-1} & g_m & & & \\ \vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & \\ \vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & \\ \vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \\ \vdots & \vdots & & \vdots & & g_0 & & & g_m \\ f_0 & \vdots & & \vdots & & & \ddots & & \vdots \\ & f_0 & & \vdots & & & & \ddots & \vdots \\ & & \ddots & \vdots & & & & & \ddots & \vdots \\ & & & f_0 & & & & & & g_0 \end{pmatrix}. \tag{3.4}$$

De eerste m kolommen van S bestaan uit f_j's, laatste n kolommen van S bestaan uit g_j's. De posities van S buiten de "parallellogrammen" zijn allemaal nul. Stellen we nu

$$s = \sum_{j=0}^{m-1} y_j x^j,\ t = \sum_{j=0}^{n-1} z_j x^j,\ sf + tg = \sum_{j=0}^{n+m-1} u_j x^j\,,$$

met alle elementen $y_j, z_j, u_j \in F$, dan vinden we

$$\begin{pmatrix} u_{n+m-1} \\ \vdots \\ \vdots \\ \vdots \\ u_0 \end{pmatrix} = S \cdot \begin{pmatrix} y_{m-1} \\ \vdots \\ y_0 \\ z_{n-1} \\ \vdots \\ z_0 \end{pmatrix}. \tag{3.5}$$

Met deze definities kan Stelling 3.2.2 als volgt geformuleerd worden.

Gevolg 3.2.3. *Stel $f, g \in F[x] \setminus \{0\}$ twee polynomen van respectieve graad n en m.*

(i) $\gcd(f, g) = 1 \iff \det S \neq 0$

(ii) *Als* $\gcd(f, g) = 1$, $n+m \geq 1$, *en* $y_0, \cdots, y_{m-1}, z_0, \cdots, z_{n-1} \in F$ *voldoen aan vergelijking* (3.5) *voor* $u_0 = 1$ *en* $u_i = 0$, $i \neq 0$, *dan zijn* $s_l = \sum_{i=0}^{m-1} y_i x^i$ *en* $t_l = \sum_{i=0}^{n-1} z_i x^i$ *de Bézout coëfficiënten berekend met het uitgebreid algoritme van Euclides, en* $s_l f + t_l g = 1$.

Voor polynomen over een ring R wordt de matrix S de *Sylvester matrix* genoemd, genoteerd $\mathrm{Syl}(f, g)$. De *resultante* van f en g, genoteerd $\mathrm{res}(f, g)$ is de determinant van $\mathrm{Syl}(f, g)$. Als $n = m = 0$, dan stellen we $\mathrm{res}(f, g)$ gelijk aan 1. Verder is het ook gebruikelijk om $\mathrm{res}(f, 0) = \mathrm{res}(0, f) = 0$ te definiëren als f nul is of niet constant en $\mathrm{res}(f, 0) = \mathrm{res}(0, f) = 1$ als f een niet nul constante is. Met deze definitie geldt Gevolg 3.2.3 voor alle paren polynomen.

Voorbeeld 3.2.4. Stel $f = r_0 = x^4 - 3x^3 + 2x$ en $g = r_1 = x^3 - 1$, twee polynomen over $\mathbb{Q}$. Uitvoering van het uitgebreid algoritme van Euclides levert

$$\begin{aligned} r_0 &= q_1 r_1 + \rho_2 r_2 = (x-3) r_1 + 3(x-1), \\ r_1 &= q_2 r_2 = (x^2 + x + 1) r_2 \end{aligned}$$

Dus $\gcd(f, g) = r_2 = x - 1$ (zowel in $\mathbb{Z}[x]$ als in $\mathbb{Q}[x]$). De Sylvester matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -3 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & -3 & -1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

dus $\operatorname{res}(f,g) = \det \operatorname{Syl}(f,g) = 0$. Beide polynomen delen door hun gcd levert uiteraard twee polynomen die copriem zijn. Met $r_0 = f/\gcd(f,g) = x^3 - 2x^2 - 2x$ en $r_1 = g/\gcd(f,g) = x^2 + x + 1$ hebben we

$$\begin{aligned} r_0 &= q_1 r_1 + r_2 = (x-3)r_1 + 3 \cdot 1\,, \\ r_1 &= q_2 r_2 = (x^2 + x + 1)r_2 \end{aligned}$$

en de Sylvester matrix van r_0 en r_1 is

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ -2 & 1 & 1 & 1 & 0 \\ -2 & -2 & 1 & 1 & 1 \\ 0 & -2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

waaruit $\operatorname{res}(r_0, r_1) = 9$.

Gevolg 3.2.5. *Stel F een veld, en $f, g \in F[x] \setminus \{0\}$ twee polynomen over F. De volgende uitspraken zijn equivalent:*

(i) $\gcd(f,g) = 1$*;*

(ii) $\operatorname{res}(f,g) \neq 0$*;*

(iii) *er bestaan geen polynomen $s, t \in F[x] \setminus \{0\}$ zodat*

$$sf + tg = 0, \quad \deg s < \deg g, \quad \deg t < \deg f\,.$$

Gevolg 3.2.6. *Stel $F \subset K$ zijn twee velden. Stel $f, g \in F[x]$ en $h \in K[x]$. Stel dat h niet constant is en f en g deelt, dan bestaat er ook een niet constant polynoom in $F[x]$ dat f en g deelt.*

Bewijs. Aangezien f, g een niet-triviale gcd hebben, geldt $\operatorname{res}(f,g) = 0$. Maar aangezien f, g ook tot $F[x]$ behoren, volgt hieruit onmiddellijk dat f, g ook een niet-triviale gcd hebben in $F[x]$. □

Deze situatie contrasteert een beetje met het feit dat het mogelijk is dat een polynoom $f \in F[x]$ een niet constante factor van graad kleiner dan $\deg f - 1$ kan hebben over K maar niet over F. Een eenvoudig voorbeeld is het polynoom $x^2 - 2$, dat irreduciebel is over $\mathbb{Q}$ maar niet over $\mathbb{R}$.

Gevolg 3.2.7. *Stel dat R een domein is en $f, g \in R[x] \setminus \{0\}$, met $\deg f + \deg g \geq 1$. Dan bestaan er polynomen $s, t \in R[x] \setminus \{0\}$ zodat $sf + tg = \operatorname{res}(f,g)$, $\deg s < \deg g$, en $\deg t < \deg f$.*

Bewijs. Stel dat F het breukenveld is van R. Als $r = \text{res}(f,g) = 0$, dan impliceert Gevolg 3.2.5 het bestaan van de polynomen $s, t \in F[x]$ zoals beschreven in de opgave. De uitspraak volgt door s en t te vermenigvuldigen met de gemeenschappelijke noemer van hun coëfficiënten. Als $r \neq 0$, dan zijn f en g copriem in $F[x]$, en door Steling 1.3.19 (ii) bestaan er twee polynomen $s^*, t^* \in F[x]$ die voldoen aan de voorwaarde op hun graad en waarvoor $s^*f + t^*g = 1$. Stelling 3.2.2 garandeert dat deze s^* en t^* de unieke oplossing zijn van een stelsel lineaire vergelijkingen met coëfficiëntenmatrix $S = \text{Syl}(f,g)$. De *regel van Cramer* impliceert dat zowel s^* en t^* het quotiënt zijn van een determinant van een matrix die ontstaat door de juiste kolom van S te vervangen en de determinant van S, dus $s = rs^*$ en $t = rt^*$ in $R[x]$. □

Gevolg 3.2.8. *Stel dat R een UFD is en $f, g \in R[x] \setminus \{0\}$. Dan is* $\gcd(f,g)$ *niet constant in $R[x]$ als en slechts als* $\text{res}(f,g) = 0$ *in R.*

Bewijs. Door Lemma 3.2.1 en Gevolg 3.2.7 □

3.3 Modulaire gcd algoritmen

In deze en de volgende secties houden we de volgende notaties aan. We beschouwen een domein R. Beschouw een ideaal $I \triangleleft R$. De reductie van een willekeurig element $f \in R[x]$ modulo I noteren we steeds als $\overline{f}$. Uiteraard gebruiken we dezelfde notatie als $I = (p)$, met $p \in R$ een priemelement.

Beschouw twee polynomen $f, g \in R[x]$, R een UFD. Stel $p \in R$ is een priemelement. Gevolg 3.2.8 stelt dat $\gcd(f,g) \in R$ (dus een constant polynoom) als en slechts als $\text{res}(f,g) \neq 0$. De resultante $\text{res}(f,g)$ is een polynomiale uitdrukking (afhankelijk van de graad van f en g) in de coëfficiënten van f en g. Men zou kunnen vermoeden dat $\overline{\text{res}(f,g)} = \text{res}(\overline{f}, \overline{g})$, dat dus $\overline{f}$ en $\overline{g}$ copriem zijn in $(R/(p))[x]$ als en slechts als $p \nmid \text{res}(f,g)$. Het volgende voorbeeld toont aan dat dit vermoeden niet correct is.

Voorbeeld 3.3.1. Stel $R = \mathbb{Z}$ en $p = 2$. Stel $f = x + 2$ en $g = x$. Dan $\text{res}(f,g) = -2 \neq 0$ en $\text{res}(\overline{f}, \overline{g}) = 0$, zoals verwacht. Stel nu $f = 4x^3 - x$ en $g = 2x + 1$, dan is $\text{res}(f,g) = 0$ en $\text{res}(\overline{f}, \overline{g}) = \text{res}(x, 1) = 1 \neq 0$. Dus $\overline{\text{res}(f,g)} \neq \text{res}(\overline{f}, \overline{g})$.

Het volgend lemma verklaart het voorbeeld en verschaft een mogelijkheid om het probleem te vermijden.

Lemma 3.3.2. *Beschouw een ring R, twee polynomen $f, g \in R[x] \setminus \{0\}$, en $I \triangleleft R$ een ideaal. Veronderstel dat $\overline{\text{lc}(f)}$ geen nuldeler is in R/I. Dan geldt*

(i) $\overline{\operatorname{res}(f,g)} = 0 \iff \operatorname{res}(\overline{f},\overline{g}) = 0$

(ii) *Als R/I een UFD is, dan is $\overline{r} = 0$ als en slechts als* $\gcd(\overline{f},\overline{g})$ *een niet constant polynoom is.*

Bewijs. We stellen $f = \sum_{j=0}^{n} f_j x^j$ en $g = \sum_{j=0}^{m} g_j x^j$ met $f_n \neq 0 \neq g_m$ en alle $f_j, g_j \in R$. Als $\deg f = 0$, dan zijn $\operatorname{Syl}(f,g)$, respectievelijk $\operatorname{Syl}(\overline{f},\overline{g})$ diagonaalmatrices met f_0, respectievelijk $\overline{f_0}$ op de hoofddiagonaal. Zowel $\overline{r}$ als $\operatorname{res}(\overline{f},\overline{g})$ zijn dan verschillend van nul, omdat $\overline{f_0^m}$ en $\overline{f_0}^m$ verschillend van nul zijn volgens de veronderstelling.

Stel nu dat $\deg f \geq 1$. Als $\overline{g} = 0$, dan is $\operatorname{res}(\overline{f},\overline{g}) = 0$ en elke kolom van g_j's in de matrix $\operatorname{Syl}(f,g)$ is nul modulo I, zodat $\overline{r} = 0$. We veronderstellen nu dat $\overline{g} \neq 0$, en stel i is de kleinste waarde waarvoor $\overline{g}_{m-i} \neq 0$. Beschouw de $(n+m-i) \times (n+m-i)$-deelmatrix van $\operatorname{Syl}(f,g)$

$$
S = \left(\begin{array}{ccc|ccc}
f_n & & & g_m & & \\
\vdots & \ddots & & \vdots & \ddots & \\
f_{n-i} & & & f_n & & g_m \\
\vdots & & & \vdots & \ddots & \vdots \\
f_0 & & & \vdots & & \vdots \\
 & \ddots & & \vdots & & g_{m-i} \\
 & & & f_0 & & \vdots \\
 & & & & \ddots & \vdots \\
 & & & & & g_0
\end{array}\right)
\begin{array}{l}
\left.\begin{array}{c} \\ \\ \end{array}\right\} i \\
\left.\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} n+m-i
\end{array}
$$

$$
\underbrace{}_{i} \quad \underbrace{}_{m-i} \quad \underbrace{}_{n}
$$

Deze matrix, modulo I, is $\operatorname{Syl}(\overline{f},\overline{g})$. Alle g_j in de eerste i rijen zijn nul modulo I, en herhaalde Laplace expansie van $r = \det \operatorname{Syl}(f,g)$ langs de eerste rij impliceert dat $\overline{r} = \overline{f_n^i} \operatorname{res}(\overline{f},\overline{g})$. Dit toont (i) aan. Deel (ii) van de stelling volgt door Gevolg 3.2.8. □

Het is duidelijk dat de conclusie niet opgaat als beide leidende coëfficiënten van f en g nul zijn modulo I. Dit is exact wat zich voordoet in Voorbeeld 3.3.1.

Stelling 3.3.3. *Stel dat R een Euclidisch domein is, $p \in R$ priem, en $f, g \in R[x] \setminus \{0\}$. Noteer $h = \gcd(f,g) \in R[x]$, $e = \deg h$, $\alpha = \operatorname{lc}(h)$, $e^* = \deg \gcd(\overline{f},\overline{g})$, en veronderstel dat p geen deler is van $b = \gcd(\operatorname{lc}(f), \operatorname{lc}(g)) \in R$. Dan geldt*

(i) $\alpha \mid b$;

(ii) $e^* \geq e$;

(iii) $e^* = e \iff \overline{\alpha} \cdot \gcd(\overline{f}, \overline{g}) = \overline{h} \iff p \nmid \operatorname{res}(f/h, g/h)$ *in* R.

Bewijs. Omdat h een deler is van f en g in $R[x]$, is $\operatorname{lc}(h)$ een deler van $\operatorname{lc}(f)$ en $\operatorname{lc}(g)$ in R, waaruit (i) volgt. Stel $u = f/h$ en $v = g/h$. Dan is $\deg \overline{h} = e$ door (i) en $p \nmid b$. De vergelijkingen

$$\overline{u}\overline{h} = \overline{f} \quad \text{en} \quad \overline{v}\overline{h} = \overline{g} \tag{3.6}$$

impliceren dat $\overline{h}$ een deler is van $\gcd(\overline{f}, \overline{g})$, hetgeen (ii) aantoont en de eerste equivalentie in (iii). (Merk immers op dat over een veld zoals $R/(p)$ altijd een monische gcd gekozen wordt).

Nu impliceert $p \nmid b$ dat p geen deler is van zowel $\operatorname{lc}(u)$ als $\operatorname{lc}(v)$. Zonder aan de algemeenheid te schaden mogen we veronderstellen dat $p \nmid \operatorname{lc}(u)$. Lemma 3.3.2 (ii) impliceert dat $p \mid \operatorname{res}(u, v)$ als en slechts als $\gcd(\overline{u}, \overline{v}) \neq 1$ in $R/(p)$. Vergelijking (3.6) impliceert dat $\gcd(\overline{f}, \overline{g}) = \gcd(\overline{u}, \overline{v}) \cdot \overline{h}/\overline{\alpha}$, hetgeen de tweede equivalentie uit (iii) oplevert. □

Voorbeeld 3.3.4. Beschouw, zoals in Voorbeeld 3.1.9, $R = \mathbb{Z}$ en $f = 18x^3 - 42x^2 + 30x - 6$, $g = -12x^2 + 10x - 2$. Stel $p = 17$. Dan is $h = \gcd(f, g) = 6x - 2$, dus $e = 1$ en $p \nmid \operatorname{lc}(h)$. Verder is $f/h = 3x^2 - 6x + 3$ en $g/h = -2x + 1$, en

$$\operatorname{res}\left(\frac{f}{h}, \frac{g}{h}\right) = \det \begin{pmatrix} 3 & -2 & 0 \\ -6 & 1 & -2 \\ 3 & 0 & 1 \end{pmatrix} = 3 \not\equiv 0 \bmod 17\,,$$

waaruit we besluiten dat $\deg \gcd(\overline{f}, \overline{g}) = 1$. Merk op dat $\gcd(\overline{f}, \overline{g}) = x + 11$ in $\mathsf{GF}(17)[x]$, en $\overline{\gcd(f, g)} = 6 \cdot \gcd(\overline{f}, \overline{g})$.

3.4 Een big prime gcd algoritme voor $F[x, y]$ en $\mathbb{Z}[x]$

In deze sectie beschrijven we een eerste modulair algoritme voor de gcd in de ringen $F[x, y]$, F een veld en $\mathbb{Z}[x]$. Het algoritme dat we beschrijven bepaalt enkel de gcd van twee elementen. Indien we ook geïnteresseerd zijn om de Bézout coëfficiënten te bepalen, dan moeten we opnieuw de theoretische concepten uit dit hoofdstuk verder uitdiepen.

We starten met de beschrijving van een modulair algoritme voor $F[x, y]$, F een veld. Men zou kunnen stellen dat om tot een resultaat te bekomen

via een modulaire redenering, we een priemelement p moeten hebben dat aan twee voorwaarden voldoet. Het gekozen priemelement p moet "groot" genoeg zijn, zodat alle coëfficiënten of de gcd kunnen gevonden worden uit hun reductie modulo p. Het gekozen priemelement mag ook geen deler zijn van de leidende coëfficiënt van de gcd, omdat anders de graad van de gcd verandert bij reductie modulo p. Het feit dat het priemgetal p groot genoeg moet zijn, is de aanleiding om dit soort modulaire algoritmen "big prime" modulaire algoritmen te noemen.

Het algoritme dat we zullen bespreken in deze sectie is probabilistisch. De strategie is om een willekeurig priemelement zo te kiezen dat de berekening van de coëfficiënten uit hun gereduceerde waarde altijd mogelijk is, en dat de niet deelbaarheidsvereiste vervuld is *met een hoge probabiliteit*.

Algoritme 3.3 Big prime modulair gcd algoritme voor $F[x, y]$

input: Primitieve polynomen $f, g \in F[x, y] = R[x]$, met $R = F[y]$, F een veld, $\deg_x f = n \geq \deg_x g \geq 1$ en $\deg_y f$, $\deg_y g \leq d$.
output: $h = \gcd(f, g) \in R[x]$.

1 $b \leftarrow \gcd(\mathrm{lc}_x(f), \mathrm{lc}_x(g)) \in R$
2 **repeat**
3 Kies een random monisch irreduciebel polynoom $p \in R$, $\deg p = d + 1 + \deg b$.
4 $\overline{f} \leftarrow f \mod p$, $\overline{g} \leftarrow g \mod p$
5 Bepaal de monische $v = \gcd(\overline{f}, \overline{g}) \in (R/\langle p \rangle)[x]$,
6 Bepaal $w, f^*, g^* \in R[x]$ zodat $w \equiv bv \mod p$, $f^* w \equiv bf \mod p$, $g^* w \equiv bg \mod p$
7 **until** $\deg_y(f^* w) = \deg_y(bf)$ en $\deg_y(g^* w) = \deg_y(bg)$
8 **return** $\mathrm{pp}_x(w)$

In principe zijn ook de polynomen f/h en g/h berekend door het algoritme, het volstaat daartoe op het einde $\mathrm{pp}_x(f^*)$ en $\mathrm{pp}_x(g^*)$ terug te geven. Voor de berekeningen in Lijn (6) gebeuren, wordt er eerst gecontroleerd of de constante coëfficiënten van w een deler zijn van de constante coëfficiënten van bf en bg. Als deze test faalt, dan keren we terug naar Lijn (3). De polynomen f^* en g^* kunnen berekend worden als $f^* \equiv f/v \mod p$ en $g^* \equiv g/v \mod p$.

Om tenslotte de gcd van niet primitieve polynomen te bepalen, bepalen we eerst de gcd van hun contents, en nadien, met bovenstaand algoritme, de gcd van hun primitief deel.

Stelling 3.4.1. *Stel $R = F[y]$, $f, g \in R[x]$ en $h = \gcd(f, g) \in R[x]$, en $r = \mathrm{res}_x(f/h, g/h) \in R$. De stopvoorwaarde in Lijn (7) van Algoritme 3.3 geldt als en slechts als $p \nmid r$, en dan is $\mathrm{pp}_x(w) = h$.*

Bewijs. Omdat $\deg_y b < \deg_y p$, geldt dat $p \nmid b$, een nodige voorwaarde om Stelling 3.3.3 te kunnen gebruiken. Verder geldt dat $\gcd(f/h, g/h) = 1$, dus $r \neq 0$ door Gevolg 3.2.8.

We veronderstellen eerst dat $p \nmid r$. Met $\alpha = \mathrm{lc}(h) \in R$, volgt uit Stelling 3.3.3 (iii) dat $\overline{\alpha} v = \overline{h}$. Ook geldt dat $\alpha \mid b = \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, dus $\deg \alpha \leq \deg b < \deg p$, dus $\overline{\alpha} = \alpha$ en $\alpha v \equiv h \bmod p$. Dus $bv \equiv (b/\alpha)h \bmod p$. De berekening van $w := bv$ gebeurt in $R/(p)$. Beschouwen we w als een polynoom in $R[x]$, dan geldt $\deg_y(w) < \deg p$. Anderzijds geldt door de keuze van p dat $\deg_y((bh)/\alpha) < \deg p$, dus $w = (bh)/\alpha)$. Ook de bepaling van f^* en g^* gebeurt in $R/(p)$. Aangezien echter $w = (bh)/\alpha$, zal $f^* = \alpha f/h$ en $g^* = \alpha g/h$. Dus ook in $R[x]$ gelden de gelijkheden $f^* w = bf$ en $f^* w = bg$, en dus aan de voorwaarden in Lijn (7) is voldaan. Door Gevolg 3.1.8 (ii) is h primitief, dus $\mathrm{pp}_x(w) = \mathrm{pp}_x((b/\alpha)h) = h$ is de gezochte gcd.

We veronderstellen nu dat $p \mid r$. Dan volgt uit Stelling 3.3.3 dat $\deg_x w = \deg_x v > \deg_x h$. Als de graadvoorwaarden in Lijn (7) vervuld zijn na berekening van w, f^* en g^*, dan zijn de congruenties in Lijn (6) gelijkheden, en dan is $\mathrm{pp}_x(w)$ een gemeenschappelijke deler van f en g van graad in x groter dan $\deg_x h$, een contradictie. Daarmee is het gestelde aangetoond. □

Om de complexiteit van het algoritme te analyseren, nemen we zonder bewijs het volgende resultaat aan.

Stelling 3.4.2. *Stel $R = F[y]$, $p \in R$ en $\deg p \geq 1$. De kost van een optelling of een vermenigvuldiging in $R/(p)$ bedraagt $\mathcal{O}((\deg p)^2)$ operaties in F.*

Zonder bewijs. □

Lemma 3.4.3. *De kost voor een uitvoering van de repeat lus van Algoritme 3.3 bedraagt $\mathcal{O}(n^2 d^2)$ operaties in F. Lijnen (1) en (8) vergen $\mathcal{O}(nd^2)$ operaties in F.*

Bewijs. Het reduceren van f en g modulo p is computationeel gezien overbodig. De kost van het uitvoeren van het uitgebreid algoritme van Euclides is uitgedrukt in bewerkingen in $R/(p)$. Deze kost bedraagt $\mathcal{O}(n^2)$ operaties in $R/(p)$, (zie Stelling 1.3.13). Uit Stelling 3.4.2 en $\deg p = d+1+\deg b \leq 2d+1$ volgt dat Lijn (5) $\mathcal{O}(n^2 d^2)$ operaties vergt in F, Lijnen (1) en (8) vergen ten hoogste $n+1$ gcd berekeningen en delingen van polynomen in $F[y]$ van graad ten hoogste $2d$, of $\mathcal{O}(nd^2)$ operaties in F. □

Opmerking 3.4.4. Omdat we in Algoritme 3.3 geen verdere veronderstellingen maken over het veld F, is een verdere analyse van de complexiteit zinloos. We hebben immers de kost om een polynoom p te vinden (Lijn (3)) niet bepaald, en het valt te verwachten dat deze afhankelijk is van het veld F. Een gedetailleerde analyse vindt met in [vzGG03], waar onder andere de kost besproken wordt om het polynoom p te bepalen met F een eindig veld.

Om Algoritme 3.3 aan te passen voor $\mathbb{Z}[x]$ moeten we eigenlijk een grens kunnen bepalen voor de coëfficiënten van h. We starten met de uitbreiding van de 2-norm voor complexe polynomen. Stel $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$. Dan definiëren we

$$\|f\|_2 := (\sum_{i=0}^{n} |f_i|^2)^{(1/2)} ,$$

met $|\cdot|$ de gekende modulus in $\mathbb{C}$. Het is de bedoeling om een norm voor alle factoren van f af te leiden in functie van $\|f\|_2$.

Lemma 3.4.5. *Voor een polynoom $f \in \mathbb{C}[x]$ en $z \in \mathbb{C}$, geldt $\|(x-z)f\|_2 = \|(\overline{z}x - 1)f\|_2$.*

Bewijs. Stel $f = \sum_{i=0}^{n} f_i x^i$ en stel $f_{-1} = f_{n+1} = 0$, dan is

$$\begin{aligned}
\|(x-z)f\|_2^2 &= \sum_{i=0}^{n+1} |f_{i-1} - z f_i|^2 = \sum_{i=0}^{n+1} (f_{i-1} - z f_i)(\overline{f_{i-1}} - \overline{z}\,\overline{f_i}) \\
&= \|f\|_2^2 (1 + |z|^2) - \sum_{i=0}^{n+1} (z\overline{f_{i-1}} f_i + \overline{z} f_{i-1} \overline{f_i}) \\
&= \sum_{i=0}^{n+1} (\overline{z} f_{i-1} - f_i)(z\overline{f_{i-1}} - \overline{f_i}) = \sum_{i=0}^{n+1} |\overline{z} f_{i-1} - f_i|^2 \\
&= \|(\overline{z}x - 1)f\|_2^2 .
\end{aligned}$$

□

Stel $f = \sum_{i=0}^{n} f_i x^i = f_n \prod_{i=1}^{n} (x - z_i)$, met $f_0, \cdots, f_n, z_1, \cdots, z_n \in \mathbb{C}$. We definiëren

$$M(f) = |f_n| \cdot \prod_{i=1}^{n} \max\{1, |z_i|\} ,$$

en merken op dat $M(f) \geq |\operatorname{lc}(f)|$ en $M(f) = M(g)M(h)$ als $f = gh$ voor $g, h \in \mathbb{C}[x]$. In de volgende stelling beschrijven we het verband tussen $M(f)$ en $\|f\|_2$.

Stelling 3.4.6 (De ongelijkheid van Landau (1905))**.** *Voor elk polynoom $f \in \mathbb{C}[x]$ geldt $M(f) \leq \|f\|_2$.*

Bewijs. Nummer de wortels van f zodanig dat $|z_1|, |z_2|, \ldots, |z_k| > 1$ en $|z_{k+1}|, \ldots, |z_n| \leq 1$ voor een zekere $k \in \{0, \ldots, n\}$, Daaruit volgt dat $M(f) = |f_n \cdot z_1 \cdots z_k|$. Stel

$$g = f_n \prod_{i=1}^{k} (\overline{z_i}x - 1) \prod_{i=k+1}^{n} (x - z_i) = g_n x^n + \cdots g_0 \in \mathbb{C}[x]\,.$$

Dan geldt

$$\begin{aligned} M(f)^2 = |f_n \overline{z_1} \cdots \overline{z_k}|^2 = |g_n|^2 \leq \|g\|_2^2 = \left\| \frac{g}{\overline{z_1}x - 1}(x - z_1) \right\|_2^2 = \cdots \\ = \left\| \frac{g}{(\overline{z_1}x - 1) \cdots (\overline{z_k}x - 1)}(x - z_1) \cdots (x - z_k) \right\|_2^2 = \|f\|_2^2 \end{aligned}$$

waarbij we herhaaldelijk Lemma 3.4.5 gebruikt hebben. Daarmee is de stelling bewezen. □

In hetgeen volgt zullen we nog twee andere normen gebruiken voor polynomen. Stel $f \in \mathbb{C}[x]$. De *max-norm* van f is gedefinieerd als $\|f\|_\infty = \max\{|f_i| \mid 0 \leq i \leq n\}$. De 1-norm van f is gedefinieerd als $\|f\|_1 = \sum_{i=0}^{n} |f_i|$

Een gekende relatie tussen deze verschillende normen is de volgende:

$$\|f\|_\infty \leq \|f\|_2 \leq \|f\|_1 \leq (n+1)^{\frac{1}{2}} \|f\|_\infty.$$

Stelling 3.4.7. *Als het polynoom $h = \sum_{i=0}^{m} h_i x^i \in \mathbb{C}[x]$ een deler is van het polynoom $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$, $n \geq m$, dan geldt*

$$\|h\|_2 \leq \|h\|_1 \leq 2^m M(h) \leq \left| \frac{h_m}{f_n} \right| 2^m \|f\|_2\,.$$

Bewijs. We schrijven $h = h_m \prod_{i=1}^{m} (x - u_i)$ met $u_i \in \mathbb{C}$, $1 \leq i \leq m$. Merk op dat omdat $h \mid f$, elke wortel u_i gelijk is aan een wortel z_j van f. De coëfficiënten van h kunnen uitgedrukt worden in functie van de wortels, we noteren S_{m-i}^m de verzameling van alle deelverzamelingen van kardinaliteit $m - i$ van $\{1, \ldots, m\}$:

$$h_i = (-1)^{m-i} h_m \sum_{S \in S_{m-i}^m} \prod_{j \in S} u_j\,,$$

waarbij de som het $(m-i)$-de elementair symmetrisch polynoom in $u_1, \cdots, u_m$ is. Bijgevolg is

$$|h_i| \leq |h_m| \cdot \sum_{S} \prod_{j \in S} |u_j| \leq \binom{m}{i} M(h)$$

voor alle $0 \leq i \leq m$. Dus

$$\|h\|_2 \leq \|h\|_1 = \sum_{i=0}^{m} |h_i| \leq 2^m M(h) \leq \left|\frac{h_m}{f_n}\right| 2^m M(f) \leq \left|\frac{h_m}{f_n}\right| 2^m \|f\|_2 \,,$$

door de som formule voor de binomiaal coëfficiënten en de ongelijkheid van Landau. □

Gevolg 3.4.8 (Mignotte's grens)**.** *Veronderstel dat $f, g, h \in \mathbb{Z}[x]$ respectieve graden n, m en k hebben, $n \geq 1$, en dat $gh \mid f$ in $\mathbb{Z}[x]$. Dan geldt*

(i) $\|g\|_\infty \|h\|_\infty \leq \|g\|_2 \|h\|_2 \leq \|g\|_1 \|h\|_1 \leq 2^{m+k}\|f\|_2 \leq (n+1)^{1/2} 2^{m+k} \|f\|_\infty$,

(ii) $\|h\|_\infty \leq \|h\|_2 \leq 2^k \|f\|_2 \leq 2^k \|f\|_1$ *en* $\|h\|_\infty \leq \|h\|_2 \leq (n+1)^{1/2} 2^k \|f\|_\infty$.

Bewijs. Stelling 3.4.7 en de ongelijkheid van Landau impliceren

$$\|g\|_1 \|h\|_1 \leq 2^{m+k} M(g) M(h) \leq 2^{m+k} M(f) \leq 2^{m+k} \|f\|_2 \,.$$

Het tweede gedeelte van de uitspraak volgt door $g = 1$ te stellen. □

Gevolg 3.4.9. *Stel dat $f, g \in \mathbb{Z}[x]$, $\deg f, \deg g \leq n$ en $\|f\|_\infty, \|g\|_\infty \leq A$. Stel $h = \gcd(f, g)$, dan is $\|h\|_\infty \leq (n+1)^{1/2} 2^n A$*

Bewijs. Pas Gevolg 3.4.8 (ii) toe, en gebruik dat $\deg h \leq \deg f$. □

Het volgende algoritme is het analogon van Algoritme 3.3 voor $\mathbb{Z}[x]$.

Algoritme 3.4 Big prime modulair gcd algoritme voor $\mathbb{Z}[x]$.

input: Primitieve polynomen $f, g \in \mathbb{Z}[x]$, $\deg f = n \geq \deg g \geq 1$ en $\|f\|_\infty, \|g\|_\infty \leq A$.
output: $h = \gcd(f, g) \in \mathbb{Z}[x]$.

1 $b \leftarrow \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, $B \leftarrow (n+1)^{1/2} 2^n A b$
2 **repeat**
3 Kies een random priemgetal $p \in \mathbb{N}$, $2B + 1 < p \leq 4B + 2$
4 $\overline{f} \leftarrow f \mod p$, $\overline{g} \leftarrow g \mod p$
5 Bepaal de monische $v = \gcd(\overline{f}, \overline{g}) \in \mathsf{GF}(p)[x]$, met $\|v\|_\infty < p/2$
6 Bepaal $w, f^*, g^* \in \mathbb{Z}[x]$ met max-norm kleiner dan $p/2$ zodat $w \equiv bv \mod p$, $f^* w \equiv bf \mod p$, $g^* w \equiv bg \mod p$
7 **until** $\|f^*\|_1 \|w\|_1 \leq B$ en $\|g^*\|_1 \|w\|_1 \leq B$
8 **return** $\mathrm{pp}(w)$

De correctheid van het algoritme wordt in de volgende stelling aangetoond. Het bewijs is in feite volkomen analoog aan het bewijs van Stelling 3.4.1. Ook de analyse van de complexiteit is vergelijkbaar.

Stelling 3.4.10. *Stel $f, g \in \mathbb{Z}[x]$, $h = \gcd(f, g)$ en $r = \operatorname{res}(f/h, g/h)$. De stopvoorwaarde in Lijn (7) van Algoritme 3.4 geldt als en slechts als $p \nmid r$, en dan is* $\operatorname{pp}(w) = h$.

Bewijs. Omdat $p > b$, geldt dat $p \nmid b$, een nodige voorwaarde om Stelling 3.3.3 te kunnen gebruiken. Verder geldt dat $\gcd(f/h, g/h) = 1$, dus $r \neq 0$ door Gevolg 3.2.8.

We veronderstellen eerst dat $p \nmid r$. Met $\alpha = \operatorname{lc}(h) \in \mathbb{Z}$, volgt uit Stelling 3.3.3 (iii) dat $\alpha v \equiv h \bmod p$. Ook geldt dat $\alpha \mid b = \gcd(\operatorname{lc}(f), \operatorname{lc}(g))$, en dus $bv \equiv (b/\alpha)h \bmod p$. Door de keuze van p en Gevolg 3.4.9 geldt dat $\|((bh)/\alpha)\|_\infty < \frac{p-1}{2}$. Daarenboven gelden de normvoorwaarden op Lijn (7) door Gevolg 3.4.8 (i), waardoor de equivalenties in Lijn (6) gelijkheden zijn. Door Gevolg 3.1.8 (ii) is h primitief, dus $\operatorname{pp}(w) = \operatorname{pp}((b/\alpha)h) = h$ is de gezochte gcd.

We veronderstellen nu dat $p \mid r$. Dan volgt uit Stelling 3.3.3 dat $\deg w = \deg v > \deg h$. Stel nu dat de normvoorwaarden in Lijn (7) vervuld zijn na berekening van w, f^* en g^*. dan geldt $\|f^* w\|_\infty \leq \|f^* w\|_1 \leq \|f^*\|_1 \|w\|_1 \leq B < p/2$, en $\|bf\|_\infty < p/2$, en analoog voor $\|g^* w\|_\infty$, waardoor de equivalenties op Lijn (6) gelijkheden zijn. Dus $\operatorname{pp}(w)$ is een gemeenschappelijke deler van f en g in $\mathbb{Z}[x]$ met $\deg \operatorname{pp}(w) > \deg h$, a contradictie. □

Lemma 3.4.11. *De kost voor een uitvoering van de repeat lus van Algoritme 3.4 bedraagt $\mathcal{O}(n^2(n^2 + \log^2 A))$ woordoperaties. Lijnen (1) en (8) vergen $\mathcal{O}(n^2(n^2 + \log^2 A))$ woordoperaties.*

Bewijs. Lijn (5) vergt $\mathcal{O}(n^2)$ aritmetische operaties in $\mathsf{GF}(p)$, elk van deze operaties vraagt $\mathcal{O}(\log^2 p)$ woordoperaties. Nu is $\log p \leq \log(4B) \in \mathcal{O}(n + \log A)$, waaruit we besluiten dat Lijn (5) $\mathcal{O}(n^2(n^2 + \log^2 A))$ woordoperaties vergt; hetzelfde geldt voor de delingen in Lijn (6). Lijnen (1) en (8) tenslotte vergen elk $\mathcal{O}(n)$ gcd berekeningen en delingen met gehele getallen van lengte $\mathcal{O}(n + \log A)$, of dus $\mathcal{O}(n(n^2 + \log^2 A))$ woordoperaties. □

Opmerking 3.4.12. Ook hier hebben we de kost in Lijn (3) niet bepaald. Men kan echter aantonen dat in minder dan $\mathcal{O}(n^2(n^2 + \log^2 A))$ woordoperaties een priemgetal $p \in \mathbb{Z}$ kan gevonden, zodat $2B < p \leq 4B$, en dat met waarschijnlijkheid van minstens de helft geen deler is van r. Het verwacht aantal uitvoeringen van de repeat lus is dan hoogstens twee, en met verwaarlozing van de logaritmische factoren, kan het volledige algoritme dus uitgevoerd worden in $\mathcal{O}(n^4)$ woordoperaties.

