2012•2013
FACULTEIT WETENSCHAPPEN
master in de informatica: databases

Masterproef
Uniforme Random Generatie van Strings

Promotor :
Prof. dr. Frank NEVEN

Raf Vandeput

Masterproef voorgedragen tot het bekomen van de graad van master in de informatica , afstudeerrichting databases

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

universiteit hasselt
KNOWLEDGE IN ACTION

Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt
Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek

universiteit hasselt | Maastricht University

2012•2013
FACULTEIT WETENSCHAPPEN
master in de informatica: databases

Masterproef

Uniforme Random Generatie van Strings

Promotor :
Prof. dr. Frank NEVEN

Raf Vandeput

Masterproef voorgedragen tot het bekomen van de graad van master in de informatica , afstudeerrichting databases

universiteit hasselt | Maastricht University

Dankwoord

Via deze weg zou ik iedereen willen bedanken die rechtstreeks of onrechtstreeks heeft meegeholpen aan de totstandkoming van deze thesis.

In de eerste plaats wil ik mijn promotor Prof. dr. Frank Neven bedanken voor het promoten van mijn thesis, het aanreiken van interessante en nuttige literatuur en voor zijn hulp en ideeën die deze thesis mogelijk gemaakt hebben.

Mijn bijzondere dank gaat ook uit naar mijn begeleider Jonny Daenen voor de vele hulp en ideeën die hij me gegeven heeft tijdens het werken aan deze thesis. Verder wil ik hem ook uitermate bedanken voor het nalezen en verbeteren van deze hele thesistekst.

Ten slotte wil ik graag mijn familieleden en vrienden bedanken voor de steun die ze me gegeven hebben doorheen het zware werk van de voorbije jaren.

Bedankt iedereen!

Abstract

In deze thesis wordt een methode beschreven die toelaat om strings uit formele talen op een uniforme random manier te genereren. Hiervoor wordt er gebruikgemaakt van combinatorische klassen. Dit zijn verzamelingen van objecten die een bepaalde structuur hebben. Een formele taal is in feite zo'n combinatorische klasse. Daarom bestuderen we combinatorische klassen eerst grondig. We beschouwen enkel de combinatorische klassen die formeel kunnen beschreven worden d.m.v. specificaties, wat in essentie grammatica's zijn.

Een belangrijke toepassing van combinatorische klassen is het tellen van het aantal objecten van een bepaalde grootte die in een dergelijke klasse zitten. Hiervoor kan gebruikgemaakt worden van generating functions, die in feite gereduceerde voorstellingen zijn van combinatorische klassen in de zin van aantallen van objecten. Het aantal objecten van iedere grootte van een bepaalde klasse zit namelijk opgeslagen in de bijhorende generating function. Bijgevolg zijn deze generating functions zeer nuttig wanneer men het aantal objecten van een bepaalde grootte uit een combinatorische klasse wil tellen.

Het hoofddoel van deze thesis is het beschrijven van een methode die toelaat om een object uit een combinatorische klasse op een uniforme random manier te genereren. Deze methode steunt op het tellen van objecten van een bepaalde grootte uit een combinatorische klasse. Deze aantallen worden namelijk gebruikt in de kansberekeningen van de random generatie methode om de objecten te genereren volgens een bepaalde kans. De tijdscomplexiteit van de beschreven random generatie methode kan ruwweg als kwadratisch ($O(n^2)$) geschat worden.

Omdat gelabelde combinatorische klassen geschikt zijn om formele talen voor te stellen, kunnen de aangereikte methodes gebruikt worden om strings uit een formele taal op een uniforme random manier te genereren. Het random genereren van combinatorische objecten is m.a.w. een generiek probleem, terwijl het random genereren van strings een specifieker probleem is. Daarom stellen we een theorie op die het random genereren van combinatorische objecten aanpakt om deze vervolgens toe te passen op strings uit formele talen.

Inhoudsopgave

Hoofdstuk 1

Inleiding

In deze thesis bestuderen we hoe we strings uit een formele taal op een uniforme random manier kunnen genereren. Hiervoor maken we gebruik van een tak van de wiskunde die de "combinatoriek" genoemd wordt, meer bepaald de "enumeratieve combinatoriek".

Zulke strings kunnen bijvoorbeeld gebruikt worden als invoer van een bepaald programma om het uitgebreid te kunnen testen. Omdat de generatie uniform gebeurt, is de kans dat een bepaalde string van een bepaalde grootte uit een taal gegenereerd wordt even groot als de kans dat een andere string van dezelfde grootte gegenereerd wordt. Bijgevolg is het mogelijk om betrouwbare testresultaten te verkrijgen bij het testen van programma's d.m.v. random gegenereerde strings.

Wanneer het random genereren niet op een uniforme manier zou gebeuren, is het mogelijk dat de kans dat een "goede" string (een string waarmee het te testen programma correct werkt) gegenereerd wordt groter is dan de kans dat een "slechte" string gegenereerd wordt. In dat geval zijn de testresultaten minder betrouwbaar doordat er waarschijnlijk meer strings gebruikt zullen worden waarmee het te testen programma correct werkt. Dit willen we juist vermijden omdat we tijdens het testen van programma's op zoek zijn naar onderdelen die niet correct werken.

Een toepassing hiervan is bijvoorbeeld het testen van een compiler voor een bepaalde programmeertaal. Stel bijvoorbeeld dat we random een syntactisch correct programma, wat in feite een string is, kunnen genereren uit deze programmeertaal, dan kunnen we dit programma als invoer meegeven aan deze compiler. Zo kan getest worden of de syntactische analyse van deze compiler correct werkt of niet.

Zoals reeds werd aangehaald, maken we gebruik van de combinatoriek. Dit is de studie van eindige of aftelbaar oneindige verzamelingen die bestaan uit eindige objecten die opgebouwd worden volgens een eindig aantal regels. Deze verzamelingen worden combinatorische klassen genoemd en de elementen ervan noemt men combinatorische objecten.

De bedoeling is om een object uit de "reële wereld" (bv. een boom) abstract voor te stellen als een object uit een combinatorische klasse en vervolgens een verzameling objecten uit de "reële wereld" te beschrijven als een combinatorische klasse. Daarom moeten we een combinatorische klasse formeel kunnen beschrijven. Hiervoor maken we gebruik van combinatorische specificaties, wat in essentie grammatica's zijn die opgebouwd worden m.b.v. een aantal basisoperaties (basisconstructors). Deze basisoperaties maken gebruiken van (andere) hulpklassen, die we in dezelfde specificatie definiëren, om de beschouwde combinatorische klasse op te bouwen.

In de combinatoriek bestaan er twee universa, namelijk het gelabelde en het ongelabelde universum. In het gelabelde universum worden gelabelde objecten beschouwd. Hierbij dragen de atomen, de kleinste bouwstenen, ervan een uniek nummer. Op die manier kunnen we de atomen van een gelabelde object van elkaar onderscheiden. In het ongelabelde universum beschouwen we ongelabelde objecten. Hier worden alle atomen van een object als gelijk aanzien.

Het gevolg van het werken met twee universa is dat we gelabelde klassen op een andere manier beschrijven als ongelabelde klassen. Een tweede gevolg is dat het tellen van gelabelde objecten ook op een verschillende manier gebeurt dan het tellen van ongelabelde objecten.

Een belangrijk onderdeel van de combinatoriek is namelijk het tellen van het aantal objecten die voldoen aan een bepaalde eigenschap. Een voorbeeld hiervan is het tellen van het aantal objecten van een bepaalde grootte uit een bepaalde combinatorische klasse. Hiervoor kan o.a. gebruikgemaakt worden van speciale wiskundige veeltermen, die generating functions genoemd worden en waarvoor bepaalde eigenschappen m.b.t. combinatorische klassen gelden. Deze generating functions zijn in feite gereduceerde representaties van combinatorische klassen die handig zijn als we enkel geïnteresseerd zijn in het tellen van objecten van een bepaalde grootte. Deze generating functions hangen dan ook nauw samen met de beschouwde combinatorische klassen. Het principe is dat een aantal bewerkingen uit de verzamelingleer, die gebruikt worden om de objecten uit deze combinatorische klassen op te bouwen, vertaald kunnen worden naar bewerkingen met deze veeltermen. Wanneer we

combinatorische klassen beschrijven m.b.v. basisconstructors en hulpklassen, kunnen we ze dus rechtstreeks vertalen naar de overeenkomstige generating functions. Op die manier kunnen we de generating functions van de hulpklassen combineren tot de generating function van de beschouwde klasse.

Merk op dat het tellen van objecten niet hetzelfde is als het opsommen van deze objecten. Bij het tellen zijn we enkel geïnteresseerd in het aantal objecten van een bepaalde grootte en niet in de vorm van deze objecten. Bij het opsommen worden de objecten volledig geconstrueerd m.b.v. de constructors van een specificatie. In het algemeen gebeurt het tellen van objecten sneller dan het opsommen ervan.

Een ander belangrijk aspect is het uniform random genereren van combinatorische objecten van een bepaalde grootte. Een naïeve manier om dit te doen, zou zijn door eerst alle objecten van deze grootte uit de beschouwde klasse op te sommen en vervolgens hieruit één object te kiezen. Deze methode neemt echter enorm veel tijd in beslag. Daarom willen we een efficiëntere methode gebruiken om het probleem van het random genereren van één object op te lossen.

In deze thesis wordt een efficiënte methode beschreven die dit probleem aanpakt. Deze methode veronderstelt dat het aantal objecten tot en met een bepaalde grootte van de beschouwde klasse en hulpklassen reeds berekend is. Vervolgens wordt voor elke operatie uit de specificatie van de beschouwde klasse een procedure uitgevoerd die gebruikmaakt van deze aantallen om op een uniforme random manier een object uit het resultaat van deze operatie te genereren. Uiteindlijk worden al deze procedures op een hiërarchische manier gecombineerd om één object van een gewenste grootte uit de beschouwde combinatorische klasse te genereren.

Formele talen die bestaan uit strings die een bepaalde structuur hebben zijn voorbeelden van combinatorische klassen. De grammatica die een formele taal beschrijft, beschrijft m.a.w. ook een combinatorische klasse. De verzameling van alle syntactisch correcte Java-programma's is dus bijvoorbeeld een combinatorische klasse.

Omdat formele talen combinatorische klassen zijn, kunnen we de aangereikte methoden toepassen op deze talen om uniform random een string van een bepaalde grootte uit een dergelijke taal te genereren. Hierbij vertrekken we vanaf de grammatica die deze taal beschrijft om eerst de aantallen van objecten te berekenen. Vervolgens worden deze aantallen gebruikt in de random generatie methode.

De achtergrond van de combinatoriek komt voornamelijk uit [20], [21], [22] en [23].. Deze boeken en papers beschrijven namelijk algemene eigenschappen van combinatorische klassen, die handig zijn bij het uniform random genereren van strings. Verder werd ook de nodige informatie gehaald uit [2], [12], [15] en [29]. Hieruit hebben we namelijk o.a. notaties en definities omtrent strings en verzamelingen gehaald.

In de onderstaande opsomming wordt een globaal overzicht van deze thesis weergegeven:

- In Hoofdstuk 2 worden een aantal veelgebruikte begrippen en notaties vastgelegd die doorheen de hele thesis gebruikt worden.
- In Hoofdstuk 3 worden combinatorische klassen geïntroduceerd en wordt uitgelegd hoe ze formeel kunnen beschreven worden.
- In Hoofdstuk 4 worden een aantal methoden beschreven die toelaten om het aantal objecten van een bepaalde grootte uit een combinatorische klasse te berekenen.
- In Hoofdstuk 5 wordt een methode beschreven om objecten uit een combinatorische klasse uniform random te genereren (en dus ook om strings uit een formele taal uniform random te genereren).
- In Hoofdstuk 6 beschrijven we onze implementatie die we gemaakt hebben om de theorie uit de vorige hoofdstukken in de praktijk om te zetten.
- In Hoofdstuk 7 beschrijven we vervolgens een aantal experimenten die we hebben uitgevoerd met onze implementatie.
- In Hoofdstuk 8 ten slotte maken we een conclusie op basis van de beschreven methoden.

Hoofdstuk 2

Terminologie

In dit hoofdstuk worden een aantal begrippen en notaties uitgelegd die veel gebruikt worden doorheen deze thesis. Een aantal begrippen hieruit komen uit [12], [20], [23] en [29].

2.1 Notaties

We starten met een aantal handige notaties, die zeer veel gebruikt worden doorheen deze thesis.

2.1.1 Sommatie

Een *sommatie* kan in de wiskunde compacter geschreven worden m.b.v. de Griekse hoofdletter Σ:

$$\sum_{i=m}^{n} x_i \quad = \quad x_m + x_{m+1} + \ldots + x_{n-1} + x_n.$$

Hierbij is elke x_i een expressie, i de index, m de ondergrens en n de bovengrens (met $n \geq m$). De onder- en bovengrens kunnen mogelijk gelijk zijn aan $\pm\infty$. In dat geval spreken we van een *oneindige sommatie*. De sommatie begint bij de expressie x_m en vervolgens wordt de index telkens met 1 verhoogd totdat x_n bereikt is.