3.5 Een small prime gcd algoritme voor $\mathbb{Z}[x]$

In tegenstelling tot big prime modulaire algoritmen, zullen small prime modulaire algoritmen niet één "groot" priemelement gebruiken maar wel *meerdere "kleine"* priemelementen. De voordelen van een big prime algoritme blijven behouden, namelijk de reductie van het probleem naar een Euclidisch domein. De reconstructie van het resultaat in de originele UFD moet nu aangepast worden. In het geval $R = \mathbb{Z}[x]$ beschikken we over uitstekend gereedschap om deze reconstructie te doen: de Chinese reststelling en het bijhorend algoritme.

Het voorbereidend werk uit de vorige secties stelt ons in staat om onmiddellijk het volgend algoritme te beschouwen dat de gcd van twee polynomen $f, g \in \mathbb{Z}[x]$ bepaalt.

Algoritme 3.5 Small prime modulair gcd algoritme voor $\mathbb{Z}[x]$.

---

input: Primitieve polynomen $f, g \in \mathbb{Z}[x]$, $\deg f = n \geq \deg g \geq 1$ en $\|f\|_\infty, \|g\|_\infty \leq A$.
output: $h = \gcd(f, g) \in \mathbb{Z}[x]$.

1 $b \leftarrow \gcd(\mathrm{lc}(f), \mathrm{lc}(g))$, $B \leftarrow (n+1)^{1/2} 2^n A b$, $l \leftarrow \lceil \log_2(2B+1) \rceil$
2 **repeat**
3 Kies een verzameling S met $2l$ priemgetallen $p \in \mathbb{N}$
4 $S \leftarrow \{p \in S \| p \nmid b\}$, $\overline{f} \leftarrow f \mod p$, $\overline{g} \leftarrow g \mod p$,
Bepaal voor elke $p \in S$ de monische $v_p = \gcd(\overline{f}, \overline{g}) \in \mathsf{GF}(p)[x]$
5 $e \leftarrow \min\{\deg v_p \| p \in S\}$, $S \leftarrow \{p \in S \| \deg v_p = e\}$
if $|S| > l$, verwijder dan $|S| - l$ elementen uit S **else** ga naar Lijn (3)
6 Bepaal voor elke $p \in S$ polynomen $w_p, f_p^*, g_p^* \in \mathbb{Z}[x]$ zodat
$w_p \equiv b v_p \mod p$, $f_p^* w_p \equiv bf \mod p$, $g_p^* w_p \equiv bg \mod p$, voor alle $p \in S$
7 Gebruik Algoritme 1.7 om elke coëfficiënt van de unieke polynomen
$w, f^*, g^* \in \mathbb{Z}[x]$ te bepalen met max-norm kleiner dan $(\prod_{p \in S} p)/2$ en
$w \equiv w_p \mod p$, $f^* \equiv f_p^* \mod p$, $g^* \equiv g_p \mod p$, voor alle $p \in S$
8 **until** $\|f^*\|_1 \|w\|_1 \leq B$ en $\|g^*\|_1 \|w\|_1 \leq B$
9 **return** $\mathrm{pp}(w)$

De volgende stelling is vergelijkbaar met Stelling 3.4.10 en toont de correctheid van het algoritme aan.

Stelling 3.5.1. *Stel $f, g \in \mathbb{Z}[x]$, $h = \gcd(f, g)$ en $r = \mathrm{res}(f/h, g/h)$. De stopvoorwaarde in Lijn (7) van Algoritme 3.5 geldt als en slechts als* $\mathrm{pp}(w) =$

h

Bewijs. De verzameling S bevat, na voltooiing van Lijn (5), met eventueel één of meerdere sprongen naar Lijn (3), juist l priemgetallen. Voor elk priemgetal $p \in S$ geldt dat $\deg v_p = e$. Dus ofwel geldt voor elke $p \in S$ dat $p \nmid r$, ofwel dat $p \mid r$, door Stelling 3.4.10 (iii) en de constructie van de verzameling. Door de keuze van de priemgetallen geldt $\|bf\|_\infty < (\prod_{p\in S} p)/2$. De met Algoritme 1.7 bepaalde polynomen w, f^*, g^* voldoen aan de equivalenties

$$f^*w \equiv bf \mod (\prod_{p\in S} p) \quad \text{en} \quad g^*w \equiv bg \mod (\prod_{p\in S} p) \tag{3.7}$$

Stel nu dat de normvoorwaarden in Lijn (8) vervuld zijn voor de berekende w, f^* en g^*. Dan geldt $\|f^*w\|_\infty \leq \|f^*w\|_1 \leq \|f^*\|_1\|w\|_1 \leq B < (\prod_{p\in S} p)/2$, en analoog voor $\|g^*w\|_\infty < (\prod_{p\in S} p)/2$. Bovenstaande equivalenties worden dan gelijkheden, en $w \in \mathbb{Z}[x]$ is dus een gemeenschappelijke factor van f en g. Dus $\deg w \leq \deg h$. Opdat $\deg w < \deg h$ zou zijn, moeten alle v_p graad kleiner dan h hebben, een contradictie met Stelling 3.4.10 (ii). Dus $\deg w = \deg h$. Dus w en h verschillen hoogstens gehele factor van elkaar. Door Gevolg 3.1.8 (ii) is h primitief, dus $\mathrm{pp}(w) = h$ is de gezochte gcd.

Omgekeerd, stel nu dat $\mathrm{pp}(w) = h$. Merk op dat alle v_p monische polynomen zijn, dus $\mathrm{lc}(bv_p) \equiv b \mod p$. Daaruit volgt dat $\mathrm{lc}(w) = b$. Dus $\mathrm{pp}(w) = h$ impliceert dat $w \mid bf$ en $w \mid bg$. Uit Gevolg 3.4.8 (Mignotte's grens) volgt dat $\|bf/w\|_\infty \leq B < (\prod_{p\in S} p)/2$. Dus de equivalentie $f^* \equiv bf/w \mod (\prod_{p\in S} p)$, afgeleid uit de respectieve equivalentie 3.7 is in feite een gelijkheid $f^* = bf/w$ Volledig analoog vinden we de gelijkheid $g^* = bg/w$. Passen we opnieuw Gevolg 3.4.8 toe, dan vinden we de normvoorwaarden op Lijn (8). □

Wanneer men geschikte voorwaarden oplegt voor de keuze van de priemgetallen in S, kan men aantonen dat men de verzameling S kan opstellen in $\widetilde{\mathcal{O}}(n(n + \log A))$ woordoperaties.

Lemma 3.5.2. *Beschouw Algoritme 3.5. De uitvoering van Lijnen (4) tot en met (7) vergt $\mathcal{O}(n(n^2+\log^2 A)(\log n+\log\log A)^2)$ woordoperaties, evenveel als de uitvoering van Lijnen (1) en (8).*

Bewijs. Voor elk priemgetal $p \in S$ geldt $p \in \mathcal{O}(\log k)$. We schatten eerst het aantal operaties om Lijn (4) uit te voeren. Per priemgetal p zijn er $\mathcal{O}(n \log A \log k)$ woordoperaties nodig om b en alle coëfficiënten van f en g te reduceren modulo p. Om de gcd te bepalen zijn er $\mathcal{O}(n^2)$ operaties nodig in $\mathsf{GF}(p)$, dus $\mathcal{O}(n^2 \log^2 k)$ operaties in $\mathbb{Z}$. In totaal zijn er dus $\mathcal{O}(n(n \log k + \log A)l \log k)$ woord operaties nodig.

In Lijn (6) worden er eerst twee delingen met rest uitgevoerd (f/v_p en g/v_p) modulo p, hetgeen dus $\mathcal{O}(n^2 \log^2 k)$ woordoperaties vereist. De Chinese reststelling wordt toegepast op ten hoogste $2n+2$ coëfficiënten van de polynomen w, f^* en g^*. Elke toepassing van de Chinese reststelling vergt $\mathcal{O}(l^2 \log^2 k)$ woordoperaties (Stelling 1.4.4). Gebruiken we $\log(\prod_{p \in S} p) = \sum_{p \in S} \log p \in \mathcal{O}(l \log k)$, dan vinden we dat er in totaal $\mathcal{O}(nl^2 \log^2 k)$ woordoperaties nodig zijn.

Zoals in Lemma 3.4.11 is de kost voor Lijnen (1) en (8) $\mathcal{O}(n(n^2 + \log^2 A))$ woordoperaties. Het lemma volgt nu uit $l \in \mathcal{O}(n + \log A)$ en $\log k \in \mathcal{O}(\log n + \log \log A)$. □

Van Algoritme 3.5 kan, net zoals van Algoritme 3.4 een versie ontwikkeld worden voor de gcd in $F[x, y]$, F een veld. De keuze van een priemgetal $p \in \mathbb{Z}$ komt dan overeen met de keuze van een priemelement $p(y) \in F[y]$. Twee polynomen $f, g \in F[x, y]$ worden dan beschouwd in het Euclidisch domein $F[y]/(p)[x]$. Wanneer voor $p(y)$ een polynoom van de vorm $y - u$, $u \in F$ gekozen wordt, dan komt "reduceren modulo p" overeen met "evalueren in het punt $y = u$". De reconstructie via de Chinese reststelling komt dan overeen met interpolatie, waar er een breed gamma aan algoritmen beschikbaar voor is, met een vergelijkbare complexiteit als de Chinese reststelling.

De geïnteresseerde lezer verwijzen we naar [vzGG03].

3.6 Opmerkingen

Om de gcd van twee polynomen $f, g \in R$, met $R = \mathbb{Z}[x]$ of $R = F[x, y]$, F een veld, te berekenen beschikken we nu over vier verschillende algoritmen (waarbij we het small prime algoritme voor $F[x, y]$ niet in detail beschreven hebben). De volgende tabel geeft een overzicht van deze algoritmen en hun bijhorende complexiteit. De logaritmische factoren zijn hier telkens verwaarloosd. Voor elementen $f \in \mathbb{Z}[x]$ slaat n op de maximale graad van f en de maximale getallengte van de coëfficiënten. Voor elementen $f \in F[x, y]$ slaat n op de maximale graad in x en op de maximale graad in y van de coëfficiënten van f. De uitdrukking in n slaat op het aantal veldoperaties in F of het aantal woordoperaties.

Algoritme 3.1 gebruikt algoritme 1.6. Om de in de tabel gegeven complexiteit aan te tonen voor deze algoritmen, moet men aan tonen dat de getallengtes die optreden bij het uitvoeren van Algoritme 1.6 over $\mathbb{Q}$ beperkt zijn, en analoog voor de uitvoering over $F[x, y]$, F een veld. Ondanks deze aantoonbare grenzen is het nu duidelijk dat de modulaire algoritmen veel beter presteren. Ook zien we dat het small prime algoritme beter presteert

Algoritme	$\mathbb{Z}[x]$	$F[x,y]$	complexiteit
Euclides	Algoritme 3.1		n^6
primitief	Algoritme 3.2		n^6
big prime modulair	Algoritme 3.4	Algoritme 3.3	n^4
small prime modulair	Algoritme 3.5		n^3

Tabel 3.1: Vergelijkende tabel voor gcd algoritmen in $\mathbb{Z}[x]$ en $F[x,y]$

dan het big prime algoritme. Dit is volledig te wijten aan het sterk reduceren van de getallengtes over $\mathbb{Z}$ of de graden van de polynomen in $F[x,y]$. Wel dient vermeld te worden dat de reconstructie in het geval van een big prime algoritme wiskundig gezien eenvoudiger is, maar dat de winst van de berekeningen in het small prime algoritme niet teniet gedaan wordt omdat we beschikken over een efficiënt algoritme dat in dit geval de reconstructie kan uitvoeren (Chinese reststelling en interpolatie).

Naast big prime en small prime algoritmen bestaan er ook nog "prime power" algoritmen. In dergelijke algoritmen wordt, in plaats van één groot priemgetal, één klein priemgetal gekozen. Het product van de kleinere priemgetallen (dat groot genoeg moet zijn), wordt nu vervangen door een macht van het gekozen priemgetal. Kiest men de exponent groot genoeg, dan zijn opnieuw alle voorwaarden vervult die de reconstructie theoretisch mogelijk maken. Indien men op een efficiënte wijze de reconstructie kan uitvoeren, dan zal het bekomen algoritme net zo goed kunnen presteren als een small prime algoritme. We zullen dit illustreren in het *Hensel lifting* principe, wat toelaat een factorisatie van een polynoom te reconstrueren uit een factorisatie van datzelfde polynoom over een eindig veld.

Modulaire gcd algoritmen voor polynomen (in meer veranderlijken) over een UFD werden ontdekt door Collins en Brown ([Col71] en [Bro71]). Moses en Yun beschrijven een prime power modulair algoritme voor de gcd van polynomen in meer veranderlijken gebaseerd op het Hensel lifting principe ([MY73]).

De ontwikkeling van snellere algoritmen voor de gcd van gehele getallen wordt voor het eerst beschreven in [Leh38]. Aansluitende referenties die specifiek algoritmen beschrijven voor de gcd van gehele getallen zijn [Knu71], [Sch71] en [Moe73]. In [Ste67] wordt een recursief algoritme beschreven voor de bepaling van de gcd in $\mathbb{Z}$. De complexiteit van dit *binair algoritme van Euclides* wordt uitvoerig onderzocht in [Val98]. De uitgebreide versie van dit algoritme wordt beschreven in [Knu97]. Een variant van het binair algoritme van Euclides voor de ring $\mathbb{Z}[i]$ wordt beschreven in [Wei00].

Snellere algoritmen om de gcd van polynomen over een veld te bepalen

worden onder andere beschreven in [Sch80], [BGY80] en [Str83]. We vermelden de volgende stelling zonder bewijs.

Stelling 3.6.1. *Stel F een veld, $f, g \in F[x]$ monisch en $\deg f = n \geq \deg g$. Er bestaat een algoritme om $gcd(f, g)$ te bepalen dat $\mathcal{O}(\mathsf{M}(n) \log n)$ operaties in F vereist.*

Zonder bewijs. □

Ook voor de Chinese Reststelling kunnen snellere algoritmen ontwikkeld worden. We verwijzen naar [BM75] voor een overzicht van aan aantal resultaten die leiden naar een $\mathcal{O}(\mathsf{M}(n) \log n)$ algoritme bij dezelfde voorwaarden als Stelling 1.4.3.

De geïnteresseerde lezer verwijzen we verder naar [vzGG03].

Hoofdstuk 4

Priemtesten en factorisatie van gehele getallen

4.1 Priemtesten

We vragen ons af of een gegeven geheel getal priem is of niet. Uiteraard zouden we dit probleem kunnen beantwoorden door het getal te factoriseren. Maar dit is verre van efficiënt; één van de belangrijke vaststellingen in dit gebied is precies dat het veel eenvoudiger is om een priemtest uit te voeren dan te factoriseren. Men kan gehele getallen met vele duizenden digits controleren op het priem zijn, maar factorisatie van getallen met amper 300 digits is in het algemeen een onmogelijke opgave (althans met de huidige methoden). In dit deel zullen we een belangrijke *probabilistische* priemtest opstellen.

Definitie 4.1.1. Zij $N \in \mathbb{N}^*$, en beschouw $\mathbb{Z}/N\mathbb{Z}$, de ring van de gehele getallen modulo N.

(i) $(\mathbb{Z}/N\mathbb{Z})^\times := \{a \pmod N \in \mathbb{Z}/N\mathbb{Z} \mid \gcd(a, N) = 1\}$ is de multiplicatieve groep van de eenheden in $\mathbb{Z}/N\mathbb{Z}$;

(ii) $\varphi(N) := |(\mathbb{Z}/N\mathbb{Z})^\times|$; φ is de zogenaamde *Euler totiënt functie*.

(iii) Voor elke $a \in \mathbb{Z}$ met $\gcd(a, N) = 1$ definiëren we $\mathrm{ord}_N(a)$, de *orde van a modulo N*, als de orde van het element $a \pmod N$ in de groep $(\mathbb{Z}/N\mathbb{Z})^\times$. Met andere woorden, $n = \mathrm{ord}_N(a)$ is het kleinste positief natuurlijk getal n zodat $a^n \equiv 1 \pmod N$.

Stelling 4.1.2. *Zij $N \in \mathbb{N}^*$, en ontbind N in priemfactoren als $N = \prod_{i=1}^k p_i^{n_i}$, met p_i twee aan twee verschillende priemgetallen, en $n_i \in \mathbb{N}^*$. Dan is*

$$\varphi(N) = \prod_{i=1}^{k} p_i^{n_i - 1}(p_i - 1)\,.$$

Bewijs. Uit de Chinese reststelling 1.4.2 volgt dat

$$(\mathbb{Z}/N\mathbb{Z})^\times \cong (\mathbb{Z}/p_1^{n_1}\mathbb{Z})^\times \times \cdots \times (\mathbb{Z}/p_k^{n_k}\mathbb{Z})^\times\,,$$

en het is dus voldoende om de formule aan te tonen voor priemmachten $N = p^n$. Het aantal getallen in $\{0, \ldots, p^n - 1\}$ onderling ondeelbaar met p^n,

of equivalent, niet deelbaar door p, is precies $p^n \cdot \frac{p-1}{p} = p^{n-1}(p-1)$, en we bekomen de gezochte formule voor $\varphi(N)$. □

Stelling 4.1.3 (Stelling van Euler)**.** *Zij $N \in \mathbb{N}^*$, en $\gcd(a, N) = 1$. Dan is $a^{\varphi(N)} \equiv 1 \pmod{N}$.*

Bewijs. Dit volgt onmiddellijk uit de definitie van φ en uit de stelling van Lagrange, die zegt dat de orde van een element in een eindige groep steeds een deler is van de orde van de groep. □

Opmerking 4.1.4. In het bijzondere geval dat $N = p$ priem is, bekomen we $a^{p-1} \equiv 1 \pmod{p}$; dit staat bekend als de *kleine stelling van Fermat.*

Lemma 4.1.5. *Zij $N \in \mathbb{N}$ met $n \geq 2$. Als $a \in \mathbb{Z}$ copriem is met N en $k \in \mathbb{N}$ zo dat $a^k \equiv 1 \pmod{N}$, dan is k een veelvoud van $\mathrm{ord}_N(a)$. In het bijzonder is $\mathrm{ord}_N(a) \mid \varphi(N)$.*

Bewijs. Dit volgt onmiddellijk door alle beweringen te vertalen naar de groep $(\mathbb{Z}/N\mathbb{Z})^\times$. □

We beschouwen nu de Fermat test, een eenvoudige probabilistische priemtest die gebaseerd is op de kleine stelling van Fermat.

Algoritme 4.1 Fermat test

input: een oneven getal $N \geq 3$
output: "samengesteld" of "mogelijk priem"
1 Kies $a \in \{2, \ldots, N-2\}$ uniform **random**
2 $b \leftarrow a^{N-1} \text{ rem } N$ (met behulp van Algoritme 1.5)
3 **if** $b \neq 1$
4 **then return** "samengesteld"
5 **else return** "mogelijk priem"

Opmerking 4.1.6. De Fermat test gebruikt $\mathcal{O}(\log N \cdot \mathsf{M}(\log N))$ woordoperaties.

We analyseren nu deze test. Indien $\gcd(a, N) \neq 1$, dan zal ook $\gcd(b, N) \neq 1$, en dan zal de test de correcte output "samengesteld" geven. Dus we mogen veronderstellen dat $\gcd(a, N) = 1$. Wegens de kleine stelling van Fermat zal de output correct zijn indien die "samengesteld" is. Echter, als de test

"mogelijk priem" geeft, is het getal N niet noodzakelijk priem. Inderdaad, beschouw de deelgroep

$$L_N := \{g \in (\mathbb{Z}/N\mathbb{Z})^\times \mid g^{N-1} = 1\} \leq (\mathbb{Z}/N\mathbb{Z})^\times .$$

Als N priem is, dan zal $L_N = (\mathbb{Z}/N\mathbb{Z})^\times$. Als $L_N \neq (\mathbb{Z}/N\mathbb{Z})^\times$, dan zal $|L_N| \leq \frac{1}{2}|(\mathbb{Z}/N\mathbb{Z})^\times|$ aangezien de orde van een deelgroep steeds een deler is van de orde van de groep. Als het element a gekozen in regel 1 van het algoritme, modulo N, in $(\mathbb{Z}/N\mathbb{Z})^\times \setminus L_N$ blijkt te liggen, dan zal de test "samengesteld" weergeven. Een dergelijk element a (alsook zijn residuklasse modulo N) wordt een *Fermat getuige* voor de samengesteldheid van N genoemd; als a (mod N) echter in L_N ligt, wordt het een *Fermat leugenaar* genoemd voor N.

Indien dus N samengesteld is en L_N een *echte* deelgroep van $(\mathbb{Z}/N\mathbb{Z})^\times$ is, dan zal bij elke uitvoering van de test de kans op de correcte output "samengesteld" minstens $1/2$ zijn, en door het herhaaldelijk uitvoeren van de test kan dus met een zo goed als gewenste zekerheid vastgesteld worden dat N geen priemgetal is.

Echter, indien N samengesteld is en $L_N = (\mathbb{Z}/N\mathbb{Z})^\times$, dan zal de test nooit het antwoord "samengesteld" geven, met andere woorden, in dit geval heeft N geen Fermat getuigen.

Definitie 4.1.7. Een getal $N \in \mathbb{N}^*$ wordt een *Carmichael getal* genoemd, indien N samengesteld is, en $a^{N-1} \equiv 1 \pmod{N}$ voor alle $a \in \mathbb{Z}$ waarvoor $\gcd(a, N) = 1$, of equivalent, als $L_N = (\mathbb{Z}/N\mathbb{Z})^\times$.

Het probleem bij de Fermat test is dat er wel degelijk Carmichael getallen bestaan, en zelfs oneindig veel. De kleinste drie zijn $561 = 3 \cdot 11 \cdot 17$, $1105 = 5 \cdot 13 \cdot 17$ en $1729 = 7 \cdot 13 \cdot 19$. Men kan ze karakteriseren in termen van hun priemdelers:

Stelling 4.1.8 (Stelling van Korselt (1899))**.** *Zij $N \in \mathbb{N}^*$ een samengesteld getal. Dan is N een Carmichael getal als en slechts als N kwadraatvrij is en er voor elke priemdeler $p \mid N$ geldt dat ook $p - 1 \mid N - 1$. Bovendien is elk Carmichael getal oneven.*

Bewijs. Veronderstel eerst dat N een Carmichael getal is, en dat er toch een priemgetal p is zodat $p^2 \mid N$; schrijf $N = p^n \cdot t$ met $p \nmid t$. Uit Stelling 4.1.2 volgt dan dat $p \mid \varphi(p^n)$; bijgevolg bestaat er een $b \in \mathbb{Z}$ met $\mathrm{ord}_{p^n}(b) = p$. Uit de Chinese reststelling 1.4.2 volgt dat er een $a \in \mathbb{Z}$ bestaat met $a \equiv b \pmod{p^n}$ en $a \equiv 1 \pmod{t}$; in het bijzonder is $\mathrm{ord}_N(a) = p$ en $\gcd(a, N) = 1$. Maar $a^{N-1} \equiv 1 \pmod{N}$, dus volgt er uit Lemma 4.1.5 nu dat $p \mid N - 1$.

Aangezien ook $p \mid N$ bekomen we $p \mid 1$, een strijdigheid. Bijgevolg is N kwadraatvrij.

Zij nu $p \mid N$ een priemdeler van N; merk op dat $(\mathbb{Z}/p\mathbb{Z})^\times$ een cyclische groep is van orde $p-1$. Kies een generator $b \pmod p$, i.e. een element $b \in \mathbb{Z}$ zodat $\mathrm{ord}_p(b) = p-1$. Uit de Chinese reststelling 1.4.2 volgt dat er een $a \in \mathbb{Z}$ bestaat met $a \equiv b \pmod p$ en $a \equiv 1 \pmod{N/p}$; dan zal $\gcd(a, N) = 1$. Omdat N een Carmichael getal is, hebben we $a^{N-1} \equiv 1 \pmod N$, en dus ook $a^{N-1} \equiv 1 \pmod p$. Uit Lemma 4.1.5 volgt nu dat $\mathrm{ord}_p(a) = p-1 \mid N-1$.

Zij omgekeerd N een kwadraatvrij natuurlijk getal zodat voor elke priem $p \mid N$ ook $p-1 \mid N-1$. Dan is N te schrijven als $N = p_1 \cdots p_k$. Wegens de kleine stelling van Fermat hebben we $a^{p_i-1} \equiv 1 \pmod{p_i}$ voor alle i, en aangezien $p_i - 1 \mid N-1$ hebben we bijgevolg $a^{N-1} \equiv 1 \pmod{p_i}$ voor alle i. Uit de Chinese reststelling 1.4.2 volgt nu $a^{N-1} \equiv 1 \pmod N$, en dus is N een Carmichael getal.

Zij nu p een willekeurige oneven priemdeler van het Carmichael getal N; dan is ook $p-1 \mid N-1$, en omdat $p-1$ even is volgt hieruit dat ook $N-1$ even is, i.e. N is oneven. (Men kan dit ook heel eenvoudig rechtstreeks inzien: als N een even Carmichael getal zou zijn, dan zou $1 \equiv (-1)^{N-1} \equiv -1 \pmod N$ en dus $N = 2$, strijdig.) □

Opmerking 4.1.9. Er zijn 1 401 644 Carmichael getallen tussen 1 en 10^{18}, dus dit is ongeveer 1 per 700 miljard. Pas in 1994 werd aangetoond door W. R. (Red) Alford, Andrew Granville and Carl Pomerance dat er oneindig veel Carmichael getallen bestaan [AGP94]; meer bepaald toonden ze aan dat voor n voldoende groot, er minstens $n^{2/7}$ Carmichael getallen zijn kleiner dan n.

Dankzij voorgaande analyse kunnen we de Fermat test nu verbeteren.

Algoritme 4.2 Sterke pseudopriemtest

input: een oneven getal $N \geq 3$
output: "samengesteld", "mogelijk priem", of een eigenlijke factor van N.

```
 1  Kies a ∈ {2, ..., N − 2} uniform random
 2  g ← gcd(a, N)
 3  if g > 1
 4      then return g
 5  Schrijf N − 1 = 2^k · m met k, m ∈ ℕ en m oneven
 6  b_0 ← a^m rem N  (met behulp van Algoritme 1.5)
 7  if b_0 = 1
 8      then return "mogelijk priem"
 9  for i = 1, ..., k
10          do b_i ← b_{i−1}^2 rem N
11  if b_k ≠ 1
12      then return "samengesteld"
13  j ← min{i ∈ {0, ..., k − 1} | b_{i+1} = 1}
14  g ← gcd(b_j + 1, N)
15  if g = 1 or g = N
16      then return "mogelijk priem"
17      else  return g
```

Stelling 4.1.10. *Als N priem is, dan geeft Algoritme 4.2 "mogelijk priem" terug. Als N samengesteld is en geen Carmichael getal is, dan geeft het algoritme "samengesteld" terug met een kans van ten minste $1/2$. Als N een Carmichael getal is, dan geeft het algoritme een eigenlijke deler van N terug met een kans van ten minste $1/2$. Het algoritme gebruikt $\mathcal{O}(\log N \cdot \mathsf{M}(\log N))$ woordoperaties.*

Bewijs. Merk vooraf op dat $b_i \equiv a^{2^i \cdot m} \pmod{N}$ voor elke $i \in \{0, \ldots, k\}$, dus in het bijzonder is $b_k \equiv a^{N-1} \pmod{N}$. Ook zien we dat als $b_i = 1$ voor een zekere i, dan ook $b_{i+1} = 1$ en dan zo verder gaand eveneens $b_k = 1$.

Indien N samengesteld is en niet Carmichael, dan is een willekeurige $a \in \{2, \ldots, N-2\}$ met probabiliteit ten minste $1/2$ een Fermat getuige voor N, dus $b_k \neq 1$, en we bereiken regel 12 die "samengesteld" teruggeeft.

Indien N priem is, dan zal $b_k = 1$. Als reeds $b_0 = 1$, dan geeft het algoritme correct in regel 8 "mogelijk priem" terug. Indien $b_0 \neq 1$, dan wordt j in regel 12 zodanig vastgelegd dat $b_j \neq 1$ maar $b_j^2 \equiv b_{j+1} = 1 \pmod{N}$. De enige elementen modulo N die als kwadraat 1 hebben, zijn echter 1 en -1,

en dus moet $b_j \equiv -1 \pmod{N}$, dus $b_j = N-1$ en $\gcd(b_j+1, N) = N$. We belanden bij regel 16, die als antwoord "mogelijk priem" geeft.

Veronderstel tenslotte dat N een Carmichael getal is. Stel P gelijk aan de verzameling van priemdelers van N; omdat N wegens Stelling 4.1.8 kwadraatvrij is, hebben we $N = \prod_{p \in P} p$. Beschouw nu

$$I := \{i \in \{0, \ldots, k\} \mid g^{2^i \cdot m} = 1 \text{ voor alle } g \in (\mathbb{Z}/N\mathbb{Z})^\times\}\,.$$

Omdat N Carmichael is, hebben we $k \in I$; het is ook duidelijk dat als $i \in I$ met $i < k$, dan ook $i+1 \in I$. Merk ook op dat $0 \notin I$, omdat $(-1)^m \equiv -1 \not\equiv 1 \pmod{N}$. Er bestaat dus een $\ell < k$ zodat $\ell \notin I$ maar $\ell + 1 \in I$. Stel nu

$$G := \{g \in (\mathbb{Z}/N\mathbb{Z})^\times \mid g^{2^\ell \cdot m} = \pm 1\} \leq (\mathbb{Z}/N\mathbb{Z})^\times\,.$$

We tonen nu aan dat $G \neq (\mathbb{Z}/N\mathbb{Z})^\times$. Inderdaad, er bestaat een $p \in P$ en een $b \in \mathbb{Z}$ copriem met p zodat $b^{2^\ell \cdot m} \not\equiv 1 \pmod{p}$, aangezien anders $\ell \in I$ zou zijn. Door de Chinese reststelling 1.4.2 bestaat er een $c \in \mathbb{Z}$ met $c \equiv b \pmod{p}$ en $c \equiv 1 \pmod{N/p}$; dan zal inderdaad $c \pmod{N} \in (\mathbb{Z}/N\mathbb{Z})^\times \setminus G$. Aangezien G dus een eigenlijke deelgroep is, heeft G ten hoogste $\varphi(N)/2$ elementen.

We beweren nu dat als a in regel 1 zodanig gekozen werd dat $a \pmod{N} \in (\mathbb{Z}/N\mathbb{Z})^\times \setminus G$, dan zal het algoritme een eigenlijke priemdeler van N vinden. Aangezien $|(\mathbb{Z}/N\mathbb{Z})^\times \setminus G| \geq \varphi(N)/2$ is de kans dat dit zal gebeuren ten minste $1/2$. Omdat $b_{\ell+1} \equiv a^{2^{\ell+1} \cdot m} \equiv 1 \pmod{N}$, zal voor elke $p \in P$ ook $b_{\ell+1} \equiv 1 \pmod{p}$. Dit impliceert dat $a^{2^\ell \cdot m} \equiv \pm 1$ voor elke $p \in P$. Aangezien echter $b_l \equiv a^{2^\ell \cdot m} \pmod{N}$ zelf noch 1 noch -1 is, zullen beide mogelijkheden (1 en -1 modulo p) zich effectief voordoen. Merk op dat $j = \ell$ in regel 12; in regel 14 is dan

$$g = \gcd(b_\ell + 1, N) = \prod_{p \in P \,|\, a^{2^\ell \cdot m} \equiv -1 \pmod{p}} p\,,$$

en dit is inderdaad een eigenlijke deler van N.

De details in verband met de complexiteit van het algoritme laten we achterwege. □

Als N samengesteld is, dan wordt een getal $a \in \{1, \ldots, N-1\}$, copriem met N, zodanig dat Algoritme 4.2 "samengesteld" teruggeeft, of een eigenlijke factor van N geeft, een *sterke getuige* voor N genoemd; indien een dergelijke a het antwoord "mogelijk priem" geeft, wordt het een *sterke leugenaar* voor N genoemd. Het is duidelijk dat elke Fermat getuige ook een sterke getuige is.

Om de kans op een fout resultaat onder een vooropgegeven positieve constante ϵ te krijgen, voeren we de test onafhankelijk $\lceil \log \epsilon^{-1} \rceil$ keer uit met verschillende a's, en we geven "waarschijnlijk priem" terug indien *elke* uitvoering van de test "mogelijk priem" gaf; in het andere geval geven we "samengesteld" terug, of een eigenlijke factor van N indien die gevonden werd.

Opmerking 4.1.11. Algoritme 4.2 is gebaseerd op de Miller-Rabin priemtest [Mil76, Rab80]. In feite is de kans op een correcte output "samengesteld" ten minste 3/4, maar het zou ons te ver leiden om dit aan te tonen.

Er bestaan ook tal van *exacte* priemtesten (ook wel *deterministische* priemtesten genoemd), die dus formeel kunnen bewijzen dat een gegeven getal priem is. Een eerste dergelijke test is de APR test, genoemd naar Adleman, Pomerance en Rumely [APR83]. Deze werd later verbeterd door Cohen en Lenstra, en staat bekend onder de naam APRT-CL [CL84]. De test heeft complexiteit $(\log N)^{\mathcal{O}(\log\log\log N)}$.

Een geavanceerdere exacte priemtest is de ECPP (Elliptic Curve Primality Proving), die gebaseerd is op de theorie van de elliptische krommen [AM93]. De complexiteit ervan is vermoedelijk $\mathcal{O}\big((\log N)^{5+\epsilon}\big)$ voor een zekere kleine $\epsilon > 0$; men kan dit tot dusver enkel aantonen indien men bepaalde vermoedens uit de analytische getaltheorie aanneemt. Het grootste getal dat men tot op heden (september 2009) met ECPP heeft kunnen controleren op het priem-zijn, is het 20562 digits lange getal

$$(((((((((2^3+3)^3+30)^3+6)^3+80)^3+12)^3+450)^3+894)^3+3636)^3+70756)^3+97220\,;$$

dit record dateert van juni 2006. Wel zijn 6 van de 20 grootste gekende ECPP-priemgetallen in 2009 gevonden.

Indien de veralgemeende Riemann hypothese waar is, kan de Miller-Rabin test aangepast worden tot een exacte priemtest, met een complexiteit $\mathcal{O}\big((\log N)^4\big)$. In de praktijk blijkt dit algoritme trager te zijn dan de andere exacte priemtesten, aangezien de *crossover* bijzonder hoog ligt.

In 2002 hebben Agrawal, Kayal en Saxena een test gevonden (de AKS priemtest) waarvan ze effectief kunnen bewijzen hebben dat die in polynomiale tijd loopt, meer bepaald complexiteit $\mathcal{O}\big((\log N)^{12}\big)$ heeft [AKS04]. Indien een vermoeden in verband met de distributie van Sophie-Germain priemgetallen (dit zijn priemgetallen p waarvoor ook $2p+1$ priem is) waar is, dan is de complexiteit maximaal $\mathcal{O}\big((\log N)^6\big)$. Dit resultaat is vooral van theoretisch belang; in de praktijk werkt deze test beduidend trager dan de probabilistische priemtesten.

Tenslotte vermelden we nog dat er testen bestaan die ontworpen zijn voor priemgetallen met een zeer specifieke gedaante. Typisch maken deze testen

gebruik van een gekende factorisatie van $N+1$ of $N-1$; in het bijzonder komen hier dus priemgetallen van de vorm $p^n \pm 1$ in aanmerking. Twee voorbeelden van dergelijke testen zijn de Lucas-Lehmer test en Proth's test.