Voorbeeld 2.1 *De notatie*

$$S = \sum_{i=5}^{10} i^2$$

is een compacte schrijfwijze voor de sommatie van de kwadraten van natuurlijke getallen met de onder- en bovengrens respectievelijk gelijk aan 5 en 10. Voluit betekent dit de volgende som:

$$\begin{aligned} S &= 5^2 + 6^2 + 7^2 + 8^2 + 9^2 + 10^2 \\ &= 25 + 36 + 49 + 64 + 81 + 100 \\ &= 355. \end{aligned}$$

◁

2.1.2 Product

Voor het *product* bestaat er een analoge notatie, die geschreven wordt m.b.v. de Griekse hoofdletter Π:

$$\prod_{i=m}^{n} x_i = x_m \times x_{m+1} \times \ldots \times x_{n-1} \times x_n.$$

Voorbeeld 2.2 *De notatie*

$$P = \prod_{i=2}^{4} i^2$$

is een compacte schrijfwijze voor het product van de kwadraten van natuurlijke getallen met de onder- en bovengrens respectievelijk gelijk aan 2 en 4. Voluit betekent dit het volgende product:

$$\begin{aligned} P &= 2^2 \times 3^2 \times 4^2 \\ &= 4 \times 9 \times 16 \\ &= 576. \end{aligned}$$

◁

2.2 Veeltermen

In deze sectie wordt eerst het begrip "veelterm" toegelicht. Vervolgens worden een aantal bewerkingen met veeltermen beschreven.

2.2.1 Begrippen en notaties

Definitie 2.3 Een *veelterm* (of *polynoom*) in een variabele z is een uitdrukking van de vorm

$$f(z) \quad = \quad a_0 + a_1 z + a_2 z^2 + \ldots + a_n z^n \tag{2.1}$$

waarin n een natuurlijk getal is en de getallen a_i (met $i = 0 \ldots n$) de *coëfficiënten* van de veelterm worden genoemd. ⧫

Wanneer n gelijk is aan $+\infty$ spreken we van een *oneindige veelterm*. In deze thesis beschouwen we veeltermen waarvan de coëfficiënten natuurlijke getallen zijn.

Definitie 2.4 De *graad* van een veelterm is gelijk aan de exponent van de hoogste macht van z waarvan de coëfficiënt verschillend is van 0. ⧫

Voorbeeld 2.5 *De graad van de veelterm*

$$f(z) \quad = \quad 5z^0 - 2z^1 + 0z^2 - 4z^3 + 2z^4 + 0z^5$$

is gelijk aan 4. ◁

In hetgeen volgt worden de termen waarvan de coëfficiënt gelijk is 0 meestal weggelaten, tenzij het expliciet nodig is om ze te vermelden.

Gebruikmakende van de sommatie-operator uit Sectie 2.1.1 kunnen we de veelterm $f(z)$ in formule (2.1) compacter noteren als volgt:

$$f(z) \quad = \quad \sum_{i=0}^{n} a_i z^i. \tag{2.2}$$

Vaak willen we in een veelterm $f(z)$ de coëfficiënt van de n-de term $a_n z^n$ aanduiden. Dit doen we m.b.v. de volgende notatie:

$$a_n \quad = \quad [z^n] f(z). \tag{2.3}$$

Dit noemen we de *extractie* van de coëfficiënt van z^n in $f(z)$.

Voorbeeld 2.6 *Beschouw de veelterm $f(z)$ uit Voorbeeld 2.5. Hiervoor geldt hetvolgende:*

$$[z^3]f(z) \;=\; -4. \qquad \triangleleft$$

2.2.2 Bewerkingen met veeltermen

In deze sectie worden een aantal bewerkingen met veeltermen toegelicht. Al deze bewerkingen resulteren telkens opnieuw in een veelterm. Hierbij valt op te merken dat we in deze bewerkingen telkens enkel gebruikmaken van veeltermen in dezelfde variabele z. We beschouwen in deze sectie ook enkel eindige veeltermen, maar alles wat hier beschreven staat geldt ook voor oneindige veeltermen.

Optelling

Zij $f(z)$ en $g(z)$ twee veeltermen in de variabele z:

$$\begin{aligned} f(z) &= \sum_{i=0}^{n} a_i z^i, \\ g(z) &= \sum_{j=0}^{m} b_j z^j. \end{aligned}$$

De som van deze veeltermen is gelijk aan:

$$f(z) + g(z) \;=\; \sum_{i=0}^{\max(n,m)} (a_i + b_i) z^i. \tag{2.4}$$

Bij het optellen van veeltermen moeten de coëfficiënten van de overeenkomstige term dus opgeteld worden. Merk op dat beide veeltermen een verschillende graad kunnen hebben. Daarom is de bovengrens van de sommatie in (2.4) gelijk aan het maximum van de graden. In dat geval zal de veelterm met de kleinste graad impliciet aangevuld worden met nullen als coëfficiënten in de termen waarvan de exponent groter is dan diens graad.

Het verschil van twee veeltermen is analoog met formule (2.4). Alleen moeten de coëfficiënten in dit geval van elkaar afgetrokken worden.

Voorbeeld 2.7 *Beschouw de volgende veeltermen $f(z)$ en $g(z)$:*

$$\begin{aligned} f(z) &= 4z^0 - 2z^1 - 4z^2, \\ g(z) &= 3z^0 + 3z^2 + 2z^3. \end{aligned}$$

De som van $f(z)$ en $g(z)$ is gelijk aan:

$$\begin{aligned} f(z) + g(z) &= (4+3)z^0 + (-2+0)z^1 + (-4+3)z^2 + (0+2)z^3 \\ &= 7z^0 - 2z^1 - 1z^2 + 2z^3. \end{aligned}$$

Het verschil van $f(z)$ en $g(z)$ is gelijk aan:

$$\begin{aligned} f(z) - g(z) &= (4-3)z^0 + (-2-0)z^1 + (-4-3)z^2 + (0-2)z^3 \\ &= 1z^0 - 2z^1 - 7z^2 - 2z^3. \end{aligned}$$ ◁

Vermenigvuldiging met een factor

Zij $f(z) = \sum_{i=0}^{n} a_i z^i$ een veelterm en zij $\lambda \in \mathbb{R}$ een willekeurige factor. Het product van λ en $f(z)$ is dan gelijk aan:

$$\lambda \times f(z) = \sum_{i=0}^{n} (\lambda \times a_i) z^i. \tag{2.5}$$

Bij het vermenigvuldigen van een veelterm met een factor moet elke coëfficiënt van de veelterm dus vermenigvuldigd worden met deze factor.

Voorbeeld 2.8 *Beschouw de veelterm $f(z)$ uit Voorbeeld 2.7 en zij de factor $\lambda = 3$. Het product van λ en $f(z)$ is dan gelijk aan:*

$$\begin{aligned} \lambda \times f(z) &= (3 \times 4)z^0 + (3 \times -2)z^1 + (3 \times -4)z^2 \\ &= 12z^0 - 6z^1 - 12z^2. \end{aligned}$$ ◁

Vermenigvuldiging van veeltermen

Zij $f(z)$ en $g(z)$ twee veeltermen in de variabele z:

$$\begin{aligned} f(z) &= \sum_{i=0}^{n} a_i z^i, \\ g(z) &= \sum_{j=0}^{m} b_j z^j. \end{aligned}$$

Het product van deze veeltermen is gelijk aan:

$$\begin{aligned} f(z) \times g(z) &= \sum_{i=0}^{n} a_i z^i \times \sum_{j=0}^{m} b_j z^j \\ &= \sum_{i=0}^{n} \sum_{j=0}^{m} a_i z^i \times b_j z^j \\ &= \sum_{i=0}^{n} \sum_{j=0}^{m} (a_i \times b_j)(z^i \times z^j) \\ &= \sum_{i=0}^{n} \sum_{j=0}^{m} (a_i \times b_j) z^{i+j}. \end{aligned}$$

Bij het vermenigvuldigen van twee veeltermen moeten alle coëfficiënten van beide veeltermen dus onderling met elkaar vermenigvuldigd worden. Voorts moeten ook alle exponenten van beide veeltermen onderling bij elkaar opgeteld worden. Bijgevolg zal het product altijd een hogere graad hebben dan de twee veeltermen die vermenigvuldigd worden, tenzij minstens één van de twee veeltermen graad 0 heeft.

Voorbeeld 2.9 *Beschouw de volgende veeltermen $f(z)$ en $g(z)$:*

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

Het product van $f(z)$ en $g(z)$ is dan gelijk aan:

$$\begin{aligned} f(z) \times g(z) &= (4 \times 3) z^1 z^2 + (4 \times 2) z^1 z^3 + (-2 \times 3) z^2 z^2 + (-2 \times 2) z^2 z^3 \\ &= 12z^3 + 8z^4 - 6z^4 - 4z^5 \\ &= 12z^3 + 2z^4 - 4z^5. \end{aligned}$$

◁

2.3 Verzamelingen

Verzamelingen zijn de basisbouwstenen van de wiskunde. Ze worden veelal gebruikt om begrippen in de wiskunde formeel te definiëren. In deze sectie lichten we een aantal veelgebruikte begrippen en bewerkingen i.v.m. verzamelingen toe.

2.3.1 Begrippen en notaties

Definitie 2.10 Een *verzameling* is een collectie van objecten. De objecten die tot een verzameling behoren worden ook wel de *elementen* van de verzameling genoemd. ⧫

Een verzameling is volledig bepaald door haar elementen. Bijgevolg speelt de volgorde van de elementen geen rol in een verzameling. Een tweede gevolg is dat herhaling van elementen ook geen rol speelt in een verzameling.

Een verzameling wordt in deze thesis altijd aangeduid met een kalligrafische hoofdletter en een willekeurig element van een verzameling met een kleine Griekse letter. Lidmaatschap van een element in een verzameling wordt aangeduid met het symbool $\in$. Stel bijvoorbeeld dat het element α lid is van de verzameling $\mathcal{A}$, dan noteren we dit met $\alpha \in \mathcal{A}$. Analoog wordt niet-lidmaatschap aangeduid met het symbool $\notin$.

Om te kunnen werken met een bepaalde verzameling moeten we ze nauwkeurig kunnen beschrijven. Een eerste manier om een verzameling te beschrijven is het opsommen van (een deel van) de elementen. Deze opsomming wordt tussen accolades geplaatst.

Voorbeeld 2.11 *De verzameling $\mathcal{A}$ van de eerste tien letters van het Nederlandse alfabet wordt beschreven als volgt:*

$$\mathcal{A} \;=\; \{a, b, c, d, e, f, g, h, i, j\}. \qquad \triangleleft$$

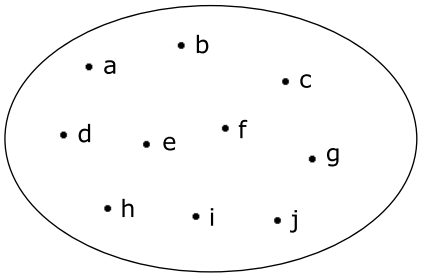
Een andere manier om een verzameling te beschrijven is het geven van één of meerdere eigenschappen waaraan de elementen van de verzameling voldoen.

Voorbeeld 2.12 *De verzameling $\mathcal{A}$ van alle studenten die ouder zijn dan 20 jaar wordt beschreven als volgt:*

$$\mathcal{A} \;=\; \{\alpha \mid \alpha \textit{ is student en leeftijd van } \alpha > 20\}.$$

De letter α wordt hier gebruikt om een willekeurig element van de verzameling $\mathcal{A}$ voor te stellen. ◁

Een verzameling kan schematisch voorgesteld worden d.m.v. een *Venn-diagram*. Dit is een grafische weergave, waarin de elementen van de verzameling worden afgebeeld als punten in een gesloten kromme in het vlak. De verzameling $\mathcal{A}$ uit Voorbeeld 2.11 wordt weergegeven in het Venn-diagram in Figuur 2.1.



Figuur 2.1: *Venn-diagram van de verzameling $\mathcal{A}$ uit Voorbeeld 2.11*

Een veelgebruikte verzameling is de verzameling van de natuurlijke getallen, die aangeduid wordt met het symbool $\mathbb{N}$:

$$\mathbb{N} \;=\; \{0, 1, 2, 3, 4, 5, \ldots\}.$$

Het getal 0 is per definitie een element van $\mathbb{N}$. De verzameling van de natuurlijke getallen zonder het getal 0 wordt aangeduid met het symbool $\mathbb{N}_0$.

Definitie 2.13 Het aantal elementen van een verzameling $\mathcal{A}$ noemt men de *cardinaliteit* van $\mathcal{A}$ en wordt genoteerd met $\text{card}(\mathcal{A})$ of $|\mathcal{A}|$. De verzameling waarvan de cardinaliteit gelijk is aan 0 noemt men de *lege verzameling*. Deze wordt aangeduid met $\emptyset$ of $\{\}$. Als de cardinaliteit van een verzameling gelijk is aan 1 noemt men deze verzameling een *singleton*. ♦

De cardinaliteit van een verzameling kan ook gelijk zijn aan $+\infty$. In dat geval spreken we van een *oneindige verzameling*. Als we echter de eerste elementen van een oneindige verzameling $\mathcal{A}$ kunnen opsommen, noemen we deze verzameling *aftelbaar oneindig*. In feite betekent dit dat er een bijectieve functie

$$f \;:\; \mathcal{A} \to \mathbb{N}_0$$

bestaat, die elk element van $\mathcal{A}$ afbeeldt op een volgnummer. De verzameling van de gehele getallen is een voorbeeld van een aftelbaar oneindige verzameling.

Een verzameling kan elementen bevatten die op hun beurt zelf verzamelingen zijn. In dat geval spreken we van een *geneste verzameling*.

Voorbeeld 2.14 *Beschouw de volgende geneste verzameling $\mathcal{C}$:*

$$\mathcal{C} \;=\; \{\{1,2\},\{3,4\},5\}\,.$$

De cardinaliteit van $\mathcal{C}$ is gelijk aan 3. Merk op dat de elementen 1, 2, 3 en 4 geen elementen zijn van $\mathcal{C}$. ◁

Definitie 2.15 Een *deelverzameling* $\mathcal{B}$ van een verzameling $\mathcal{A}$ wordt genoteerd met $\mathcal{B} \subseteq \mathcal{A}$ en is formeel gedefinieerd als volgt:

$$\mathcal{B} \subseteq \mathcal{A} \;\Leftrightarrow\; \forall \alpha \in \mathcal{B} : \alpha \in \mathcal{A}. \qquad \blacklozenge$$

Een verzameling $\mathcal{B}$ is dus een deelverzameling van een verzameling $\mathcal{A}$ als en slechts als elk element van $\mathcal{B}$ ook een element is van $\mathcal{A}$.

Voorbeeld 2.16 *Beschouw de verzameling $\mathcal{A}$ uit Voorbeeld 2.11. De verzameling $\mathcal{B} = \{a, b, c, d, e\}$ is een deelverzameling van $\mathcal{A}$.* ◁

In een Venn-diagram wordt een deelverzameling voorgesteld door een gesloten kromme die volledig omvat is in een andere gesloten kromme.

Definitie 2.17 De verzameling van alle deelverzamelingen van een verzameling $\mathcal{A}$ noemt men de *machtsverzameling* van $\mathcal{A}$ en wordt genoteerd met $\mathcal{P}(\mathcal{A})$. $\blacklozenge$

Voorbeeld 2.18 *Beschouw de verzameling $\mathcal{A} = \{1, 2\}$. De machtsverzameling $\mathcal{P}(\mathcal{A})$ is dan gelijk aan $\{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.* ◁

Merk op dat de lege verzameling $\emptyset$ een deelverzameling is van iedere verzameling. Bijgevolg is $\emptyset$ een element van de machtsverzameling van iedere verzameling.

2.3.2 Bewerkingen met verzamelingen

In deze sectie worden een aantal bewerkingen met verzamelingen toegelicht. Het resultaat van al deze bewerkingen is opnieuw een verzameling.

Unie

Definitie 2.19 De *unie* van twee verzamelingen $\mathcal{A}$ en $\mathcal{B}$ wordt genoteerd met $\mathcal{A} \cup \mathcal{B}$ en is gedefinieerd als volgt:

$$\mathcal{A} \cup \mathcal{B} \;=\; \{\alpha \mid \alpha \in \mathcal{A} \vee \alpha \in \mathcal{B}\}. \qquad \blacklozenge$$

De unie wordt dus gecreëerd door alle elementen van $\mathcal{A}$ en $\mathcal{B}$ samen te voegen in één verzameling. Merk op dat een element dat zowel in $\mathcal{A}$ als in $\mathcal{B}$ zit maar één keer in de unie zit, omdat herhaling niet mogelijk is in een verzameling. De cardinaliteit van de resulterende verzameling is dus niet noodzakelijk gelijk aan de som van de cardinaliteiten van de gebruikte verzamelingen. Dit is enkel het geval wanneer de doorsnede (zie volgende Sectie) van beide verzamelingen de lege verzameling is. Voor de cardinaliteit van de unie geldt in het algemeen dat:

$$|\mathcal{A} \cup \mathcal{B}| \;=\; |\mathcal{A}| + |\mathcal{B}| - |\mathcal{A} \cap \mathcal{B}|.$$

In het algemeen wordt de unie van n verzamelingen gecreëerd door alle elementen van al deze verzamelingen samen te voegen in één verzameling.

Voorbeeld 2.20 *Beschouw de verzamelingen* $\mathcal{A} = \{a, b\}$ *en* $\mathcal{B} = \{b, c, d\}$. *De unie van deze twee verzamelingen is gelijk aan* $\{a, b, c, d\}$. *We zien dat de cardinaliteit van deze unie gelijk is aan* $2 + 3 - 1 = 4$. $\triangleleft$

Doorsnede

Definitie 2.21 De *doorsnede* van twee verzamelingen $\mathcal{A}$ en $\mathcal{B}$ wordt genoteerd met $\mathcal{A} \cap \mathcal{B}$ en is gedefinieerd als volgt:

$$\mathcal{A} \cap \mathcal{B} \;=\; \{\alpha \mid \alpha \in \mathcal{A} \wedge \alpha \in \mathcal{B}\}. \qquad \blacklozenge$$

De doorsnede wordt dus gecreëerd door alle elementen die zowel in $\mathcal{A}$ als in $\mathcal{B}$ zitten, samen te voegen in één verzameling. Merk op dat als $\mathcal{A}$ en $\mathcal{B}$ geen gemeenschappelijke elementen hebben, dat de doorsnede ervan dan de lege verzameling is. In dat geval noemen we $\mathcal{A}$ en $\mathcal{B}$ *disjunct*.

De cardinaliteit van de doorsnede van twee verzamelingen is gelijk aan het aantal elementen dat ze gemeenschappelijk hebben.

In het algemeen wordt de doorsnede van n verzamelingen gecreëerd door alle gemeenschappelijke elementen van al deze verzamelingen samen te voegen in één verzameling.

Voorbeeld 2.22 *Beschouw de verzamelingen $\mathcal{A}$ en $\mathcal{B}$ uit Voorbeeld 2.20. De doorsnede van deze twee verzamelingen is gelijk aan $\{b\}$. We zien dat de cardinaliteit van deze doorsnede gelijk is aan 1.* ◁

De unie en doorsnede van n verzamelingen kunnen net zoals de sommatie en het product (zie Sectie 2.1) compacter genoteerd worden m.b.v. een index. Beschouw de n verzamelingen $\mathcal{A}_1, \ldots, \mathcal{A}_n$. De unie en doorsnede van deze verzamelingen worden respectievelijk compacter genoteerd met:

$$\bigcup_{i=1}^{n} \mathcal{A}_i$$

en

$$\bigcap_{i=1}^{n} \mathcal{A}_i.$$

Merk op dat het in dit geval vereist is dat de n verzamelingen aangeduid worden met dezelfde letter met telkens een verschillende index.

Verschil

Definitie 2.23 Het *verschil* van twee verzamelingen $\mathcal{A}$ en $\mathcal{B}$ wordt genoteerd met $\mathcal{A} \setminus \mathcal{B}$ en is gedefinieerd als volgt:

$$\mathcal{A} \setminus \mathcal{B} \;=\; \{\alpha \mid \alpha \in \mathcal{A} \wedge \alpha \notin \mathcal{B}\}.$$ ♦

Het verschil wordt dus gecreëerd door alle elemeten die in $\mathcal{A}$, maar niet in $\mathcal{B}$ zitten, samen te voegen in één verzameling. De cardinaliteit van het verschil van deze twee verzamelingen is gelijk aan het aantal elementen van $\mathcal{A}$ dat niet in $\mathcal{B}$ zit. Of met andere woorden:

$$|\mathcal{A} \setminus \mathcal{B}| \;=\; |\mathcal{A}| - |\mathcal{A} \cap \mathcal{B}|.$$

Voorbeeld 2.24 *Beschouw de verzamelingen $\mathcal{A}$ en $\mathcal{B}$ uit Voorbeeld 2.20. Het verschil van $\mathcal{A}$ en $\mathcal{B}$ is gelijk aan $\{a\}$. We zien dat de cardinaliteit van dit verschil gelijk is aan $2 - 1 = 1$.* ◁

Cartesisch product

Definitie 2.25 Zij $\mathcal{A}$ en $\mathcal{B}$ twee verzamelingen en zij $\alpha \in \mathcal{A}$ en $\beta \in \mathcal{B}$. Een groepering van α en β in een bepaalde volgorde noemt men een *geordend paar* (ook wel een *koppel* genoemd), dat genoteerd wordt met (α, β). De elementen α en β worden de *componenten* van het geordend paar genoemd. ⧫

Merk op dat, in tegenstelling tot verzamelingen, herhaling van componenten in een geordend paar wel mogelijk is en dat de volgorde van deze componenten wel belangrijk is. Zo zijn bijvoorbeeld de geordende paren $(1, 2)$ en $(2, 1)$ niet gelijk aan elkaar, maar de verzamelingen $\{1, 2\}$ en $\{2, 1\}$ wel.

In het algemeen is het mogelijk om n elementen van n verzamelingen (met $n \geq 0$) te groeperen in een bepaalde volgorde. In dat geval spreekt men van een *geordend n-tal* of een *(n-)tupel*. Als n gelijk is aan 0 spreken we van het *lege tupel*, dat we aanduiden met $()$.

De componenten van een tupel kunnen, net zoals in het geval van verzamelingen, op hun beurt zelf ook tupels zijn (met mogelijk een verschillend aantal componenten). In dat geval spreken we van een *genest tupel*.

Definitie 2.26 De *grootte* van een geordend n-tal is gelijk aan de som van de groottes van de componenten ervan. ⧫

Merk op dat de grootte van een geordend n-tal in het algemeen dus niet gelijk is aan het aantal componenten ervan.

Voorbeeld 2.27 *Zij $((a, a), a, a)$ een genest tupel met 3 componenten. De grootte van dit tupel is gelijk aan 4, want de grootte van de eerste component, die op zijn beurt ook een tupel is, is gelijk aan 2 en de groottes van de twee andere componenten zijn beide gelijk aan 1.* ◁

Definitie 2.28 Het *cartesisch product* van twee verzamelingen $\mathcal{A}$ en $\mathcal{B}$ wordt genoteerd met $\mathcal{A} \times \mathcal{B}$ en is gedefinieerd als volgt:

$$\mathcal{A} \times \mathcal{B} \;=\; \{(\alpha, \beta) \mid \alpha \in \mathcal{A} \wedge \beta \in \mathcal{B}\}. \qquad ⧫$$

Het cartesisch product wordt dus gecreëerd door alle elementen van $\mathcal{A}$ en $\mathcal{B}$ onderling te combineren tot geordende paren. Hieruit volgt dat de cardinaliteit van het cartesisch product precies gelijk is aan het product van de cardinaliteiten van

de twee gebruikte verzamelingen. Of met andere woorden:

$$|\mathcal{A} \times \mathcal{B}| \quad = \quad |\mathcal{A}| \times |\mathcal{B}|.$$

In het algemeen wordt het cartesisch product van n verzamelingen gecreëerd door alle elementen van al deze verzamelingen onderling te combineren tot geordende n-tallen.

Voorbeeld 2.29 *Beschouw de verzamelingen $\mathcal{A}$ en $\mathcal{B}$ uit Voorbeeld 2.20. Het cartesisch product van deze twee verzamelingen is gelijk aan:*

$$\{(a, b), (a, c), (a, d), (b, b), (b, c), (b, d)\}.$$

We zien dat de cardinaliteit van dit cartesisch product gelijk is aan $2 \times 3 = 6$. $\triangleleft$

2.3.3 Relaties tussen verzamelingen

In de sectie beschrijven we relaties tussen verzamelingen. Hierbij besteden we aandacht aan een belangrijke soort relatie, namelijk de equivalentierelatie.

Definitie 2.30 Zij $\mathcal{A}$ en $\mathcal{B}$ twee verzamelingen. Een *relatie* R van $\mathcal{A}$ naar $\mathcal{B}$ is gedefinieerd als een deelverzameling van het cartesisch product $\mathcal{A} \times \mathcal{B}$. $\blacklozenge$

In plaats van $(a, b) \in R$ schrijven we vaak aRb en zeggen we dat "a in relatie is met b t.o.v. R". Een relatie van een verzameling $\mathcal{A}$ naar dezelfde verzameling $\mathcal{A}$ noemen we een relatie *in* $\mathcal{A}$.