De bekendste priemgetallen die met behulp van de Lucas-Lehmer test gecontroleerd worden, zijn de *Mersenne priemgetallen*, dit zijn priemgetallen van de vorm 2^n-1. Het reusachtige project GIMPS (Great Internet Mersenne Prime Search) is hierop gebaseerd — samen met tal van andere optimalisaties — en het tot op heden (september 2009) grootste gekende priemgetal is door dit project gevonden [KW]. Dit getal is $2^{43\,112\,609}-1$, werd ontdekt op 23 augustus 2008, en telt maar liefst 12 978 189 digits. Op het moment van ontdekken was het het 46^{e} gekende Mersenne priemgetal, maar in april 2009 werd een 47^{e} Mersenne priem ontdekt, die met 12 837 064 digits slechts een "tikje kleiner" is dan de recordhouder.

Een *Proth priemgetal* is een priemgetal van de vorm $k2^n+1$, met k oneven en $k < 2^n$. Het grootst gekende Proth priemgetal is $19\,249 \cdot 2^{13\,018\,586}+1$ en telt 3 918 990 digits; dit is tevens het grootst gekende priemgetal dat geen Mersenne priem is. Het werd ontdekt door het (eveneens distributed computing) project "Seventeen or Bust" [HN].

Tenslotte vermelden we nog zonder bewijs een belangrijk resultaat in verband met de distributie van de priemgetallen.

Stelling 4.1.12 (Prime number theorem)**.** *Noteer het n^{e} priemgetal als p_n, en stel $\pi(x)$ gelijk aan het aantal priemgetallen kleiner dan of gelijk aan x. Dan is bij benadering*

$$\pi(x) \approx \frac{x}{\ln x}\,, \quad p_n \approx n \ln n\,.$$

Meer bepaald hebben we

$$\frac{x}{\ln x}\Big(1+\frac{1}{2\ln x}\Big) \;<\; \pi(x) \;<\; \frac{x}{\ln x}\Big(1+\frac{3}{2\ln x}\Big) \qquad \textit{als } x \geq 59,$$

$$n\Big(\ln n + \ln\ln n - \frac{3}{2}\Big) \;<\; p_n \;<\; n\Big(\ln n + \ln\ln n - \frac{1}{2}\Big) \textit{ als } n \geq 20,$$

en in het bijzonder hebben we $p_n < 2n\ln n$ voor alle $n \geq 2$.

4.2 Factorisatie van gehele getallen

In dit deel zullen we een aantal algoritmen bespreken om gehele getallen te ontbinden in priemfactoren. Het zal duidelijk worden dat er heel wat

wiskundige vindingrijkheid is nodig geweest voor het ontwikkelen van deze methodes, en we zullen de meest ingenieuze ervan, met name Lenstra's elliptische krommen methode [Len87], en de NFS-methode (number field sieve) [LLMP93], niet kunnen bespreken binnen het raam van deze cursus.

Het feit dat gehele getallen zo moeilijk te factoriseren zijn, heeft zo ook zijn voordelen: het is een essentieel gegeven voor de veiligheid van moderne cryptosystemen, die dikwijls gebaseerd zijn op een "sleutel" die het product is van twee grote priemgetallen; zie sectie 4.3.

We zullen zien dat we bij de bespreking van algoritmen voor factorisatie, naast de complexiteit van de berekening ook rekening zullen moeten houden met de grootte-orde van de benodigde *opslagcapaciteit* voor tussenberekeningen, een element waar we tot dusver nog geen rekening mee hoefden te houden.

Het is gebruikelijk om bij factorisatie-algoritmen niet de big-O, maar de soft-O notatie te gebruiken:

Definitie 4.2.1 (Soft-O notatie)**.** Zij $f, g : \mathbb{R} \to \mathbb{R}$. Dan is

$$g \in \widetilde{\mathcal{O}}(f) \iff \exists k \in \mathbb{R} \mid g \in \mathcal{O}\big(f \cdot (\log f)^k\big) .$$

Met andere woorden, de soft-O notatie negeert de logaritmische factoren.

Voor heel wat algoritmen zal het nodig zijn te eisen dat de input niet priem is, en ook geen volkomen macht is. We vermelden dat er algoritmen bestaan (gebruik makend van p-adische Newton iteratie) die voor een gegeven getal N kunnen nagaan of het een volkomen macht is, en de corresponderende machtswortel kunnen trekken, in tijd $\mathcal{O}(\log N \cdot \mathsf{M}(\log N))$. Dit is verwaarloosbaar vergeleken met de tijd die zal nodig zijn voor het factoriseren; dit is immers van de orde $\widetilde{\mathcal{O}}(1)$.

4.2.1 Trial division

Het eenvoudigst denkbare factorisatie-algoritme is ongetwijfeld het volgende.

Algoritme 4.3 Trial division

input: $N \in \mathbb{N}$, $N \geq 3$, niet priem en geen volkomen macht; $b \in \mathbb{N}$.
output: De kleinste priemfactor van N als die $\leq b$, anders "gefaald".

```
1  for p = 2, ..., b
2        do if p | N
3              then return p
4  return "gefaald"
```

Om dan alle priemfactoren van N te vinden, delen we de gevonden priemfactor zo vaak mogelijk uit, en vervolgens gaan we verder met het overblijvende getal N/p^ℓ. Uiteraard mogen we de lus in regel 1 dan starten vanaf p. De procedure zal eindigen als p de tweede grootste priemfactor is van N; na uitdelen ervan blijft dan immers een (macht van een) priemgetal over.

Notatie 4.2.2. Voor een gegeven $N \in \mathbb{N}$ zullen we de grootste priemdeler van N steeds noteren als $S_1(N)$, en de tweede grootste priemdeler als $S_2(N)$.

Uiteraard hebben we steeds $S_2(N) < N^{0.50}$. Het aantal lussen vereist bij trial division is van de orde $S_2(N)(\log N)^{\mathcal{O}(1)}$, dus $\widetilde{\mathcal{O}}(S_2(N))$. Merk op dat de uitvoertijd dus afhangt van $S_2(N)$ en niet rechtstreeks van N zelf; dit fenomeen zullen we ook bij de andere factorisatie-algoritmen vaststellen. Om een idee te krijgen van de concrete uitvoertijd geven we nog mee dat voor random $N \in \mathbb{N}$ de kans dat $S_2(N)$ groter is dan $N^{0.3}$ ongeveer 0.20 is. Dit wil dus zeggen dat in 80% van de gevallen de benodigde tijd $\widetilde{\mathcal{O}}(N^{0.3})$ is.

4.2.2 De ρ-methode van Pollard

In deze paragraaf bespreken we een probabilistisch algoritme voor factorisatie, gevonden door Pollard [Pol75]. Zoals we zullen zien, bevat het een heuristisch argument, zodat we niet in staat zullen zijn de vermoedelijke complexiteit van het algoritme rigoureus aan te tonen.

Zij N het getal dat we willen factoriseren. We nemen aan dat N minstens twee verschillende priemfactoren heeft: het is dus geen priemgetal of macht van een priemgetal (deze voorwaarde kan eenvoudig gecontroleerd worden).

Het idee van de ρ-methode is als volgt. We kiezen een veeltermfunctie $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$ en een startwaarde $x_0 \in \mathbb{Z}/N\mathbb{Z}$, en we definiëren recursief $x_i := f(x_{i-1})$ voor alle $i > 0$. We hopen dat de rij $x_0, x_1, x_2, \ldots$ zich gedraagt als een rij van onafhankelijke random elementen in $\mathbb{Z}/N\mathbb{Z}$. Als p

een (onbekende) priemdeler is van N, dan hebben we een *herhaling* modulo p, als er twee getallen t, ℓ zijn, met $\ell > 0$, zodat $x_t \equiv x_{t+\ell} \pmod{p}$. Zij nu q een andere priemdeler van N. Als de x_i's daadwerkelijk random resten zijn modulo N, dan zijn $x_i \pmod{p}$ en $x_i \pmod{q}$ onafhankelijke random variabelen, omwille van de Chinese reststelling 1.4.2. Het is dus zeer waarschijnlijk dat $x_t \not\equiv x_{t+\ell} \pmod{q}$, zodat $\gcd(x_{t+\ell} - x_t, N)$ een niet-triviale factor van N is.

Een eerste vraag is: hoe klein kunnen we verwachten dat t en ℓ zullen zijn? Uiteraard zal $t + \ell \leq p$, maar we zullen zien dat de verwachte waarde slechts $\mathcal{O}(\sqrt{p})$ is voor een *random* rij $(x_i)_{i \in \mathbb{N}}$.

Lemma 4.2.3 (Verjaardagsparadox)**.** *We beschouwen random keuzes met teruglegging, uit een verzameling* $\{1, 2, \ldots, p\}$*. De verwachtingswaarde van het aantal nodige keuzes voor we een herhaling krijgen, is* $\mathcal{O}(\sqrt{p})$*.*

Bewijs. Stel $s \geq 2$ gelijk aan het aantal keuzes voor we een herhaling krijgen; dit is een random variabele. Voor elke $j \geq 2$ is de kans dat $s \geq j$ gelijk aan de kans dat er in de eerste $j - 1$ keuzes geen herhaling zat, dus

$$\begin{aligned} \text{prob}(s \geq j) &= \frac{1}{p^{j-1}} \prod_{i=1}^{j-1} \big(p - (i-1)\big) = \prod_{i=1}^{j-1} \left(1 - \frac{i-1}{p}\right) \\ &\leq \prod_{i=1}^{j-1} e^{-\frac{i-1}{p}} = e^{-\frac{(j-1)(j-2)}{2p}} \leq e^{-(j-2)^2 \,/\, 2p}, \end{aligned}$$

waarbij we gebruik hebben gemaakt van het feit dat $1 - x \leq e^{-x}$ voor alle reële x met $0 < x < 1$. Bijgevolg geldt voor de verwachtingswaarde $E(s)$

$$\begin{aligned} E(s) = \sum_{j=2}^{\infty} \text{prob}(s \geq j) &\leq \sum_{j=0}^{\infty} e^{-j^2 \,/\, 2p} \leq 1 + \int_0^{\infty} e^{-x^2 \,/\, 2p} \, dx \\ &\leq 1 + \sqrt{2p} \cdot \int_0^{\infty} e^{-x^2} \, dx = 1 + \sqrt{p\pi/2} \;\in\; \mathcal{O}(\sqrt{p}) \,. \end{aligned}$$

□

Beschouw nu opnieuw de veeltermfunctie $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$ zoals hierboven, waarbij we op zoek gaan naar een herhaling $x_i = x_j$ met $i \neq j$. Een vanzelfsprekende methode is om alle x_i neer te schrijven totdat we een waarde bekomen die reeds eerder in de rij voorkwam. Het nadeel van deze methode is dat deze $\mathcal{O}(t + \ell)$ opslag vereist.

Het volgende algoritme, bekend als "Floyd's cycle detection trick", neemt slechts constante opslag in. Het idee van Floyd is om een tweede rij $(y_i)_{i \in \mathbb{N}}$ te beschouwen die f itereert met dubbele snelheid, zodat dus $y_i = x_{2i}$ voor alle

i; op elk moment i houden we enkel de waarden x_i en y_i zelf bij, en dus niet de voorgaande waarden. Intuïtief is het wellicht duidelijk dat de snellere rij (y_i) de rij (x_i) zal "inhalen", zodat we op een bepaald moment $x_{2i} = y_i = x_i$ zullen krijgen.

Algoritme 4.4 Floyd's cycle detection trick

input: $1 \leq N \in \mathbb{N}$, een functie $f : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$, en een startwaarde $x_0 \in \{0, \ldots, N-1\}$.
output: Een index $i > 0$ zodat $x_i = x_{2i}$.

```
1  x ← x_0; y ← x_0
2  i ← 0
3  repeat  i ← i + 1
4          x ← f(x)
5          y ← f(f(y))
6     until x = y
7  return i
```

Veronderstel dat t de minimale waarde is zodat er een $\ell_0 > 0$ bestaat met $x_t = x_{t+\ell_0}$, en stel vervolgens ℓ gelijk aan de minimale waarde voor ℓ_0 die hieraan voldoet.

Stelling 4.2.4. *Algoritme 4.4 werkt correct, en stopt na ten hoogste $t + \ell$ stappen.*

Bewijs. Merk op dat bij het bereiken van regel 6 steeds $x = x_i$ en $y = x_{2i}$. We hebben $x_i = x_{2i}$ als en slechts als $i \geq t$ en $\ell \mid (2i - i)$. Als $t = 0$, dan zal $i = \ell$ voldoen. Als $t > 0$, dan zal $i = t + (-t \operatorname{rem} \ell)$ voldoen. In beide gevallen vinden we $i \leq t + \ell$. □

We beschrijven nu de ρ-methode van Pollard om het getal N te factoriseren. Daartoe gebruiken we de methode die we net beschreven hadden, toegepast op de functie

$$f(x) = x^2 + 1 \operatorname{rem} N\,.$$

Het magische aan deze functie is dat ze zich voldoende random blijkt te gedragen, hoewel ze zeer eenvoudig van voorschrift is.

Zij p de kleinste priemdeler van N; dan hebben we uiteraard $x_{i+1} \equiv x_i^2 + 1 \pmod{p}$ voor alle $i \geq 0$. Het heuristische feit dat de rij $(x_i)_{i \in \mathbb{N}}$ voldoende

random is, samen met Lemma 4.2.3, vertelt ons dat we een herhaling modulo p kunnen verwachten na $\mathcal{O}(\sqrt{p})$ stappen. Deze herhaling kunnen we detecteren met behulp van Floyd's cycle detection trick.

Algoritme 4.5 De ρ-methode van Pollard

input: $N \in \mathbb{N}$, $N \geq 3$, niet priem en geen volkomen macht.
output: De (meestal kleinste) priemfactor van N, of "gefaald".

```
1  Kies x ∈ {0, ..., N − 1} random
2  y ← x
3  repeat  x ← x^2 + 1 rem N
4          y ← (y^2 + 1)^2 + 1 rem N
5          if x = y
6            then return "gefaald"
7          g ← gcd(x − y, N)
8          if g > 1
9            then return g
```

Opmerking 4.2.5. Ook als N samengesteld is, kan de methode "gefaald" teruggeven; het zou immers kunnen dat toevallig voor alle i, j met $x_i \equiv x_j \pmod{p}$ steeds ook $x_i \equiv x_j \pmod{N}$. In dat geval kunnen we het algoritme opnieuw uitvoeren met een andere startwaarde x_0. Een andere mogelijkheid is om de functie $f(x) = x^2 + 1$ te vervangen door de functie $f(x) = x^2 + a$ voor een geheel getal $a \not\equiv 0, -2 \pmod{N}$.

Stelling 4.2.6. *Zij $N \in \mathbb{N}$ samengesteld, p de kleinste priemfactor van N, en $f(x) = x^2 + 1$. In de veronderstelling dat de rij $\left(f^i(x_0)\right)_{i\in\mathbb{N}}$ zich modulo p gedraagt als een random rij, is de verwachte looptijd van Algoritme 4.5 voor het vinden van de kleinste priemfactor p van N gelijk aan $\mathcal{O}(\sqrt{p} \cdot \mathsf{M}(\log N) \cdot \log\log N)$. Door het algoritme recursief toe te passen, kan N volledig gefactoriseerd worden in een verwachte tijd $S_2(N)^{1/2} \cdot \widetilde{\mathcal{O}}((\log N)^2)$, of dus $\widetilde{\mathcal{O}}(N^{1/4})$.*

Zonder bewijs. □

Opmerking 4.2.7. Zoals ook al blijkt uit voorgaande stelling, is dit algoritme zeer efficiënt voor het vinden van relatief kleine priemdelers. Zo heeft het zijn bekendheid verworven door het factoriseren van het Fermat getal $F_8 = 2^{2^8} + 1$ als product van twee priemgetallen, één van 16 digits en één van 62 digits. (In feite werd hier een geoptimaliseerde versie van dit algoritme gebruikt, ontwikkeld door Brent [Bre80].)

4.2.3 De kwadratische zeef

In deze paragraaf zullen we de kwadratische zeef van Pomerance bespreken, een algoritme voor het volledig factoriseren van een oneven getal N dat geen priemmacht is. In tegenstelling tot de voorgaande methodes helpt het niet als N kleine factoren heeft, de looptijd hangt enkel af van de grootte van N en niet van de priemdelers van N.

De kwadratische zeef vertrekt van het volgende idee van Fermat: als $N = a \cdot b$, dan geldt

$$N = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2.$$

Als nu a en b dicht bij elkaar liggen, dan is de tweede term klein en is dus $(a+b)/2$ ongeveer gelijk aan $\sqrt{N}$. Dit suggereert onmiddellijk het volgende factorisatie-algoritme, door Fermat: controleer voor alle $t = \lceil\sqrt{N}\rceil, \lceil\sqrt{N}\rceil + 1, \ldots$ of $t^2 - N$ een volkomen kwadraat is. Indien we zo'n kwadraat vinden, stel $t^2 - N = s^2$, dan kunnen we N factoriseren als $N = (t-s)(t+s)$.

De basisidee van de kwadratische zeef is nu als volgt: stel dat we $t_1, \ldots, t_\ell$ vinden waarvoor het *product* $(t_1^2 - N)\ldots(t_\ell^2 - N)$ een kwadraat is, stel s^2. Dan hebben we $(t_1 \ldots t_\ell)^2 \equiv s^2 \pmod{N}$, of ook $(t_1 \ldots t_\ell/s)^2 \equiv 1 \pmod{N}$. (Als s niet inverteerbaar is hebben we ook een factor van N gevonden, dus we mogen onderstellen dat $\gcd(s, N) = 1$.) Nu komt volgende stelling van pas:

Stelling 4.2.8. *Zij N een oneven geheel getal dat geen priemgetal of macht van een priemgetal is. Stel dat we een $x \in \mathbb{Z}/N\mathbb{Z}$ gevonden hebben waarvoor $x^2 \equiv 1 \pmod{N}$. Voor zo'n x is de kans minstens 50% dat $\gcd(x-1, N)$ een echte factor van N is.*

Bewijs. Door het onderstelde kunnen we N schrijven als $N = a \cdot b$ met $a \geq 3$, $b \geq 3$ en $\gcd(a, b) = 1$. De Chinese Reststelling zegt dat $x^2 \equiv 1 \pmod{N}$ equivalent is met het stelsel

$$\begin{cases} x^2 \equiv 1 \pmod{a} \\ x^2 \equiv 1 \pmod{b} \end{cases}.$$

Beide vergelijkingen hebben minstens twee oplossingen, namelijk 1 en -1. Met de Chinese Reststelling hebben we dus in totaal minstens vier oplossingen van $x^2 \equiv 1 \pmod{N}$.

Wanneer zal $\gcd(x-1, N)$ geen echte deler van N zijn? Als $\gcd(x-1, N) = N$, dan moet $x \equiv 1 \pmod{N}$. Als $\gcd(x-1, N) = 1$, dan kunnen

we in $(x^2-1) \equiv 0 \pmod{N}$ de factor $(x-1)$ wegdelen en vinden we $x \equiv -1 \pmod{N}$. Dit zijn de enige twee "slechte" oplossingen van $x^2 \equiv 1 \pmod{N}$. Als $x \not\equiv 1 \pmod{N}$ en $x \not\equiv -1 \pmod{N}$, dan geeft $\gcd(x-1, N)$ een echte factor. $\square$

De kwadratische zeef is een methode om een congruentie $t^2 \equiv s^2 \pmod{N}$ te vinden, dus een congruentie $(t/s)^2 \equiv 1 \pmod{N}$. Dan geeft Stelling 4.2.8 hopelijk een factorisatie van N.

We beschouwen een *factorbasis* $\mathcal{F}$ bestaande uit alle priemgetallen kleiner dan of gelijk aan een voorop gekozen getal $B \in \mathbb{R}$ en ook het getal -1, dat we hier als "priemgetal" beschouwen (dus bijvoorbeeld -24 heeft dan als factorisatie $-1 \cdot 2^3 \cdot 3$). Stel $\mathcal{F} = \{p_1, \ldots, p_h\}$. Een getal $b \in \mathbb{N}$ wordt een *B-getal* genoemd indien $b^2 \operatorname{rem} N$ een product is van getallen in $\mathcal{F}$. Voor elk B-getal b schrijven we

$$b^2 \equiv p_1^{\alpha_1} \cdots p_h^{\alpha_h} \pmod{N}.$$

Zo'n vergelijking noemen we een *relatie*. Met b associëren we de *binaire exponentvector*

$$\epsilon(b) = (\alpha_1 \bmod 2, \ldots, \alpha_h \bmod 2) \in V,$$

waarbij V de h-dimensionale vectorruimte over $\mathsf{GF}(2)$ is.

Veronderstel nu dat we genoeg relaties hebben voor B-getallen $b_1, \ldots, b_\ell$, zodanig dat

$$\epsilon(b_1) + \cdots + \epsilon(b_\ell) = 0 \text{ in } V\,. \tag{4.1}$$

We schrijven $b_i^2 \equiv \prod_{j=1}^{h} p_j^{\alpha_{ij}} \pmod{N}$ voor alle $i \in \{1, \ldots, \ell\}$; dan zegt vergelijking (4.1) precies dat

$$\gamma_j := \frac{1}{2} \sum_{i=1}^{\ell} \alpha_{ij} \in \mathbb{N}$$

voor alle $j \in \{1, \ldots, h\}$. We stellen vast dat

$$\left(\prod_{i=1}^{\ell} b_i \right)^2 \equiv \left(\prod_{j=1}^{h} p_j^{\gamma_j} \right)^2 \pmod{N},$$

en dit is de gezochte congruentie $t^2 \equiv s^2 \pmod{N}$. Stelling 4.2.8 impliceert dat deze met kans minstens $1/2$ een factorisatie van N geeft. Om dus met zeer grote waarschijnlijkheid de factorisatie van N te vinden, moeten we ervoor zorgen dat we genoeg onafhankelijke congruenties $t^2 \equiv s^2 \pmod{N}$ hebben. Als we $h + d$ relaties hebben voor een zekere $d > 0$, zegt lineaire

algebra ons dat we dan minstens d onafhankelijke oplossingen hebben voor (4.1). De kans is dan hoogstens 2^{-d} dat we *geen* factorisatie vinden.

Om de relaties te vinden, werken we zoals bij de methode van Fermat: we berekenen $t^2 - N$ voor alle gehele getallen t in een zeker interval rond $\sqrt{N}$ en controleren of $t^2 - N$ het product is van getallen uit de factorbasis $\mathcal{F}$ (gebruik makend van *trial division*). Aangezien $t \approx \sqrt{N}$, zal $|t^2 - N|$ klein zijn, de kans is dus groter dat t inderdaad een B-getal is.

We werken het algoritme uit aan de hand van het voorbeeld $N = 53953$. Als factorbasis nemen we $\mathcal{F} = \{-1, 2, 3, 5, 7, \ldots, 47\}$. We berekenen $\sqrt{N} = 232,278\ldots$. We proberen $t^2 - N$ te factoriseren voor alle gehele getallen $t \in [222, 242]$. Dit is het resultaat:

$$
\begin{aligned}
222^2 &\equiv -1 \cdot 7 \cdot 23 \cdot 29 & \qquad 232^2 &\equiv -1 \cdot 3 \cdot 43 \\
223^2 &\equiv -1 \cdot 2^7 \cdot 3 \cdot 11 & 233^2 &\equiv 2^4 \cdot 3 \cdot 7 \\
225^2 &\equiv -1 \cdot 2^8 \cdot 13 & 238^2 &\equiv 3^2 \cdot 13 \cdot 23 \\
229^2 &\equiv -1 \cdot 2^3 \cdot 3^3 \cdot 7 & 239^2 &\equiv 2^5 \cdot 3^2 \cdot 11 \\
230^2 &\equiv -1 \cdot 3^4 \cdot 13 & 241^2 &\equiv 2^5 \cdot 3 \cdot 43 \\
231^2 &\equiv -1 \cdot 2^4 \cdot 37
\end{aligned}
$$

Op basis van deze relaties kunnen we onze factorbasis verkleinen tot $\mathcal{F}' := \{-1, 2, 3, 7, 11, 13, 23, 29, 37, 43\}$. De gevonden relaties kunnen we voorstellen in een matrix. In elke rij zetten we de exponentvector van een relatie, de kolommen staan voor de priemen in de factorbasis. De oneven getallen zijn vetjes gedrukt:

$$
M = \begin{array}{c}
\begin{array}{cccccccccc}
-1 & 2 & 3 & 7 & 11 & 13 & 23 & 29 & 37 & 43
\end{array} \\
\left(\begin{array}{cccccccccc}
\mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\
\mathbf{1} & \mathbf{7} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & 8 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
\mathbf{1} & \mathbf{3} & \mathbf{3} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & 0 & 4 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
\mathbf{1} & 4 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 4 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{5} & 2 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
0 & \mathbf{5} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}
\end{array}\right)
\end{array}
$$

Om een oplossing van (4.1) te vinden, zoeken we een element uit de linkse kern van M over $\mathsf{GF}(2)$. We vinden dat $\epsilon(225) + \epsilon(230) = 0$.

Hieruit halen we

$$\begin{aligned} 225^2 \cdot 230^2 &\equiv (-1 \cdot 2^8 \cdot 13) \cdot (-1 \cdot 3^4 \cdot 13) \pmod{N} \\ (225 \cdot 230)^2 &\equiv (2^4 \cdot 3^2 \cdot 13)^2 \pmod{N} \end{aligned}$$

Zoals in Stelling 4.2.8 bereken we

$$\gcd(N, 225 \cdot 230 - 2^4 \cdot 3^2 \cdot 13) = \gcd(53953, 49878) = 163.$$

Dit geeft de factorisatie $53953 = 163 \cdot 331$.

Hier is het resulterende algoritme.

Algoritme 4.6 Kwadratische zeef (naïeve versie)

input: $N \in \mathbb{N}$, $N \geq 3$, oneven, niet priem en geen volkomen macht;
$B \in \mathbb{R}_{>0}$.
output: Een eigenlijke deler van N, of "gefaald".

Bereken alle "priemgetallen" $p_1 = -1$, $p_2 = 2$, $p_3 = 3$, $\ldots$, p_h kleiner dan of gelijk aan B
if $p_i \mid N$ voor een $i \in \{2, \ldots, h\}$
  then return p_i
Kies een interval I van gehele getallen rond $\sqrt{N}$
$M \leftarrow$ matrix met 0 rijen en h kolommen
$n \leftarrow 0$ (aantal rijen in M)
for $b \in I$
  do $g \leftarrow \gcd(b, N)$
    if $g > 1$
      then return g
    $a \leftarrow b^2 - N$
    for $i = 1, \ldots, h$
      do $\alpha_i \leftarrow 0$
        while $p_i \mid a$
          do $a \leftarrow a/p_i$
            $\alpha_i \leftarrow \alpha_i + 1$
    if $a = 1$
      then
        $n \leftarrow n + 1$
        $b_n \leftarrow b$
        Voeg de rij $(\alpha_1, \ldots, \alpha_h)$ toe aan de matrix M (dit wordt de n-de rij)
Bereken een basis K voor de linkse kern van de matrix M over $\mathsf{GF}(2)$
for $k \in K$
  do Lift k naar een vector over $\mathbb{Q}$ ($0_{\mathsf{GF}(2)} \mapsto 0_{\mathbb{Q}}$ en $1_{\mathsf{GF}(2)} \mapsto 1_{\mathbb{Q}}$)
    $(\gamma_1, \ldots, \gamma_h) \leftarrow (k \cdot M)/2$
    $s \leftarrow \prod_{i=1}^{n} b_i^{k_i}$
    $t \leftarrow \prod_{j=1}^{h} p_j^{\gamma_j}$
    $g \leftarrow \gcd(N, s - t)$
    if $1 < g < N$
      then return g
return "gefaald"

Het algoritme, zoals hier beschreven, is maar een naïeve versie. De trial division stap van de a's is niet zo efficiënt. In plaats daarvan gebruiken we een *zeef*: Als $a = b^2 - N$, dan zal $p_i \mid a$ als en slechts als

$$b^2 \equiv N \pmod{p_i}. \tag{4.2}$$

Voor elke p_i kunnen we deze kwadratische vergelijking oplossen[1] over $\mathsf{GF}(p_i)$ om onmiddellijk te zien voor welke b's we de corresponderende a kunnen delen door p_i. We beschouwen als voorbeeld $N = 53953$ en $p = 11$. Vergelijking (4.2) wordt dan $b^2 \equiv 9 \pmod{11}$, met als oplossingen $b \equiv 3 \pmod{11}$ en $b \equiv 8 \pmod{11}$. Voor ons interval $I = [222, 242]$ hebben we dus een factor 11 voor $b = 223, 228, 234, 239$. Met deze methode moeten we wel nog trial division doen om te controleren of de getallen deelbaar zijn door hogere machten van p_i.

In het Algoritme 4.6 zitten nog een aantal onbekende parameters, zoals de grootte van het interval I en de zeef-grens B. Wat I betreft is er geen probleem: men kan gedurende de loop van het algoritme I vergroten als er te weinig relaties gevonden zijn. Het is duidelijk dat de moeilijkheid van het algoritme is om een goede waarde voor B te vinden. Inderdaad, indien B te klein is, zijn de B-getallen te zeldzaam en kan het lang duren om er te vinden; als B te groot is wordt h groter, wat de lineaire algebra (het berekenen van de kern van M) trager maakt. We laten deze moeilijke discussie achterwege, en vermelden enkel het resultaat.

Vermoeden 4.2.9. *Zij $N \in \mathbb{N}$, $N \geq 3$, oneven, niet priem, en geen volkomen macht. De ideale waarde voor de parameter B voor de factorisatie van N met behulp van Algoritme 4.6 is*

$$B = \exp\left(\tfrac{1}{\sqrt{2}} \cdot (\ln N)^{\frac{1}{2}} \cdot (\ln \ln N)^{\frac{1}{2}}\right).$$

Met deze keuze van B heeft het algoritme complexiteit

$$\mathcal{O}\left(\exp\left((1+\epsilon) \cdot (\ln N)^{\frac{1}{2}} \cdot (\ln \ln N)^{\frac{1}{2}}\right)\right)$$

voor elke waarde van $\epsilon > 0$.

Voor meer informatie verwijzen we de geïnteresseerde student naar het zeer vlot geschreven en voor algemeen wiskundig publiek bedoelde overzichtsartikel van Pomerance [Pom96].

[1] Dit gebeurt bijvoorbeeld door middel van de factorisatie van de veelterm $x^2 - N$ over $\mathsf{GF}(p_i)$, zie Hoofdstuk 5.

Historisch gezien is de kwadratische zeef een verbetering van de *random kwadraten mathode* van Dixon ([Dix81]). Hierin worden de b's niet in een vast interval gekozen, maar gewoon volledig willekeurig. Dit was de eerste methode waarvan men kon bewijzen dat ze sneller is dan $\widetilde{\mathcal{O}}(N^\epsilon)$ voor elke $\epsilon > 0$.

4.2.4 Factorisatie in de praktijk

We vermelden nog twee andere zeer belangrijke factorisatie-methodes die beide gebaseerd zijn op diepere getaltheorie.

Ten eerste is er de *number field sieve* (NFS) van Lenstra, Lenstra, Manasse en Pollard [LLMP93] die vermoedelijke complexiteit

$$\mathcal{O}\Big(\exp\Big((c+\epsilon)(\ln N)^{\frac{1}{3}} \cdot (\ln\ln N)^{\frac{2}{3}}\Big)\Big)$$

heeft voor alle $\epsilon > 0$. Voor getallen van een speciale vorm zoals $a^n \pm 1$ is er de *special number field sieve* (SNFS) met $c = \sqrt[3]{32/9}$ en voor algemene getallen de *general number field sieve* (GNFS) met $c = \sqrt[3]{64/9}$. Dit is in de praktijk de snelste methode om getallen te factoriseren met meer dan pakweg 100 cijfers.

Lenstra's elliptische krommen methode (ECM) [Len87] heeft vermoedelijke complexiteit

$$\mathcal{O}\Big(\exp\Big(\big(\sqrt{2}+\epsilon\big) \cdot (\ln p)^{\frac{1}{2}} \cdot (\ln\ln p)^{\frac{1}{2}}\Big)\Big)\widetilde{\mathcal{O}}(\ln N)\,,$$

voor alle $\epsilon > 0$. Deze hangt dus bijna niet af van de grootte van N, maar vooral van de (onbekende) priemdeler p van N.

Als men in de praktijk een getal wil factoriseren zal men altijd een combinatie gebruiken van de methodes die we hier beschreven hebben. De volgende stappen zijn typisch:

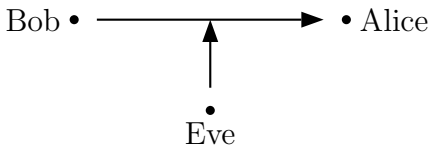
1. Trial division tot een grens van bijvoorbeeld 1 miljoen.
2. Pollard's ρ-methode. Hiermee kunnen we priemfactoren vinden met een grootte tot 10 à 15 cijfers.
3. ECM om nog zoveel mogelijk "kleine" factoren te vinden. Typisch betekent dit factoren tot 20 à 50 cijfers, afhankelijk van de grootte van N.
4. Als ons getal nog altijd niet volledig gefactoriseerd is: kwadratische zeef voor $N < 10^{100}$ en number field sieve voor $N > 10^{100}$.

Het huidige record voor factorisatie van getallen zonder speciale vorm is de factorisatie van RSA-768, een getal dat gegeven werd als *RSA challenge*. In de volgende paragraaf staat een cryptografisch systeem beschreven dat gekraakt kan worden als we grote getallen kunnen factoriseren. Om de sterkte van dat systeem te testen, zijn de RSA challenges in het leven geroepen: het zijn een reeks van gecodeerde boodschappen van verschillende moeilijkheid. Het getal RSA-768 heeft 768 bits (232 decimale cijfers) en is gefactoriseerd met het GNFS algoritme op een cluster van een honderdtal computers. De factorisatie duurde anderhalf jaar en werd bekend gemaakt op 12 december 2009.

4.3 Toepassing: Het RSA cryptosysteem

Het bekende RSA cryptosysteem, genoemd naar Rivest, Shamir en Adleman [RSA78], is gebaseerd op het feit dat het factoriseren van gehele getallen een (vermoedelijk) zeer moeilijk probleem is. We schetsen eerst even kort het idee van een cryptosysteem.

Veronderstel dat Bob een bericht wil sturen naar Alice, op een zodanige manier dat een afluisteraar Eve[2] die luistert naar het transmissiekanaal, het bericht niet kan begrijpen. Dit gebeurt door het bericht te coderen zodanig dat enkel Alice, die de juiste *sleutel* bezit, het bericht kan decoderen.



Dit klinkt simpel genoeg, maar het blijkt uitermate moeilijk om een waterdicht systeem te bedenken. Sterker nog, het enige bewijsbaar veilige cryptosysteem (in een computertheoretische betekenis) is de zogenaamde *one-time pad*, waarbij de sleutel even lang is als de te coderen boodschap, en waarbij een sleutel niet herbruikt mag worden. Uiteraard is een dergelijk systeem in de praktijk onbruikbaar.

Diffie en Hellman stelden in 1976 een toen revolutionair idee voor, dat nu bekend staat onder de naam *public key cryptography* [DH76]. Hun idee is om twee verschillende sleutels K en S te gebruiken voor encryptie en decryptie, respectievelijk, zodanig dat zowel encryptie als decryptie "gemakkelijk" zijn, maar zodat decryptie zonder de kennis van S "moeilijk" is. Hier betekent gemakkelijk in polynomiale tijd, liefst zelfs quasi-lineair of kwadratisch, terwijl

[2]De namen Alice, Bob en Eve zijn standaard in de cryptografie. Alice en Bob hebben hun naam te danken aan hun beginletters A en B, terwijl Eve doet denken aan "eavesdropper".

moeilijk betekent dat er geen efficiënt algoritme bestaat, zodat het in theorie vele honderden jaren zou duren om de decryptie uit te voeren. We willen met andere woorden een functie die gemakkelijk te berekenen is én inverteerbaar is, maar waarvan de inverse moeilijk te berekenen is zonder bijkomende kennis. Een dergelijke functie wordt een *trapdoor* functie genoemd. De sleutels K en S worden respectievelijk *public key* en *private key* genoemd, en een dergelijk cryptosysteem wordt *asymmetrisch* genoemd. Bij een dergelijk systeem zijn n public-secret sleutelparen voldoende om een veilige communicatie tussen n partijen te garanderen.