Voorbeeld 2.31 *Zij $\mathcal{L} = \{België, Duitsland, Nederland\}$ een verzameling die bestaat uit drie landen en $\mathcal{S} = \{Berlijn, Hasselt, Amsterdam, Brussel, Maastricht\}$ een verzameling die bestaat uit vijf steden.*

Een mogelijke relatie R die we kunnen definiëren van $\mathcal{L}$ naar $\mathcal{S}$ is de volgende:

$$\begin{array}{l}\{\ (België, Hasselt),\\ \ \ (Nederland, Amsterdam),\\ \ \ (Nederland, Maastricht)\ \}\end{array}$$

Definitie 2.32 Zij R een relatie van een verzameling $\mathcal{A}$ naar een verzameling $\mathcal{B}$. Het *domein* van R, genoteerd met $\text{dom}(R)$ is gedefineerd als volgt:

$$\text{dom}(R) \quad = \quad \{a \in \mathcal{A} \mid \exists b \in B : aRb\}. \qquad \blacklozenge$$

Het domein van een relatie R van $\mathcal{A}$ naar $\mathcal{B}$ is m.a.w. de verzameling van alle elementen a uit $\mathcal{A}$ waarvoor er een element b in $\mathcal{B}$ bestaat zodat het geordende paar (a, b) in R zit.

Definitie 2.33 Zij R een relatie van een verzameling $\mathcal{A}$ naar een verzameling $\mathcal{B}$. Het *beeld* van R, genoteerd met $\text{beeld}(R)$ is gedefineerd als volgt:

$$\text{beeld}(R) \quad = \quad \{b \in \mathcal{B} \mid \exists a \in A : aRb\}.$$

Het beeld van een relatie R van $\mathcal{A}$ naar $\mathcal{B}$ is dus de verzameling van alle elementen b uit $\mathcal{B}$ waarvoor er een element a in $\mathcal{A}$ bestaat zodat het geordende paar (a, b) in R zit. $\blacklozenge$

Voorbeeld 2.34 *Beschouw de relatie R uit Voorbeeld 2.31. Het domein en het beeld van R zijn gelijk aan:*

$$\begin{aligned} \text{dom}(R) &= \{België, Nederland\}. \\ \text{beeld}(R) &= \{Hasselt, Amsterdam, Maastricht\}. \end{aligned} \qquad \triangleleft$$

Een belangrijke soort van relaties *in* een verzameling zijn de zogenaamde equivalentierelaties.

Definitie 2.35 Een relatie R in een *niet-lege* verzameling $\mathcal{A}$ wordt een *equivalentierelatie* genoemd als en slechts als R voldoet aan de volgende drie eigenschappen:

1. $\forall a \in \mathcal{A} : aRa$
2. $\forall a, b, c \in \mathcal{A} : (aRb \wedge bRc) \Rightarrow aRc$
3. $\forall a, b \in \mathcal{A} : aRb \Rightarrow bRa$ $\blacklozenge$

Opdat een relatie R in een verzameling $\mathcal{A}$ een equivalentierelatie is, moet ze dus aan drie voorwaarden voldoen. Ten eerste moet R *reflexief* zijn. Dit wil zeggen dat elk element uit $\mathcal{A}$ in relatie met zichzelf moet zijn t.o.v. R.

Ten tweede moet R *transitief* zijn. Dit wil zeggen dat als een element a uit $\mathcal{A}$ in relatie is met een element b uit $\mathcal{A}$ t.o.v. R en b op zijn beurt in relatie is met een element c uit $\mathcal{A}$ t.o.v. R, dan moet a in relatie zijn met c t.o.v. R.

Ten derde moet R *symmetrisch* zijn. Dit wil zeggen dat als een element a uit $\mathcal{A}$ in relatie staat met een element b uit $\mathcal{A}$, dan moet b op zijn beurt in relatie staan met a.

Voorbeeld 2.36 *Beschouw de verzameling* $\mathcal{A} = \{-2, -1, 0, 1, 2\}$. *Hierin kunnen we een relatie AW definiëren waarin alle getallen met dezelfde absolute waarde in relatie staan met elkaar. Of met andere woorden:*

$$\begin{aligned} AW = \{\ & (-2,-2), (-2,2), \\ & (-1,-1), (-1,1), \\ & (0,0), \\ & (1,1), (1,-1), \\ & (2,2), (2,-2)\ \} \end{aligned}$$

De relatie AW is een equivalentierelatie, omdat ze voldoet aan de drie voorwaarden uit Definitie 2.35. ◁

Een equivalentierelatie in een verzameling verdeelt deze verzameling in disjuncte deelverzamelingen.

Definitie 2.37 Zij R een equivalentierelatie in een verzameling $\mathcal{A}$. Beschouw verder een element $a \in \mathcal{A}$. De equivalentieklasse K_a van a is gedefinieerd als volgt:

$$K_a \ = \ \{x \in \mathcal{A} \mid xRa\}. \qquad \blacklozenge$$

Een equivalentieklasse volgens een equivalentierelatie in een verzameling $\mathcal{A}$ is m.a.w. een niet-lege deelverzameling van $\mathcal{A}$, waarin alle elementen zitten die equivalent (of gelijkaardig) zijn met elkaar t.o.v. de beschouwde equivalentierelatie.

Wanneer we een equivalentierelatie R definiëren in een verzameling $\mathcal{A}$, noteren we dat met $\mathcal{A}\,/\mathbf{R}$. Een equivalentieklasse volgens een dergelijke equivalentierelatie stellen we meestal voor door één element ervan. Dit element noemen we de *representant* van de equivalentieklasse. De equivalentieklasse zelf noteren we vervolgens door de representant ervan tussen vierkante haken te zetten, gevolgd door de naam van de equivalentierelatie in subscript.

Voorbeeld 2.38 *Beschouw de verzameling $\mathcal{A}$ en de equivalentierelatie AW uit Voorbeeld 2.36. De equivalentieklassen volgens de relatie AW zijn de volgende:*

$$\begin{aligned} K_{-2} = K_2 &= \{-2, 2\} \\ K_{-1} = K_1 &= \{-1, 1\} \\ K_0 &= \{0\} \end{aligned}$$

Deze equivalentieklassen noteren we met $[-2]_{AW}, [-1]_{AW}$ en $[0]_{AW}$. ◁

Ten slotte beschouwen we de zogenaamde *triviale* equivalentierelatie in een verzameling $\mathcal{A}$, die alle elementen van $\mathcal{A}$ in één grote equivalentieklasse groepeert die gelijk is aan de verzameling $\mathcal{A}$.

2.4 Strings en formele talen

In deze sectie worden eerst een aantal begrippen en eigenschappen omtrent strings en formele talen beschreven. Vervolgens worden er een aantal bewerkingen met strings en formele talen gedefinieerd.

Definitie 2.39 Een *alfabet* is een eindige verzameling van symbolen. ⧫

Een alfabet wordt meestal voorgesteld door de griekse letter Σ en willekeurige symbolen ervan worden meestal aangeduid met kleine letters vooraan in het Latijns alfabet, zoals a, b, c, of door cijfers.

Een voorbeeld van een alfabet is de verzameling van alle ASCII-symbolen. De letters a, b, c, de spatie en de leestekens zijn voorbeelden van symbolen van dit alfabet.

Definitie 2.40 Een *string* (ook soms *woord* genoemd) over een alfabet Σ is een eindige rij symbolen over Σ. ⧫

Een willekeurige string wordt meestal voorgesteld door kleine letters achteraan in het Latijns alfabet, zoals u, v, w, x, y of z.

Een string over een alfabet kan ook wiskundig gedefinieerd worden als een functie van een beginsegment van de natuurlijke getallen naar het alfabet. Zij w een string over het alfabet Σ, gedefinieerd als de functie $w : \{1, \ldots, n\} \to \Sigma$, dan noemen we het getal n de *lengte* van w, wat genoteerd wordt met $|w|$. Er is slechts één string van lengte 0. Deze string wordt de *lege string* genoemd en wordt aangeduid met

de Griekse letter ϵ. Als w een string is en als $1 \leq i \leq |w|$, dan duiden we het i-de symbool van w aan met $w(i)$. Merk ten slotte op dat de symbolen van een bepaald alfabet kunnen gezien worden als strings van lengte 1 over dat alfabet.

Voorbeeld 2.41 *Over het Nederlandse alfabet is "informatica" een string van lengte 11.* ◁

Definitie 2.42 Zij u en v twee strings over een alfabet Σ. De *concatenatie* van u en v, genoteerd met $u.v$ of uv, is de string $w : \{1, \ldots, |u|+|v|\} \to \Sigma$ die gedefinieerd is al volgt:

$$w(i) = \begin{cases} u(i) & \text{als } 1 \leq i \leq |u| \\ v(i - |u|) & \text{als } |u| + 1 \leq i \leq |u| + |v|. \end{cases}$$

De concatenatie van twee strings wordt dus bekomen door alle symbolen ervan achter elkaar te schrijven. In het algemeen wordt de concatenatie van n strings bekomen door alle symbolen van alle strings achter elkaar te schrijven. Een string kan ook n keer geconcateneerd worden met zichzelf. Daarom voeren we, naar analogie met de vermenigvuldiging en machtsverheffing van getallen, de volgende notatie in:

$$\begin{aligned} w^0 &= \epsilon \\ w^1 &= w \\ w^{n+1} &= w^n.w \qquad \text{als } n \geq 1. \end{aligned}$$

Voorbeeld 2.43 *Als w de string "ab" is over het alfabet $\Sigma = \{a, b\}$, dan is $w^3 = w.w.w$ de string "abbabb".* ◁

Definitie 2.44 Een *formele taal* over een alfabet Σ is een verzameling van strings over Σ. ⧫

Merk op dat een taal niet noodzakelijk eindig moet zijn. Een taal wordt dikwijls aangeduid met de hoofdletter L (eventueel met een subscript). Er is slechts één taal die geen enkele string bevat. Deze taal wordt, naar analogie met verzamelingen, de *lege taal* genoemd en aangeduid met $\emptyset$ of $\{\}$.

Voorbeeld 2.45 *De verzameling van alle syntactisch correcte Java-programma's is een taal over het vocabulair (alfabet) van Java.* ◁

De bewerkingen die we gedefinieerd hebben voor verzamelingen (zie Sectie 2.3.2) kunnen ook gebruikt worden voor talen. Het cartesisch product van talen wordt

echter de *concatenatie van talen* genoemd. De concatenatie van de talen L_1 en L_2, genoteerd met $L_1.L_2$ of L_1L_2, is gedefinieerd als volgt:

$$L_1.L_2 \;=\; \{v.w \mid v \in L_1 \wedge w \in L_2\}.$$

Merk op dat strings v en w eigenlijk gegroepeerd worden in een geordend paar (v, w), dat we voor de eenvoud noteren als de concatenatie van de strings v en w.

De concatenatie van twee talen bestaat dus uit alle concatenaties van een string van de eerste taal met een string van de tweede taal. In het algemeen wordt de concatenatie van n talen bekomen door alle strings van al deze talen te concateneren met elkaar.

Voorbeeld 2.46 *Beschouw de talen $L_1 = \{a, b\}$ en $L_2 = \{c, d\}$ over het alfabet $\Sigma = \{a, b, c, d\}$. De concatenatie van deze twee talen is gelijk aan $\{ac, ad, bc, bd\}$.*$\triangleleft$

Naar analogie met de concatenatie van een string met zichzelf voeren we nu de volgende notatie in voor de concatenatie van een taal L met zichzelf:

$$\begin{array}{rcll} L^0 & = & \{\epsilon\} & \\ L^1 & = & L & \\ L^{n+1} & = & L^n.L & \text{als } n \geq 1. \end{array}$$

Merk op de taal $\{\epsilon\}$ niet gelijk is aan de lege taal $\emptyset$. De taal $\{\epsilon\}$ bevat namelijk één string van lengte 0.

Deze notatie leidt tot een nieuwe bewerking op talen:

Definitie 2.47 Zij L een taal over een alfabet Σ. De *Kleene-sluiting* van L, genoteerd met L^*, is gedefinieerd als volgt:

$$L^* \;=\; \bigcup_{n \in \mathbb{N}} L^n. \qquad \blacklozenge$$

De Kleene-sluiting van een taal wordt dus bekomen door de taal n keer te concateneren met zichzelf (voor elke $n \geq 0$).

Voorbeeld 2.48 *Beschouw de taal L_1 uit Voorbeeld 2.46. Hiervoor geldt dat:*

$$\begin{array}{rcl} L_1^0 & = & \{\epsilon\}; \\ L_1^1 & = & \{a, b\}; \end{array}$$

$$\begin{aligned} L_1^2 &= \{aa, ab, ba, bb\}; \\ L_1^3 &= \{aaa, aab, aba, abb, baa, bab, bba, bbb\}; \\ &\vdots \end{aligned}$$

We zien dus dat L_1^* bestaat uit alle strings die enkel uit a's en b's bestaan. ◁

Merk op dat een formele taal over een alfabet Σ altijd een deelverzameling is van Σ^*.