We beschrijven nu RSA, dat een voorbeeld is van een asymmetrisch cryptosysteem. Het idee is dat Alice twee verschillende grote random priemgetallen kiest (van pakweg 150 digits), noem ze p en q; stel dan $N = pq$. Als N zou kunnen gefactoriseerd worden, dan is het cryptosysteem gekraakt, maar voor getallen van 300 digits zoals N ligt dit nog niet in het bereik van de huidige methoden. We zullen berichten coderen als getallen modulo N, of dus als elementen van $\mathbb{Z}/N\mathbb{Z}$; als we uitgaan van een alfabet van 26 letters, kunnen we op die manier boodschappen van lengte $\lfloor \log_{26} 10^{300} \rfloor = 212$ coderen in 1 zo'n getal, door gebruik te maken van de 26-adische voorstelling van dat getal. Zo wordt bijvoorbeeld de boodschap CAESAR gecodeerd als

$$2 \cdot 26^0 + 0 \cdot 26^1 + 4 \cdot 26^2 + 18 \cdot 26^3 + 0 \cdot 26^4 + 17 \cdot 26^5 = 202\,302\,466 \in \mathbb{Z}/N\mathbb{Z}\,.$$

Als Alice nu berichten wil ontvangen van Bob, kiest ze een random $e \in \{2, \ldots, \varphi(N) - 2\}$ copriem met $\varphi(N)$, waarbij $\varphi(N)$ de Euler totiënt functie is die we ingevoerd hebben in Definitie 4.1.1(ii). De clue is nu dat $\varphi(N) = (p-1)(q-1)$ berekenen even moeilijk is als het factoriseren van N, zoals we in Stelling 4.3.5 zullen aantonen. Maar Alice kent $\varphi(N)$, en kan dus eenvoudig $d \in \{2, \ldots, \varphi(N) - 2\}$ berekenen zodat $de \equiv 1 \pmod{\varphi(N)}$, met behulp van het uitgebreid algoritme van Euclides 1.6. Ze maakt het paar (N, e) beschikbaar als haar public key, en houdt het paar (N, d) geheim voor zichzelf als private key. (De oorspronkelijke priemgetallen p en q mogen nu zelfs "vernietigd" worden.)

We definiëren nu de encryptie- en decryptiefuncties ϵ en δ als

$$\epsilon : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z} : x \mapsto x^e\,,$$
$$\delta : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z} : x \mapsto x^d\,.$$

Lemma 4.3.1. *De functies ϵ en δ zijn elkaars inverse.*

Bewijs. We hebben $\epsilon\delta = \delta\epsilon : x \mapsto x^{de}$ voor alle $x \in \mathbb{Z}/N\mathbb{Z}$. Aangezien $de \equiv 1 \pmod{\varphi(N)}$, volstaat het te bewijzen dat

$$x^{\varphi(N)+1} \equiv x \pmod{N} \tag{4.3}$$

voor *alle* $x \in \mathbb{Z}/N\mathbb{Z}$. Indien $\gcd(x, N) = 1$, volgt dit onmiddellijk uit de stelling van Euler. Stel nu $\gcd(x, N) = p$; dan volgt opnieuw uit de stelling van Euler dat $x^{\varphi(N)+1} \equiv x \pmod{q}$, terwijl trivialerwijze $x^{\varphi(N)+1} \equiv x \pmod{p}$ omdat $p \mid x$. Uit de Chinese reststelling 1.4.2 volgt nu opnieuw de gelijkheid (4.3). Het geval $\gcd(x, N) = q$ is analoog. □

Opmerking 4.3.2. Vergelijking (4.3) is niet langer geldig voor willekeurige N. (Beschouw bijvoorbeeld $N = 12$ en $x = 2$.)

Opmerking 4.3.3. Indien de boodschap x toevallig zo zou zijn dat $\gcd(x, N)$ verschillend is van 1, dan is het cryptosysteem gekraakt, omdat $\gcd(x, N)$ dan hetzij p hetzij q is. Gelukkig is die kans verwaarloosbaar klein, met name

$$1 - \frac{\varphi(N)}{N} = \frac{1}{p} + \frac{1}{q} - \frac{1}{pq}\,,$$

hetgeen van de orde 10^{-150} is als p en q 150 digits lang zijn.

We willen nu formeel nagaan dat het berekenen van de secret key even moeilijk is als het factoriseren van N zelf.

Definitie 4.3.4. (i) Een *polynomiale-tijd reductie* van een probleem X naar een probleem Y is een deterministisch algoritme voor X in polynomiale tijd, gebruik makend van een (onbekende) subroutine die probleem Y oplost.

(ii) Twee problemen X en Y worden *(polynomiale-tijd) equivalent* genoemd, indien er een polynomiale-tijd reductie bestaat van X naar Y en een polynomiale-tijd reductie van Y naar X.

Stelling 4.3.5. *Gegeven is een $N \in \mathbb{N}$ die het product is van twee (onbekende) priemgetallen p en q. De volgende drie problemen zijn equivalent:*

(a) *factoriseer N;*

(b) *bereken $\varphi(N)$;*

(c) *gegeven $e \in \mathbb{N}$, bepaal of $\gcd(e, \varphi(N)) = 1$, en indien wel, bereken een $d \in \mathbb{N}$, $d < \varphi(N)$, met de $\equiv 1 \pmod{\varphi(N)}$.*

Bewijs. Uit Stelling 4.1.2 volgt dat er een polynomiale reductie bestaat van (b) naar (a); in ons specifiek geval volgt uit de factorisatie $N = pq$ onmiddellijk $\varphi(N) = (p-1)(q-1)$.

Veronderstel nu dat N en $\varphi(N)$ gekend zijn; we zoeken de onbekende priemfactoren p en q (in polynomiale tijd). Merk op dat $p+q = N-\varphi(N)+1$; bijgevolg zijn p en q de oplossingen van de kwadratische vergelijking

$$x^2 - (N - \varphi(N) + 1) \cdot x + N = 0\,.$$

Hieruit volgt dat (a) en (b) equivalent zijn.

Het uitgebreid algoritme van Euclides 1.6 verschaft ons een reductie van (c) naar (b). Veronderstel omgekeerd dat we een (onbekend) algoritme hebben voor het oplossen van probleem (c). We beschrijven dan een algoritme dat, hiervan gebruik makend, $\varphi(N)$ zal berekenen. Vooreerst zoeken we het kleinste natuurlijk getal $e \geq 2$ met $\gcd(e, \varphi(N)) = 1$; hiervoor lopen we alle natuurlijke getallen af, en ons algoritme voor (c) vertelt ons of $\gcd(e, \varphi(N)) = 1$. Aangezien $\varphi(N)$ ten hoogste $r = \log \varphi(N)$ verschillende priemdelers heeft, is er zeker een priemgetal p_s copriem met $\varphi(N)$ voor een $s \leq r + 1$, waarbij p_s het s^{e} priemgetal aanduidt. Uit de Prime number theorem 4.1.12 volgt dan dat p_s, en dus ook e, in elk geval kleiner is dan

$$2(r+1)\ln(r+1) \in \mathcal{O}(\log \varphi(N) \log\log \varphi(N)) \,.$$

Het algoritme voor (c) geeft ons nu ook een $d \in \mathbb{N}$ met $de \equiv 1 \pmod{\varphi(N)}$, en met $d < \varphi(N)$; dan is $t := de - 1$ een veelvoud is van $\varphi(N)$, kleiner dan $e \cdot \varphi(N)$.

Voor elk natuurlijk getal u tussen 1 en e testen we nu of t/u gelijk zou kunnen zijn aan $\varphi(N)$. Hiertoe moet t/u uiteraard $\in \mathbb{N}$ zijn; indien dit zo is, dan lossen we opnieuw de kwadratische vergelijking

$$x^2 - (N - t/u + 1) \cdot x + N = 0$$

op. Indien beide wortels natuurlijke getallen zijn, stel a en b, dan hebben we N geschreven als $N = ab$. Aangezien het niet mogelijk is dat één van beide gelijk is aan 1, moeten we wel hebben dat $\{a, b\} = \{p, q\}$, en dan is t/u inderdaad gelijk aan $\varphi(N)$. Op die manier hebben we een reductie van (b) naar (c) geconstrueerd. □

Opmerking 4.3.6. Voorgaande stelling impliceert nog *niet* dat het kraken van RSA even moeilijk is als het factoriseren van natuurlijke getallen; er zou immers een methode kunnen bestaan om RSA te kraken die de private key zelf niet nodig heeft.

Hoofdstuk 5

Factorisatie van polynomen

In dit hoofdstuk beschrijven we een aantal algoritmen die ons uiteindelijk in staat stellen polynomen in één veranderlijke over eindige velden, het veld der rationale getallen, en de ring der gehele getallen, te factoriseren.

De factorisatie van een polynoom in één veranderlijke geschiedt in vermelde gevallen altijd in meerdere stappen. Zonder uitzondering wordt een polynoom eerst in kwadraatvrije factoren gefactoriseerd. Kwadraatvrije polynomen over eindige velden kunnen gefactoriseerd worden met het algoritme van Berlekamp, het oudste algoritme dat deze klus in polynomiale tijd klaart. Het algoritme van Zassenhaus, dat een kwadraatvrij polynoom over $\mathbb{Z}$ factoriseert, is een modulair algoritme, en maakt gebruik van het algoritme van Berlekamp.

5.1 Kwadraatvrije factorisatie

Stel dat F een willekeurig veld is. Gegeven een polynoom $f = \sum_{i=0}^{n} a_i x^i$, dan definiëren we de symbolische afgeleide van f als $f' = \sum_{i=1}^{n} i a_i x^{i-1}$. We veronderstellen voor deze sectie steeds dat $f \in F[x]$ een monisch polynoom is, met als factorisatie $f = \prod_{i=1}^{r} f_i^{e_i}$, waarbij de f_i's de verschillende monische factoren zijn en waarbij $e_j \in \mathbb{N} \setminus \{0\}$. Het *kwadraatvrij gedeelte van* f is het polynoom $\prod_{i=1}^{r} f_i$. Het is eenvoudig om aan te tonen dat

$$f' = \sum_{i=1}^{r} e_i \frac{f}{f_i} f_i' \,. \tag{5.1}$$

We zien reeds snel dat uit deze gedaante een verband volgt tussen de afgeleide van een polynoom, en het kwadraatvrij zijn ervan.

Lemma 5.1.1. *Zij $f \in F[x]$ een polynoom met $\gcd(f, f') = 1$. Dan is f kwadraatvrij.*

Bewijs. Als f niet kwadraatvrij is, dan is er in de factorisatie $f = \prod_{i=1}^{r} f_i^{e_i}$ een i met $e_i \geq 2$. Uit vergelijking (5.1) volgt dan dat $f_i \mid \gcd(f, f')$, en bijgevolg $\gcd(f, f') \neq 1$. □

Het is in het algemeen niet waar dat een kwadraatvrij polynoom f voldoet aan $\gcd(f, f') = 1$; zie echter Lemma 5.1.2 en Lemma 5.1.10(ii).

We behandelen de kwadraatvrije factorisatie van polynomen nu eerst voor velden met karakteristiek nul; nadien zullen we perfecte velden in karakteristiek p nader bekijken.

Het volgende lemma geeft meteen aanleiding tot een algoritme om het kwadraatvrij gedeelte te bepalen als $\mathrm{char}(F) = 0$.

Lemma 5.1.2. *Als* $\mathrm{char}(F) = 0$*, dan geldt*

$$\prod_{i=1}^{r} f_i = \frac{f}{\gcd(f, f')},$$

en f *is kwadraatvrij als en slechts als* $\gcd(f, f') = 1$*.*

Bewijs. Elke term in vergelijking (5.1) is deelbaar door $f_i^{e_i}$, behalve misschien de i-de. Deze term, $\frac{e_i f}{f_i} f_i'$, is zeker deelbaar door $f_i^{e_i - 1}$. Deze term is deelbaar door $f_i^{e_i}$ als en slechts als $f_i \mid e_i f_i'$. Dit is onmogelijk want $f_i' \neq 0$ en $\deg(e_i f_i') = \deg f_i' < \deg f_i$. Bijgevolg geldt $\gcd(f, f') = \prod_{i=1}^{r} f_i^{e_i - 1}$, en is het gestelde bewezen. □

Algoritme 5.1 Kwadraatvrij gedeelte in karakteristiek nul

input: $f \in F[x]$, monisch met F een veld en $\mathrm{char}(F) = 0$.
output: het kwadraatvrije gedeelte van f.
1 $u \leftarrow \gcd(f, f')$.
2 **return** $\dfrac{f}{u}$

Stelling 5.1.3. *Algoritme 5.1 bepaalt correct het kwadraatvrij gedeelte van een polynoom* $f \in F[x]$*, met* F *een veld en* $\mathrm{char}(F) = 0$*. Er zijn daartoe* $\mathcal{O}(M(n) \log n)$ *operaties nodig, met* $n = \deg f$*.*

Bewijs. De correctheid volgt onmiddellijk uit Lemma 5.1.2. Uit Stellingen 2.2.5 en 3.6.1 volgt onmiddellijk dat er $\mathcal{O}(\mathsf{M}(n) \log n)$ operaties vereist zijn. □

Beschouw nu een monisch polynoom $f \in F[x] \setminus \{0\}$, F een willekeurig veld van karakteristiek 0. De unieke factorisatie van de vorm

$$f = g_1 g_2^2 \cdots g_m^m,$$

met alle g_i kwadraatvrij, $\gcd(g_i, g_j) = 1$, $i \neq j$, en $g_m \neq 1$ wordt de *kwadraatvrije factorisatie* van f genoemd. We noteren deze ook als $(g_1, \ldots, g_m)$.

Indien men Algoritme 5.1 uitvoert en dan met $u = \gcd(f, f')$ en $v = g_1 g_2 \cdots g_m = f/u$, $g_1 = v/\gcd(u, v)$ berekent, en dan het algoritme opnieuw uitvoert met f vervangen door u en zo recursief verder, dan vinden we de kwadraatvrije factorisatie van f. Het is duidelijk dat deze methode $\mathcal{O}(m\mathsf{M}(n)\log n)$ operaties vereist. Het volgende algoritme is efficiënter.

Algoritme 5.2 Yun's algoritme voor kwadraatvrije factorisatie in karakteristiek nul

---

input: $f \in F[x]$, monisch met F een veld en $\mathrm{char}(F) = 0$, $\deg f \geq 0$.
output: De kwadraatvrije factorisatie van f.

1 $u \leftarrow \gcd(f, f')$, $v_1 \leftarrow \dfrac{f}{u}$, $w_1 \leftarrow \dfrac{f'}{u}$
2 $i \leftarrow 1$
3 **repeat**
4 $\quad h_i \leftarrow \gcd(v_i, w_i - v_i')$, $v_{i+1} \leftarrow \dfrac{v_i}{h_i}$, $w_{i+1} \leftarrow \dfrac{w_i - v_i'}{h_i}$
5 $\quad i \leftarrow i + 1$
6 **until** $v_i = 1$
7 $k \leftarrow i - 1$
8 **return** $(h_1, \ldots, h_k)$

Het volgende lemma zal gebruikt worden om de correctheid aan te tonen.

Lemma 5.1.4. *Stel dat F een veld is, met* $\mathrm{char}(F) = 0$, *$g_1, \ldots, g_m \in F[x]$ monische, kwadraatvrije en paarsgewijze coprieme polynomen, $g = g_1 \cdots g_m$, en $h = \sum_{i=1}^{m} \frac{c_i g_i' g}{g_i}$ met $c_i \in F$. Dan geldt*

$$\gcd(g, h - cg') = \prod_{c_j = c} g_j \,,$$

voor alle $c \in F$.

Bewijs. Het is duidelijk dat

$$g' = \sum_{i=1}^{m} \frac{g_i' g}{g_i} \,, \quad \text{waaruit} \quad h - cg' = \sum_{i=1}^{m} (c_i - c) g_i' \frac{g}{g_i} \,.$$

Het element g_j is een deler van elke term in deze som, behalve misschien de j-de, en $\gcd(g_j, g_j') = \gcd(g_j, g/g_j) = 1$ omdat $\mathrm{char}(F) = 0$, g_j en g kwadraatvrij zijn, en de g_i's copriem zijn. De stelling volgt nu uit

$$\gcd(g_j, h - cg') = \gcd\left(g_j, (c_j - c)g_j'\frac{g}{g_j}\right) = \gcd(g_j, c_j - c)\,. \qquad \square$$

Stelling 5.1.5. *Algoritme 5.2 bepaalt de kwadraatvrije factorisatie van f in $\mathcal{O}(\mathsf{M}(n)\log n)$ operaties, met $n = \deg f$.*

Bewijs. Stel dat $g_1 g_2^2 \ldots g_m^m$ de kwadraatvrije factorisatie is van f. Dan is $u = \gcd(f, f') = g_2 g_3^2 \cdots g_m^{m-1}$. We bewijzen door middel van inductie op i dat

$$h_i = g_i \;\text{ als } i \geq 1, \quad v_{i+1} = \prod_{j=i+1}^{m} g_j, \quad w_{i+1} = \sum_{j=i+1}^{m} (j-i)g_j'\frac{v_{i+1}}{g_j}$$

voor $0 \leq i \leq m$. De eigenschap is duidelijk voor v_1. Voor w_1 volgt deze uit

$$f' = \sum_{j=1}^{m} j\frac{f}{g_j}g_j' = u\sum_{j=1}^{m} j\frac{v_1}{g_j}g_j'\,.$$

Voor $i \geq 1$ geeft Lemma 5.1.4 (met $g = v_i$, $h = w_i$ en $c = 1$) dat $h_i = g_i$. Daaruit volgt $v_{i+1} = \prod_{j=i+1}^{m} g_j$, en

$$\begin{aligned} w_{i+1} &= \frac{\left(\displaystyle\sum_{j=i}^{m}(j-(i-1))g_j'\frac{v_i}{g_j} - \sum_{j=i}^{m} g_j'\frac{v_i}{g_j}\right)}{g_i} \\ &= \sum_{j=i+1}^{m}(j-i)g_j'\frac{v_i}{g_j g_i} = \sum_{j=i+1}^{m}(j-i)g_j'\frac{v_{i+1}}{g_j}\,. \end{aligned}$$

We schatten nu het aantal vereiste operaties. Stel $d_j = \deg g_j$, $1 \leq j \leq m$. Om de berekeningen op lijn 1 uit te voeren, zijn $\mathcal{O}(\mathsf{M}(n)\log n)$ operaties vereist. Er geldt ook $\deg v_i = \sum_{j=i}^{m} d_j$ en $\deg w_i = \deg v_i - 1$. De berekening van de gcd in stap i vereist $\mathcal{O}(\mathsf{M}(\deg v_i)\log n)$, en de beide delingen vereisen $\mathcal{O}(\mathsf{M}(\deg v_i))$ operaties in F. Gebruikmakend van de superlineaire eigenschappen van M vinden we

$$\begin{aligned} \sum_{i=1}^{m}\mathsf{M}(\deg v_i) &\leq \mathsf{M}\Big(\sum_{i=1}^{m}\deg v_i\Big) = \mathsf{M}\Big(\sum_{i=1}^{m}\sum_{j=i}^{m} d_j\Big) \\ &= \mathsf{M}\Big(\sum_{i=1}^{m} i d_i\Big) = \mathsf{M}(n)\,. \end{aligned} \qquad \square$$

Vervolgens behandelen we perfecte velden in karakteristiek p.

Definitie 5.1.6. Zij F een veld. Dan wordt F *perfect* genoemd, als en slechts als ofwel $\mathrm{char}(F) = 0$, ofwel $\mathrm{char}(F) = p > 0$ en $F^p = F$, i.e. elk element van F is te schrijven als de p-de macht van een element van F.

Voorbeeld 5.1.7. Eindige velden zijn steeds perfect. Inderdaad, stel $F = \mathsf{GF}(q)$, $q = p^h$, p priem, $h \geq 1$; dan is $\mathrm{char}(F) = p$. Zij nu $a \in F$ willekeurig; dan is $a = b^p$ voor $b = a^{p^{h-1}}$. Dus F is perfect.

Lemma 5.1.8. *Zij F een perfect veld met $\mathrm{char}(F) = p > 0$, en beschouw een polynoom $f = \sum_{i=0}^{n} a_i x^i \in F[x]$. Dan is $f' = 0$ als en slechts als er een polynoom $g \in F[x]$ bestaat zodat $f = g^p$.*

Bewijs. Als $f = g^p$, dan is uiteraard $f' = 0$. Veronderstel dus omgekeerd dat $f' = 0$; dit impliceert dat p een deler is van alle voorkomende exponenten, en dus

$$f = \sum_{i=0}^{n/p} a_{ip} x^{ip} .$$

Aangezien F perfect is, is elke a_{ip} te schrijven als $a_{ip} = b_i^p$ voor zekere $b_i \in F$, en dus is

$$f = \sum_{i=0}^{n/p} b_i^p x^{ip} = \Big(\sum_{i=0}^{n/p} b_i x^i \Big)^p . \qquad \square$$

Gevolg 5.1.9. *Zij f een veelterm over een perfect veld. Als f irreduciebel is, dan geldt $f' \neq 0$.*

Voor perfecte velden in karakteristiek p vertaalt Lemma 5.1.2 zich als volgt.

Lemma 5.1.10. *Zij F een perfect veld met $\mathrm{char}(F) = p$, en beschouw een polynoom $f \in F[x]$, met factorisatie $f = \prod_{i=1}^{r} f_i^{e_i}$.*

(i) $\dfrac{f}{\gcd(f, f')} = \displaystyle\prod_{e_i f_i' \neq 0} f_i = \prod_{p \nmid e_i} f_i .$

(ii) *f is kwadraatvrij als en slechts als $\gcd(f, f') = 1$.*

(iii) *Stel $u = \gcd(f, f')$ en $v = f/u$, en stel $n = \deg f$. Dan is*

$$\frac{u}{\gcd(u, v^n)} = \prod_{p \mid e_i} f_i^{e_i} .$$

Bewijs. (i) Kies $1 \leq i \leq r$. Zoals reeds opgemerkt (in het bewijs van Lemma 5.1.2) is $f_i^{e_i-1}$ altijd een deler van f' en $f_i^{e_i}$ een deler van f' als en slechts $e_i f_i' = 0$. Het polynoom f_i^2 is dus nooit een deler van $f/\gcd(f,f')$, en f_i is een deler ervan als en slechts als $e_i f_i' \neq 0$.
Merk op dat uit het irreduciebel zijn van de factoren volgt dat $f_i' \neq 0$ voor alle i, wegens Lemma 5.1.8. Bijgevolg is $e_i f_i' = 0$ als en slechts als $p \mid e_i$.

(ii) Als $\gcd(f,f') = 1$ dan volgt uit Lemma 5.1.1 (of uit (i)) dat f kwadraatvrij is. Stel omgekeerd dat f kwadraatvrij is, of dus dat $e_i = 1$ voor $1 \leq i \leq r$. Dan volgt uit (i) dat $f/\gcd(f,f') = f$, en dus is $\gcd(f,f') = 1$.

(iii) Omdat $e_i \leq n$ voor alle $1 \leq i \leq r$, geldt

$$\gcd(u,v^n) = \gcd\left(\prod_{p\nmid e_i} f_i^{e_i-1} \prod_{p\mid e_i} f_i^{e_i}, \prod_{p\nmid e_i} f_i^n\right) = \prod_{p\nmid e_i} f_i^{e_i-1}. \qquad \square$$

Opmerking 5.1.11. De equivalentie in Lemma 5.1.10(ii) is in het algmeen niet geldig voor niet-perfecte velden. Beschouw bijvoorbeeld de transcendente uitbreiding $F = \mathsf{GF}(2)(\alpha)$, en stel $f = 1 + \alpha x^2$; dan is f kwadraatvrij, terwijl toch $\gcd(f,f') \neq 1$. (Merk op dat f niet langer kwadraatvrij is over de algebraïsche sluiting van F.)

Hoewel hetgeen nu volgt, theoretisch mogelijk is voor willekeurige perfecte velden in karakteristiek p, beperken we ons hier toch tot eindige velden, omdat we daarin eenvoudig de p-de machtswortels van een element kunnen bepalen (zie Voorbeeld 5.1.7), terwijl we voor willekeurige perfecte velden niet over een dergelijk algoritme beschikken.

Onze voorgaande beschouwingen leiden ons nu tot het volgend algoritme.

Algoritme 5.3 Kwadraatvrij gedeelte over $\mathsf{GF}(q)$

input: $f \in F[x]$, monisch met $F = \mathsf{GF}(q)$, $\deg f = n \geq 1$.
output: het kwadraatvrije gedeelte van f.

1 $u \leftarrow \gcd(f,f')$, $v \leftarrow \dfrac{f}{u}$, $w \leftarrow \dfrac{u}{\gcd(u,v^n)}$.
2 **if** $w = 1$
3 **then return** v
4 **else** Bepaal het kwadraatvrij gedeelte z van $w^{1/p}$ met dit algoritme
5 **return** vz

Om de complexiteit van dit algoritme te schatten, maken we gebruik van Lemma 5.1.10, en de daar gedefinieerde notaties.

Stelling 5.1.12. *Algoritme 5.3 bepaalt correct het kwadraatvrij gedeelte van een polynoom $f \in \mathsf{GF}(q)[x]$, $q = p^h$, $h \geq 1$. Als q/p klein is in verhouding tot n, zijn daartoe $\mathcal{O}(M(n) \log n)$ operaties nodig.*

Bewijs. De correctheid van het algoritme volgt onmiddellijk uit Lemma 5.1.10.

Het uitvoeren van lijn (1) vergt $\mathcal{O}(\mathsf{M}(n) \log n)$ operaties. Stel $m = \deg w$. Uit Stelling 1.2.4 weten we dat er ten hoogste $2(m/p) \log(q/p)$ operaties in $\mathsf{GF}(q)$ nodig zijn om de m/p (q/p)-de machten te bepalen, nodig voor de oproep in lijn (4). We vinden dus dat $S(n) \in \mathcal{O}(\mathsf{M}(n) \log n)$.

De graad van z in de eerste recursieve oproep is $m/p \leq \lfloor n/p \rfloor$. Voor $T(n)$ hebben we dus de relatie $T(n) \leq T(\lfloor n/p \rfloor) + S(n)$. De functies S en T voldoen dus aan de voorwaarden van Lemma 2.1.2, met $b = 1$. Bijgevolg geldt $T(n) \in \mathcal{O}(S(n))$. □

Ook het algoritme van Yun kan aangepast worden voor polynomen over eindige velden. De observatie dat Lemma 5.1.4 in feite geldig blijft voor perfecte velden, samen met wat extra werk, zal ons het volgende algoritme leveren.

Algoritme 5.4 Yun's algoritme voor kwadraatvrije factorisatie over eindige velden

input: $f \in \mathsf{GF}(q)[x]$, monisch met $\deg f \geq 0$.
output: De kwadraatvrije factorisatie van f.

1 $u \leftarrow \gcd(f, f')$, $v_1 \leftarrow \dfrac{f}{u}$, $w_1 \leftarrow \dfrac{f'}{u}$
2 $i \leftarrow 1$
3 **repeat**
4 $\quad h_i \leftarrow \gcd(v_i, w_i - v_i')$, $v_{i+1} \leftarrow \dfrac{v_i}{h_i}$, $w_{i+1} \leftarrow \dfrac{w_i - v_i'}{h_i}$
5 $\quad i \leftarrow i + 1$
6 **until** $v_i = 1$
7 $k \leftarrow i - 1$
8 $z \leftarrow \dfrac{f}{h_1 h_2^2 \cdots h_k^k}$
9 **if** $z = 1$
10 **then return** $(h_1, \ldots, h_k)$
11 **else** Bepaal de kwaadraatvrije factorisatie $(s_1, \ldots, s_l)$ van $z^{1/p}$ met dit algoritme
12 **for** $i = k+1, \ldots, p-1$ **do** $h_i \leftarrow 1$
13 **for** $i = 1, \ldots, p-1$ en $j = 1, \cdots, l$ **do** $t_{jp+i} \leftarrow \gcd(h_i, s_j)$
14 **for** $j = 1, \ldots, l$ **do** $t_{jp} \leftarrow \dfrac{s_j}{t_{jp+1} t_{jp+2} \cdots t_{(j+1)p-1}}$
15 **for** $i = 1, \ldots, p-1$ **do** $t_i \leftarrow \dfrac{h_i}{t_{p+i} t_{2p+i} \cdots t_{lp+i}}$
16 **return** $(t_1, \ldots, t_r)$ met $r < (l+1)p$ maximaal waarvoor $t_r \neq 1$.

Het bewijs van Lemma 5.1.4 steunt essentieel op het feit dat $\gcd(g_i, g_i') = 1$. Voor kwadraatvrije polynomen over perfecte velden blijft dit geldig. We hebben dus onmiddellijk het volgende lemma.

Lemma 5.1.13. *Stel dat F een eindig veld $\mathsf{GF}(q)$ is, $g_1, \ldots, g_m \in F[x]$ monische, kwadraatvrije en paarsgewijze coprieme polynomen, $g = g_1 \cdots g_m$, en $h = \sum_{i=1}^{m} \frac{c_i g_i' g}{g_i}$ met $c_i \in F$. Dan geldt*

$$\gcd(g, h - cg') = \prod_{c_j = c} g_j \,,$$

voor alle $c \in F$.

Lemma 5.1.14. *Stel dat $f \in \mathsf{GF}(q)[x]$ kwadraatvrije factorisatie $(g_1, \ldots, g_m)$ heeft. Dan bepaalt Algoritme* 5.4

$$h_i = \prod_{j \equiv i \pmod p} g_j, \ 1 \leq i < p \tag{5.2}$$

en $h_i = 1$ voor $i \geq p$.

Bewijs. Volkomen analoog als in Stelling 5.1.5 toont men aan dat

$$h_i = \prod_{j \equiv i \pmod p} g_j \text{ als } i \geq 1, \quad v_{i+1} = \prod_{\mathrm{rem}(j,p) > i}^{m} g_j, \quad w_{i+1} = \sum_{\mathrm{rem}(j,p) > i}^{m} (j-i) g_j' \frac{v_{i+1}}{g_j} .$$

Hiermee volgt (5.2), en ook dat het algoritme zal stoppen met $k < p$. □

Stelling 5.1.15. *Algoritme* 5.4 *bepaalt de kwadraatvrije factorisatie van een polynoom $f \in \mathsf{GF}(q)[x]$, $q = p^h$, p priem.*

Bewijs. Als $m < p$, dan volgt uit Lemma 5.1.14 dat $z = 1$ (lijn (8)) en dat $(h_1, \ldots, h_k)$ de gezochte kwadraatvrije factorisatie is. Stel nu dat $m \geq p$. We gebruiken vergelijking (5.2) (Lemma 5.1.14) om f te schrijven als

$$\begin{aligned} f &= (g_1 g_{p+1}^{p+1} g_{2p+1}^{2p+1} \cdots)(g_2^2 g_{p+2}^{p+2} g_{2p+2}^{2p+2} \cdots) \cdots (g_p^p g_{2p}^{2p} g_{3p}^{3p} \cdots) \\ &= h_1 (g_{p+1} g_{2p+1}^2 \cdots)^p h_2^2 (g_{p+2} g_{2p+2}^2 \cdots)^p \cdots (g_p g_{2p}^2 \cdots)^p , \end{aligned}$$

waaruit

$$z^{\frac{1}{p}} = (g_p g_{p+1} \cdots)(g_{2p} g_{2p+1} \cdots)^2 \cdots ,$$

en

$$s_j = \prod_{jp \leq i < (j+1)p} g_i \quad \text{voor } 1 \leq j \leq l . \tag{5.3}$$

Dus $g_k \mid h_i$ als en slechts als $i = \mathrm{rem}(k,p)$ en $g_k \mid s_j$ als en slechts als $j = \mathrm{quo}(k,p)$. Dus $t_{jp+i} = g_{jp+i}$, voor $1 \leq i < p$ en $1 \leq j \leq l$. Vergelijking (5.3) verklaart ook dat lijn 14 de gezochte t_{jp}, $1 \leq j \leq l$, levert, en vergelijking (5.2) (Lemma 5.1.14(ii)) toont aan dat lijn (15) de gezochte t_i, $1 \leq i < p$, levert. □

Om de complexiteit van het algoritme te berekenen, gebruiken we de volgende stelling zonder bewijs.

Stelling 5.1.16. *Stel F een veld, $f \in F[x]$, $\deg f = n$, en $m_1, \ldots, m_r \in F[x]$ zodat $\sum_{i=1}^{r} \deg m_i \leq n$. De polynomen $\gcd(f, m_1), \ldots, \gcd(f, m_r)$ kunnen samen berekend worden met $\mathcal{O}(\mathsf{M}(n) \log n)$ operaties in F.*

Zonder bewijs. □

Stelling 5.1.17. *Om Algoritme 5.4 uit te voeren, zijn* $\mathcal{O}(\mathsf{M}(n)\log n)$ *operaties nodig in* $\mathsf{GF}(q)$.

Bewijs. Om z te bepalen in lijn 8 zijn $\mathcal{O}(\mathsf{M}(n)\log n)$ operaties in $\mathsf{GF}(q)$ nodig. Om $z^{1/p}$ te bepalen zijn $\mathcal{O}((n/p)\log(q/p)) \subset \mathcal{O}(n)$ operaties nodig. Om de gcds in lijn 13 op efficiënte wijze te berekenen, gaan we als volgt te werk. Eerst wordt $s = s_1 \cdots s_l$ bepaald, een polynoom van graad ten hoogste $\lfloor n/p \rfloor$, hetgeen $\mathcal{O}(\mathsf{M}(\lfloor n/p \rfloor)\log n)$ operaties vergt. De reductie van elke h_i modulo s vergt $\mathcal{O}(\mathsf{M}(\deg h_i))$ operaties, in totaal $\mathcal{O}(\mathsf{M}(n))$ omdat $\sum_{i=1}^{p-1} \deg h_i \leq n$ en de superlineariteit van M. De bepaling van de $\gcd(h_i, s_j) = \gcd(\mathrm{rem}(h_i, s), s_j)$, $1 \leq j \leq l$, vergt $\mathcal{O}(\mathsf{M}(\lfloor n/p \rfloor)\log n)$ operaties (Stelling 5.1.16), en dus $\mathcal{O}(\mathsf{M}(n)\log n)$ voor alle i. De delingen in lijn 14 vergen $\mathcal{O}(\mathsf{M}(\deg s_j))$ operaties en de delingen in lijn 15 vergen $\mathcal{O}(\mathsf{M}(\deg h_i))$ operaties. Alle delingen samen in lijn 14 en 15 vergen dus in totaal $\mathcal{O}(\mathsf{M}(n))$ operaties. De totale kost van de eerste 7 lijnen werd bepaald in Stelling 5.1.5 en bedraagt $\mathcal{O}(\mathsf{M}(n))$ operaties. De kost voor alle lijnen, uitgezonderd de recursieve oproep is dus $\mathcal{O}(\mathsf{M}(n)\log n)$. Zoals in het bewijs van Stelling 5.1.12 volgt dat de totale kost $\mathcal{O}(\mathsf{M}(n)\log n)$ is. □

5.2 Factorisatie van kwadraatvrije polynomen over eindige velden

In deze sectie beschouwen we steeds een *kwadraatvrij* polynoom $f \in \mathsf{GF}(q)[x]$. We bespreken het algoritme van Berlekamp ([Ber67] en [Ber70]), dit is het oudste algoritme dat de factorisatie van f bepaalt in polynomiale tijd. De bepaling van de factorisatie maakt gebruik van lineaire algebra.

Stel dat f monisch en kwadraatvrij is en van graad n. Beschouw het Frobenius automorfisme:

$$\sigma : \left\{ \begin{array}{rcl} \mathsf{GF}(q^n) & \longrightarrow & \mathsf{GF}(q^n) \\ \alpha & \longmapsto & \alpha^q \,. \end{array} \right.$$

Merk op dat σ de Galoisgroep van de velduitbreiding $\mathsf{GF}(q^n)/\mathsf{GF}(q)$ voortbrengt, en dat $\sigma^n = \mathbf{1}$. Beschouw nu de ring $R = \mathsf{GF}(q)[x]/(f)$. Deze ring is een n-dimensionale vectorruimte over $\mathsf{GF}(q)$. De afbeelding

$$\sigma : \left\{ \begin{array}{rcl} R & \longrightarrow & R \\ \alpha & \longmapsto & \alpha^q \,. \end{array} \right.$$

wordt ook het *Frobenius automorfisme* van R genoemd. Als f irreduciebel is, dan is $R \cong \mathsf{GF}(q^n)$, en dan vinden we het klassieke Frobenius automorfisme terug.