2.5 Grammatica's

In deze sectie worden grammatica's geïntroduceerd die dienen om bepaalde verzamelingen formeel te beschrijven. Traditioneel dienen grammatica's om formele talen te beschrijven. In deze thesis worden grammatica's echter ook gebruikt om andere verzamelingen van objecten te beschrijven. We geven daarom een algemene definitie van grammatica's.

2.5.1 Begrippen en notaties

Definitie 2.49 Een *objectconstructor* is een functie Φ_o die op basis van een sequentie van k objecten $(r_1, \ldots, r_k)$ een nieuw object creëert dat genoteerd wordt met $\Phi_o(r_1, \ldots, r_k)$. ♦

De objecten r_i (met $1 \leq i \leq k$) worden de *directe componenten* van het nieuwe object $\Phi_o(r_1, \ldots, r_k)$ genoemd. De *componenten* van een object *in de brede betekenis* zijn het object zelf, de directe componenten ervan en de componenten in de brede betekenis van deze directe componenten. De *componenten* van een object *in de strikte betekenis* zijn de componenten ervan in de brede betekenis met uitzondering van het object zelf.

Voorbeeld 2.50 *Beschouw het object $r = ((a, b), (c, d))$. De directe componenten van r zijn (a, b) en (c, d). De componenten van r in de brede betekenis zijn $((a, b), (c, d)), (a, b), (c, d), a, b, c$ en d.* ◁

Definitie 2.51 Een *(verzameling)constructor* is een functie Φ_v die op basis van een sequentie van k verzamelingen $(\mathcal{R}_1, \ldots, \mathcal{R}_k)$ een nieuwe verzameling creëert die genoteerd wordt met $\Phi_v(\mathcal{R}_1, \ldots, \mathcal{R}_k)$. Een dergelijke verzamelingconstructor wordt geïnduceerd door een objectconstructor als volgt:

$$\Phi_v(\mathcal{R}_1, \ldots, \mathcal{R}_k) \;=\; \bigcup_{r_1 \in \mathcal{R}_1, \ldots, r_k \in \mathcal{R}_k} \{\Phi_o(r_1, \ldots, r_k)\}. \qquad \blacklozenge$$

Een verzamelingconstructor past dus een objectconstructor toe op alle mogelijke combinaties van objecten uit de beschouwde verzamelingen $(\mathcal{R}_1, \ldots, \mathcal{R}_k)$.

Het natuurlijke getal k wordt de *ariteit* van de verzamelingconstructor genoemd. Als we een verzamelingconstructor Φ_v toepassen op één verzameling $\mathcal{R}$ kan de ariteit k van deze constructor mogelijk elke (natuurlijke) waarde aannemen. In dat geval noemen we Φ_v een *multi-constructor* die gebruikmaakt van k objecten van dezelfde verzameling om een nieuw object te creëeren. Dit noteren we verkort op de volgende manier:

$$\Phi_v(\mathcal{R}) \;=\; \bigcup_{k \in \mathbb{N}} \Phi_v(\underbrace{\mathcal{R}, \ldots, \mathcal{R}}_{k}) \,/\mathbf{E},$$

waarbij $\mathbf{E}$ een equivalentierelatie is, die de gecreëerde objecten groepeert in equivalentieklassen volgens deze relatie. Merk op dat deze equivalentierelatie ook de triviale equivalentierelatie kan zijn, die de gecreëerde objecten samen in een grote verzameling groepeert.

Een multi-constructor is m.a.w. een verzameling van functies, waarin er voor elke mogelijk ariteit telkens één functie aanwezig is. Omdat een multi-constructor in feite één argument heeft, spreken we af dat de ariteit ervan gelijk is aan 1.

Voorbeelden van verzamelingconstructors zijn het cartesisch product en de unie, die beschreven werden in Sectie 2.3.2. Deze constructors zijn geen multi-constructors, want hun ariteit is gelijk aan 2. Daarom worden de unie en het cartesisch product ook wel *binaire* constructors genoemd.

In Definitie 2.25 werd reeds de volgende objectconstructor gedefinieerd:

$$\Phi_o(r_1, r_2) \;=\; (r_1, r_2).$$

Hierbij zijn r_1 en r_2 twee objecten. Deze objectconstructor Φ_o creëert koppels waarbij de volgorde van de twee componenten r_1 en r_2 belangrijk is. De grootte

van het koppel (r_1, r_2) is, zoals eerder vermeld, gelijk aan de som van de groottes van r_1 en r_2.

Voorbeeld 2.52 *Beschouw de verzamelingen $\mathcal{R}_1$ en $\mathcal{R}_2$ en de bovenstaande objectconstructor Φ_o. Hiermee kunnen we het cartesisch product op het niveau van verzamelingen formeel definiëren als volgt:*

$$\Phi_c(\mathcal{R}_1, \mathcal{R}_2) \;=\; \bigcup_{r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2} \{\Phi_o(r_1, r_2)\}.$$

De verzamelingconstructor Φ_c past dus de objectconstructor Φ_o toe op elke mogelijk combinatie van een object uit $\mathcal{R}_1$ en een uit $\mathcal{R}_2$. ◁

De unie kunnen we echter niet op een dergelijke manier definiëren omdat er geen objectconstructor bestaat die de unie van twee objecten creëert. Deze objectconstructor zou twee objecten moeten creëeren, maar dat is niet toegelaten volgens Definitie 2.49. Er is m.a.w. geen objectconstructor die we als basis gebruiken om de unie-constructor op verzamelingen te definiëren. Bijgevolg definiëren we de unie enkel op het niveau van verzamelingen, namelijk als volgt:

$$\Phi_u(\mathcal{R}_1, \mathcal{R}_2) \;=\; \mathcal{R}_1 \cup \mathcal{R}_2.$$

Andere verzamelingconstructors worden geïntroduceerd in Sectie 3.3.

Definitie 2.53 Een *grammatica* G is een viertupel (T, N, S, P) bestaande uit:

$$G \begin{cases} \text{Een verzameling } \textit{terminals } T \\ \text{Een verzameling } \textit{nonterminals } N \\ \text{Een } \textit{startsymbool } S \in N \\ \text{Een verzameling } \textit{producties } P \end{cases}$$

De producties zijn van de vorm:

$$U \to a \quad \text{of} \quad U \to V \quad \text{of} \quad U \to \Phi_v(R_1, \ldots, R_k)$$

waarbij a een terminal is, U, V, $R_1, \ldots, R_k$ nonterminals zijn en Φ_v een verzamelingconstructor is. Verder geldt dat U verschillend is van V. Merk op dat er in elke productie hoogstens één verzamelingconstructor voorkomt. ♦

De nonterminals van een grammatica (ook wel variabelen genoemd) worden aangeduid met hoofdletters en de terminals (ook wel atomen genoemd) met kleine

letters, tenzij anders vermeld wordt. De producties dienen om nieuwe objecten te construeren op basis van andere (kleinere) objecten. Aanvankelijk worden hiervoor de terminals gebruikt, want deze vormen de basisbouwstenen van alle objecten die uit een grammatica gegenereerd worden.

Als een grammatica gebruikt wordt om een formele taal te beschrijven, dan is de verzameling van terminals het alfabet van deze taal. Verder zijn de gegenereerde objecten dan strings uit deze taal.

Opmerking: dikwijls beschouwen we grammatica's waarin $R_1, \ldots, R_k$ ook terminals kunnen zijn. Deze notatie is een afkorting waarbij meerdere producties samen verkort worden tot één productie. Zo is bijvoorbeeld de notatie $A \to \Phi(a)$ een afkorting van de producties $A \to \Phi(B)$ en $B \to a$, waarbij a een terminal is en A en B nonterminals zijn.

Een andere belangrijk begrip m.b.t. grammatica's is het afleiden van een object uit een grammatica. Hiervoor voeren we eerst de begrippen "compleet object" en "partieel object" in.

Definitie 2.54 Een *(compleet) object* is een object dat enkel opgebouwd is m.b.v. terminals. ⧫

Definitie 2.55 Een *partieel object* is een object dat opgebouwd kan worden m.b.v. terminals, nonterminals en constructors. ⧫

In de context van grammatica's kan een partieel object gezien worden als een niet-afgewerkt object, waarvan alle nonterminals en constructors moeten vervangen worden door terminals om een compleet object te verkrijgen.

Definitie 2.56 Een partieel object ω wordt *vervangen (of afgeleid) in één stap* uit een nonterminal W van een grammatica $G = (T, N, S, P)$, genoteerd met $W \Rightarrow \omega$, als en slechts als één van de volgende drie voorwaarden geldt:

- ω is een terminal én de productie $W \to \omega$ zit in P;
- ω is een nonterminal én de productie $W \to \omega$ zit in P;
- er zit een productie $W \to \Phi(R_1, \ldots, R_k)$ in P én $\omega \in \Phi(R_1, \ldots, R_k)$. ⧫

Definitie 2.57 Een partieel object $a\omega b$ wordt *vervangen (of afgeleid) in één stap* uit een partieel object aUb (waarbij U een nonterminal is), genoteerd met $aUb \Rightarrow a\omega b$ als en slechts als een productie voor U het partiële object ω creëert. ⧫

Definitie 2.58 Een compleet object ω wordt *afgeleid (in één of meerdere stappen)* uit een nonterminal W van een grammatica $G = (T, N, S, P)$, genoteerd met $W \stackrel{*}{\Rightarrow} \omega$, als en slechts als er een sequentie van vervangingen in één stap bestaat, die startend vanaf W, ω oplevert. ⧫

Deze sequentie van vervangingen in één stap wordt bekomen door te starten met een productie waar aan de linkerkant van de pijl de nonterminal W staat. In het partiële object aan de rechterkant worden vervolgens vervangingen in één stap uitgevoerd om uiteindelijk een compleet object te verkrijgen.

Opmerking: de verzameling van alle complete objecten die afgeleid worden uit een nonterminal U van een grammatica duiden we aan met dezelfde letter $\mathcal{U}$ in kalligrafisch schrift. Omgekeerd, wanneer we een verzameling $\mathcal{A}$ gebruiken in een grammatica gebruiken we de naam ervan in gewoon schrift als nonterminal A.

Definitie 2.59 Een object ω wordt *afgeleid* uit een grammatica $G = (T, N, S, P)$ als en slechts als ω afgeleid wordt uit het startsymbool S van deze grammatica.⧫

Definitie 2.60 De verzameling van alle objecten die afgeleid worden uit (het startsymbool S van) een grammatica $G = (T, N, S, P)$ noemt men de *taal* die aanvaard wordt door deze grammatica. Deze taal wordt genoteerd met $\mathcal{L}(G)$. ⧫

De afleiding van een object uit een grammatica kan voorgesteld worden d.m.v. een *afleidingsboom*. Dit is een grafische weergave die duidelijk de verschillende stappen (sequentie van vervangingen in één stap) van de afleiding weergeeft.

Voorbeeld 2.61 *Beschouw de volgende grammatica G_1, die een formele taal beschrijft:*

$$G_1 \begin{cases} T = \{[,], \epsilon\} \\ N = \{S\} \\ P_1 = S \to \epsilon \\ P_2 = S \to [S] \\ P_3 = S \to SS \end{cases}$$

Het startsymbool van G_1 is de enige gebruikte nonterminal, namelijk S.

Om de leesbaarheid te verhogen hebben we de bovenstaande grammatica G_1 verkort genoteerd. In de tweede en derde productie wordt namelijk impliciet gebruikgemaakt van het cartesisch product als constructor, dat in dit geval de concatenatie van twee

talen voorstelt. Zo is de productie $S \to [S]$ in feite een afkorting van de producties

$$\begin{aligned} S &\to A \times C \\ C &\to S \times B \\ A &\to [\\ B &\to] \end{aligned}$$

waarbij A, B en C nieuwe nonterminals zijn. Verder is de productie $S \to SS$ een afkorting van de productie

$$S \to S \times S$$

Opmerking: wanneer we een grammatica beschouwen die een formele taal beschrijft, maken we in het vervolg dikwijls gebruik van een dergelijke impliciete notatie.