Stel dat de factorisatie van f gegeven wordt door $f = f_1 \cdots f_r$, dan leert de Chinese reststelling (Stelling 1.4.2) dat

$$R \cong \mathsf{GF}(q)[x]/(f_1) \times \cdots \times \mathsf{GF}(q)[x]/(f_r)\,. \tag{5.4}$$

In het bijzonder zien we dat de Frobenius afbeelding een automorfisme van R is omdat f kwadraatvrij is.

Beschouw nu de afbeelding $\beta = \sigma - \mathbf{1}$: $R \longrightarrow R$, $\beta(a) = a^q - a$. Deze afbeelding is $\mathsf{GF}(q)$-lineair. We bepalen de kern van β.

Elke component $\mathsf{GF}(q)[x]/(f_i)$ is een veld met $q^{\deg f_i}$ elementen en bevat $\mathsf{GF}(q)$ als een deelveld. Voor elk element $a \in \mathsf{GF}(q)[x]$ hebben we

$$\begin{aligned} a \bmod f \in \ker\beta &\iff a^q \equiv a \bmod f \iff a^q \equiv a \bmod f_i \text{ voor } 1 \le i \le r \\ &\iff a \bmod f_i \in \mathsf{GF}(q) \text{ voor } 1 \le i \le r\,. \end{aligned}$$

Dus is $\mathcal{B} = \ker\beta \cong \mathsf{GF}(q)^r$. In feite is $\mathcal{B}$ een $\mathsf{GF}(q)$-deelalgebra van R, deze wordt ook wel de *Berlekamp deelalgebra* genoemd. Noteren we het isomorfisme in (5.4) met χ, dan geldt

$$a \bmod f \in \ker\beta \iff \chi(a \bmod f) = (a_1 \bmod f_1, \ldots, a_r \bmod f_r)$$

voor welbepaalde $a_1, \ldots, a_r \in \mathsf{GF}(q)$. Merk op dat $a_i = 0$ als en slechts als $f_i \mid a$.

De Frobenius afbeelding in R is een $\mathsf{GF}(q)$-lineaire afbeelding. Ze kan dus voorgesteld worden door een matrix $Q \in \mathsf{GF}(q)^{n\times n}$, ten opzichte van een basis voor de vectorruimte R. Een natuurlijke basis is de verzameling

$$\{1 \bmod f, x \bmod f, \ldots, x^{n-1} \bmod f\,\}.$$

Een basis voor $\mathcal{B}$ bepalen kan gebeuren door Gauss eliminatie toe te passen op de matrix $Q - I$. Merk op dat

$$f \text{ is irreduciebel} \iff r = 1 \iff \operatorname{rang}(Q - I) = n - 1\,.$$

Het algoritme van Berlekamp is een random algoritme. Tijdens de uitvoering zal er een randomgenerator gebruikt moeten worden om willekeurige elementen van een eindig veld te bepalen. Binnen de randomalgoritmen onderscheidt men twee zeer belangrijke klassen.

Een *Monte Carlo algoritme* is een random algoritme dat mogelijks een fout antwoord oplevert. Maar het is een vereiste dat één van de input data

een reëel getal ϵ is, $0 < \epsilon < 1$ en dat de waarschijnlijkheid op een fout antwoord kleiner dan ϵ is voor alle mogelijke waarden van de resterende input data. De performantie van het algoritme zal afhangen van ϵ, een grotere nauwkeurigheid zal resulteren in langere rekentijden.

Een *Las Vegas algoritme* is een algoritme dat *nooit* een fout antwoord oplevert, maar *geen* antwoord (i.e. "gefaald") zal opleveren met probabiliteit ϵ, waarbij deze laatste parameter mogelijks behoort tot de input data. In de praktijk kunnen we een dergelijk algoritme meerdere keren uitvoeren met dezelfde input data, en wachten tot het een antwoord oplevert. Na een aantal iteraties N, waarbij dat aantal afhangt van ϵ, zal het met een aan zekerheid grenzende waarschijnlijkheid $1 - \epsilon^N$ een antwoord geven. In vele implementaties van dergelijke algoritmen gebeurt dit automatisch.

Het algoritme van Berlekamp is een Las Vegas algoritme. Voor we aan de beschrijving beginnen, zullen we eerste enkele lemma's opstellen die ons in staat stellen statistische uitspraken te doen over bepaalde elementen in eindige velden.

Lemma 5.2.1. *Stel $q = p^h$, p een priem en $h \geq 1$, k een deler van $q - 1$, en $S = \{b^k \mid b \in \mathsf{GF}(q)^*\}$ de verzameling van k-de machten in $\mathsf{GF}(q)^*$. Dan geldt*

(i) *S is een deelgroep van orde $(q-1)/k$;*

(ii) *$S = \{a \in \mathsf{GF}(q)^* \mid a^{(q-1)/k} = 1\}$.*

Bewijs. Het gestelde volgt onmiddellijk uit $\mathsf{GF}(q)^* \cong C_{q-1}$. □

We passen Lemma 5.2.1 nu toe voor p oneven en $k = 2$:

Lemma 5.2.2. *Stel $q = p^h$, p een oneven priem en $h \geq 1$. Stel*

$$S = \{a^2 \mid a \in \mathsf{GF}(q)^*\}$$

is de verzameling van kwadraten in $\mathsf{GF}(q)^*$. Dan geldt

(i) *$S \subseteq \mathsf{GF}(q)^*$ is een multiplicatieve deelgroep van orde $(q-1)/2$,*

(ii) *$S = \{a \in \mathsf{GF}(q)^* \mid a^{(q-1)/2} = 1\}$,*

(iii) *$a^{(q-1)/2} \in \{1, -1\}$ voor alle $a \in \mathsf{GF}(q)^*$.*

Met behulp van Lemma 5.2.2 en de eigenschappen van de Berlekamp deelalgebra kunnen we nu het volgende algoritme voor oneven karakteristiek opstellen.

Algoritme 5.5 Algoritme van Berlekamp (oneven karakteristiek)

input: Een monisch kwadraatvrij polynoom $f \in \mathsf{GF}(q)[x]$, q oneven, $\deg f = n \geq 1$
output: Een niet triviale factor g van f of "gefaald"

1 Bepaal $\mathrm{rem}(x^q, f)$ in $\mathsf{GF}(q)[x]$ met behulp van Algoritme 1.5
2 **for** $i = 0, \ldots, n-1$
3 **do** Bepaal $\mathrm{rem}(x^{qi}, f) = \sum_{j=0}^{n} q_{ij} x^j$
4 $Q \leftarrow (q_{ij})_{0 \leq i,j \leq n}$
5 Bepaal een basis voor de kern van $Q - I$, bepaal daarmee $r = \dim \mathcal{B}$, en een basis $\{b_1, \ldots, b_r\}$ voor de Berlekamp deelalgebra met $\deg b_i < n$.
6 Kies onafhankelijke uniform verdeelde random elementen $c_1, \ldots, c_r \in \mathsf{GF}(q)$
7 $a \leftarrow c_1 b_1 + \cdots + c_r b_r$
8 $g_1 \leftarrow \gcd(a, f)$
9 **if** $g_1 \neq 1$
10 **then return** g_1
11 Bepaal $b = a^{(q-1)/2}$ in $\mathsf{GF}(q)[x]/f$ en $\mathrm{rem}(b-1, f)$ met Algoritme 1.5 en 2.8.
12 $g_2 \leftarrow \gcd(b-1, f)$
13 **if** $g_2 \neq 1$ en $g_2 \neq f$
14 **then return** g_2
15 **else return** "gefaald"

Stelling 5.2.3. *Als f reduciebel is dan bepaalt Algoritme 5.5 een niet triviale factor van f of faalt met kans ten hoogste* $1/2$.

(vergelijk deze stelling met Stelling 4.2.8.)

Bewijs. Stel dat $a \in \mathcal{B}$. Als $\gcd(a, f) \neq 1$, dan levert dit onmiddellijk een niet-triviale factor op van f, omdat $\deg a < \deg f$. Merk op dat deze factor niet noodzakelijk irreduciebel is. Als $\gcd(a, f) = 1$, dan volgt dat $f_i \nmid a$ voor alle $1 \leq i \leq r$. Voor een dergelijke $a \in \mathcal{B}$ zullen alle $\chi_i(a)$ in $\chi(a) = (\chi_1(a), \ldots, \chi_r(a))$ verschillend zijn van nul.

Aangezien $a \in \mathcal{B}$, geldt $\chi_i(a) \in \mathsf{GF}(q)$ voor alle $1 \leq i \leq r$. Lemma 5.2.2 impliceert dat $a^e \bmod f_i \in \{-1, 1\}$, voor $e = (q-1)/2$. Stel $b = a^e$, definieer $\chi_i(a^e) = \epsilon_i$, dan is $\chi_i(b-1) = \epsilon_i - 1$. Bijgevolg is

$$\gcd(b-1, f) = \prod_{\epsilon_i = 1} f_i \,.$$

Als alle $\epsilon_i = 1$, dan is $\gcd(b-1, f) = f$ en dus een triviale factor van f; als alle $\epsilon_i = -1$, dan is $\gcd(b-1, f) = 1$, en levert dit eveneens een triviale

factor van f. In alle andere gevallen is $\gcd(b-1, f)$ een echte factor van f. De situatie $\epsilon_1 = \epsilon_2 = \cdots = \epsilon_r$ doet zich voor met probabiliteit $2 \cdot 2^{-r}$; de kans op succes is dus $1 - 2 \cdot 2^{-r} \geq 1/2$. □

Lemma 5.2.2 speelt een belangrijke rol om het algoritme uiteindelijk op te stellen. Als $q = p^h$, met $p = 2$, dan geldt dit lemma echter niet. In essentie zegt het dat de niet nul elementen van een eindig veld van oneven karakteristiek voor de helft uit kwadraten bestaat. In een eindig veld van even karakteristiek zijn alle elementen een kwadraat. Zoals welbekend wordt de rol van de kwadraten en de niet kwadraten in het even geval overgenomen door de elementen van spoor 0 en spoor 1. Het spoor is als volgt gedefinieerd:

$$T : \mathsf{GF}(q) \longrightarrow \mathsf{GF}(2) : x \longmapsto x^{2^{h-1}} + x^{2^{h-2}} + \cdots + x^4 + x^2 + x.$$

Uit de definitie van T volgt er gemakkelijk dat $T(x)^2 + T(x) = 0$ voor elke $x \in \mathsf{GF}(q)$. Daaruit volgt dat $T(x) = 0$ of $T(x) = 1$.

Het volgende lemma stelt ons in staat Algoritme 5.5 te herformuleren voor even q.

Lemma 5.2.4. *Stel $p = 2$, $q = 2^h$ met $h \geq 1$. Stel*

$$S = \{a \in \mathsf{GF}(q) \mid T(a) = 0\}.$$

Dan geldt:

(i) *T is $\mathsf{GF}(2)$-lineair (met $\mathsf{GF}(q)$ gezien als $\mathsf{GF}(2)$-vectorruimte).*

(ii) *T is surjectief op $\mathsf{GF}(2)$.*

(iii) *S is een additieve deelgroep van $\mathsf{GF}(q)$ van orde $q/2$.*

Bewijs. (i) Dit volgt onmiddellijk uit de definitie van T.

(ii) Aangezien $T(0) = 0$ zit 0 al zeker in het beeld van T. De spoorafbeelding wordt gedefinieerd aan de hand van een polynoom van graad $q/2$. Dit betekent dat $T(x)$ hoogstens $q/2$ nulpunten heeft. Het kan dus niet dat alle elementen van $\mathsf{GF}(q)$ spoor nul hebben.

(iii) Dat S een deelgroep is volgt uit (i). Uit de eigenschappen van groepshomomorfismen volgt

$$q = |\ker T| \cdot |\operatorname{im} T| = 2 \cdot |\ker T|. \qquad \square$$

Beschouw nu een willekeurig polynoom $a \in \mathsf{GF}(q)[x]$, $a = \sum_{i=1}^{n} a_i x^i$. We definiëren $T(a) = T(\sum_{i=1}^{n} a_i x^i) = \sum_{i=1}^{n} T(a_i x^i)$. Voor even q vervangen we

in lijn 11 in Algoritme 5.5 $b = a^{(q-1)/2}$ door $b = T(a)$, en $\mathrm{rem}(b-1, f)$ door $\mathrm{rem}(b, f)$.

Algoritme 5.6 Algoritme van Berlekamp (even karakteristiek)

input: Een monisch kwadraatvrij polynoom $f \in \mathsf{GF}(q)[x]$, q oneven, $\deg f = n \geq 1$
output: Een niet triviale factor g van f of "gefaald"
1 Bepaal $\mathrm{rem}(x^q, f)$ in $\mathsf{GF}(q)[x]$ met behulp van Algoritme 1.5
2 **for** $i = 0, \cdots, n-1$
3 **do** Bepaal $\mathrm{rem}(x^{qi}, f) = \sum_{j=0}^{n} q_{ij} x^j$
4 $Q \leftarrow (q_{ij})_{0 \leq i,j \leq n}$
5 Bepaal een basis voor de kern van $Q - I$, bepaal daarmee $r = \dim \mathcal{B}$, en een basis $\{b_1, \ldots, b_r\}$ voor de Berlekamp deelalgebra met $\deg b_i < n$.
6 Kies onafhankelijke uniform verdeelde random elementen $c_1, \ldots, c_r \in \mathsf{GF}(q)$
7 $a \leftarrow c_1 b_1 + \cdots + c_r b_r$
8 $g_1 \leftarrow \gcd(a, f)$
9 **if** $g_1 \neq 1$
10 **then return** g_1
11 Bepaal $b = T(a)$ en $\mathrm{rem}(b, f)$ met Algoritme 1.5 en 2.8.
12 $g_2 \leftarrow \gcd(b, f)$
13 **if** $g_2 \neq 1$ en $g_2 \neq f$
14 **then return** g_2
15 **else return** "gefaald"

Volkomen analoog als Stelling 5.2.3 kunnen we nu de volgende stelling bewijzen.

Stelling 5.2.5. *Algoritme 5.6 bepaalt een niet-triviale factor van f of faalt met kans ten hoogste* $1/2$.

In de analyse van de complexiteit van het algoritme van Berlekamp moeten we enkel voor Lijn (11) een onderscheid maken tussen q oneven en q even. Een belangrijke bijdrage tot de complexiteit wordt geleverd door Lijn (5), waar door Gauss eliminatie een basis bepaald wordt voor $\ker(\beta)$. Het klassieke algoritme voor Gauss eliminatie behoort tot dezelfde complexiteitsklasse als het klassieke algoritme voor matrixvermenigvuldiging, namelijk $\mathcal{O}(n^3)$. Het algoritme van Strassen ([Str69]) voert echter matrixvermenigvuldiging uit in $\mathcal{O}(n^\omega)$ operaties, met $\omega = \log_2 7$. We vermelden hier enkel dat deze ontdekking heeft geleid tot verder geavanceerd onderzoek naar snelle

algoritmen voor matrixvermenigvuldiging. De geïnteresseerde lezer verwijzen we naar [vzGG03]. We gaan er hier van uit dat Gauss eliminatie kan in $\mathcal{O}(n^\omega)$ operaties, met $2 < \omega \leq 3$.

Stelling 5.2.6. *Het algoritme van Berlekamp vereist $\mathcal{O}(n^\omega + n\mathsf{M}(n))$ operaties in $\mathsf{GF}(q)$, op voorwaarde dat lijn 5 kan uitgevoerd worden in $\mathcal{O}(n^\omega)$, $2 < \omega \leq 3$, operaties.*

Bewijs. Lijnen (1) tot en met (4) vergen $\mathcal{O}(n\mathsf{M}(n))$ operaties in $\mathsf{GF}(q)$ (Stelling 2.2.5). De Gauss eliminatie in Lijn (5) vergt $\mathcal{O}(n^\omega)$ operaties in $\mathsf{GF}(q)$. Lijn (7) vergt $\mathcal{O}(nr)$ veldoperaties. Lijnen (8) en (12) vergen $\mathcal{O}(\mathsf{M}(n)\log n)$ veldoperaties. Lijn (11) vergt $\mathcal{O}(\mathsf{M}(n)\log q) + \mathcal{O}(\mathsf{M}(n))$ operaties als q oneven is, en $\mathcal{O}(n) + \mathcal{O}(\mathsf{M}(n))$ operaties als q even is. In elk geval domineren $\mathcal{O}(n\mathsf{M}(n))$ en $\mathcal{O}(n^\omega)$ in deze afschatting. □

Yun's algoritme en het algoritme van Berlekamp stellen ons in staat om de factorisatie van een willekeurig polynoom over $\mathsf{GF}(q)$ te bepalen. Er bestaan nog efficiëntere algoritmen, en we vermelden de volgende stelling zonder bewijs.

Stelling 5.2.7. *De factorisatie van een polynoom $f \in \mathsf{GF}(q)[x]$ kan bepaald worden in $\mathcal{O}(\mathsf{M}(n^2)\log n + \mathsf{M}(n)\log n \log q)$, of nog, $\widetilde{\mathcal{O}}(n^2 + n\log q)$ operaties in $\mathsf{GF}(q)$.*

Zonder bewijs. □

5.3 Een big prime modulair algoritme voor factorisatie in $\mathbb{Z}[x]$

Zoals in Hoofdstuk 3 zullen we nu modulaire algoritmen bestuderen. Stel dat $f \in \mathbb{Z}[x]$ een kwadraatvrij primitief polynoom is. Zij p een priemgetal; we zullen de reductie van f modulo p noteren als $\overline{f}$; zie pagina 63. Stel dat $\overline{f}$ kwadraatvrij is in $\mathsf{GF}(p)[x]$. Met behulp van het algoritme van Berlekamp kunnen we een factorisatie $\overline{f} = \overline{f_1}\cdots\overline{f_k}$ bepalen. Indien g een factor is van f in $\mathbb{Z}[x]$, dan zal $\overline{g} = \overline{f_{i_1}}\cdots\overline{f_{i_s}}$. Indien we de factor g kunnen reconstrueren uit een aantal factoren van $\overline{f}$, dan is het factorisatie probleem in $\mathbb{Z}[x]$ opgelost. Dit is echter niet zo eenvoudig als het lijkt. Immers, een irreduciebele factor g van f in $\mathbb{Z}[x]$ is niet noodzakelijk irreduciebel in $\mathsf{GF}(p)[x]$. Om de factoren van f te bepalen aan de hand van een factorisatie in $\mathsf{GF}(p)[x]$ zullen we de juiste combinaties moeten vinden om de irreduciebele factoren in $\mathbb{Z}[x]$ te vinden. Om het principe om te zetten in een algoritme moeten we de volgende vragen kunnen beantwoorden.

(i) Hoe groot moet het priemgetal p gekozen worden opdat alle factoren van f gereconstrueerd kunnen worden uit hun beeld modulo p? In Hoofdstuk 3 hebben we via de grens van Mignotte hierop reeds een antwoord gevonden.

(ii) Voor welke priemgetallen zal $\overline{f}$ kwadraatvrij zijn? Het antwoord zullen we kunnen afleiden uit de eigenschappen van de resultante ingevoerd in Hoofdstuk 3.

(iii) Hoe vinden we welke factoren in $\mathsf{GF}(p)[x]$ gecombineerd moeten worden om hun oorspronkelijke irreduciebele factor in $\mathbb{Z}[x]$ te vinden? We zullen ons beperken tot de eenvoudigste oplossing: we proberen alle mogelijke combinaties uit. Dat dit niet de meest efficiënte oplossing is zal echter blijken uit een eenvoudig voorbeeld.

Alle factorisatie algoritmen die we in deze en volgende secties zullen beschouwen bestaan uit twee delen: een gedeelte waar factorisatie modulo één of meerdere priemen, of een priemmacht, uitgevoerd wordt; en een gedeelte waar de factoren over $\mathbb{Z}$ gereconstrueerd worden. In deze sectie beschrijven we een algoritme waar factorisatie modulo één priemgetal gebruikt wordt. Het is dus een zogenaamd big prime algoritme. Voor we het antwoord op de drie vragen zoeken, moeten we ook nog opmerken dat de hele operatie zinloos is als $p \mid \mathrm{lc}(f)$, omdat dan $\deg f \neq \deg \overline{f}$ en we dus nooit alle factoren gaan kunnen reconstrueren.

Het antwoord op vraag (ii) kunnen we snel vinden.

Definitie 5.3.1. Stel R is een UFD. De *discriminant* van een polynoom $f \in R[x]$ is gedefinieerd als $\mathrm{disc}(f) = \mathrm{res}(f, f')$.

Lemma 5.3.2. *Stel $f \in \mathbb{Z}[x] \setminus \{0\}$ kwadraatvrij. Stel $p \in \mathbb{N}$ is een priemgetal zodat $p \nmid \mathrm{lc}(f)$. Het polynoom $\overline{f}$ is kwadraatvrij als en slechts als $p \nmid \mathrm{disc}(f)$.*

Bewijs. We hebben de volgende keten van equivalenties:

$$\begin{aligned}
\overline{f} \text{ kwadraatvrij} &\iff \gcd(\overline{f}, \overline{f}') = 1 && \text{(Lemma 5.1.10(ii))} \\
&\iff \mathrm{res}(\overline{f}, \overline{f}') \neq 0 && \text{(Gevolg 3.2.5)} \\
&\iff \overline{\mathrm{res}(f, f')} \neq 0 && \text{(Lemma 3.3.2(i))} \\
&\iff p \nmid \mathrm{res}(f, f') \\
&\iff p \nmid \mathrm{disc}(f)
\end{aligned}$$

□

Gevolg 5.3.3. *Stel $f \in \mathbb{Z}[x] \setminus \{0\}$ kwadraatvrij. Stel $p \in \mathbb{N}$ is een priemgetal zodat $p \nmid \mathrm{disc}(f)$. Dan is $\overline{f}$ kwadraatvrij.*

Bewijs. Uit het feit dat $\mathrm{lc}(f) \mid \mathrm{disc}(f)$. $\square$

Stel nu dat $f = f_1 \cdots f_k$ de factorisatie is van het primitief kwadraatvrij polynoom $f \in \mathbb{Z}[x]$, $\deg f = n$. Kies een priemgetal $p \in \mathbb{N}$ zodat $p \nmid \mathrm{disc}(f)$. Dan geldt

$$\overline{f} = \overline{f_1} \cdots \overline{f_k} = \overline{\mathrm{lc}(f)} g_1 \cdots g_r$$

in $\mathsf{GF}(p)[x]$, met $g_1, \ldots, g_r$ monische irreduciebele factoren van $\overline{f}$ in $\mathsf{GF}(p)[x]$. Veronderstel dat we de factoren g_i berekend hebben door $\overline{f}$ te factoriseren in $\mathsf{GF}(p)[x]$. Stel dat f_1 een irreduciebele factor van f in $\mathbb{Z}[x]$ is en stel $S = \{i \in \{1, \ldots, r\} \text{ zodat } g_i \mid \overline{f_1}\}$. Dan geldt

$$f_1 \frac{\mathrm{lc}(f)}{\mathrm{lc}(f_1)} \equiv \mathrm{lc}(f) \prod_{i \in S} g_i \bmod p\,. \tag{5.5}$$

Als $p/2$ groter is dan $(n+1)^{1/2} 2^n |\mathrm{lc}(f)| \cdot \|f\|_\infty$, dan zijn de coëfficiënten van $\mathrm{lc}(f) f_1 / \mathrm{lc}(f_1)$ gehele getallen in absolute waarde kleiner dan $p/2$ door Gevolg 3.4.8(ii) (toegepast met $h = f_1 \,\mathrm{lc}(f)/\,\mathrm{lc}(f_1)$ en $f = \mathrm{lc}(f) \cdot f$). De equivalentie in Vergelijking (5.5) wordt dan een gelijkheid (als we $\mathsf{GF}(p)$ voorstellen door $\{-(p-1)/2, \ldots, -1, 0, 1, \ldots, (p-1)/2\}$), waardoor we f_1 kunnen reconstrueren uit de g_i's, $i \in S$. Daarmee zijn we dus aanbeland bij vraag (iii): vind de verzameling S. In het volgende algoritme proberen we elke mogelijke deelverzameling voor S van $\{1, \ldots, r\}$.

Algoritme 5.7 Factorisatie in $\mathbb{Z}[x]$ (big prime)

input: een polynoom $f \in \mathbb{Z}[x]$, kwadraatvrij, primitief, $\deg f = n \geq 1$ met $\mathrm{lc}(f) > 0$ en $\|f\|_\infty = A$.
output: de irreducibele factoren $\{f_1, \ldots, f_k\} \subseteq \mathbb{Z}[x]$ van f.

1 **if** $n = 1$ **then return** $\{f\}$
2 $b \leftarrow \mathrm{lc}(f)$, $B \leftarrow (n+1)^{1/2} 2^n A b$
3 **repeat**
4 kies een random priemgetal p, $2B < p < 4B$.
5 $\overline{f} \leftarrow f \bmod p$
6 **until** $\gcd(\overline{f}, \overline{f}') = 1 \in \mathsf{GF}(p)$.
7 Bepaal monische $g_1, \ldots, g_r \in \mathbb{Z}[x]$, $\|g_i\|_\infty < p/2$, $\deg g_i > 0$ waarvoor $f \equiv b g_1 \cdots g_r \bmod p$
8 $T \leftarrow \{1, \ldots, r\}$, $s \leftarrow 1$, $G \leftarrow \emptyset$, $f^* \leftarrow f$
9 **while** $2s \leq |T|$
10 **do for** alle $S \subseteq T$, $|S| = s$
11 **do** bepaal $g^*, h^* \in \mathbb{Z}[x]$, $\|g^*\|_\infty, \|h^*\|_\infty < p/2$ waarvoor $g^* \equiv b \prod_{i \in S} g_i \bmod p$ en $h^* \equiv b \prod_{i \in T \setminus S} g_i \bmod p$
12 **if** $\|g^*\|_1 \|h^*\|_1 \leq B$
13 **then** $T \leftarrow T \setminus S$, $G \leftarrow G \cup \{\mathrm{pp}(g^*)\}$
14 $f^* \leftarrow \mathrm{pp}(h^*)$, $b \leftarrow \mathrm{lc}(f^*)$
15 onderbreek de **for** lus gestart op lijn 10 en ga naar lijn 9.
16 $s \leftarrow s + 1$
17 **return** $G \cup \{f^*\}$.

Stelling 5.3.4. *Algoritme 5.7 bepaalt de irreduciebele factoren van f.*

Bewijs. Merk vooreerst op dat de voorwaarde in lijn 6 ons garandeert dat $p \nmid \mathrm{disc}(f)$, zodat $\overline{f}$ wegens Gevolg 5.3.3 kwadraatvrij zal zijn.

We tonen nu eerst aan dat telkens we in lijn 12 van het algoritme beland zijn, de voorwaarde

$$g^* h^* \equiv b f^* \pmod{p} \qquad (*)$$

vervuld is. De voorwaarde $(*)$ is initieel geldig, omdat dan $f^* = f$, en per definitie van g^* en h^* is dan $g^* h^* \equiv b^2 \prod_{i \in T} g_i \pmod{p}$, terwijl anderzijds in lijn 7 de g_i's precies zo gedefinieerd zijn dat $f \equiv f^* \equiv b \prod_{i \in T} g_i \pmod{p}$.

Merk vervolgens op dat $g^* h^*$ enkel afhangt van b en T, en niet van S. Zowel de verzameling T als het polynoom f^* en de waarde van $b = \mathrm{lc}(f^*)$

wijzigen enkel in lijnen 13–14. Noteer de nieuwe waarden van f^*, g^*, h^* en b als we na deze wijziging weer bij lijn 12 zijn aanbeland, als $\tilde{f}^*$, $\tilde{g}^*$, $\tilde{h}^*$ en $\tilde{b}$, respectievelijk. Dan is $\tilde{g}^*\tilde{h}^* \equiv \tilde{b}^2 \prod_{i\in T\setminus S} g_i \pmod p$. Anderzijds is $\tilde{f}^* = \mathrm{pp}(h^*) \equiv c\prod_{i\in T\setminus S} g_i \pmod p$ voor een zekere constante $c \in \mathsf{GF}(p)$. Aangezien de g_i monisch zijn, volgt hieruit $\mathrm{lc}(\tilde{f}^*) \equiv c \pmod p$, en dus $\tilde{f}^* \equiv \tilde{b}\prod_{i\in T\setminus S} g_i \pmod p$. Dit bewijst dat $(*)$ opnieuw geldig is.

We tonen vervolgens aan dat de voorwaarde in lijn 12 waar is als en slechts als $g^*h^* = bf^*$. Stel dat deze gelijkheid geldig is, dan is $g^*h^* \mid bf$, en dan volgt uit Gevolg 3.4.8 dat $\|g^*\|_1\|h^*\|_1 \le B$. Omgekeerd, veronderstel dat $\|g^*\|_1\|h^*\|_1 \le B$. Dan is

$$\|g^*h^*\|_\infty \le \|g^*h^*\|_1 \le \|g^*\|_1\|h^*\|_1 \le B < p/2\,,$$

en bijgevolg is de congruentie in $(*)$ een gelijkheid, omdat de absolute waarde van elke coëfficiënt van beide leden kleiner is dan $p/2$.

Veronderstel nu dat g een irreduciebele factor van f^* is, die modulo p reduceert in s irreduciebele factoren. Aangezien $\mathsf{GF}(p)[x]$ een UFD is, vormen deze factoren een deelverzameling van $\{g_1, \ldots, g_r\}$. Stel dus $S \subseteq T$ zodanig dat

$$g \equiv \mathrm{lc}(g)\prod_{i\in S} g_i \pmod p\,;$$

we beweren dat dan $g^*h^* = bf^*$, wat betekent dat de voorwaarde in lijn 12 is vervuld, zodat het algoritme de factor g zal vinden. We gaan dit nu na. Stel $h = f^*/g \in \mathbb{Z}[x]$; dan is $\mathrm{lc}(g)\,\mathrm{lc}(h) = \mathrm{lc}(f^*) = b$, zodat

$$\mathrm{lc}(h)g \equiv b\prod_{i\in S} g_i \equiv g^* \pmod p \quad\text{en}\quad \mathrm{lc}(g)h \equiv b\prod_{i\in T\setminus S} g_i \equiv h^* \pmod p\,.$$

De grens van Mignotte (Gevolg 3.4.8) impliceert dat de coëfficiënten van $\mathrm{lc}(h)g$ en $\mathrm{lc}(g)h$ in absolute waarde ten hoogste $B < p/2$ zijn. Omdat ook $\|g^*\|_\infty, \|h^*\|_\infty < p/2$, hebben we dat $\mathrm{lc}(h)g = g^*$, $\mathrm{lc}(g)h = h^*$, en dus $g^*h^* = bf^*$, wat onze bewering bewijst.

Merk vervolgens op dat we bij het doorlopen van het algoritme zeker zijn, als we een factor g van f^* vinden die modulo p reduceert in s factoren, dat deze dan irreduciebel is; zoniet zou deze factor van g op zijn beurt te schrijven zijn als het product van minder dan s factoren modulo p, maar dan hadden we deze al eerder in het algoritme gevonden wegens de redenering van de vorige paragraaf. Bijgevolg is elk polynoom dat we aan G toevoegen, inderdaad irreduciebel en primitief. Het is duidelijk dat ook f^* op elk moment van het algoritme primitief is.

We moeten enkel nog aantonen dat f^* irreduciebel is als $2s > |T|$. Stel dus dat $g \in \mathbb{Z}[x]$ een irreduciebele factor is van f^* met $\deg(g) < \deg(f^*)$, die

modulo p reduceert in $s > |T|/2$ factoren; stel dan $h = f^*/g$. Dan reduceert h modulo p in $|T| - s < |T|/2$ factoren, wat betekent dat het algorimte reeds alle irrecuciebele factoren van h gevonden heeft, maar dan kan h geen deler meer zijn van f^*. Met deze contradictie besluiten we dat in lijn 17 het resterende polynoom f^* inderdaad irreduciebel (en primitief) is. □

Stelling 5.3.5. *Beschouw Algoritme 5.7. Stel $\beta = \log B$, dan geldt $\beta \in \mathcal{O}(n + \log A)$, en de kost van lijnen 3 tot en met 7 is*

$$\mathcal{O}\left(\beta^2 \mathsf{M}(\beta) \log \beta + (\mathsf{M}(n^2) + \mathsf{M}(n)\beta) \log n \cdot \mathsf{M}(\beta) \log \beta\right) \text{ of } \widetilde{\mathcal{O}}(n^3 + \log^3 A)\,.$$

De uitvoering van lijnen 11 tot en met 15 vergt

$$\mathcal{O}((\mathsf{M}(n) \log n + n \log \beta)\mathsf{M}(\beta)) \text{ of } \widetilde{\mathcal{O}}(n^2 + n \log A)$$

operaties, en er zijn ten hoogste 2^{n+1} iteraties.

Zonder bewijs. □

Voorbeeld 5.3.6. Stel $f = 6x^4 + 5x^3 + 15x^2 + 5x + 4 \in \mathbb{Z}[x]$. Het polynoom f is primitief, $n = 4$, $A = 15$, $b = \mathrm{lc}(f) = 6$ en $f' = 24x^3 + 15x^2 + 30x + 5$ en $\mathrm{disc}(f) = 19250814$, dus f is kwadraatvrij; $B = (n+1)^{1/2}2^n Ab = 1440\sqrt{5} \approx 3219,9$. Stel dat $p = 6473 > 2B$ gekozen wordt. Dan $p \nmid b$, en f en f' zijn copriem modulo p. In feite zijn de enige "slechte" priemgetallen (een priemgetal p, $p \mid \mathrm{lc}(f)$ of f mod p is niet kwadraatvrij) de priemdelers van $\mathrm{disc}(f)$, dit zijn $2, 3, 11, 31$ en 97. De factorisatie in lijn 7 wordt

$$f \equiv bg_1g_2g_3g_4 = 6(x - 819)(x + 605)(x + 2632)(x + 2977) \bmod p\,.$$

Voor alle deelverzamelingen met kardinaliteit $s = 1$ is de voorwaarde in lijn 12 vals, en daaruit volgt dat $f^* = f$ geen lineaire factor heeft. Stel $s = 2$ en $S = \{1, 2\}$. We bepalen

$$\begin{aligned} g^* &\equiv bg_1g_2 = 6(x - 819)(x + 605) \equiv 6x^2 - 1284x - 1863 \bmod p\,, \\ h^* &\equiv bg_3g_4 = 6(x + 2632)(x + 2977) \equiv 6x^2 + 1289x - 615 \bmod p \end{aligned}$$

in lijn 11. Het is duidelijk dat $\|g^*\|_1\|h^*\|_1 \geq \|g^*\|_\infty\|h^*\|_\infty = 1863 \cdot 1289 > B$ in lijn 12, en in feite $g^*h^* \neq bf^*$, hetgeen ook duidelijk wordt door de constante coëfficiënten te vergelijken

$$g^*(0)h^*(0) = (-1863) \cdot (-615) \neq 24 = 6 \cdot 4 = bf^*(0)\,.$$

Nemen we $S = \{1, 4\}$, dan vinden we

$$\begin{aligned} g^* &\equiv bg_1g_4 = 6(x - 819)(x + 2977) \equiv 6x^2 + 2x + 2 \bmod p\,, \\ h^* &\equiv bg_2g_3 = 6(x + 605)(x + 2632) \equiv 6x^2 + 3x + 12 \bmod p\,. \end{aligned}$$

In lijn 12 zien we dat $\|g^*\|_1\|h^*\|_1 = 210 < B$, en dus in feite is er gelijkheid $g^*h^* = bf^*$, dus $\text{pp}(g^*) = 3x^2 + x + 1$ en $\text{pp}(h^*) = 2x^2 + x + 4$ zijn de irreduciebele factoren van $f \in \mathbb{Z}[x]$.

Computationeel gezien zit de zwakheid van Algoritme 5.7 in het feit dat er combinaties moeten getest worden van factoren in $\mathsf{GF}(p)$, zonder dat er enige aanwijzing is welke combinaties een factor in $\mathbb{Z}[x]$ reconstrueren. Het volgende voorbeeld is een worst case scenario.