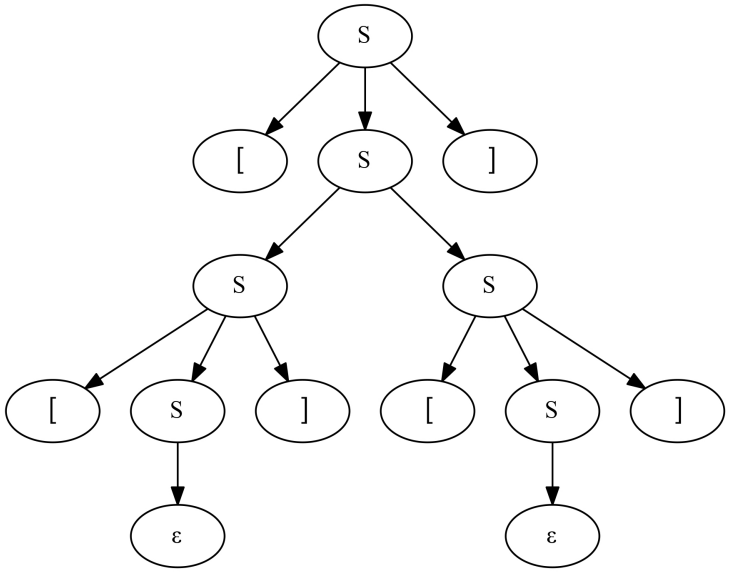
Het partiële object "$[\times S \times]$" kan bijvoorbeeld afgeleid worden uit de nonterminal S. Voorts kan bijvoorbeeld het complete object (in dit geval een string) "$[[][]]$" afgeleid worden uit G_1 op de volgende manier:

$$\begin{aligned} S &\xrightarrow{P_2} [S] \\ &\xrightarrow{P_3} [SS] \\ &\xrightarrow{P_2} [[S]S] \\ &\xrightarrow{P_2} [[S][S]] \\ &\xrightarrow{P_1} [[\epsilon][S]] \\ &\xrightarrow{P_1} [[\epsilon][\epsilon]] = [[][]] \end{aligned}$$

De afleidingsboom van deze string wordt weergegeven in Figuur 2.2.

Andere strings die gegenereerd worden door G_1 zijn bijvoorbeeld "ϵ", "$[]$", "$[][]$", "$[[[]]]$" en "$[[[[]][]]]$". Het is makkelijk in te zien dat $\mathcal{L}(G_1)$ de taal van de goedgeneste vierkante haakjesparen is. ◁

In hetgeen volgt zal een grammatica enkel nog gerepresenteerd worden d.m.v. de producties i.p.v. het volledige viertupel. De producties bevatten namelijk impliciet alle terminals en nonterminals van de grammatica en het is daarom niet nodig om deze nog eens expliciet te vermelden. In deze representatie is het startsymbool de nonterminal die aan de linkerkant van de pijl van de eerste productie staat.



Figuur 2.2: *Afleidingsboom van de string "[[][]]" uit grammatica G_1 uit Voorbeeld 2.61.*

2.5.2 Contextvrije grammatica's

Definitie 2.62 Een grammatica wordt *contextvrij* genoemd als er in de producties alleen gebruikgemaakt wordt van de verzamelingconstructors unie en cartesisch product. De taal die aanvaard wordt door een contextvrije grammatica wordt een *contextvrije taal* genoemd. ⧫

Voorbeeld 2.63 *De grammatica G_1 uit Voorbeeld 2.61 is contextvrij, omdat er in de producties enkel gebruikgemaakt wordt van de unie (zie ook Sectie 2.5.4) en het cartesisch product. Bijgevolg is $\mathcal{L}(G_1)$ een contextvrije taal.* ◁

2.5.3 Ambiguïteit

Definitie 2.64 Een grammatica wordt *ambigu* genoemd als er minstens één object bestaat dat op minstens twee verschillende manieren kan afgeleid worden uit deze grammatica. Voor een dergelijk object bestaan er m.a.w. twee (of meer) verschillende afleidingsbomen. In het andere geval wordt de grammatica *niet-ambigu* genoemd. ⧫

Definitie 2.65 Een (contextvrije) taal wordt *inherent ambigu* genoemd als ze alleen beschreven kan worden d.m.v. ambigue grammatica's [15, p. 108]. ⧫

Stelling 2.66 *Als er in een productie van een grammatica de unie gecreëerd wordt van twee niet-disjuncte verzamelingen, dan is de grammatica ambigu.*

Bewijs: Beschouw een grammatica G die een productie P bevat van de vorm $U \to A \cup B$, waarbij U, A en B nonterminals zijn en stel dat zowel A als B het object ω kunnen afleiden.

Stel dat we een object willen afleiden dat ω bevat. Wanneer P moet verwerkt worden om ω af te leiden, is er de keuze om de afleiding te vervolgen m.b.v. A of m.b.v. B, want beide nonterminals kunnen ω afleiden. Als we op dit punt kiezen om de afleiding te vervolgen via A zal deze afleiding in essentie verschillend zijn van de afleiding die we verkrijgen door B te volgen (en andersom). Het afgeleide object zal echter niet verschillend zijn, omdat we in beide gevallen ω toevoegen aan het reeds afgeleide object. We zien dus dat er een object bestaat dat op meerdere manieren kan afgeleid worden uit G en daarom is G een ambigue grammatica. □

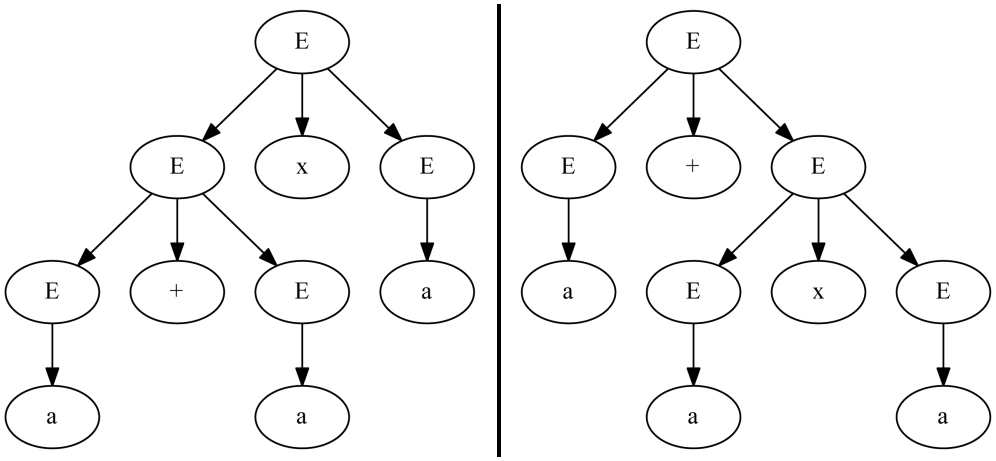
We kunnen dit ook in de grafische weergave bekijken. Beschouw de afleidingsbomen van de twee afleidingen uit het bewijs. Op het punt waar we moeten kiezen om de afleiding te vervolgen via A of B kunnen we in elke boom een deelboom op dit punt beschouwen. Beide deelbomen zijn verschillend omdat hun wortel respectievelijk gelabeld is met A en B. Bijgevolg zijn de twee afleidingsbomen van de volledige afleiding ook verschillend. Hieruit kunnen besluiten dat de grammatica G ambigu is.

Voorbeeld 2.68 *Beschouw de volgende contextvrije grammatica G_2, die ook een formele taal beschrijft:*

$$G_2 \begin{cases} E \to E + E \\ E \to E \times E \\ E \to (E) \\ E \to a \end{cases}$$

Deze grammatica genereert de taal van alle syntactisch correcte infix algebraïsche expressies, die alleen de operatoren $+$ en $\times$ en de variabele a gebruiken. Merk op dat $+$ en $\times$ terminals zijn van de grammatica G_2 en geen constructors. Merk ook op dat er in de eerste drie producties impliciet gebruikgemaakt wordt van het cartesisch product.

De grammatica is echter ambigu, omdat bijvoorbeeld de expressie "$a + a \times a$" op twee verschillende manieren kan afgeleid worden uit het startsymbool E. De twee verschillende afleidingsbomen worden weergegeven in Figuur 2.3.



Figuur 2.3: *Twee verschillende afleidingen van de expressie "$a+a\times a$" uit grammatica G_2 uit Voorbeeld 2.68.*

De taal die aanvaard wordt door grammatica G_2 is echter niet inherent ambigu. Er bestaat namelijk een andere, niet-ambigue grammatica die exact dezelfde taal aanvaardt als $\mathcal{L}(G_2)$. Die grammatica is de volgende:

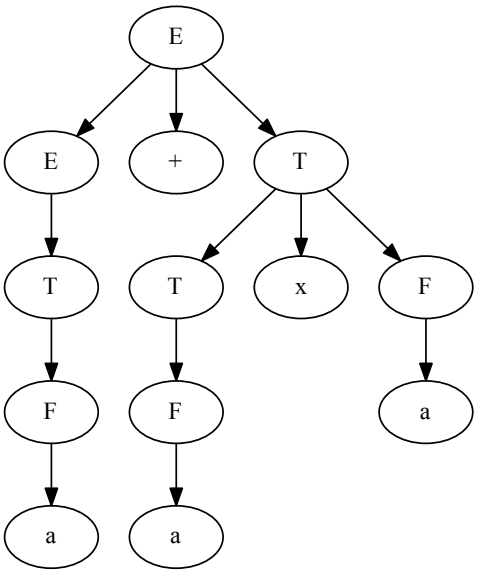
$$G_3 \begin{cases} E \to E + T \\ E \to T \\ T \to T \times F \\ T \to F \\ F \to (E) \\ F \to a \end{cases}$$

Om een algebraïsche expressie op te bouwen wordt in G_3 gebruikgemaakt van de extra nonterminals T en F, die respectievelijk termen en factoren van de expressie voorstellen. Merk opnieuw op dat $+$ en $\times$ atomen van G_3 zijn.

De expressie "$a + a \times a$" kan slechts op één manier afgeleid worden uit G_3. De afleidingsboom hiervan is weergegeven in Figuur 2.4. In het algemeen geldt dat G_3 geen enkel object kan genereren op meer dan één manier [15, p. 108]. Bijgevolg is G_3 een niet-ambigue contextvrije grammatica en is de taal die aanvaard wordt door G_3 (en dus ook door G_2) niet inherent ambigu. ◁

In het algemeen is het onbeslisbaar of een grammatica ambigu is of niet. Voor contextvrije grammatica's werd dit bewezen in o.a. [4].

In deze thesis beschouwen we enkel niet-ambigue grammatica's om problemen te vermijden bij het genereren van objecten. Omdat we bij het genereren van



Figuur 2.4: *Afleiding van de expressie "a+a×a" uit grammatica G_3 uit Voorbeeld 2.68.*

objecten in feite de afleidingsboom ervan creëeren, wordt een object dat meerdere afleidingsbomen heeft op meerdere manieren gegenereerd (zie Sectie 7.2).

2.5.4 Verschillende vormen van een grammatica

In deze sectie definiëren we een aantal vormen van grammatica's die het werken ermee vereenvoudigen.

Compacte vorm

De grammatica

$$A \rightarrow B \mid C \mid D$$

is een *compacte vorm* van de grammatica

$$\begin{aligned} A &\rightarrow B \\ A &\rightarrow C \end{aligned}$$

$$A \rightarrow D$$

Een dergelijke compacte vorm gebruiken we als afkorting om de leesbaarheid te verhogen. In de compacte vorm representeert het symbool | de unie van producties. Door een grammatica in de compacte vorm te noteren, zorgen we er dus voor dat elke nonterminal aan de linkerkant van precies één productie voorkomt.

Verder gebruiken we in het vervolg van deze thesis de volgende twee normaalvormen.

Chomsky Normal Form (CNF)

Definitie 2.69 Een grammatica staat in *Chomsky Normal Form (CNF)* als en slechts als de ariteit van de gebruikte verzamelingconstructors in de producties gelijk is aan 1 of 2. ⧫

De producties van een grammatica in CNF zijn dus van de vorm:

$$U \rightarrow a \quad \text{of} \quad U \rightarrow V \quad \text{of} \quad U \rightarrow \Phi_v(R_1) \quad \text{of} \quad U \rightarrow \Phi_v(R_1, R_2)$$

waarbij a een terminal is, U, V, R_1 en R_2 nonterminals zijn en Φ_v een verzamelingconstructor is. Merk op dat in de productie $U \rightarrow \Phi_v(R_1)$ Φ_v een multi-constructor kan zijn.

Zoals reeds in Sectie 2.5.1 vermeld, gebruiken we de notatie $\Phi(a, b)$ als afkorting van de producties $\Phi(A, B)$, $A \rightarrow a$ en $B \rightarrow b$.

Single Production Normal Form (SPNF)

Definitie 2.70 Een grammatica staat in *Single Production Normal Form (SPNF)* als en slechts als ze in CNF staat én er voor elke nonterminal van de grammatica precies één productie gedefinieerd is. ⧫

In een grammatica die in SPNF staat wordt dus in elke productie een nieuwe verzameling van objecten gedefinieerd, die benoemd wordt met de nonterminal van deze productie.

In Sectie 3.3.6 wordt beschreven hoe de grammatica's die we nodig hebben in deze thesis kunnen omgezet worden naar SPNF.

2.5.5 Recursieve grammatica's

Een grammatica G kan ook vereenvoudigd grafisch voorgesteld worden d.m.v. de bijhorende *afhankelijkheidsgraaf*. Dit is een *gerichte* graaf, waarin lussen mogen voorkomen. De nodes van deze graaf zijn gelabeld met de nonterminals van G. Een node n van deze graaf is verbonden met een node m als en slechts als de overeenkomstige nonterminal van m voorkomt aan de rechterkant van de pijl in een productie van de overeenkomstige nonterminal van n. Merk op dat een node ook met zichzelf kan verbonden zijn d.m.v. een lus.

Met behulp van de afhankelijkheidsgraaf van een grammatica kunnen we bepalen of de grammatica recursief is of niet.

Definitie 2.71 Een grammatica G wordt *recursief* genoemd als en slechts als de afhankelijkheidsgraaf van G minstens één cyclus bevat. In het andere geval noemen we de grammatica *iteratief* (of *niet-recursief*). ⧫

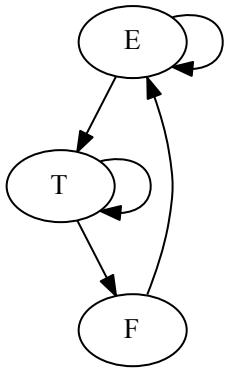
Onderzoeken of de afhankelijkheidsgraaf van een grammatica een cyclus bevat, kan op een efficiënte manier gebeuren met bijvoorbeeld het *depth-first-search (DFS) algoritme* [25, p. 603 - 612]. Wanneer we d.m.v. DFS bij het doorlopen van de afhankelijkheidsgraaf een node bezoeken die we reeds bezocht hebben, betekent dit dat de graaf een cyclus bevat. De enige aanpassing die we moeten maken aan DFS in dit geval is dat het algoritme mag stoppen van zodra er een cyclus gevonden is.

De tijdscomplexiteit van DFS in een graaf met n nodes en v verbindingen is $O(n + v)$ [25, p. 606]. Omdat een gerichte graaf met lussen met n nodes hoogstens n^2 verbindingen kan hebben, komt deze complexiteit overeen met $O(n + n^2) = O(n^2)$, wat als efficiënt beschouwd wordt.

Voorbeeld 2.72 *Beschouw de grammatica G_3 uit Voorbeeld 2.68. De afhankelijkheidsgraaf van G_3 wordt weergegeven in Figuur 2.5. Met behulp van deze afhankelijkheidsgraaf zien we dat G_3 recursief is, want (E, T, F, E) is bijvoorbeeld een cyclus in deze graaf.* ◁

Om te kunnen werken met recursieve grammatica's, beschouwen we ze meestal op een iteratieve manier. Beschouw bijvoorbeeld de volgende recursieve grammatica G:

$$G \begin{cases} A \to \epsilon \\ A \to B \times A \\ B \to b \end{cases}$$



Figuur 2.5: *Afhankelijkheidsgraaf van de grammatica G_3 uit Voorbeeld 2.68.*

Hierin wordt de recursie veroorzaakt door de tweede productie. We zien dat de objecten uit de verzameling $\mathcal{A}$ gecreëerd worden o.b.v. de objecten die reeds aanwezig zijn in $\mathcal{A}$. De verzameling $\mathcal{A}$ wordt m.a.w. in verschillende stappen opgebouwd. Hierbij hangt het resultaat van elke stap i af van het resultaat van de vorige stap $i-1$. Als we een recursieve grammatica op deze manier uitvoeren, leiden we in feite de objecten er op een iteratieve manier uit af.

De bovenstaande grammatica G kunnen we daarom op de volgende manier als iteratief beschouwen:

$$G \begin{cases} A^{[i]} \to \epsilon \\ A^{[i]} \to B \times A^{[i-1]} \\ B \to b \end{cases}$$

We starten met de verzameling $A^{[0]}$ gelijk te stellen aan de lege verzameling $\emptyset$. Vervolgens laten we de index i starten vanaf 1. Voor de eerste waarden van i krijgen we dan de volgende resultaten:

$$\begin{aligned} A^{[1]} &= \epsilon \mid B \times A^{[0]} \\ &= \epsilon \mid B \times \emptyset \\ &= \{\epsilon\} \end{aligned}$$

$$A^{[2]} = \epsilon \mid B \times A^{[1]}$$

$$
\begin{aligned}
&= \epsilon \mid B \times \{\epsilon\} \\
&= \{\epsilon, b\}
\end{aligned}
$$

$$
\begin{aligned}
A^{[3]} &= \epsilon \mid B \times A^{[2]} \\
&= \{\epsilon, b, bb\}
\end{aligned}
$$

Als we deze stappen tot in het oneindige zouden uitvoeren, verkrijgen we op deze manier alle objecten die uit de nonterminal A kunnen afgeleid worden.

Hoofdstuk 3

Combinatorische klassen

In dit hoofdstuk behandelen we combinatorische klassen. Dit zijn in essentie verzamelingen van objecten die een bepaalde structuur hebben. We starten met het introduceren van een aantal begrippen en notaties omtrent combinatorische klassen. Belangrijk hierbij is het onderscheid tussen gelabelde en ongelabelde klassen.

Vervolgens wordt er uitgelegd hoe (bepaalde) klassen formeel beschreven kunnen worden d.m.v. grammatica's. Hierbij wordt het begrip specificatie ingevoerd, samen met de cruciale notie van well-defined specificaties. Tot slot worden er enkele voorbeelden gegeven van specificaties van combinatorische klassen.

De begrippen die in dit hoofdstuk beschreven worden komen voornamelijk uit [20], [21] en [23].

3.1 Definities en notaties

Definitie 3.1 Een *combinatorische klasse* (of kortweg *klasse*) $\mathcal{A}$ is een eindige of aftelbaar oneindige verzameling waarop een *groottefunctie* $|\cdot|$ gedefinieerd is die voldoet aan de volgende eigenschappen:

1. de grootte van een element is een natuurlijk getal (mogelijk nul);
2. het aantal elementen van een bepaalde grootte is eindig.

De elementen van een combinatorische klasse worden *combinatorische objecten* of *combinatorische structuren* genoemd. ♦

In wat volgt beschouwen we enkel *discrete* objecten. Voorbeelden hiervan zijn strings, grafen en bomen. Zulke objecten worden opgebouwd m.b.v. *atomen*, die

de kleinst mogelijke bouwstenen van een object vormen. In het geval van strings zijn dat bijvoorbeeld de symbolen van de string. In het geval van grafen en bomen zijn dat de nodes ervan.

Om met zulke objecten uit de "reële wereld" te kunnen werken in de combinatorische wereld, stellen we ze op een abstracte manier voor als een combinatorisch object. Zo stellen we atomen voor als combinatorische objecten van grootte 1. Verder stellen we grotere objecten die opgebouwd worden m.b.v. atomen voor als samengestelde combinatorische objecten. Doorheen dit hoofdstuk en voornamelijk in Sectie 3.4 worden een aantal voorbeelden gegeven van deze abstracte voorstelling.

Een alternatieve manier om een combinatorische klasse te definiëren is als een koppel $(\mathcal{A}, |\cdot|)$, waarbij de verzameling $\mathcal{A}$ eindig of aftelbaar oneindig is en de mapping $|\cdot| \in (\mathcal{A} \to \mathbb{N})$ zó is dat het inverse beeld (in dit geval een deelverzameling van $\mathcal{A}$) van elk natuurlijk getal eindig is.

Omdat een klasse een verzameling is, noteren we een klasse ook altijd met een kalligrafische hoofdletter en een object ervan met een kleine Griekse letter. De grootte van het object $\alpha \in \mathcal{A}$ wordt dan genoteerd met $|\alpha|$ (of $|\alpha|_{\mathcal{A}}$ als het expliciet duidelijk moet zijn uit welke klasse het object komt). De eindige deelverzameling (ook wel *subklasse* genoemd) van de klasse $\mathcal{A}$ met daarin alleen de objecten van grootte n duiden we aan met $\mathcal{A}_n$. Deze subklasse $\mathcal{A}_n$ is m.a.w. gelijk aan de verzameling $\{\alpha \in \mathcal{A} \mid |\alpha| = n\}$. De cardinaliteit van deze subklasse wordt genoteerd met A_n (of soms ook met $|\mathcal{A}_n|$ of $\text{card}(\mathcal{A}_n)$).

Dikwijls is het intuïtief duidelijk welke groottefunctie bij een combinatorische klasse hoort. In dat geval wordt de klasse enkel gerepresenteerd door de verzameling van objecten. In de gevallen waarin het niet duidelijk is welke groottefunctie beschouwd wordt, wordt deze wel expliciet vermeld. Als we bijvoorbeeld een klasse beschouwen die bestaat uit strings stellen we de grootte van zo'n string gelijk aan diens lengte, tenzij anders wordt vermeld.

Een formele taal is een combinatorische klasse, omdat enerzijds elke string uit de taal per definitie een eindige grootte heeft. Anderzijds is het aantal strings van een bepaalde grootte uit een formele taal steeds eindig omdat het gebruikte alfabet per definitie eindig is. Een string is m.a.w. een combinatorisch object.

3.2 Gelabelde versus ongelabelde universum

In de combinatoriek wordt een onderscheid gemaakt tussen het *gelabelde* en het *ongelabelde* universum. In het gelabelde universum beschouwt men combinatorische objecten waarvan elk atoom een *label* (of nummer) met zich meedraagt. Dit label moet uniek zijn in het gehele object. Hierdoor worden alle atomen van één object van elkaar onderscheiden.

In het ongelabelde universum dragen de atomen van objecten geen label. Daardoor worden alle atomen van één ongelabeld object als gelijken beschouwd. Dit onderscheid is voornamelijk belangrijk bij het beschrijven van combinatorische klassen (zie Sectie 3.3) en het tellen van objecten uit een dergelijke klasse (zie Hoofdstuk 4).

Definitie 3.2 Een *gelabeld* object van grootte n is een graaf, waarvan elke node (of atoom) genummerd is met een uniek natuurlijk getal. ⧫

Wat voornamelijk van belang is bij de gebruikte grafen is hun verzameling van nodes en hun labels. De verbindingen tussen de nodes zijn niet van cruciaal belang in deze thesis.

We maken verder ook een onderscheid tussen zwak-gelabelde en goed-gelabelde objecten:

Definitie 3.3 Een *zwak-gelabeld* object van grootte n is gelabeld (genummerd) met n willekeurige unieke getallen uit de verzameling van de natuurlijke getallen. Een *goed-gelabeld* (of gelabeld) object van grootte n is gelabeld met de getallen uit het interval $[1 \dots n]$. ⧫

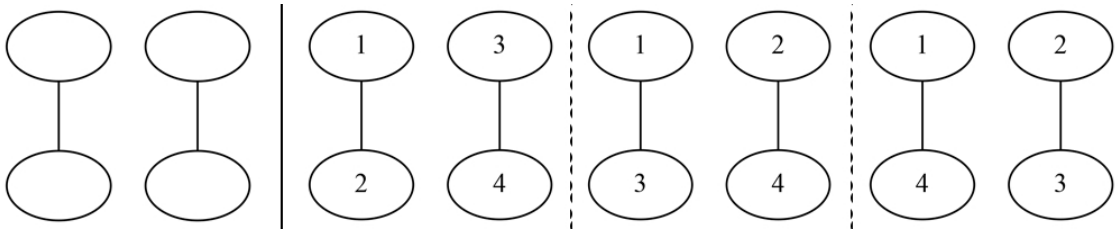
Een combinatorische klasse kan nooit zowel ongelabelde als gelabelde objecten bevatten, daarom spreken we in hetgeen volgt enkel over "ongelabelde klassen" en "gelabelde klassen". We nemen verder ook aan dat een gelabelde klasse enkel bestaat uit goed-gelabelde objecten.

Een gelabelde klasse bevat over het algemeen meer objecten van een bepaalde grootte dan zijn ongelabelde tegenhanger[1]. Dit komt doordat de verzameling van labels van een gelabeld object gepermuteerd kan worden. Vervolgens kunnen we

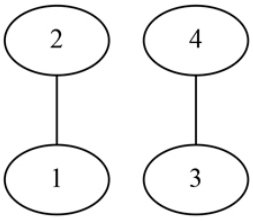
[1]De ongelabelde tegenhanger van een gelabeld object is het object met dezelfde onderliggende structuur, waarbij de labels zijn weggelaten. De ongelabelde tegenhanger van een gelabelde klasse is de ongelabelde klasse die de ongelabelde tegenhangers van alle objecten uit de gelabelde klasse bevat.

elk label uit een dergelijke permutatie mogelijk in volgorde toekennen aan een atoom uit het beschouwde object om op die manier een ander object te verkrijgen. Bijgevolg kunnen de atomen van een gelabeld object op maximaal $n!$ manieren herverdeeld worden.

In Figuur 3.1 wordt dit geïllustreerd m.b.v. ongerichte grafen. De drie gelabelde grafen in deze figuur zijn verschillend van elkaar omdat hun verzameling van bogen telkens verschillend is. De grafen hebben wel dezelfde onderliggende structuur. Zulke grafen noemen we *isomorfe* grafen. De graaf in Figuur 3.2 is echter wél gelijk aan de eerste gelabelde graaf in Figuur 3.1, omdat hun verzamelingen van bogen gelijk zijn aan elkaar.