Voorbeeld 5.3.7. Beschouw het i-de *Swinnerton-Dyer* polynoom:

$$f := \prod(x \pm \sqrt{2} \pm \sqrt{3} \pm \sqrt{5} \pm \cdots \pm \sqrt{p_i})$$

met p_i het i-de priemgetal en waarbij het product berekend wordt met alle mogelijke combinaties van de $+$ en de $-$ tekens. Men kan aantonen dat $f \in \mathbb{Z}[x]$ een irreduciebel polynoom is van graad $n = 2^i$. Omdat elk veld $\mathsf{GF}(p^2)$ de elementen $\sqrt{2} \bmod p, \ldots, \sqrt{p_i} \bmod p$, bevat, factoriseert f in lineaire factoren in $\mathsf{GF}(p^2)$. Bijgevolg is elke irreduciebele factor van $\overline{f}$ lineair of kwadratisch. Als $p \nmid \text{disc}(f)$, dan blijkt dat ofwel alle factoren lineair zijn (als $2, 3, \ldots, p_i$ kwadraten modulo p zijn), ofwel alle factoren kwadratisch zijn. Er zijn tenminste $n/2$ factoren, en Algoritme 5.7 moet minstens $2^{n/4}$ deelverzamelingen controleren om te concluderen dat f irreduciebel is.

De criteria opdat een kwadraatvrij polynoom $f \in \mathbb{Z}[x]$ kwadraatvrij blijft modulo p zijn duidelijk. Krachtige stellingen van Frobenius ([Fro96]) en Chebotarëv ([Che26]) (o.a. de beroemde dichtheidsstelling van Chebotarëv) geven aan dat men niet kan stellen dat irreduciebele polynomen over $\mathbb{Z}$, "meestal" irreduciebel blijven modulo p. Het zou ons in deze nota's veel te ver leiden om hierop in te gaan, maar meer achtergrond en een bewijs kan men vinden in [SL96].

5.4 Hensel lifting en factorisatie in $\mathbb{Z}[x]$

In deze sectie leggen we de basis voor de beschrijving van een zogenaamd prime power modulair algoritme om polynomen in $\mathbb{Z}[x]$ te factoriseren. Een groot gedeelte van het werk is reeds gedaan in de theoretische studie van Algoritme 5.7. In deze sectie bespreken we een theoretisch aspect dat rechtstreeks naar het algoritme zal leiden: het zogenaamde "liften" van een factorisatie modulo p naar een factorisatie modulo p^l. We starten de beschrijving met een voorbeeld.

Voorbeeld 5.4.1. Stel R is een ring en stel $f, g, h \in R[x]$, en $m \in R$ zodat $f \equiv gh \bmod m$. We veronderstellen dat er een $s, t \in R[x]$ bestaat waarvoor $sg + th \equiv 1 \bmod m$. Indien $R/(m)$ een veld is, en g, h copriem modulo m, dan kunnen we s en t bepalen met behulp van het uitgebreid Euclidisch algoritme. We berekenen in $R[x]$

$$e = f - gh, \quad \hat{g} = g + te, \quad \hat{h} = h + se \tag{5.6}$$

en vinden

$$\begin{aligned} f - \hat{g}\hat{h} &= f - gh - gse - hte - ste^2 = f - gh - (sg + th)e - ste^2 \\ &= (1 - sg - th)e - ste^2 \equiv 0 \bmod m^2 , \end{aligned}$$

aangezien $e \equiv 0 \bmod m$ en $1 - sg - th \equiv 0 \bmod m$. Bijgevolg geldt $f \equiv \hat{g}\hat{h} \bmod m^2$, zodat $\hat{g}\hat{h}$ een factorisatie is modulo m^2.

Ons doel is om met $m = p$, p een priemelement van R, op een inductieve wijze een factorisatie modulo m te liften naar een factorisatie modulo m^l. Merk op dat we daartoe ook de Bézout relatie $sg + tg \equiv 1$ moeten kunnen meeliften.

We beschouwen echter eerst een tweede voorbeeld, waaruit nog een bijkomend probleem zal blijken.

Voorbeeld 5.4.2. Gegeven de factorisatie $x^4 - 1 \equiv (x - 2)(x^3 + 2x^2 - x - 2) \bmod 5$. We hebben $f = x^4 - 1$, $g = x^3 + 2x^2 - x - 2$ en $h = x - 2$. De polynomen g en h zijn copriem modulo 5. Het uitgebreid algoritme van Euclides levert $s = -2$ en $t = 2x^2 - 2x - 1$ zodat $sg + th \equiv 1 \bmod 5$. We vinden

$$\begin{aligned} e &= f - gh = 5x^2 - 5, \\ \hat{g} &= g + te = 10x^4 - 9x^3 - 13x^2 + 9x + 3, \\ \hat{h} &= h + se = -10x^2 + x + 8, \end{aligned}$$

en in feite

$$f - \hat{g}\hat{h} = 25 \cdot (4x^6 - 4x^5 - 8x^4 + 7x^3 + 5x^2 - 3x - 1) \equiv 0 \bmod 25 ,$$

zodat $f \equiv \hat{g}\hat{h} \bmod 25$.

In dit voorbeeld duikt er een probleem op: de graden van $\hat{g}$ en $\hat{h}$ zijn hoger dan die van g en h, en samen hoger dan die van f. Met het oog op de reconstructie van een factorisatie over $\mathbb{Z}$ vanuit een factorisatie modulo p^l is dit dus een probleem.

We zullen dit probleem oplossen door deling met rest uit te voeren, maar omdat R geen veld is, is dit niet altijd mogelijk. Als de leidende coëfficiënt van het polynoom waardoor we delen, een eenheid is, kunnen we deze deling wel uitvoeren. Meer bepaald kunnen we het volgende lemma aantonen.

Lemma 5.4.3. *Stel R een ring.*

(i) *Stel $f, g \in R[x]$, met g niet nul en monisch. Dan bestaan er unieke polynomen $q, r \in R[x]$ met $f = qg + r$ en $\deg r < \deg g$.*

(ii) *Met f, g, q, r zoals in* (i) *en $f \equiv 0 \bmod m$ voor een $m \in R$, geldt er $q \equiv r \equiv 0 \bmod m$.*

Bewijs. Deel (i) volgt onmiddellijk uit sectie 2.2. Stel $f = mf^*$ voor een bepaald polynoom $f^* \in R[x]$. Uit (i) volgt $f^* = q^*g + r^*$ voor bepaalde $q^*, r^* \in R[x]$ met $\deg r^* < \deg g$. Daaruit volgt dat $f = mf^* = (mq^*)g + mr^*$ en $f = qg + r$ beide een deling met rest voorstellen van f door g. Uit de uniciteit van q en r in (i) volgt dat $q = mq^*$ en $r = mr^*$. □

Merk op dat we de coëfficiënten van de nieuwe polynomen enkel modulo m^2 moeten kennen. Het volgende algoritme voert de lifting-stap uit.

Algoritme 5.8 Hensel step

---

input: Een element $m \in R$, R een ring, en polynomen $f, g, h, s, t \in R[x]$ zodat $f \equiv gh \bmod m$ en $sg + th \equiv 1 \bmod m$
$\mathrm{lc}(f)$ is geen nuldeler modulo m, h is monisch, $\deg f = n = \deg g + \deg h$, $\deg s < \deg h$, en $\deg t < \deg g$.
output: polynomen $g^*, h^*, s^*, t^* \in R[x]$ zodat $f \equiv g^*h^* \bmod m^2$ en $s^*g^* + t^*h^* \equiv 1 \bmod m^2$,
h^* is monisch, $g^* \equiv g \bmod m$, $h^* \equiv h \bmod m$, $s^* \equiv s \bmod m$, $t^* \equiv t \bmod m$, $\deg g^* = \deg g$, $\deg h^* = \deg h$, $\deg s^* < \deg h^*$, $\deg t^* < \deg g^*$.

1 Bepaal $e, q, r, g^*, h^* \in R[x]$ zodat $\deg r < \deg h$ en

$$e \equiv f - gh \pmod{m^2}, \qquad se \equiv qh + r \pmod{m^2}$$
$$g^* \equiv g + te + qg \pmod{m^2}, \qquad h^* \equiv h + r \pmod{m^2}$$

2 Bepaal $b, c, d, s^*, t^* \in R[x]$ zodat $\deg d < \deg h^*$ en

$$b \equiv sg^* + th^* - 1 \pmod{m^2}, \quad sb \equiv ch^* + d \pmod{m^2}$$
$$s^* \equiv s - d \pmod{m^2}, \qquad t^* \equiv t - tb - cg^* \pmod{m^2}$$

3 **return** g^*, h^*, s^*, t^*

Voorbeeld 5.4.4. Stel $m = 5$ en $f, g, h, s, t, e \in \mathbb{Z}[x]$ zoals in Voorbeeld 5.4.2. Deling van se door h levert $q = -10x+5$ en $r = -5$ met $se \equiv qh+r \bmod 25$. Daarenboven berekenen we

$$g^* \equiv g + te + qg \equiv x^3 + 7x^2 - x - 7 \bmod 25,\ h^* \equiv h + r \equiv x - 7 \bmod 25\,.$$

Dan is $f \equiv g^*h^* \bmod 25$, en $\deg g^* = \deg g$ en $\deg h^* = \deg h$. We bepalen nu s^* en t^*. Daartoe berekenen we

$$b \equiv sg^* + th^* - 1 \equiv 5x^3 + 10x^2 + 5x \bmod 25\,.$$

Deling met rest levert $c = -5$ en $d = -10x^3 + 10x + 10$ met $sb \equiv ch^* + d \bmod 25$. Nu is

$$s^* \equiv s - d \equiv 8 \bmod 25,\ t^* \equiv t - tb - cg^* \equiv -8x^2 - 12x - 1 \bmod 25\,.$$

We vinden $s^*g^* + t^*h^* \equiv 1 \bmod 25$ en $\deg s^* = \deg s$ en $\deg t^* = \deg t$. We kunnen dus de volgende iteratie uitvoeren.

Stelling 5.4.5. *Algoritme 5.8 werkt correct zoals gespecifieerd. Voor de ring $R = \mathbb{Z}$ zijn $\mathcal{O}(\mathsf{M}(n)\mathsf{M}(\log m))$ woordoperaties nodig als $m > 1$ en alle inputs maxnorm kleiner dan m^2 hebben.*

Bewijs. We berekenen

$$\begin{aligned} f - g^*h^* &\equiv f - (g + te + qg)(h + se - qh) \\ &\equiv f - gh - (sg + th)e - ste^2 - (sg - th)qe + ghq^2 \\ &\equiv (1 - sg - th)e - ste^2 - (sg - th)qe + ghq^2 \equiv 0 \bmod m^2\,, \end{aligned}$$

omdat door de veronderstelling $1-sg-th \equiv e \equiv 0 \bmod m$ en door Lemma 5.4.3 $q \equiv 0 \bmod m$. Daaruit volgt ook dat $g^* \equiv g \bmod m$ en $h^* \equiv h \bmod m$. Uit de constructie volgt ook dat $\deg r < \deg h$, en bijgevolg is h^* ook monisch en $\deg h^* = \deg h$. Tenslotte impliceert $f \equiv g^*h^* \bmod m^2$ en h^* monisch samen met $\mathrm{lc}(f)$ is geen nuldeler modulo m dat $\deg g^* = \deg f - \deg h^* = \deg f - \deg h = \deg g$. Op een analoge wijze volgen de resultaten voor s^* en t^*. We hebben

$$\begin{aligned} s^*g^* + t^*h^* - 1 &\equiv (s - sb + ch^*)g^* + (t - tb - cg^*)h^* - 1 \\ &= sg^* + th^* - 1 - (sg^* + th^*)b \equiv -b^2 \equiv 0 \bmod m^2\,, \end{aligned}$$

omdat door de veronderstelling $b \equiv 0 \bmod m$. Lemma 5.4.3 impliceert echter dat $c \equiv d \equiv 0 \bmod m$, en bijgevolg $s^* \equiv s \bmod m$ en $t^* \equiv t \bmod m$. Omdat $\deg d < \deg h^*$ hebben we ook dat $\deg s^* < \deg h^*$, en $s^*g^* + t^*h^* \equiv$

$1 \bmod m^2$, samen met het feit dat h^* monisch is impliceert dit $\deg t^* \leq \deg s^* + \deg g^* - \deg h^* < \deg g^*$.

Merk op dat alle polynomen die voorkomen in het algoritme graad kleiner dan $2n$ hebben. De coëfficiënten die voorkomen zijn kleiner dan m^4 en hebben dus woordlengte in $\mathcal{O}(\log m)$. De kost voor een arithmetische operatie in $\mathbb{Z}$ is dus $\mathcal{O}(\mathsf{M}(\log m))$ woordoperaties. Het aantal arithmetische operaties in $\mathbb{Z}$ voor een optelling, vermenigvuldiging of deling met rest van polynomen is $\mathcal{O}(\mathsf{M}(n))$. Hieruit volgt de afschatting van de kost. □

De volgende stelling is een eenvoudig gevolg.

Stelling 5.4.6. *Gegeven een element $p \in R$, $l \in \mathbb{N} \setminus \{0\}$, en gegeven de input van Algoritme 5.8, dan kunnen we polynomen berekenen zoals in de output van Algoritme 5.8, waarbij m^2 vervangen is door p^l.*

Tenslotte bewijzen we de volgende stelling over de uniciteit.

Stelling 5.4.7. *Stel R een ring, $p \in R$ geen nuldeler, $l \in \mathbb{N} \setminus \{0\}$, en $g, h, g^*, h^*, s, t \in R[x]$ niet nul zodat $sg + th \equiv 1 \bmod p$. Stel ook dat $\mathrm{lc}(g)$ en $\mathrm{lc}(h)$ geen nuldelers zijn modulo p, en dat de polynomen g^* en g dezelfde graad en leidende coëfficiënt hebben, en dat ze gelijk zijn modulo p, en analoog voor h en h^*. Als $gh \equiv g^*h^* \bmod p^l$, dan geldt $g \equiv g^* \bmod p^l$ en $h \equiv h^* \bmod p^l$.*

Bewijs. We bewijzen de stelling uit het ongerijmde. Stel dat $g \not\equiv g^* \bmod p^l$ of $h \not\equiv h^* \bmod p^l$, en stel i is maximaal met $1 \leq i < l$ waarvoor p^i een deler is van zowel $g^* - g$ als $h^* - h$. Dus $g^* - g = up^i$ en $h^* - h = vp^i$ voor een bepaalde $u, v \in R[x]$ en $p \nmid u$ of $p \nmid v$. We mogen veronderstellen dat $p \nmid u$. Nu geldt

$$0 \equiv g^*h^* - gh = g^*(h^* - h) + h(g^* - g) = (g^*v + hu)p^i \bmod p^l\,.$$

Aangezien p geen nuldeler is, hebben we $p \mid p^{l-i} \mid (g^*v + hu)$. Zoals gebruikelijk noteren we $s \bmod p$ als $\overline{s}$. Dan geldt $\overline{sg} + \overline{th} = 1$, $\overline{g^*} = \overline{g}$, en $\overline{g^*}\,\overline{v} + \overline{h}\,\overline{u} = 0$. Dus

$$0 = \overline{t}(\overline{g^*}\,\overline{v} + \overline{h}\,\overline{u}) = \overline{t}\,\overline{g}\,\overline{v} + (1 - \overline{sg})\overline{u} = (\overline{t}\,\overline{v} - \overline{su})\overline{g} + \overline{u}\,,$$

en bijgevolg is $\overline{g} \mid \overline{u}$. Aangezien $\mathrm{lc}(g) = \mathrm{lc}(g^*)$ en $\deg g = \deg g^*$, hebben we $\deg \overline{u} \leq \deg u < \deg g = \deg \overline{g}$. Aangezien $\mathrm{lc}(\overline{g}) = \overline{\mathrm{lc}(g)}$ geen nuldelers is, is $\overline{u}$ het nulpolynoom. Dit is in tegenspraak met de veronderstelling $p \nmid u$. □

We komen nu tot een veralgemening van Algoritme 5.8. Daartoe veralgemenen we eerst de notie copriem. Stel dat R een willekeurige ring is, $p \in R$, en $g, h \in R[x]$. De polynomen g en h noemen we *Bézout copriem* modulo p

als er polynomen $s, t \in R[x]$ bestaan zodat $sg + th \equiv 1 \bmod p$. Als $R/(p)$ een veld is, dan zijn g en h copriem in de gebruikelijke zin.

Algoritme 5.9 Multifactor Hensel lifting

input: Een priem element $p \in R$, R een domein, $f \in R[x]$, $\deg f = n \geq 1$, $\mathrm{lc}(f)$ is een eenheid in R modulo p, monische niet constante polynomen $f_1, \ldots, f_r \in R[x]$, paarsgewijze Bezout copriem modulo p
$f \equiv \mathrm{lc}(f) f_1 \cdots f_r \bmod p$, $l \in \mathbb{N}$.
output: Monische polynomen $f_1^*, \cdots, f_r^* \in R[x]$ met
$f \equiv \mathrm{lc}(f) f_1^* \cdots f_r^* \bmod p^l$ en $f_i^* \equiv f_i \bmod p$ voor alle i.

1 **if** $r = 1$ **then** bepaal $f_1^* \in R[x]$ met $f \equiv \mathrm{lc}(f) f_1^* \bmod p^l$ en **return** f_1^*
2 $k \leftarrow \lfloor r/2 \rfloor$, $d \leftarrow \lceil \log l \rceil$.
3 Bepaal $g_0, h_0 \in R[x]$ zodat $g_0 \equiv \mathrm{lc}(f) f_1 \cdots f_k \bmod p$ en $h_0 \equiv f_{k+1} \cdots f_r \bmod p$
4 Bepaal $s_0, t_0 \in R[x]$ zodat $s_0 g_0 + t_0 h_0 \equiv 1 \bmod p$,
$\deg s_0 < \deg h_0$ en $\deg t_0 < \deg g_0$ (Algoritme 1.6).
5 **for** $j = 1, \ldots, d$
6 **do** Stel $m = p^{2^{j-1}}$ en lift de congruenties $f \equiv g_{j-1} h_{j-1}$ en
$s_{j-1} g_{j-1} + t_{j-1} h_{j-1} \equiv 1$ modulo $p^{2^{j-1}}$ naar congruenties
$f \equiv g_j h_j$ en $s_j g_j + t_j h_j \equiv 1$ modulo p^{2^j} (Algoritme 5.8)
7 $g \leftarrow g_d$, $h \leftarrow h_d$
8 Gebruik dit algoritme recursief om $f_1^*, \ldots, f_k^* \in R[x]$ te bepalen
waarvoor $g \equiv \mathrm{lc}(g) f_1^* \cdots f_k^* \bmod p^l$ en $f_i^* \equiv f_i \bmod p$ voor $1 \leq i \leq k$.
9 Gebruik dit algoritme recursief om $f_{k+1}^*, \ldots, f_r^* \in R[x]$ te bepalen
waarvoor $h \equiv f_{k+1}^* \cdots f_r^* \bmod p^l$ en $f_i^* \equiv f_i \bmod p$ voor $k < i \leq r$.
10 **return** $f_1^*, \ldots, f_r^*$

Stelling 5.4.8. *Algoritme* 5.9 *werkt correct zoals gespecifieerd.*

Bewijs. De correctheid volgt onmiddellijk uit Stelling 5.4.5 door inductie op j en r. □

De kost van het algoritme in het geval $R = \mathbb{Z}$ beschrijven we in een stelling zonder bewijs.

Stelling 5.4.9. *Stel $R = \mathbb{Z}$, $p \in \mathbb{N}$ priem, $f \in R[x]$ met $\deg(f) = n$, met $\|f\|_\infty < p^l$ en $\|f_i\|_\infty < p$ voor alle i, dan vereist de uitvoering van Algoritme* 5.9

$$\mathcal{O}\left((\mathsf{M}(n)\mathsf{M}(l\mu) + \mathsf{M}(n)\log n \cdot \mathsf{M}(\mu) + n\mathsf{M}(\mu)\log\mu)\log r\right),$$

of nog, $\widetilde{\mathcal{O}}(nl\mu)$ woordoperaties in $\mathbb{Z}$, met $\mu = \log p$.

We eindigen uiteindelijk met de beschrijving van een algoritme om polynomen in $\mathbb{Z}[x]$ te factoriseren gebruikmakend van Hensel lifting. Zoals reeds vermeld is het grootste gedeelte van het theoretisch werk geleverd door de beschrijving van Algoritme 5.7. Het algoritme werd voor het eerst beschreven door Zassenhaus ([Zas69]).

Algoritme 5.10 Factorisatie in $\mathbb{Z}[x]$ (prime power): algoritme van Zassenhaus

---

input: een polynoom $f \in \mathbb{Z}[x]$, kwadraatvrij, primitief, $\deg f = n \geq 1$ met $\mathrm{lc}(f) > 0$ en $\|f\|_\infty = A$.
output: de irreducibele factoren $\{f_1, \ldots, f_k\} \subseteq \mathbb{Z}[x]$ van f.

1 **if** $n = 1$ **then return** $\{f\}$
2 $b \leftarrow \mathrm{lc}(f)$, $B \leftarrow (n+1)^{1/2} 2^n A b$,
$C \leftarrow (n+1)^{2n} A^{2n-1}$, $\gamma \leftarrow \lceil 2 \log C \rceil$
3 **repeat**
4 kies een random priemgetal p, $p \leq 2\gamma \ln \gamma$
5 $\overline{f} \leftarrow f \bmod p$
6 **until** $p \nmid b$ en $\gcd(\overline{f}, \overline{f}') = 1 \in \mathsf{GF}(p)$.
7 $l \leftarrow \lceil \log_p(2B+1) \rceil$
8 Bepaal monische $h_1, \ldots, h_r \in \mathbb{Z}[x]$, $\|h_i\|_\infty < p/2$, $\deg h_i > 0$ waarvoor $f \equiv b h_1 \cdots h_r \bmod p$
9 Bepaal een factorisatie $f \equiv b g_1 \cdots g_r \bmod p^l$, alle $g_i \in \mathbb{Z}[x]$ monisch, $\|g_i\|_\infty < p^l/2$ zodat $g_i \equiv h_i \bmod p$ voor $1 \leq i \leq r$ (Algoritme 5.9).
10 $T \leftarrow \{1, \ldots, r\}$, $s \leftarrow 1$, $G \leftarrow \emptyset$, $f^* \leftarrow f$
11 **while** $2s \leq |T|$
12 **do for** alle $S \subseteq T$, $|S| = s$
13 **do** bepaal $g^*, h^* \in \mathbb{Z}[x]$, $\|g^*\|_\infty, \|h^*\|_\infty < p^l/2$ waarvoor
$g^* \equiv b \prod_{i \in S} g_i \bmod p^l$ en $h^* \equiv b \prod_{i \in T \setminus S} g_i \bmod p^l$
14 **if** $\|g^*\|_1 \|h^*\|_1 \leq B$
15 **then** $T \leftarrow T \setminus S$, $G \leftarrow G \cup \{\mathrm{pp}(g^*)\}$
16 $f^* \leftarrow \mathrm{pp}(h^*)$, $b \leftarrow \mathrm{lc}(f^*)$
17 onderbreek de **for** lus gestart op lijn 12 en ga naar lijn 11.
18 $s \leftarrow s + 1$
19 **return** $G \cup \{f^*\}$.

Stelling 5.4.10. *Algoritme 5.10 werkt correct zoals gespecifieerd.*

Bewijs. Merk vooreerst op dat de voorwaarden in lijn 6 ons garanderen dat $p \nmid \operatorname{disc}(f)$, zodat $\overline{f}$ wegens Gevolg 5.3.3 kwadraatvrij zal zijn.

We tonen nu eerst aan dat telkens we in lijn 14 van het algoritme beland zijn, de voorwaarde

$$g^*h^* \equiv bf^* \pmod{p^l} \qquad (*)$$

vervuld is. De voorwaarde $(*)$ is initieel geldig, omdat dan $f^* = f$, en per definitie van g^* en h^* is dan $g^*h^* \equiv b^2 \prod_{i\in T} g_i \pmod{p^l}$, terwijl anderzijds in lijn 9 de g_i's precies zo gedefinieerd zijn dat $f \equiv f^* \equiv b \prod_{i\in T} g_i \pmod{p^l}$.

Merk vervolgens op dat g^*h^* enkel afhangt van b en T, en niet van S. Zowel de verzameling T als het polynoom f^* en de waarde van $b = \operatorname{lc}(f^*)$ wijzigen enkel in lijnen 15–16. Noteer de nieuwe waarden van f^*, g^*, h^* en b als we na deze wijziging weer bij lijn 14 zijn aanbeland, als $\tilde{f}^*$, $\tilde{g}^*$, $\tilde{h}^*$ en $\tilde{b}$, respectievelijk. Dan is $\tilde{g}^*\tilde{h}^* \equiv \tilde{b}^2 \prod_{i\in T\setminus S} g_i \pmod{p^l}$. Anderzijds is $\tilde{f}^* = \operatorname{pp}(h^*) \equiv c \prod_{i\in T\setminus S} g_i \pmod{p^l}$ voor een zekere constante $c \in \mathbb{Z}/(p^l\mathbb{Z})$. Aangezien de g_i monisch zijn, volgt hieruit $\operatorname{lc}(\tilde{f}^*) \equiv c \pmod{p^l}$, en dus $\tilde{f}^* \equiv \tilde{b} \prod_{i\in T\setminus S} g_i \pmod{p^l}$. Dit bewijst dat $(*)$ opnieuw geldig is.

We tonen vervolgens aan dat de voorwaarde in lijn 14 waar is als en slechts als $g^*h^* = bf^*$. Stel dat deze gelijkheid geldig is, dan is $g^*h^* \mid bf$, en dan volgt uit Gevolg 3.4.8 dat $\|g^*\|_1\|h^*\|_1 \leq B$. Omgekeerd, veronderstel dat $\|g^*\|_1\|h^*\|_1 \leq B$. Dan is

$$\|g^*h^*\|_\infty \leq \|g^*h^*\|_1 \leq \|g^*\|_1\|h^*\|_1 \leq B < p^l/2\,,$$

en bijgevolg is de congruentie in $(*)$ een gelijkheid, omdat de absolute waarde van elke coëfficiënt van beide leden kleiner is dan $p^l/2$.

Veronderstel nu dat g een irreduciebele factor van f^* is, die modulo p reduceert in s irreduciebele factoren. Aangezien $\mathsf{GF}(p)[x]$ een UFD is, vormen deze factoren een deelverzameling van $\{h_1, \ldots, h_r\}$. Stel dus $S \subseteq T$ zodanig dat

$$g \equiv \operatorname{lc}(g) \prod_{i\in S} h_i \pmod{p}\,.$$

we beweren dat dan $g^*h^* = bf^*$, wat betekent dat de voorwaarde in lijn 14 is vervuld, zodat het algoritme de factor g zal vinden. We gaan dit nu na. Stel $h = f^*/g \in \mathbb{Z}[x]$; dan is $\operatorname{lc}(g)\operatorname{lc}(h) = \operatorname{lc}(f^*) = b$, zodat

$$\operatorname{lc}(h)g \equiv b \prod_{i\in S} h_i \equiv g^* \pmod{p} \quad \text{en} \quad \operatorname{lc}(g)h \equiv b \prod_{i\in T\setminus S} h_i \equiv h^* \pmod{p}\,.$$

De deelverzameling S is gekend, dus kunnen de polynomen g^* en h^* bepaald worden waarvoor

$$g^* \equiv b\prod_{i\in S} g_i \bmod p^l \text{ en } h^* \equiv b\prod_{i\in T\setminus S} g_i \bmod p^l\,,$$

en $g^* \equiv g \pmod p$ doordat $g_i \equiv h_i \pmod p$.

Dus $bf^* \equiv \mathrm{lc}(h)g\cdot\mathrm{lc}(g)h \bmod p^l$ en $bf^* \equiv g^*h^* \bmod p^l$ zijn beide liftingen van dezelfde factorisatie van bf^* modulo p. Stelling 5.4.7 impliceert dat $\mathrm{lc}(h)g \equiv g^* \bmod p^l$ en $\mathrm{lc}(g)h \equiv h^* \bmod p^l$.

De grens van Mignotte (Gevolg 3.4.8) impliceert dat de coëfficiënten van $\mathrm{lc}(h)g$ en $\mathrm{lc}(g)h$ in absolute waarde ten hoogste $B < p^l/2$ zijn. Omdat ook $\|g^*\|_\infty, \|h^*\|_\infty < p^l/2$, hebben we dat $\mathrm{lc}(h)g = g^*$, $\mathrm{lc}(g)h = h^*$, en dus $g^*h^* = bf^*$, wat onze bewering bewijst.

Merk vervolgens op dat we bij het doorlopen van het algoritme zeker zijn, als we een factor g van f^* vinden die modulo p reduceert in s factoren, dat deze dan irreduciebel is; zoniet zou deze factor van g op zijn beurt te schrijven zijn als het product van minder dan s factoren modulo p, maar dan hadden we deze al eerder in het algoritme gevonden wegens de redenering van de vorige paragraaf. Bijgevolg is elk polynoom dat we aan G toevoegen, inderdaad irreduciebel en primitief. Het is duidelijk dat ook f^* op elk moment van het algoritme primitief is.

We moeten enkel nog aantonen dat f^* irreduciebel is als $2s > |T|$. Stel dus dat $g \in \mathbb{Z}[x]$ een irreduciebele factor is van f^* met $\deg(g) < \deg(f^*)$, die modulo p reduceert in $s > |T|/2$ factoren; stel dan $h = f^*/g$. Dan reduceert h modulo p in $|T| - s < |T|/2$ factoren, wat betekent dat het algorimte reeds alle irrecuciebele factoren van h gevonden heeft, maar dan kan h geen deler meer zijn van f^*. Met deze contradictie besluiten we dat in lijn 19 het resterende polynoom f^* inderdaad irreduciebel (en primitief) is. □

We merken nog op dat het Algoritme van Zassenhaus een vergelijkbare complexiteit heeft als Algoritme 5.7. De winst zit in het feit dat het priemgetal p veel kleiner is, zodat de factorisatie over $\mathsf{GF}(p)$ minder woordoperaties vereist. Anderzijds heeft dit algoritme ook dezelfde computationele moeilijkheden: het is nog steeds mogelijk dat alle combinaties van factoren over $\mathsf{GF}(p)$ moeten getest worden om dan uiteindelijk te besluiten dat het polynoom f irreduciebel is.

5.5 Opmerkingen

Wat is het verschil tussen de factorisatie van een polynoom in $\mathbb{Z}[x]$ en de factorisatie van hetzelfde polynoom in $\mathbb{Q}[x]$? Stel dat $f \in \mathbb{Z}[x]$ een primitief polynoom is en stel dat een factorisatie in $\mathbb{Q}[x]$ gegeven wordt door

$f = f_1 \cdots f_k$, $f_i \in \mathbb{Q}[x]$ en irreduciebel. Alle factoren vermenigvuldigen met de gemeenschappelijke noemer en dan de content verwijderen levert een factorisatie $f = f_1^* \cdots f_k^*$, $f_i^* \in \mathbb{Z}[x]$, irreduciebel. Anderzijds is elke factorisatie over $\mathbb{Z}$ in irreduciebele factoren dat ook over $\mathbb{Q}$. Voor een willekeurig polynoom $f \in \mathbb{Z}[x]$ wordt een factorisatie bepaald door de factorisatie van het primitief gedeelte in $\mathbb{Z}[x]$ en de factorisatie van de content in $\mathbb{Z}$. Bijgevolg is factorisatie in $\mathbb{Z}[x]$ gelijkwaardig met factorisatie in $\mathbb{Q}[x]$ samen met factorisatie in $\mathbb{Z}$.

De modulaire algoritmen uit Hoofdstuk 5 kunnen net zoals de modulaire algoritmen uit Hoofdstuk 3 aangepast worden om een polynoom $f \in F[x, y]$, F een veld, te factoriseren. De max-norm wordt dan vervangen door de graad in y van de polynomen. Dit leidt dan zelfs tot een aantal vereenvoudigingen, net zoals in de modulaire algoritmen voor de gcd van polynomen in $F[x, y]$.

Een fundamentele aanpak van de computationele zwakheid in Algoritmen 5.7 en 5.10 wordt geleverd door het *basis reductie algoritme* van Lenstra, Lenstra en Lovász ([LLL82]). Dit algoritme is, kort samengevat, de discrete versie van het Gram Schmidt orthogonalisatie algoritme. Door polynomen te associëren met elementen in $\mathbb{Z}^n$, en gebruik te maken van het basis reductie algoritme, kunnen op een efficiëntere wijze factoren van f bepaald worden in $\mathbb{Z}$ telkens als er een factor modulo p gereconstrueerd kan worden. De meest efficiënte combinatie van het basis reductie algoritme en het Algoritme 5.10 vindt men in [vH02]. De geïnteresseerde lezer verwijzen we naar [vzGG03].

Hoofdstuk

6 Gröbnerbasissen

6.1 Polynoomidealen

In dit hoofdstuk zullen we een methode invoeren om te rekenen met idealen in veeltermringen. Tevens zullen we kort een aantal toepassingen bespreken, die doen aanvoelen waarom dit van belang kan zijn. Een standaard referentie voor het materiaal in dit hoofdstuk is het boek van Buchberger en Winkler [BW98].

Definitie 6.1.1. (i) We herhalen dat I een *ideaal* van een ring R wordt genoemd, notatie $I \trianglelefteq R$, indien I een additieve deelgroep van R is zodat voor alle $a \in I$ en alle $r \in R$ eveneens $ar \in I$.

(ii) Zij $a_1, \ldots, a_n \in R$. Het ideaal *voortgebracht door* $a_1, \ldots, a_n$, notatie $(a_1, \ldots, a_n)$, is het ideaal

$$a_1R + \cdots + a_nR = \{a_1r_1 + \cdots + a_nr_n \mid r_1, \ldots, r_n \in R\}\,.$$

(iii) Indien $I \trianglelefteq R$ een ideaal van R is, en $I = (a_1, \ldots, a_n)$ voor zekere $a_1, \ldots, a_n$, dan wordt het stel $\{a_1, \ldots, a_n\}$ een *voortbrengende verzameling* voor I genoemd. In de computeralgebra wordt dit ook wel een *basis* voor I genoemd.

Opmerking 6.1.2. De naam "basis" is verwarrend, en heeft heel andere eigenschappen dan een basis van een vectorruimte. Zo is er immers geen "onafhankelijkheid" vereist tussen de elementen, en in het bijzonder is het aantal elementen in een basis niet uniek bepaald door het ideaal.

In dit hoofdstuk zal R steeds een polynomenring zijn, dus $R = F[x_1, \ldots, x_n]$, waarbij F een veld is, en $x_1, \ldots, x_n$ variabelen zijn.

Definitie 6.1.3. Beschouw s polynomen $f_1, \ldots, f_s \in R$, en zij I het ideaal voortgebracht door deze polynomen. Dan is *de variëteit van* I de verzameling

$$\begin{aligned} V(I) &:= \{u \in F^n \mid f(u) = 0 \text{ voor alle } f \in I\} \\ &= \{u \in F^n \mid f_i(u) = 0 \text{ voor alle } i \in \{1, \ldots, s\}\}\,. \end{aligned}$$

Een aantal natuurlijke en interessante vragen dringen zich op:

- Is $V(I) \neq \emptyset$?
- Hoe "groot" is $V(I)$?
- Membership problem: gegeven een $f \in R$, zit $f \in I$?
- Trivialiteit: is $I = R$?

Deze problemen zijn uiteraard niet onafhankelijk van elkaar. Als $I = R$, dan is in het bijzonder $1 \in I$, zodat de elementen van $V(I)$ moeten voldoen aan de vergelijking $1 = 0$. Dus $I = R$ impliceert $V(I) = \emptyset$. Indien F algebraïsch gesloten is, is ook het omgekeerde hiervan waar.

Stelling 6.1.4 (Hilberts Nullstellensatz)**.** *Zij F een algebraïsch gesloten veld, $R = F[x_1, \ldots, x_n]$ een polynomenring over F, en $I \trianglelefteq R$ een ideaal in R. Dan is $V(I) = \emptyset$ als en slechts als $I = R$.*

Zonder bewijs. □

Aan de hand van een voorbeeld laten we zien dat de voorwaarde dat F algebraïsch gesloten moet zijn, niet kan weggelaten worden.

Voorbeeld 6.1.5. Zij $R = \mathbb{R}[x, y]$, beschouw de polynomen

$$f(x, y) = x^2 + y^2 - 1 \quad \text{en} \quad g(x, y) = y - 2\,,$$

en zij $I = (f, g) \trianglelefteq R$. De nulpunten van f vormen een cirkel met straal 1 rond de oorsprong; de nulpunten van g vormen een rechte die deze cirkel niet snijdt. Bijgevolg is $V(I) = \emptyset$. Nochtans is I een echt ideaal van R. Merk op dat dezelfde polynomen over $\mathbb{C}[x, y]$ wel gemeenschappelijke nulpunten hebben, met name $(\pm\sqrt{3}i, 2)$.

Merk op dat het oplossen van bovenstaande vragen eenvoudig is in het geval $n = 1$, omdat $F[x]$ een Euclidisch domein is, en dus in het bijzonder is het een hoofdideaaldomein. Bijgevolg is $(f_1, \ldots, f_n) = (\gcd(f_1, \ldots, f_n))$, en we hebben $f \in (g)$ als en slechts als $g \mid f$. Uiteraard gaat dit niet meer op indien $n > 1$; zo is $\gcd(x, y) = 1$ in $F[x, y]$, terwijl (x, y) duidelijk verschillend is van (1); het is zelfs geen hoofdideaal.

Met behulp van Gröbnerbasissen zal het mogelijk blijken om sommige van deze ogenschijnlijk verloren gegane eigenschappen toch te kunnen herstellen.

6.2 Monomiale orderelaties en multivariate deling met rest

We willen een analogon van de deling met rest introduceren voor polynomen met meerdere variabelen. Om dat mogelijk te maken, moeten we kunnen zeggen wat we bedoelen met de "leidende term" van een dergelijk polynoom. We moeten dus een orderelatie vinden op een multivariate polynomenring.

Definitie 6.2.1. (i) Een *partiële orderelatie* "$<$" op een verzameling S is een irreflexieve en transitieve relatie, met andere woorden, een relatie die voldoet aan $a \not< a$, en $a < b$ samen met $b < c$ impliceert $a < c$, voor alle $a, b, c \in S$. In het bijzonder is $a < b$ samen met $b < a$ onmogelijk. We gebruiken de notatie "$a \leq b$" om aan te duiden dat $a = b$ of $a < b$.

(ii) Een partiële orderelatie is *totaal*, indien voor alle $a, b \in S$ geldt dat ofwel $a < b$, ofwel $b < a$.

(iii) Een totale orderelatie is *welgeordend* indien elke niet-ledige deelverzameling T van S een kleinste element heeft, d.w.z. een element $a \in T$ zodat voor alle $b \in T$ geldt dat $a = b$ of $a < b$.

Zo zijn de gewone orderelaties "$<$" voor de verzamelingen $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ en $\mathbb{R}_{\geq 0}$ totale orderelaties, maar enkel de eerste is welgeordend. De inclusie $\subsetneq$ op de verzameling $2^{\mathbb{N}}$ van deelverzamelingen van $\mathbb{N}$ is een partiële orderelatie die niet totaal is.

Notatie 6.2.2. Indien $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$, dan zullen we $\mathbf{x}^\alpha$ schrijven voor het monoom $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. We zullen overigens dikwijls α en $\mathbf{x}^\alpha$ identificeren met elkaar.

Definitie 6.2.3. Een *monomiale orderelatie* voor $R = F[x_1, \ldots, x_n]$ is een relatie $\prec$ op $\mathbb{N}^n$ zodat

(i) $\prec$ is een totale orderelatie;

(ii) $\alpha \prec \beta$ impliceert $\alpha + \gamma \prec \beta + \gamma$ voor alle $\gamma \in \mathbb{N}^n$;

(iii) $\prec$ is welgeordend.

Met behulp van de identificatie die we ingevoerd hebben in Notatie 6.2.2, geeft dit dus een orderelatie op de monomen van R. Merk op dat het kleinste element van deze orderelatie steeds het element $\{0\}^n \in \mathbb{N}^n$ is (m.a.w. het monoom 1). We geven nu drie standaard voorbeelden van monomiale orderelaties op R.

Definitie 6.2.4. (i) *Lexicografische orderelatie*:

$$\alpha \prec_{\text{lex}} \beta \iff \text{de meest linkse niet-nul coördinaat van } \beta - \alpha \text{ is positief.}$$

(ii) *Gegradeerde lexicografische orderelatie*:

$$\alpha \prec_{\text{grlex}} \beta \iff \sum_{i=1}^{n}(\beta_i - \alpha_i) > 0 \text{ of } \Big(\sum_{i=1}^{n}(\beta_i - \alpha_i) = 0 \text{ en } \alpha \prec_{\text{lex}} \beta\Big).$$

(iii) *Gegradeerde omgekeerde lexicografische orderelatie*:

$$\alpha \prec_{\text{grevlex}} \beta \iff \sum_{i=1}^{n}(\beta_i - \alpha_i) > 0 \text{ of } \Big(\sum_{i=1}^{n}(\beta_i - \alpha_i) = 0 \text{ en de meest rechtse niet-nul coördinaat van } \beta - \alpha \text{ is negatief}\Big).$$

Het (eenvoudige) bewijs dat dit monomiale orderelaties zijn, laten we hier achterwege. Merk op dat voor elk van deze drie relaties $\prec$ geldt dat de variabelen (dus de monomen van graad 1) geordend worden als $x_1 \succ x_2 \succ \cdots \succ x_n$. De term "gegradeerd" verwijst naar het feit dat de totale graad $\sum_{i=1}^{n} \alpha_i$ het belangrijkste criterium is voor de orderelatie. Merk verder ook nog op dat voor $n = 1$ deze drie relaties samenvallen.

Zodra we een monomiale orderelatie hebben voor R, kunnen we deze gebruiken om de termen van een polynoom te sorteren, waarbij we de "grootste" termen eerst zetten.

Voorbeeld 6.2.5. Beschouw $f = 4xyz^2 + 4x^3 - 5y^4 + 7xy^2z \in \mathbb{Q}[x, y, z]$. (We gebruiken meestal x, y, z in plaats van x_1, x_2, x_3 indien $n \leq 3$.) Dan zijn de monomiale vectoren die met elk van de 4 termen van f corresponderen, respectievelijk gelijk aan $(1, 1, 2)$, $(3, 0, 0)$, $(0, 4, 0)$ en $(1, 2, 1)$. We verkrijgen

$$\begin{aligned} f &= 4x^3 + 7xy^2z + 4xyz^2 - 5y^4 && \text{geordend volgens } \prec_{\text{lex}}; \\ f &= 7xy^2z + 4xyz^2 - 5y^4 + 4x^3 && \text{geordend volgens } \prec_{\text{grlex}}; \\ f &= -5y^4 + 7xy^2z + 4xyz^2 + 4x^3 && \text{geordend volgens } \prec_{\text{grevlex}}. \end{aligned}$$

Definitie 6.2.6. Zij $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha \mathbf{x}^\alpha \in R$ een niet-nul polynoom met alle $c_\alpha \in F$ (met slechts een eindig aantal niet-nul coëfficiënten), en zij $\prec$ een gegeven monomiale orderelatie voor R.

(i) Elke $c_\alpha \mathbf{x}^\alpha$ met $c_\alpha \neq 0$ wordt een *term* van f genoemd.

(ii) De *multigraad* van f is $\text{mdeg}(f) = \max_{\prec}\{\alpha \in \mathbb{N}^n \mid c_\alpha \neq 0\}$.

(iii) De *leidende coëfficiënt* van f is $\mathrm{lc}(f) = c_{\mathrm{mdeg}(f)} \in F \setminus \{0\}$.

(iv) Het *leidend monoom* van f is $\mathrm{lm}(f) = \mathbf{x}^{\mathrm{mdeg}(f)} \in R$.

(v) De *leidende term* van f is $\mathrm{lt}(f) = \mathrm{lc}(f) \cdot \mathrm{lm}(f) \in R$.

Lemma 6.2.7. *Zij $R = F[x_1, \ldots, x_n]$ een polynomenring, en $\prec$ een monomiale orderelatie voor R. Voor elke $f, g \in R^*$ hebben we*

(i) $\mathrm{mdeg}(fg) = \mathrm{mdeg}(f) + \mathrm{mdeg}(g)$*;*

(ii) *als* $f + g \neq 0$*, dan is* $\mathrm{mdeg}(f + g) \preceq \max\{\mathrm{mdeg}(f), \mathrm{mdeg}(g)\}$*, met gelijkheid als* $\mathrm{mdeg}(f) \neq \mathrm{mdeg}(g)$*;*

(iii) $\mathrm{lt}(fg) = \mathrm{lt}(f) \cdot \mathrm{lt}(g)$.

Bewijs. Deze eigenschappen zijn duidelijk voldaan indien f en g monomen zijn; toon zelf aan dat hieruit volgt dat ze ook voldaan zijn voor willekeurige polynomen f en g. □

We willen nu een algoritme opstellen voor deling met rest in R. Gegeven polynomen $f, f_1, \ldots, f_s \in R$, dan zoeken we $q_1, \ldots, q_s, r \in R$ zodanig dat

$$f = q_1 f_1 + \cdots + q_s f_s + r\,.$$

Voor we het algoritme opstellen, geven we twee voorbeelden die aantonen wat er anders is dan in het univariate geval.

Voorbeeld 6.2.8. Zij $\prec = \prec_{\mathrm{lex}}$, $f = xy^2+1$, $f_1 = xy+1$, $f_2 = y+1$. Wanneer we f eerst door f_1 delen, krijgen we $f = y \cdot (xy+1) - 1 \cdot (y+1) + 2$; wanneer we echter f eerst door f_2 delen, krijgen we $f = (xy - x) \cdot (y + 1) + (x + 1)$. We hebben in geen enkele zin uniciteit van de bekomen quotiënten en rest.

Voorbeeld 6.2.9. Zij $\prec = \prec_{\mathrm{lex}}$, $f = x^2y + xy^2 + y^2$, $f_1 = xy - 1$, $f_2 = y^2 - 1$. We voeren de deling stap voor stap uit, door telkens de leidende term van f te (proberen) delen door de leidende term van f_1 of f_2. We krijgen

$$\begin{aligned} f &= x \cdot (xy - 1) + xy^2 + x + y^2 \\ &= x \cdot (xy - 1) + y \cdot (xy - 1) + \underbrace{x + y^2 + y}_{\text{rest}}\,. \end{aligned}$$

Nu is de leidende term x niet langer deelbaar door xy of y^2, dus x zal in elk geval een restterm zijn. Maar dan gaat de deling weer verder:

$$f = (x + y) \cdot (xy - 1) + (x) + (y^2 - 1) + (y + 1)\,.$$

In het resterend polynoom $y+1$ is geen van beide termen nog deelbaar door xy of y^2, dus hebben we de deling volledig uitgevoerd. We bekomen dus uiteindelijk

$$f = (x+y)\cdot(xy-1) + 1\cdot(y^2-1) + \underbrace{x+y+1}_{\text{rest}}\,.$$

Met deze voorbeelden in het achterhoofd kunnen we nu een algoritme opstellen voor multivariate deling met rest.

Algoritme 6.1 Multivariate deling met rest

input: niet-nul polynomen $f, f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$;
een monomiale orderelatie $\prec$ voor R.
output: polynomen $q_1, \ldots, q_s, r \in R$ zodat $f = q_1 f_1 + \cdots + q_s f_s + r$,
en geen enkel monoom van r deelbaar door $\mathrm{lt}(f_1), \ldots, \mathrm{lt}(f_s)$.

```
1  r ← 0
2  p ← f
3  for i = 1, ..., s
4        do q_i ← 0
5  while p ≠ 0
6        do if lt(f_i) | lt(p) voor een i ∈ {1, ..., s}
7              then kies zo een i
8                   q_i ← q_i + lt(p)/lt(f_i)
9                   p ← p − lt(p)/lt(f_i) · f_i
10             else r ← r + lt(p)
11                  p ← p − lt(p)
12 return q_1, ..., q_s, r
```

De volgende stelling impliceert de correctheid van dit algoritme.

Stelling 6.2.10. *Telkens als Algoritme 6.1 de while-lus* 6–11 *doorloopt, zijn de volgende eigenschappen geldig.*

(i) $f = p + q_1 f_1 + \cdots + q_s f_s + r$;

(ii) $\mathrm{mdeg}(p) \preceq \mathrm{mdeg}(f)$;

(iii) *als* $q_i \neq 0$, *dan* $\mathrm{mdeg}(q_i f_i) \preceq \mathrm{mdeg}(f)$ *voor elke* $i \in \{1, \ldots, s\}$;

(iv) *geen enkele term in* r *is deelbaar door eender welke* $\mathrm{lt}(f_i)$.

Bovendien geldt:

(v) *het algoritme eindigt na een eindig aantal stappen.*

Bewijs. (i) Initieel is $p = f$, $r = 0$ en alle $q_i = 0$, zodat (i) inderdaad geldt. Het is duidelijk dat in het **then**-geval 8–9 de uitdrukking $p + q_i f_i$ onveranderd blijft, en dat in het **else**-geval 10–11 de uitdrukking $p + r$ niet verandert. In beide gevallen zal dus $p + q_1 f_1 + \cdots + q_s f_s + r$ invariant blijven, en dus gelijk blijven aan f.

(ii+v) Initieel is $p = f$, dus in het bijzonder $\operatorname{mdeg}(p) = \operatorname{mdeg}(f)$. Wegens Lemma 6.2.7(iii) is

$$\begin{aligned} \operatorname{lt}\big(\operatorname{lt}(p)/\operatorname{lt}(f_i) \cdot f_i\big) &= \operatorname{lt}\big(\operatorname{lt}(p)/\operatorname{lt}(f_i)\big) \cdot \operatorname{lt}(f_i) \\ &= \operatorname{lt}(p)/\operatorname{lt}(f_i) \cdot \operatorname{lt}(f_i) \\ &= \operatorname{lt}(p)\,. \end{aligned} \tag{6.1}$$

Bijgevolg is $\operatorname{mdeg}\big(p - \operatorname{lt}(p)/\operatorname{lt}(f_i) \cdot f_i\big) \prec \operatorname{mdeg}(p)$, zodat bij elke stap in het **then**-geval de multigraad van p kleiner wordt. In het **else**-geval volgt dit op analoge wijze uit $\operatorname{lt}(\operatorname{lt}(p)) = \operatorname{lt}(p)$.
Hieruit volgt enerzijds dat $\operatorname{mdeg}(p)$ steeds kleiner blijft dan $\operatorname{mdeg}(f)$, waarmee (ii) bewezen is; anderzijds volgt hieruit dat het algoritme eindigt, wegens het welgeordend zijn van de monomiale orderelatie $\preceq$, waarmee we dus ook (v) bewezen hebben.

(iii) Uit vergelijking (6.1) weten we dat $\operatorname{mdeg}(\operatorname{lt}(p)/\operatorname{lt}(f_i) \cdot f_i) = \operatorname{mdeg}(p)$, en uit (ii) weten we dat $\operatorname{mdeg}(p) \preceq \operatorname{mdeg}(f)$. Dit samenvoegend zien we dat elke term t die in regel 8 aan q_i wordt toegevoegd, voldoet aan $\operatorname{mdeg}(tf_i) \preceq \operatorname{mdeg}(f)$.

(iv) Dit volgt onmiddellijk uit het feit dat de termen in r allen worden opgebouwd in regel 10, die zich enkel voordoet indien $\operatorname{lt}(f_i) \nmid \operatorname{lt}(p)$ voor alle $i \in \{1, \ldots, s\}$. □

Merk op dat dit algoritme niet deterministisch is, omwille van de keuze in regel 7. Om het algoritme deterministisch te maken, kunnen we steeds de kleinste i kiezen.

Notatie 6.2.11. Gegeven niet-nul polynomen $f, f_1, \ldots, f_s \in R$, dan noteren we het resultaat r van Algoritme 6.1, waarbij we in regel 7 telkens de kleinst mogelijke waarde voor i kiezen, als $f \operatorname{rem} (f_1, \ldots, f_s)$.

Het volgend voorbeeld toont echter aan dat dit niet volstaat om het membership problem te kunnen beantwoorden.

Voorbeeld 6.2.12. Stel $f = xy^2 - x$, $f_1 = xy + 1$ en $f_2 = y^2 - 1$, en stel $\prec = \prec_{\text{lex}}$. Als we het algoritme volgen, bekomen we als resultaat van de deling

met rest $f = y \cdot f_1 + 0 \cdot f_2 + (-x - y)$; in het bijzonder zien we dat $r \neq 0$. Echter, indien we niet eerst de kleinst mogelijke i kiezen in regel 8, verkrijgen we $f = 0 \cdot f_1 + x \cdot f_2 + 0$, zodat nochtans $f \in (f_1, f_2)$.

Onze bedoeling zal zijn om een speciale basis te vinden voor een willekeurig ideaal, zodanig dat de deling met rest met betrekking tot deze basis uniek is, en in het bijzonder dus het membership problem correct kan beantwoorden. Op het eerste gezicht is het helemaal niet duidelijk of er wel een dergelijke basis bestaat.

6.3 Monomiale idealen en Hilberts basisstelling

Definitie 6.3.1. Een *monomiaal ideaal* $I \trianglelefteq R$ is een ideaal voortgebracht door monomen in R, zodat er dus een deelverzameling $A \subseteq \mathbb{N}^n$ bestaat met

$$I = (\mathbf{x}^A) = (\mathbf{x}^\alpha \mid \alpha \in A)\,.$$

We bewijzen eerst enkele eigenschappen voor monomiale idealen.

Lemma 6.3.2. *Zij $I = (\mathbf{x}^A) \trianglelefteq R$ een monomiaal ideaal, en zij $\beta \in \mathbb{N}^n$. Dan is $\mathbf{x}^\beta \in I$ als en slechts als er een $\alpha \in A$ bestaat zodat $\mathbf{x}^\alpha \mid \mathbf{x}^\beta$.*

Bewijs. Het "als" gedeelte is triviaal. Veronderstel omgekeerd dat $\mathbf{x}^\beta \in I$, en schrijf $\mathbf{x}^\beta = \sum_i q_i \mathbf{x}^{\alpha_i}$, met $q_i \in R$ en $\alpha_i \in A$ voor elke i. Elke monoom in het rechterlid (na uitwerking) is deelbaar door een $\mathbf{x}^{\alpha_i}$. Aangezien het linkerlid een monoom is, moet het gelijk zijn aan één van die termen van het rechterlid, en is dus deelbaar door een $\mathbf{x}^{\alpha_i}$. □

Het volgend lemma toont aan dat het membership problem heel eenvoudig is voor monomiale idealen.

Lemma 6.3.3. *Zij $I \trianglelefteq R$ een monomiaal ideaal van R, en zij $f \in R$. Dan zijn de volgende uitspraken equivalent.*

(a) *$f \in I$;*

(b) *elke term van f zit in I;*

(c) *f is een F-lineaire combinatie van monomen in I.*

In het bijzonder zijn twee monomiale idealen gelijk aan en elkaar als en slechts als ze de zelfde monomen bevatten.

Bewijs. De implicaties (b) $\Rightarrow$ (c) en (c) $\Rightarrow$ (a) zijn triviaal, zelfs voor een willekeurig ideaal $I \trianglelefteq R$. Veronderstel dus dat $f \in I$, en schrijf $f = \sum_i q_i \mathbf{x}^{\alpha_i}$, met $q_i \in R$ en $\alpha_i \in A$ voor elke i. Elk monoom in het rechterlid (na uitwerking) is deelbaar door een $\mathbf{x}^{\alpha_i}$, en zit bijgevolg in I. Dus (a) $\Rightarrow$ (b). $\square$

Stelling 6.3.4 (Lemma van Dickson)**.** *Elk monomiaal ideaal is voortgebracht door een eindige verzameling monomen. Meer bepaald bestaat er voor elke $A \subseteq \mathbb{N}^n$ een eindige deelverzameling $B \subseteq A$ zodat $(\mathbf{x}^A) = (\mathbf{x}^B)$.*

Bewijs. Zoals we zullen zien, is het bewijs van deze stelling volledig combinatorisch. De bewering is triviaal indien $A = \emptyset$, dus we veronderstellen dat $A \neq \emptyset$. We definiëren nu een relatie "$\leq$" op $\mathbb{N}^n$ als

$$\alpha \leq \beta \iff \alpha_i \leq \beta_i \text{ voor alle } i \in \{1, \ldots, n\},$$

voor alle $\alpha = (\alpha_1, \ldots, \alpha_n)$ en $\beta = (\beta_1, \ldots, \beta_n)$ in $\mathbb{N}^n$; deze relatie is zodanig dat $\alpha \leq \beta$ als en slechts als $\mathbf{x}^\alpha \mid \mathbf{x}^\beta$. We schrijven zoals gewoonlijk $\alpha < \beta$ als $\alpha \leq \beta$ en $\alpha \neq \beta$. Dan is $<$ een partiële orderelatie op $\mathbb{N}^n$ (die niet totaal is zodra $n \geq 2$).

Stel nu $B = \{\alpha \in A \mid \text{voor alle } \beta \in A \text{ is } \beta \not< \alpha\}$ gelijk aan de verzameling van minimale elementen van A (met betrekking tot $\leq$). We beweren nu:

(a) B is een eindige deelverzameling van A;

(b) voor elke $\alpha \in A$ bestaat er een $\beta \in B$ met $\beta \leq \alpha$.

Uit deze beweringen volgt de stelling, want $\beta \leq \alpha$ als en slechts als $\mathbf{x}^\beta \mid \mathbf{x}^\alpha$ als en slechts als $\mathbf{x}^\alpha \in (\mathbf{x}^\beta)$, en dus impliceert (b) dat $(\mathbf{x}^A) \subseteq (\mathbf{x}^B)$; de andere inclusie volgt trivialerwijze uit $B \subseteq A$.

We tonen nu eerst bewering (b) aan. Voor elke $\alpha \in \mathbb{N}^n$ bestaan er slechts eindig veel elementen $\beta \in \mathbb{N}^n$ met $\beta \leq \alpha$, en dus bestaat er geen oneindige dalende keten van elementen $\alpha^{(1)} > \alpha^{(2)} > \ldots$ in $\mathbb{N}^n$. In het bijzonder is er voor elke $\alpha \in A$ een minimaal element $\beta \in B$ zodat $\beta \leq \alpha$.

Tenslotte tonen we bewering (a) aan; we gebruiken hiervoor inductie op n. Indien $n = 1$, dan is $<$ een totale orderelatie, en dan bestaat B uit het unieke kleinste element van A. Stel dus $n \geq 2$, en stel

$$A^* = \{(\alpha_1, \ldots, \alpha_{n-1}) \in \mathbb{N}^{n-1} \mid \text{er bestaat een } \alpha_n \in \mathbb{N} \text{ zodat } (\alpha_1, \ldots, \alpha_{n-1}, \alpha_n) \in A\}.$$

Door de inductiehypothese is de verzameling B^* van minimale elementen van A^* eindig. Voor elke $\beta = (\beta_1, \ldots, \beta_{n-1}) \in B^*$ kiezen we een $b_\beta \in \mathbb{N}$ zodat $(\beta_1, \ldots, \beta_{n-1}, b_\beta) \in A$, en we stellen $b = \max\{b_\beta \mid \beta \in B^*\}$. We beweren nu:

$$\text{voor elke } (\alpha_1, \ldots, \alpha_n) \in B \text{ is } \alpha_n \leq b. \tag{6.2}$$

Indien deze bewering voldaan is, dan is de laatste coördinaat van de elementen van B naar boven begrensd; uiteraard kunnen we deze redenering herhalen voor elke andere coördinaat. Hieruit besluiten we dan dat B slechts eindig veel elementen kan bevatten.

Er rest ons nog enkel bewering (6.2) aan te tonen. Veronderstel dus dat $\alpha = (\alpha_1, \ldots, \alpha_n) \in A$ zo is dat $\alpha_n > b$; we moeten aantonen dat $\alpha \notin B$. Inderdaad, er bestaat een element $\beta = (\beta_1, \ldots, \beta_{n-1}) \in B^*$ waarvoor $\beta \leq (\alpha_1, \ldots, \alpha_{n-1})$, en aangezien $b_\beta \leq b < \alpha_n$ impliceert dit

$$A \ni (\beta_1, \ldots, \beta_{n-1}, b_\beta) < (\alpha_1, \ldots, \alpha_{n-1}, \alpha_n) = \alpha\,,$$

zodat α niet minimaal is in A, dus $\alpha \notin B$. □

Definitie 6.3.5. Voor elke deelverzameling $G \subseteq R$ verschillend van $\emptyset$ en $\{0\}$, definiëren we $\mathrm{lt}(G) = \{\mathrm{lt}(g) \mid g \in G\}$.

Als $I \trianglelefteq R$ een ideaal is, dan is er wegens het lemma van Dickson een eindige deelverzameling $G \subset I$ waarvoor $(\mathrm{lt}(G)) = (\mathrm{lt}(I))$. Een dergelijke verzameling G blijkt steeds een basis te zijn voor het ideaal I zelf, zoals we nu aantonen.

Lemma 6.3.6. *Zij $I \trianglelefteq R$ een ideaal in $R = F[x_1, \ldots, x_n]$. Als $G \subset I$ een eindige deelverzameling is van I zodanig dat $(\mathrm{lt}(G)) = (\mathrm{lt}(I))$, dan is $(G) = I$.*

Bewijs. Stel $G = \{f_1, \ldots, f_s\}$. Als f een willekeurig polynoom in I is, dan geeft deling met rest dat

$$f = q_1 f_1 + \cdots + \cdots + q_s f_s + r\,,$$

met $q_1, \ldots, q_s, r \in R$, en zodanig dat ofwel $r = 0$, ofwel is geen enkele term van r deelbaar door een $\mathrm{lt}(f_i)$. Maar $r = f - q_1 f_1 - \cdots - q_s f_s \in I$, en dus is $\mathrm{lt}(r) \in \mathrm{lt}(I) \subseteq (\mathrm{lt}(G))$, i.e. het monoom $\mathrm{lt}(r)$ is bevat in het monomiaal ideaal $(\mathrm{lt}(G))$. Lemma 6.3.2 impliceert nu dat $\mathrm{lt}(r)$ toch deelbaar zou zijn door een $\mathrm{lt}(f_i)$, tenzij $r = 0$, en dus is $f \in (f_1, \ldots, f_s) = (G)$. □

Uiteindelijk bekomen we het volgend belangrijk resultaat.

Stelling 6.3.7 (Hilberts basisstelling [Hil90])**.** *Elk ideaal I in $R = F[x_1, \ldots, x_n]$ is eindig voortgebracht. Meer bepaald bestaat er een eindige deelverzameling $G \subseteq I$ zodat $(G) = I$ en $(\mathrm{lt}(G)) = (\mathrm{lt}(I))$.*

Bewijs. Dit volgt uit Dickson's lemma 6.3.4 toegepast op $(\mathrm{lt}(I))$, samen met Lemma 6.3.6. □

Een onmiddellijk gevolg hiervan is het volgende.

Gevolg 6.3.8. *Zij F een veld; dan is de polynomenring $F[x_1, \ldots, x_n]$ een noetherse ring.*

Bewijs. Zij $I_1 \subseteq I_2 \subseteq \ldots$ een stijgende keten van idealen in R; we moeten aantonen dat de keten stabiliseert. Stel daartoe $I = \bigcup_{j \geq 1} I_j$; dan is I een ideaal, dat door Hilberts basisstelling 6.3.7 eindig voortgebracht is, stel $I = (f_1, \ldots, f_s)$. Zij dan $n = \min\{j \geq 1 \mid f_1, \ldots, f_s \in I_j\}$; dan is $I_n = I_{n+1} = \cdots = I$. □

We merken op dat het wel kan gebeuren dat een eindige verzameling G wel I voortbrengt, maar dat $(\mathrm{lt}(G))$ toch een eigenlijk deelideaal is van $(\mathrm{lt}(I))$, zoals in het volgend voorbeeld.

Voorbeeld 6.3.9. Stel $f = x^3 - 2xy$ en $g = x^2y - 2y^2 + x$ in $\mathbb{Q}[x, y]$, en zij $\prec = \prec_{\mathrm{grlex}}$. Beschouw $G = \{f, g\}$ en $I = (G)$. Dan is $x^2 = -y \cdot f + x \cdot g$, zodat $x^2 \in (\mathrm{lt}(I))$, maar $x^2 \notin (\mathrm{lt}(G)) = (x^3, x^2y)$.

We komen verder nog op dit voorbeeld terug.

6.4 Gröbnerbasissen en S-polynomen

Hilberts basisstelling motiveert de volgende definitie.

Definitie 6.4.1. Zij $\prec$ een monomiale orderelatie op de veeltermring R, en zij $I \trianglelefteq R$ een ideaal. Een eindige deelverzameling $G \subseteq I$ is een *Gröbnerbasis* van I (met betrekking tot $\prec$) als $(\mathrm{lt}(G)) = (\mathrm{lt}(I))$.

Lemma 6.3.6 zegt precies dat elke Gröbnerbasis G voor I inderdaad een basis is voor I, i.e. dat $(G) = I$. Met de conventie dat $() = (\emptyset) = \{0\}$, impliceert Hilberts basisstelling onmiddellijk het volgend resultaat.

Gevolg 6.4.2. *Elk ideaal $I \trianglelefteq R = F[x_1, \ldots, x_n]$ heeft een Gröbnerbasis.*

Bewijs. Onmiddellijk uit Hilberts basisstelling 6.3.7. □

Voor we overgaan tot een constructie van Gröbnerbasissen, tonen we eerst het belang ervan aan, door na te gaan dat deling met rest met betrekking tot een Gröbnerbasis een oplossing levert voor het membership problem.

Lemma 6.4.3. *Zij G een Gröbnerbasis voor een ideaal $I \trianglelefteq R$, en zij $f \in R$. Dan bestaat er een uniek polynoom $r \in R$ zodat*

(i) $f - r \in I$;

(ii) *geen enkele term van* r *is deelbaar door een monoom van* $\mathrm{lt}(G)$.

Bewijs. Het bestaan van een dergelijke r volgt uit het algoritme voor deling met rest, meer bepaald uit Stelling 6.2.10. Veronderstel nu dat $f = h_1 + r_1 = h_2 + r_2$ met $h_1, h_2 \in I$ en geen enkele term van r_1 noch r_2 deelbaar door een element van $\mathrm{lt}(G)$. Dan is $r_1 - r_2 = h_2 - h_1 \in I$. Veronderstel nu dat $r_1 \neq r_2$, zodat $\mathrm{lt}(r_1 - r_2) \in (\mathrm{lt}(I))$; dan is, wegens Lemma 6.3.2, $\mathrm{lt}(r_1 - r_2)$ deelbaar door $\mathrm{lt}(g)$ voor een zekere $g \in G$. Aangezien elke term van $r_1 - r_2$ afkomstig is uit r_1 of r_2 (of beide), bekomen we een contradictie; bijgevolg moet $r_1 = r_2$. □

In het bijzonder zal de rest r bij deling van f door G niet afhangen van de volgorde waarin we de elementen van G delen. We kunnen dus r noteren als $f \operatorname{rem} G$.

Stelling 6.4.4. *Zij* G *een Gröbnerbasis voor een ideaal* $I \trianglelefteq R$ *met betrekking tot een monomiale orderelatie* $\prec$*, en zij* $f \in R$*. Dan is* $f \in I$ *als en slechts als* $f \operatorname{rem} G = 0$.

Bewijs. Dit volgt onmiddellijk uit Lemma 6.4.3. □

Opmerking 6.4.5. Het omgekeerde van Stelling 6.4.4 is ook waar, in de volgende betekenis. Stel $R = F[x_1, \ldots, x_n]$, en $G = \{f_1, \ldots, f_s\} \subseteq R$. Als $f \operatorname{rem} (f_1, \ldots, f_s) = 0$ voor alle $f \in I = (G)$ (in de betekenis van Notatie 6.2.11), dan is G een Gröbnerbasis voor I. We zullen echter verderop een veel sterkere versie hiervan bewijzen; zie Stelling 6.4.8.

Stelling 6.4.4 lost het membership problem op: om te controleren of een gegeven $f \in R$ ook in I zit, moeten we een Gröbnerbasis G kiezen voor I, en controleren of $f \operatorname{rem} G = 0$. Echter, het bewijs van Hilberts basisstelling is niet constructief: het vertelt ons niet hoe we een Gröbner basis kunnen berekenen voor een gegeven ideaal I vertrekkende van een (willekeurige) basis G voor I.

Om een Gröbnerbasis te kunnen construeren, moeten we eerst begrijpen wat er voor een willekeurige basis precies misloopt om een Gröbnerbasis te zijn. Eén van de mogelijkheden is dat een monomiale combinatie $a\mathbf{x}^\alpha f + b\mathbf{x}^\beta g$ van twee polynomen $f, g \in G$ met $a, b \in F$ en $\alpha, \beta \in \mathbb{N}^n$ een polynoom zou kunnen geven waarvan de leidende term niet deelbaar is door een monoom in $\mathrm{lt}(G)$, doordat de leidende termen tegenover elkaar wegvallen; zie Voorbeeld 6.3.9. Dit zullen we nu formeel vast leggen.

Definitie 6.4.6. Zij $f, g \in R \setminus \{0\}$, stel $\mathrm{mdeg}(f) = \alpha = (\alpha_1, \ldots, \alpha_n)$ en $\mathrm{mdeg}(g) = \beta = (\beta_1, \ldots, \beta_n)$, en zij $\gamma = (\max\{\alpha_1, \beta_1\}, \ldots, \max\{\alpha_n, \beta_n\})$. Het *S-polynoom* van f en g is

$$S(f,g) = \frac{\mathbf{x}^\gamma}{\mathrm{lt}(f)} \cdot f - \frac{\mathbf{x}^\gamma}{\mathrm{lt}(g)} \cdot g \in R\,.$$

Merk op dat

$$\mathrm{mdeg}\, S(f,g) \prec \gamma = \mathrm{mdeg}\big(\mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))\big)\,, \tag{6.3}$$

aangezien de leidende termen van beide termen van $S(f,g)$ precies tegenover elkaar wegvallen.

Het is duidelijk dat $S(f,g) = -S(g,f)$, en aangezien $\mathbf{x}^\gamma / \mathrm{lt}(f)$ en $\mathbf{x}^\gamma / \mathrm{lt}(g)$ tot R behoren, is $S(f,g) \in (f,g)$. In Voorbeeld 6.3.9 hebben we

$$S(f,g) = \frac{x^3y}{x^3} \cdot f - \frac{x^3y}{x^2y} \cdot g = -x^2\,.$$

Het volgend lemma zegt dat wanneer we bij een monomiale combinatie van polynomen in G de leidende termen tegenover elkaar wegvallen, dat dit dan noodzakelijk van S-polynomen komt.

Lemma 6.4.7. *Zij $f_1, \ldots, f_s \in R$, $\alpha_1, \ldots, \alpha_s \in \mathbb{N}^n$, $c_1, \ldots, c_s \in F^*$,*

$$f = \sum_{i=1}^{s} c_i\, \mathbf{x}^{\alpha_i}\, f_i \in R\,,$$

en $\delta \in \mathbb{N}^n$ zodanig dat $\alpha_i + \mathrm{mdeg}(f_i) = \delta$ voor elke $i \in \{1, \ldots, s\}$ terwijl $\mathrm{mdeg}(f) \prec \delta$. (Dit drukt uit dat de leidende termen tegenover elkaar wegvallen.)

Dan is $\mathbf{x}^{\gamma_{ij}} \mid \mathbf{x}^\delta$ voor alle $i, j \in \{1, \ldots, s\}$ met $i < j$, waarbij $\mathbf{x}^{\gamma_{ij}} = \mathrm{lcm}(\mathrm{lm}(f_i), \mathrm{lm}(f_j))$, en er bestaan elementen $c_{ij} \in F$ zodat

$$f = \sum_{1 \le i < j \le s} c_{ij}\, \mathbf{x}^{\delta - \gamma_{ij}}\, S(f_i, f_j)\,, \tag{6.4}$$

en $\mathrm{mdeg}(\mathbf{x}^{\delta-\gamma_{ij}}\, S(f_i, f_j)) \prec \delta$ voor alle $i, j \in \{1, \ldots, s\}$ met $i < j$.

Bewijs. Door elke c_i met $\mathrm{lc}(f_i)$ te vermenigvuldigen, mogen we veronderstellen dat elke $\mathrm{lc}(f_i) = 1$ en $\mathrm{lt}(f_i) = \mathrm{lm}(f_i) = \mathbf{x}^{\mathrm{mdeg}(f_i)}$ voor alle i. Stel

$i, j \in \{1, \ldots, s\}$ met $i < j$. Het monoom $\mathbf{x}^\delta = \mathbf{x}^{\alpha_i} \cdot \mathrm{lm}(f_i) = \mathbf{x}^{\alpha_j} \cdot \mathrm{lm}(f_j)$ is een gemeen veelvoud van $\mathrm{lm}(f_i)$ en $\mathrm{lm}(f_j)$, en bijgevolg is $\mathbf{x}^{\gamma_{ij}} \mid \mathbf{x}^\delta$. Nu is

$$S(f_i, f_j) = \frac{\mathbf{x}^{\gamma_{ij}}}{\mathrm{lt}(f_i)} \cdot f_i - \frac{\mathbf{x}^{\gamma_{ij}}}{\mathrm{lt}(f_j)} \cdot f_j \,,$$

en uit (6.3) volgt dat

$$\mathrm{mdeg}(\mathbf{x}^{\delta - \gamma_{ij}}\, S(f_i, f_j)) \prec \delta \,.$$

We bewijzen nu (6.4) door inductie op s. Voor $s = 1$ zijn de voorwaarden van het lemma niet mogelijk, en dus volgt het besluit (6.4) trivialerwijze uit de voorwaarden. Stel nu $s \geq 2$; dan is

$$\begin{aligned}
g &:= f - c_1\, \mathbf{x}^{\delta - \gamma_{12}}\, S(f_1, f_2) \\
&= c_1\, \mathbf{x}^{\alpha_1}\, f_1 + c_2\, \mathbf{x}^{\alpha_2}\, f_2 + \sum_{i=3}^{s} c_i\, \mathbf{x}^{\alpha_i}\, f_i - c_1\, \mathbf{x}^{\delta - \gamma_{12}} \cdot \Big(\frac{\mathbf{x}^{\gamma_{12}}}{\mathrm{lt}(f_1)}\, f_1 - \frac{\mathbf{x}^{\gamma_{12}}}{\mathrm{lt}(f_2)}\, f_2 \Big) \\
&= c_1\, (\mathbf{x}^{\alpha_1} - \mathbf{x}^{\delta - \mathrm{mdeg}(f_1)})\, f_1 + (c_2\, \mathbf{x}^{\alpha_2} + c_1\, \mathbf{x}^{\delta - \mathrm{mdeg}(f_2)})\, f_2 + \sum_{i=3}^{s} c_i\, \mathbf{x}^{\alpha_i}\, f_i \\
&= (c_1 + c_2)\, \mathbf{x}^{\alpha_2}\, f_2 + \sum_{i=3}^{s} c_i\, \mathbf{x}^{\alpha_i}\, f_i \,,
\end{aligned}$$

waarbij we in de laatste regel gebruik hebben gemaakt van $\alpha_1 + \mathrm{mdeg}(f_1) = \delta = \alpha_2 + \mathrm{mdeg}(f_2)$. Uit Lemma 6.2.7(ii) weten we dat

$$\mathrm{mdeg}(g) \preceq \max\nolimits_{\prec}\{\mathrm{mdeg}(f), \mathrm{mdeg}(\mathbf{x}^{\delta - \gamma_{12}}\, S(f_1, f_2))\} \prec \delta \,,$$

en dus voldoet g opnieuw aan de voorwaarden van het lemma, met een som van lengte $s - 1$ indien $c_1 + c_2 \neq 0$ en van lengte $s - 2$ in het andere geval. Merk op dat voor $s = 2$ dit enkel mogelijk is indien $g = 0$; in dat geval is $f = c_1\, \mathbf{x}^{\delta - \gamma_{12}}\, S(f_1, f_2)$, en is vergelijking (6.4) voldaan voor f. Veronderstel nu $s \geq 3$; de inductiehypothese impliceert dan dat

$$g = \sum_{2 \leq i < j \leq s} c_{ij}\, \mathbf{x}^{\delta - \gamma_{ij}}\, S(f_i, f_j)$$

voor zekere $c_{ij} \in F$ voor $2 \leq i < j \leq s$. Stel nu $c_{12} = c_1$ en $c_{1j} = 0$ voor alle $j \in \{3, \ldots, s\}$; we bekomen dan

$$f = g + c_1\, \mathbf{x}^{\delta - \gamma_{12}}\, S(f_1, f_2) = \sum_{1 \leq i < j \leq s} c_{ij}\, \mathbf{x}^{\delta - \gamma_{ij}}\, S(f_i, f_j) \,,$$

en opnieuw is vergelijking (6.4) voldaan voor f. □

De volgende stelling geeft ons een eenvoudige test voor Gröbnerbasissen.

Stelling 6.4.8. *Een eindige verzameling $G = \{f_1, \ldots, f_s\} \subseteq R$ is een Gröbnerbasis voor het ideaal (G) als en slechts als*

$$S(f_i, f_j) \operatorname{rem} (f_1, \ldots, f_s) = 0 \text{ voor alle } i, j \in \{1, \ldots, s\} \text{ met } i < j\,.$$

Bewijs. Stel $I = (G)$. Het feit dat de voorwaarde nodig is, volgt onmiddellijk uit Stelling 6.4.4 aangezien $S(f_i, f_j) \in (f_i, f_j) \subseteq I$. We bewijzen nu dat de voorwaarde ook voldoende is. Beschouw daartoe een $f \in I$ met $f \neq 0$; we moeten aantonen dat $\mathrm{lt}(f) \in (\mathrm{lt}(G))$. We schrijven

$$f = \sum_{i=1}^{s} q_i f_i\,, \quad \delta = \max\nolimits_{\prec}\{\mathrm{mdeg}(q_i f_i) \mid i \in \{1, \ldots, s\}\}\,, \tag{6.5}$$

met alle $q_i \in R$. Dan is $\mathrm{mdeg}(f) \preceq \delta$. Indien $\mathrm{mdeg}(f) \prec \delta$, dan vallen sommige van de termen in (6.5) weg tegenover elkaar, en

$$f^* := \sum_{\substack{1 \leq i \leq s \\ \mathrm{mdeg}(q_i f_i) = \delta}} \mathrm{lt}(q_i)\, f_i$$

voldoet aan de voorwaarden van Lemma 6.4.7. We kunnen f^* dus schrijven als een F-lineaire combinatie van polynomen van de vorm $\mathbf{x}^{\alpha_{ij}}\, S(f_i, f_j)$, met $\alpha_{ij} \in \mathbb{N}^n$ zodat $\alpha_{ij} + \mathrm{mdeg}(S(f_i, f_j)) \prec \delta$ voor alle $1 \leq i < j \leq n$. Aangezien elk van de $S(f_i, f_j)$'s deelbaar is door $\{f_1, \ldots, f_s\}$, kunnen we

$$S(f_i, f_j) = \sum_{k=1}^{s} q_k^{(ij)} f_k$$

schrijven voor zekere $q_k^{(ij)}$, die wegens Stelling 6.2.10(iii) voldoen aan

$$\mathrm{mdeg}(q_k^{(ij)} f_k) \preceq \mathrm{mdeg}(S(f_i, f_j)) \prec \delta - \alpha_{ij}\,.$$

Hieruit volgt dat

$$\begin{aligned} f^* &= \sum_{i,j} c_{ij} \mathbf{x}^{\alpha_{ij}}\, S(f_i, f_j) \\ &= \sum_{i,j} c_{ij} \mathbf{x}^{\alpha_{ij}} \sum_{k=1}^{s} q_k^{(ij)} f_k \\ &= \sum_{k=1}^{s} \underbrace{\Big(\sum_{i,j} c_{ij} \mathbf{x}^{\alpha_{ij}}\, q_k^{(ij)} \Big)}_{q_k^*} f_k\,, \end{aligned}$$

waarbij

$$\mathrm{mdeg}(q_k^* f_k) \preceq \max\nolimits_{\prec}\{\alpha_{ij} + \mathrm{mdeg}(q_k^{(ij)} f_k) \mid 1 \leq i < j \leq s\} \prec \delta\,.$$

Dit impliceert dat f^* in de vorm (6.5) kan geschreven worden voor een kleinere waarde van δ. Per definitie van f^* kan echter ook $f - f^*$ in de vorm (6.5) geschreven worden voor een kleinere waarde van δ, en bijgevolg is dit ook zo voor $f = (f - f^*) + f^*$.

Zolang $\mathrm{mdeg}(f)$ strikt kleiner is dan de nieuwe waarde voor δ, blijven we dit proces herhalen; omdat $\prec$ welgeordend is, zal dit na een eindig aantal stappen ophouden, zodat we f in de vorm (6.5) zullen geschreven hebben met $\mathrm{mdeg}(f) = \delta$, en bijgevolg $\mathrm{mdeg}(f) = \mathrm{mdeg}(q_i f_i)$ voor ten minste één index i. Dan is

$$\mathrm{lt}(f) = \sum_{\substack{1 \leq i \leq s \\ \mathrm{mdeg}(q_i f_i) = \delta}} \mathrm{lt}(q_i)\ \mathrm{lt}(f_i) \in (\mathrm{lt}(G))\,. \qquad \square$$

6.5 Het algoritme van Buchberger

We beschrijven nu een methode om een Gröbnerbasis te berekenen vertrekkende van een gegeven basis. Het essentiële idee is om, telkens wanneer aan de voorwaarde van Stelling 6.4.8 niet voldaan wordt, het strijdige S-polynoom aan onze basis toe te voegen. We krijgen dan het volgend algoritme, dat gebaseerd is op dat van Buchberger [Buc65].

Algoritme 6.2 Het algoritme van Buchberger

input: $f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$;
een monomiale orderelatie $\prec$ voor R.
output: een Gröbner basis $G \subseteq R$ voor het ideaal $I = \langle f_1, \ldots, f_s \rangle$
met betrekking tot $\prec$, met $f_1, \ldots, f_s \in G$.

```
1  G ← {f_1, ..., f_s}
2  repeat
3        S ← ∅
4        sorteer de elementen van G (willekeurig) als g_1, ..., g_t
5        for 1 ≤ i < j ≤ t
6             do r ← S(g_i, g_j) rem (g_1, ..., g_t)
7                if r ≠ 0
8                   then S ← S ∪ {r}
9        if S = ∅
10          then return G
11          else  G ← G ∪ S
```

Stelling 6.5.1. *Algoritme 6.2 berekent correct een Gröbnerbasis voor het gegeven ideaal I.*

Bewijs. We tonen eerst aan dat het algoritme eindigt. Stel dat G en G' corresponderen met twee opeenvolgende **repeat**-lussen. Dan zal $G' \supseteq G$ en $(\mathrm{lt}(G')) \supseteq (\mathrm{lt}(G))$. Dus vormen de idealen $(\mathrm{lt}(G))$ bij opeenvolgende **repeat**-lussen een stijgende keten van idealen. Omdat R noethers is (Gevolg 6.3.8), hebben we na een eindig aantal **repeat**-lussen dat $(\mathrm{lt}(G')) = (\mathrm{lt}(G))$. We beweren dat dan ook $G' = G$. Stel namelijk $g, h \in G$, en $r = S(g, h)$ rem G. Dan is $r \in G'$ en ofwel is $r = 0$, ofwel is $\mathrm{lt}(r) \in (\mathrm{lt}(G')) = (\mathrm{lt}(G))$. Uit Lemma 6.3.2 volgt dan dat $\mathrm{lt}(r)$ deelbaar is door een element van $\mathrm{lt}(G)$, in strijd met de definitie van r als een rest bij deling door G. Bijgevolg moet $r = 0$ en dus $G' = G$.

Op elk moment van het algoritme is G een basis van I en is $\{f_1, \ldots, f_s\} \subseteq G$, aangezien dit initieel zo is, en de enige elementen die aan G worden toegevoegd zijn S-polynomen van $g_i, g_j \in I$ bij deling door elementen van I. Op het ogenblik dat het algoritme stopt, zijn alle resten van S-polynomen bij deling door G gelijk aan 0, en uit Stelling 6.4.8 volgt dan dat G een Gröbnerbasis is voor I. □

In het algemeen is de Gröbnerbasis die door het algoritme van Buchberger berekend wordt, noch minimaal, noch uniek bepaald. Het is echter mogelijk

om het algoritme zodanig om te vormen dat beide eigenschappen wel voldaan zijn, zoals we nu zullen zien.

Lemma 6.5.2. *Zij G een Gröbnerbasis voor $I \trianglelefteq R$ en $g \in G$. Veronderstel dat $\mathrm{lt}(g) \in (\mathrm{lt}(G \setminus \{g\}))$. Dan is $G \setminus \{g\}$ nog steeds een Gröbnerbasis voor I.*

Bewijs. Uit het gegeven volgt $(\mathrm{lt}(G)) = (\mathrm{lt}(G \setminus \{g\}))$, dus het resultaat volgt onmiddellijk uit Definitie 6.4.1. □

Definitie 6.5.3. (i) Een Gröbnerbasis G voor $I = (G)$ wordt *minimaal* genoemd als voor alle $g \in G$ geldt dat

(a) $\mathrm{lc}(g) = 1$;

(b) $\mathrm{lt}(g) \notin (\mathrm{lt}(G \setminus \{g\}))$.

(ii) Een element g van een Gröbnerbasis G is *gereduceerd met betrekking tot* G als geen enkele monoom van g tot $(\mathrm{lt}(G \setminus \{g\}))$ behoort.

(iii) Een Gröbnerbasis G voor I wordt *gereduceerd* genoemd, als G minimaal is, en alle elementen van G gereduceerd zijn met betrekking tot G.

Stelling 6.5.4. *Elk ideaal $I \trianglelefteq R$ heeft een unieke gereduceerde Gröbnerbasis.*

Bewijs. We bewijzen eerst het bestaan van een gereduceerde Gröbnerbasis. Door herhaaldelijk Lemma 6.5.2 toe te passen, kunnen we starten met een minimale Gröbnerbasis $G = \{g_1, \ldots, g_s\}$ voor I. Voor elke $i \in \{1, \ldots, s\}$ definiëren we nu inductief

$$h_i = g_i \operatorname{rem} \{h_1, \ldots, h_{i-1}, g_{i+1}, \ldots, g_s\}\,.$$

Inductie op i toont aan dat $\mathrm{lt}(h_j) = \mathrm{lt}(g_j)$, en h_j is gereduceerd met betrekking tot $G_i = \{h_1, \ldots, h_i, g_{i+1}, \ldots, g_s\}$ voor alle $i, j \in \{1, \ldots, s\}$ met $j \leq i$. Uiteindelijk is $G_s = \{h_1, \ldots, h_s\}$ een gereduceerde Gröbnerbasis voor I.

Veronderstel nu dat G en G' twee gereduceerde Gröbnerbasissen voor I zijn. We beweren dat $\mathrm{lt}(G) = \mathrm{lt}(G')$. Zij $\mathrm{lt}(g) \in \mathrm{lt}(G) \subseteq (\mathrm{lt}(I)) = (\mathrm{lt}(G'))$; dan is er een $g' \in G'$ zodat $\mathrm{lt}(g') \mid \mathrm{lt}(g)$, wegens Lemma 6.3.2. Door een symmetrisch argument vinden we dan ook een $g'' \in G$ zodat $\mathrm{lt}(g'') \mid \mathrm{lt}(g')$, en bijgevolg is $\mathrm{lt}(g'') \mid \mathrm{lt}(g)$. Aangezien G minimaal is, kan dit enkel als $\mathrm{lt}(g'') = \mathrm{lt}(g)$ en dus ook $\mathrm{lt}(g') = \mathrm{lt}(g)$. Dit bewijst dat $\mathrm{lt}(g) \in \mathrm{lt}(G')$, dus is $\mathrm{lt}(G) \subseteq \mathrm{lt}(G')$. Door symmetrie bekomen we $\mathrm{lt}(G) = \mathrm{lt}(G')$.

Zij nu $g \in G$ willekeurig, en beschouw een $g' \in G'$ met $\mathrm{lt}(g) = \mathrm{lt}(g')$. Zowel G als G' is gereduceerd, en dus is geen enkel monoom in $g - g'$ deelbaar door een element van $\mathrm{lt}(G) = \mathrm{lt}(G')$, zodat $g - g' = g - g' \operatorname{rem} G$. Echter, $g - g' \in I$, dus is $g - g' \operatorname{rem} G = 0$, waaruit volgt dat $g = g'$. Bijgevolg is $G \subseteq G'$, en wegens symmetrie is ook $G' \subseteq G$. □

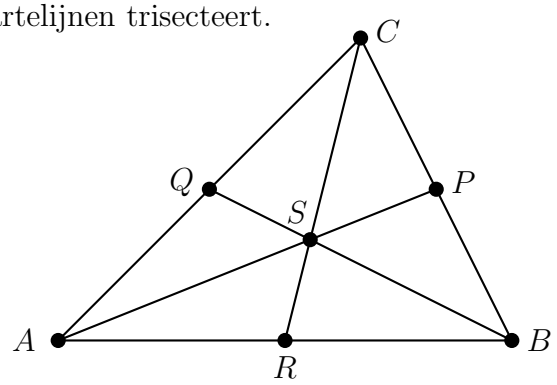
Het bestuderen van de complexiteit van Buchberger's algoritme is een bijzonder moeilijke opdracht, die pas in 1996 door Kühnle en Mayr voltooid is [KM96], voortbouwend op eerdere resultaten van Mayr en Meyer [MM82]. Het resultaat is ontgoochelend: het berekenen van een Gröbnerbasis is een zogenaamd EXPSPACE-compleet probleem, en heeft dubbel-exponentiële complexiteit. Ook de polynomen zelf lijden aan "intermediate expression swell": zelfs indien men start van een basis met weinig polynomen van lage graad met kleine coëfficiënten, produceert het algoritme dikwijls reusachtige aantallen polynomen van hoge graad en met enorme coëfficiënten. Bovendien kan men aantonen dat de oorzaak hiervan niet te wijten is aan het algoritme, maar eigen is aan het probleem. Er bestaan idealen van polynoomringen in n variabelen, waarvoor elke Gröbnerbasis ten minste $2^{2^{an}}$ polynomen bevat, elk van graad ten minste $2^{2^{bn}}$, voor bepaalde constanten $a, b \in \mathbb{R}_{>0}$.

6.6 Toepassingen

6.6.1 Automatische meetkundige bewijsvoering

Indien een meetkundig probleem zich laat vertalen naar coördinaten en vergelijkingen die polynomiaal zijn, is het mogelijk om Gröbnerbasissen te gebruiken om de correctheid van een bewering na te gaan. We geven hiervan een voorbeeld.

Een bekende stelling in de vlakke meetkunde zegt dat de drie zwaartelijnen van een driehoek elkaar snijden in 1 punt, en dat het snijpunt de zwaartelijnen trisecteert.



We coördinatiseren dit probleem, en we kiezen het assenstelsel zodanig dat $A = (0, 0)$, $B = (2, 0)$, en $C = (2x, 2y)$, met $x, y \in \mathbb{R}$; merk op dat $y \neq 0$. Dan hebben we $P = (x + 1, y)$, $Q = (x, y)$ en $R = (1, 0)$. We stellen de coördinaten van het punt S, dat we definiëren als het snijpunt van AP en BQ, gelijk aan (u, v); merk op dat eveneens $v \neq 0$.

We willen nu uitdrukken dat S op de rechte AP ligt; we doen dit door weer te geven dat de richtingscoëfficiënten van AS en AP gelijk zijn, en we krijgen

$$\frac{u}{v} = \frac{x+1}{y}\,,$$

of dus

$$f_1 := uy - v(x+1) = 0\,.$$

Analoog vertaalt de voorwaarde dat S op de rechte BQ ligt zich naar

$$f_2 := (u-2)y - v(x-2) = 0\,.$$

De bewering is nu dat S ook op de rechte CR ligt, dus dat

$$g_1 := 2(u-1)y - v(2x-1) = 0\,,$$

en dat S elk van de drie zwaartelijnen trisecteert, zodat

$$\begin{aligned}
S - A = 2\cdot(P-S) &\iff (u,v) = 2\cdot(x+1-u, y-v)\,,\\
S - B = 2\cdot(Q-S) &\iff (u-2,v) = 2\cdot(x-u, y-v)\,,\\
S - C = 2\cdot(R-S) &\iff (u-2x, v-2y) = 2\cdot(1-u, -v)\,,
\end{aligned}$$

hetgeen kan herschreven worden als

$$\begin{aligned}
g_2 &:= 2x - 3u + 2 = 0\,,\\
g_3 &:= 2y - 3v = 0\,,
\end{aligned}$$

waarbij elk van beide vergelijkingen drie keer bekomen wordt.

We hebben dus de veronderstellingen herschreven in termen van polynomen $f_1, f_2 \in R = \mathbb{R}[x,y,u,v]$, en het te bewijzen in termen van polynomen $g_1, g_2, g_3 \in R$. Meer bepaald willen we aantonen dat

$$f_1 = f_2 = 0 \implies g_1 = g_2 = g_3 = 0\,.$$

De bewering zal dus waar zijn als en slechts als $V(f_1,f_2) \subseteq V(g_1,g_2,g_3)$; in het bijzonder kunnen we de bewering aantonen door te bewijzen dat $g_1, g_2, g_3 \in (f_1, f_2)$.

We berekenen eerst een Gröbnerbasis voor het ideaal $I = (f_1, f_2)$, met betrekking tot de monomiale orderelatie $\prec = \prec_{\text{grlex}}$ met $x \succ y \succ u \succ v$. We hebben dan

$$f_1 = -xv + yu - v\,, \quad f_2 = -xv + yu - 2y + 2v\,,$$

en

$$S(f_1, f_2) = \frac{xv}{-xv} f_1 - \frac{xv}{-xv} f_2 = f_2 - f_1 = -2y + 3v\,;$$

toevallig blijkt $S(f_1, f_2) = -g_3$. We zien dat $(-g_3)$ rem $(f_1, f_2) = -g_3 \neq 0$, dus voegen we g_3 toe aan onze basis. We stellen vast dat dan

$$\begin{aligned} S(f_1, g_3) \text{ rem } (f_1, f_2, g_3) &= \left(\tfrac{3}{2}xv^2 - y^2u + yv\right) \text{ rem } (f_1, f_2, g_3) = 0\,, \\ S(f_2, g_3) \text{ rem } (f_1, f_2, g_3) &= \left(\tfrac{3}{2}xv^2 - y^2u + 2y^2 - 2yv\right) \text{ rem } (f_1, f_2, g_3) = 0\,, \end{aligned}$$

zodat $\{f_1, f_2, g_3\}$ een Gröbnerbasis is voor I. Omwille van Lemma 6.5.2 mogen we f_2 schrappen uit de Gröbnerbasis, en wanneer we vervolgens het proces uit het bewijs van Stelling 6.5.4 toepassen voor het reduceren van de Gröbnerbasis $\{f_1, g_3\}$, bekomen we

$$\begin{aligned} h_1 &= f_1 \text{ rem } g_3 = -xv + \tfrac{3}{2}uv - v\,, \\ h_2 &= g_3 \text{ rem } h_1 = g_3\,, \end{aligned}$$

zodat we uiteindelijk de unieke gereduceerde Gröbnerbasis

$$G = \{xv - \tfrac{3}{2}uv + v, y - \tfrac{3}{2}v\}$$

bekomen.

Het is nu eenvoudig na te gaan dat $g_1 \text{ rem } G = 0$ en $g_3 \text{ rem } G = 0$, maar $g_2 \text{ rem } G \neq 0$. Dus $g_1 \in I$, $g_3 \in I$, maar $g_2 \notin I$. Echter, $vg_2 \text{ rem } G = 0$, zodat $vg_2 \in I$, en omdat $v \neq 0$ betekent dit nog steeds dat uit $f_1 = f_2 = 0$ volgt dat ook $g_2 = 0$, zodat de bewering inderdaad correct is.

We zien dus dat ook de ogenschijnlijk onbelangrijke nevenvoorwaarde $v \neq 0$ bij het oplossen van het probleem opduikt. Dit lijkt een struikelblok voor automatische bewijsvoering, maar dit is het gelukkig niet; een systematische oplossing voor dit probleem wordt gegeven door *de truc van Rabinowitsch*, die erin bestaat om $1 - vz$ toe te voegen aan het ideaal I, waarbij z een nieuwe variabele is. Dit garandeert immers dat $v \neq 0$, en aangezien $g_2 = (1 - vz) \cdot g_2 + z \cdot (vg_2)$ zal inderdaad $g_2 \text{ rem } (G \cup \{1 - vz\}) = 0$.

6.6.2 Impliciete parametrisatie

Zij $f_1, \ldots, f_n \in F[t_1, \ldots, t_m]$, en veronderstel dat de algebraïsche variëteit $V \subseteq F^n$ gegeven wordt door de parametervergelijkingen

$$\begin{aligned} x_1 &= f_1(t_1, \ldots, t_m)\,, \\ &\vdots \\ x_n &= f_n(t_1, \ldots, t_m)\,, \end{aligned}$$

zodat V "expliciet" beschreven is als

$$V = \{a \in F^n \mid a = (f_1(b), \ldots, f_t(b)) \text{ voor een } b \in F^m\}\,.$$

De vraag is nu om polynomen $g_1, \ldots, g_s \in F[x_1, \ldots, x_n]$ te vinden, zodanig dat V de "impliciete" beschrijving $V = V(I)$ heeft, met $I = (g_1, \ldots, g_s)$. (In feite zal $V(I)$ de Zariski sluiting van V zijn, maar daar gaan we hier niet verder op in.)

Voorbeeld 6.6.1. Zij $V \subseteq F^3$ gegeven door

$$x = t\,, \quad y = t^2\,, \quad z = t^3\,;$$

dit is de zogenaamde *twisted cubic*. Een impliciete beschrijving van V wordt gegeven door de polynomen $g_1 = y - x^2$, $g_2 = z - x^3$.

Om dit probleem algemeen op te lossen, beschouwen we

$$J = (x_1 - f_1, \ldots, x_n - f_n) \subseteq F[t_1, \ldots, t_m, x_1, \ldots, x_n]\,,$$

en we sorteren de variabelen zodanig dat $t_1 \succ \cdots \succ t_m \succ x_1 \succ \cdots \succ x_n$; we berekenen vervolgens een Gröbnerbasis G voor J met betrekking tot $\prec = \prec_{\text{lex}}$. Dan zullen sommige elementen van G enkel afhangen van $x_1, \ldots, x_n$, en dit zijn de kandidaten voor de g_i's. In Voorbeeld 6.6.1 krijgen we als gereduceerde Gröbnerbasis voor J met betrekking tot $t \succ z \succ y \succ x$ de basis $\{t - x, z - x^3, y - x^2\}$.

Deze aanpak zal altijd werken over een oneindig veld, in die zin dat de variëteit in F^n gedefinieerd door de polynomen in $G \cap F[x_1, \ldots, x_n]$ steeds de kleinste variëteit is die het beeld van de parametrisatie bevat (Cox, Little & O'Shea, 1997).

6.6.3 Stelsels van polynoomvergelijkingen

Het oplossen van stelsels van polynoomvergelijkingen is de meest natuurlijke toepassing van Gröbnerbasissen; in feite zijn deze precies met dit doel ingevoerd. In het algemeen hebben we polynomen $f_1, \ldots, f_m \in F[x_1, \ldots, x_n]$, en willen we vragen beantwoorden over de corresponderende affiene variëteit $V = \{a \in F^n \mid f_1(a) = \cdots = f_m(a) = 0\}$. Bijvoorbeeld, is $V \neq \emptyset$, en indien dit zo is, zoek dan een punt van V. Gegeven een ander polynoom $g \in F[x_1, \ldots, x_n]$, is dan $g(a) = 0$ voor alle $a \in V$?

De eenvoudigste situatie is het geval waarin V nul-dimensionaal is, zodat het slechts uit een eindig aantal geïsoleerde punten bestaat. Een voorbeeld hiervan is de doorsnede van twee vlakke krommen. Een Gröbnerbasis met

betrekking tot de lexicografische ordening $\prec = \prec_{\text{lex}}$ en $x_1 \prec \cdots \prec x_n$ zal vaak enkele polynomen hebben die alleen de variabele x_1 bevatten, een aantal andere die enkel x_1 en x_2 bevatten, enzovoort. Deze kunnen dan opgelost worden met substitutie, door eerst naar x_1 op te lossen, vervolgens voor elk van de bekomen oplossingen voor x_1 naar x_2 op te lossen, en zo verder.

Voorbeeld 6.6.2. Zij $f, g \in \mathbb{C}[x, y]$ gegeven door

$$\begin{aligned} f &= (y^2 + 6)(x - 1) - y(x^2 + 1)\,, \\ g &= (x^2 + 6)(y - 1) - x(y^2 + 1)\,; \end{aligned}$$

we zoeken de doorsnede van deze twee vlakke krommen, met andere woorden, we proberen het stelsel $f = g = 0$ op te lossen. We berekenen eerst een gereduceerde Gröbnerbasis voor $I = (f, g)$, en we bekomen $G = (h_1, h_2, h_3)$ met

$$\begin{aligned} h_1 &= x^2 - 5x + y^2 - 5y + 12\,, \\ h_2 &= x^3 + x^2y - 6x^2 - 5xy + 11x + 6y - 6\,, \\ h_3 &= x^4 - 6x^3 + 15x^2 - 26x + 24\,. \end{aligned}$$

Het polynoom h_3 bevat enkel de variabele x, en we bepalen de wortels ervan door het symbolisch te factoriseren over $\mathbb{Q}$ met behulp van een algoritme uit Hoofdstuk 5. We bekomen

$$h_3 = (x - 2)(x - 3)(x^2 - x + 4)\,,$$

en bijgevolg

$$V(h_3) = \left\{ 2, 3, \frac{1 \pm \sqrt{15}i}{2} \right\}.$$

Vervolgens moeten we nagaan welke van deze partiële oplossingen uitbreidt naar een globale oplossing in $V(I)$. Wanneer we $x = 2$ substitueren in h_1 en h_2, dan zien we dat h_2 identisch nul wordt, terwijl we voor h_1 de vergelijking $0 = y^2 - 5y + 6 = (y - 2)(y - 3)$ krijgen, zodat we de twee globale oplossingen $(2, 2)$ en $(2, 3)$ vinden. Op gelijkaardige wijze vinden we de andere oplossingen, en uiteindelijk verkrijgen we de zes oplossingen

$$V(f, g) = \left\{ (2, 2), (2, 3), (3, 2), (3, 3), \left(\frac{1 \pm \sqrt{15}i}{2}, \frac{1 \mp \sqrt{15}i}{2} \right) \right\}.$$

Bibliografie

[AGP94] W. R. Alford, Andrew Granville, and Carl Pomerance. There are infinitely many Carmichael numbers. *Ann. of Math. (2)*, 139(3):703–722, 1994.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[AM93] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Math. Comp.*, 61(203):29–68, 1993.

[APR83] Leonard M. Adleman, Carl Pomerance, and Robert S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. of Math. (2)*, 117(1):173–206, 1983.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[Ber67] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Tech. J.*, 46:1853–1859, 1967.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.

[BGY80] Richard P. Brent, Fred G. Gustavson, and David Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1(3):259–295, 1980.

[BM75] Allan Borodin and Ian Munro. *The computational complexity of algebraic and numeric problems*. American Elsevier Publishing Co., Inc., New York-London-Amsterdam, 1975. Elsevier Computer Science Library; Theory of Computation Series, No. 1.

[Bre80] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20(2):176–184, 1980.

[Bro71] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. Assoc. Comput. Mach.*, 18:478–504, 1971.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Philosophische Fakultät an der Leopold-Franzens-Universität, Innsbruck, Austria, 1965.

[BW98] B. Buchberger and F. Winkler, editors. *Gröbner bases and applications*, volume 251 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. Papers from the Conference on 33 Years of Gröbner Bases held at the University of Linz, Linz, February 2–4, 1998.

[Che26] Nikolai Chebotarëv. Die Bestimmung der Dichtigkeit einer Menge von Primzahlen, welche zu einer gegebenen Substitutionsklasse gehören. *Math. Ann.*, 95:191–228, 1926.

[CL84] H. Cohen and H. W. Lenstra, Jr. Primality testing and Jacobi sums. *Math. Comp.*, 42(165):297–330, 1984.

[Col71] George E. Collins. The calculation of multivariate polynomial resultants. *J. Assoc. Comput. Mach.*, 18:515–532, 1971.

[Coo66] S. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, 1966.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

[Dix81] John D. Dixon. Asymptotically fast factorization of integers. *Math. Comp.*, 36(153):255–260, 1981.

[FÖ9] Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

[Fro96] G Frobenius. Über Beziehungen zwischen den Primidealen eines algebraischen Körpers und und den Substitutionen einer Gruppe.

Sitzungberichte der Königlich Preussischen Akademie der Wissenschaften, Berlin, pages 689–702, 1896.

[Hil90] David Hilbert. Ueber die theorie der algebraischen formen. *Math. Ann*, 36:473–534, 1890.

[HN] Louis Helm and David Norris. http://www.seventeenorbust.com.

[KM96] Klaus Kühnle and Ernst W. Mayr. Exponential space computation of gröbner bases. In *ISSAC '96: Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, pages 63–71, New York, NY, USA, 1996. ACM.

[Knu71] Donald E. Knuth. The analysis of algorithms. In *Actes du congrès international des mathématiciens, tome 3*, pages 269–274, Paris, 1971. Gauthier-Villars Éditeur. URL: `http://cr.yp.to/bib/entries.html\#1971/knuth-gcd`.

[Knu97] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. First edition 1969.

[KO63] Anatolii Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl.*, 7:595–596, 1963.

[KW] Scott Kurowski and George Woltman. http://www.mersenne.org.

[Leh38] D. H. Lehmer. Euclid's Algorithm for Large Numbers. *Amer. Math. Monthly*, 45(4):227–233, 1938.

[Len87] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Ann. of Math. (2)*, 126(3):649–673, 1987.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[LLMP93] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *The development of the number field sieve*, volume 1554 of *Lecture Notes in Math.*, pages 11–42. Springer, Berlin, 1993.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. System Sci.*, 13(3):300–317, 1976. Working papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N.M., 1975).

[MM82] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, 46(3):305–329, 1982.

[Moe73] R. T. Moenck. Fast computation of GCDs. In *Fifth Annual ACM Symposium on Theory of Computing (Austin, Tex., 1973)*, pages 142–151. Assoc. Comput. Mach., New York, 1973.

[MY73] Joel Moses and David Y. Y. Yun. The ez gcd algorithm. In *ACM'73: Proceedings of the annual conference*, pages 159–166, New York, NY, USA, 1973. ACM.

[Pol75] J. M. Pollard. A Monte Carlo method for factorization. *Nordisk Tidskr. Informationsbehandling (BIT)*, 15(3):331–334, 1975.

[Pom96] Carl Pomerance. A tale of two sieves. *Notices Amer. Math. Soc.*, 43(12):1473–1485, 1996.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[Sch71] A. A. Schönhage. Schnelle berechnung von kettenbruchentwicklungen. *Acta Inf.*, 1:139–144, 1971.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27(4):701–717, 1980.

[Sch77] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informat.*, 7(4):395–398, 1976/77.

[SL96] P. Stevenhagen and H. W. Lenstra, Jr. Chebotarëv and his density theorem. *Math. Intelligencer*, 18(2):26–37, 1996.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[Ste67] J. Stein. Computational Problems Associated with Racah Algebra. *Journal of Computational Physics*, 1:397–405, February 1967.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[Str83] V. Strassen. The computational complexity of continued fractions. *SIAM J. Comput.*, 12(1):1–27, 1983.

[Too63] A. L. Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. *Dokl. Akad. Nauk SSSR*, 150:496–498, 1963.

[Val98] B. Vallée. Dynamics of the binary Euclidean algorithm: functional analysis and operators. *Algorithmica*, 22(4):660–685, 1998. URL: `http://users.info.unicaen.fr/~brigitte/Publications/bin-gcd.ps`.

[vH02] Mark van Hoeij. Factoring polynomials and the knapsack problem. *J. Number Theory*, 95(2):167–189, 2002.

[vzGG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, second edition, 2003.

[Wei00] André Weilert. $(1+i)$-ary GCD computation in $\mathbf{Z}[i]$ is an analogue to the binary GCD algorithm. *J. Symbolic Comput.*, 30(5):605–617, 2000.

[Zas69] Hans Zassenhaus. On Hensel factorization. I. *J. Number Theory*, 1:291–311, 1969.

Index