Figuur 3.1: *Een ongelabelde graaf van grootte 4 en zijn drie isomorfe tegenhangers in het gelabeld universum.*



Figuur 3.2: *Een gelabelde graaf die gelijk is aan de eerste gelabelde graaf in Figuur 3.1.*

3.3 Beschrijven van combinatorische klassen

Om met combinatorische klassen te kunnen werken, moeten we ze formeel kunnen beschrijven (specificeren). Het beschrijven van (bepaalde) combinatorische klassen gebeurt d.m.v. specificaties die opgebouwd worden a.d.h.v. basisklassen en -constructors. Er valt echter wel op te merken dat niet elke specificatie die opgebouwd wordt a.d.h.v. deze basisklassen en -constructors een combinatorische klasse beschrijft.

We maken een onderscheid tussen specificaties die een ongelabelde klasse beschrijven en specificaties die een gelabelde klasse beschrijven. Voor de eenvoud noemen we deze respectievelijk "ongelabelde specificaties" en "gelabelde specificaties".

Deze sectie start met de definitie van een combinatorische (type-)specificatie. Vervolgens worden de basisklassen en -constructors gedefinieerd in zowel het ongelabelde als het gelabelde universum. Daarna wordt beschreven welke specificaties een combinatorische klasse beschrijven. Ten slotte wordt een standaardvorm voor gelabelde specificaties geïntroduceerd die het werken ermee eenvoudiger maakt.

3.3.1 Specificaties

Definitie 3.4 Een *specificatie* van een verzameling van objecten $\mathcal{A}$ is een vijftupel $(T, N, S, P, |\cdot|)$, waarbij (T, N, S, P) een grammatica is waaruit de objecten van $\mathcal{A}$ worden afgeleid en $|\cdot|$ een *groottefunctie* is, gedefinieerd van T naar $\mathbb{N}$, die op de grammatica gedefinieerd wordt. Zo is de grootte van een terminal $a \in T$ het getal $|a|$, gegeven door de specificatie. De grootte van een samengesteld object is gelijk aan de som van de groottes van de diens directe componenten. ⧫

Een specificatie definieert dus de grootte van de objecten die kunnen afgeleid worden uit een bepaalde grammatica. De groottefunctie voldoet aan de volgende twee fundamentele eigenschappen:

- de grootte van een object is altijd een natuurlijk getal (mogelijk nul);
- de grootte is *additief*, d.w.z. voor elke objectconstructor Φ_o in de producties geldt voor de objecten $r_1, \ldots, r_k$ dat $|\Phi_o(r_1, \ldots, r_k)| = |r_1| + \ldots + |r_k|$.

In het geval dat $k = 0$, definiëren we de grootte als volgt: $|\Phi_o()| = 0$.

Merk op dat de additiviteitseigenschap hier enkel gedefinieerd is op het niveau van objecten. Ze geldt echter telkens automatisch voor de bijhorende verzamelingconstructor, omdat deze zelf niets wijzigt aan de grootte van de gecreëerde objecten.

In hetgeen volgt gebruiken we de termen "grammatica" en "specificatie" door elkaar wanneer het intuïtief duidelijk is welke groottefunctie er bedoeld wordt.

In deze thesis worden specificaties gebruikt om combinatorische klassen formeel te beschrijven. De gebruikte specificaties worden opgebouwd a.d.h.v. zes (vijf in het gelabelde universum) basisverzamelingconstructors, die gedefinieerd worden in Sectie 3.3.3 en Sectie 3.3.4. Deze constructors zijn in essentie functies die allemaal werken met verzamelingen van objecten.

3.3.2 Basisklassen

In deze sectie definiëren we twee veelgebruikte soorten van basisklassen waarmee de basisconstructors kunnen werken. Deze basisklassen kunnen m.a.w. gezien worden als bouwstenen van andere combinatorische klassen. De basisklassen worden gedefinieerd voor zowel het ongelabelde als het gelabelde universum.

Neutrale klasse

Definitie 3.5 Een *neutrale klasse* bestaat uit één object van grootte 0. Een dergelijk object noemt men een *neutraal object*. ⧫

Vaak duiden we een neutrale klasse aan met de letter $\mathcal{E}$ en een neutraal object met de letter ϵ. Omdat een neutraal object grootte 0 heeft en m.a.w. geen atoom bevat, bevat het ook geen label. De representatie van een neutraal object (en dus ook van een neutrale klasse) is dus dezelfde in het ongelabelde en het gelabelde universum, maar merk op dat we in feite een ongelabeld neutraal object en een gelabeld neutraal object kunnen beschouwen.

Wanneer we gebruikmaken van verschillende neutrale klassen duiden we ze aan m.b.v. een subscript om het verschil duidelijk weer te geven. Zo stellen bijvoorbeeld $\mathcal{E}_{\square}$ en $\mathcal{E}_{\diamondsuit}$ de neutrale klassen voor die respectievelijk bestaan uit de neutrale objecten $\square$ en $\diamondsuit$.

Opmerking: deze objecten kunnen ook gezien worden als het resultaat van een objectconstructor zonder argumenten.

Om een neutrale klasse $\mathcal{E}$ te definiëren in een specificatie gebruiken we zowel in het ongelabelde universum als in het gelabelde de productie

$$E \rightarrow \epsilon.$$

Atomaire klasse

Definitie 3.6 Een *atomaire klasse* bestaat uit één object van grootte 1. Een dergelijk object noemt men ook wel eens een *atoom*. ⧫

Een gelabeld atoom bevat één label, namelijk het getal 1. We stellen een atoom in de "telwereld" meestal voor d.m.v. een cirkeltje. Als we een gelabeld atoom beschouwen, plaatsen we het label in dat cirkeltje. Als we een ongelabeld atoom beschouwen, blijft het cirkeltje leeg.

Verder kunnen we ook een *betekenis* (bv. een symbool uit een alfabet) uit de "reële wereld" aan een atoom geven. In dat geval zetten we de betekenis als markering rechtsonder in subscript. Dit is voornamelijk nuttig indien we met strings werken. Als we namelijk de markering van elk atoom van een object op een rij zetten, bekomen we op die manier de *stringrepresentatie* van het object. Merk op dat het niet noodzakelijk is om een betekenis aan een atoom te geven. In de "telwereld" heeft deze betekenis echter geen belang omdat we daarin enkel geïnteresseerd zijn in aantallen van objecten.

Voorbeeld 3.7 *Stel dat we het symbool* `while` *uit de vocabulaire van Java willen modelleren als een atoom. We modelleren m.a.w. het symbool* `while` *als een object van grootte 1. We noteren dit atoom in de "telwereld" dan als* $\bigcirc_{while}$ *als we een ongelabeld object beschouwen en als* $\textcircled{1}_{while}$ *in het gelabelde geval.*

Beschouw verder het samengestelde ongelabelde object $(\bigcirc_{public}, \bigcirc_{static}, \bigcirc_{void})$ *dat we kunnen gebruiken om een string te modelleren. De stringrepresentatie van dit object is gelijk aan* `public static void`*. Merk op dat dit een string van grootte 3 is.* ◁

Een atomaire klasse duiden we meestal aan met de letter $\mathcal{Z}$. Wanneer we gebruikmaken van verschillende atomaire klassen duiden we ze ook aan m.b.v. een subscript om het verschil duidelijk weer te geven. Zo stellen bijvoorbeeld $\mathcal{Z}_a$ en $\mathcal{Z}_b$ de atomaire klassen voor die respectievelijk de symbolen `a` en `b` modelleren.

Om een atomaire klasse $\mathcal{Z} = \{\textcircled{1}_a\}$ te definiëren in een specificatie gebruiken we in het ongelabelde universum één van de volgende twee producties:

$$\begin{aligned} Z &\rightarrow a \text{ of} \\ Z &\rightarrow \bigcirc_a. \end{aligned}$$

In het gelabelde universum gebruiken we één van de volgende twee producties:

$$\begin{aligned} Z &\rightarrow a \text{ of} \\ Z &\rightarrow \textcircled{1}_a. \end{aligned}$$

3.3.3 Basisconstructors in het ongelabelde universum

In deze sectie definiëren we zes ongelabelde basisverzamelingconstructors die we kunnen gebruiken in specificaties om ongelabelde combinatorische klassen te beschrijven.

Zoals werd vermeld in Sectie 2.5.1, creëert een verzamelingconstructor Φ_v op basis van k verzamelingen van objecten $\mathcal{R}_1, \ldots, \mathcal{R}_k$ een nieuwe verzameling $\mathcal{R}$ door de gepaste objectconstructor Φ_o toe te passen op de objecten uit $\mathcal{R}_1, \ldots, \mathcal{R}_k$.

In hetgeen volgt veronderstellen we dat de argumenten van Φ_v combinatorische klassen zijn, tenzij anders vermeld wordt. Een constructor Φ_v dient m.a.w. om een nieuwe combinatorische klasse te creëeren op basis van k combinatorische klassen. Verder wordt de objectconstructor meestal impliciet gedefinieerd.

Cartesisch product

Definitie 3.8 Het *cartesisch product* $\mathcal{A} = \mathcal{B} \times \mathcal{C}$ van twee ongelabelde klassen $\mathcal{B}$ en $\mathcal{C}$ wordt identiek gedefinieerd als het cartesisch product van twee verzamelingen:

$$\mathcal{A} = \mathcal{B} \times \mathcal{C} \quad = \quad \{(\beta, \gamma) \mid \beta \in \mathcal{B} \wedge \gamma \in \mathcal{C}\}.$$

Voor de grootte van een object $\alpha \in \mathcal{A}$ geldt dat:

$$|\alpha|_{\mathcal{A}} \quad = \quad |\beta|_{\mathcal{B}} + |\gamma|_{\mathcal{C}}. \tag{3.1}$$

♦

Het cartesisch product is dus een binaire verzamelingconstructor die een klasse $\mathcal{A}$ creëert door telkens de gepaste objectconstructor toe te passen. Deze objectconstructor creëert een koppel (wat een samengesteld object is) door een object uit $\mathcal{B}$ te combineren met een object uit $\mathcal{C}$. De grootte van een dergelijk koppel is, zoals gedefinieerd in Definitie 2.26, gelijk aan de som van de groottes van de twee directe componenten ervan. Bijgevolg geldt de additiviteitseigenschap voor het cartesisch product.

Op een analoge manier kunnen we het cartesisch product van meerdere klassen definiëren.

Definitie 3.9 Zij $\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(n)}$ n ongelabelde combinatorische klassen (met $n > 2$). Het cartesisch product van deze klassen, genoteerd als $\mathcal{R}^{(1)} \times \ldots \times \mathcal{R}^{(n)}$, is gelijk aan:

$$\mathcal{R}^{(1)} \times \ldots \times \mathcal{R}^{(n)} \quad = \quad \{(r_1, \ldots, r_n) \mid r_i \in \mathcal{R}^{(i)} \text{ (voor } 1 \leq i \leq n)\}. \qquad \blacklozenge$$

Het cartesische product van n klassen creëert dus telkens tupels (wat ook weer samengestelde objecten zijn) van grootte n, waarbij de i-de component uit de i-de gebruikte klasse komt. De grootte van een dergelijk tupel is gelijk aan de som van de groottes van de n componenten.

Combinatorische unie

Definitie 3.10 Zij $\mathcal{E}_\square = \{\square\}$ en $\mathcal{E}_\diamondsuit = \{\diamondsuit\}$ twee ongelabelde neutrale klassen. De *combinatorische unie* (ook wel *combinatorische som* genoemd) $\mathcal{A} = \mathcal{B} + \mathcal{C}$ van twee ongelabelde combinatorische klassen $\mathcal{B}$ en $\mathcal{C}$ is gedefinieerd als volgt:

$$\mathcal{A} = \mathcal{B} + \mathcal{C} \quad = \quad (\mathcal{B} \times \mathcal{E}_\square) \cup (\mathcal{C} \times \mathcal{E}_\diamondsuit). \tag{3.2}$$

Voor de grootte van een object $\alpha \in \mathcal{A}$ geldt dat:

$$|\alpha|_\mathcal{A} \quad = \quad \begin{cases} |\alpha|_\mathcal{B} & \text{als } \alpha \in \mathcal{B} \\ |\alpha|_\mathcal{C} & \text{als } \alpha \in \mathcal{C}. \end{cases} \tag{3.3}$$

$\blacklozenge$

De combinatorische unie is een binaire verzamelingconstructor die een klasse $\mathcal{A}$ creëert die de *disjuncte* unie is van twee combinatorische klassen $\mathcal{B}$ en $\mathcal{C}$. In feite worden er disjuncte kopieën van $\mathcal{B}$ en $\mathcal{C}$ gecreëerd zodat er een onderscheid wordt gemaakt tussen objecten die in beide klassen zitten, zonder iets aan de grootte ervan te wijzigen. De resulterende klasse wordt ten slotte bekomen door de unie

van deze disjuncte kopieën te nemen. De reden waarom de combinatorische unie gebruikt wordt i.p.v. de standaardunie in combinatorische specificaties is om ambigue grammatica's te vermijden [21, p. 13].

Deze constructor kunnen we echter niet op het niveau van objecten definiëren, omdat er geen objectconstructor bestaat die de unie van twee objecten kan creëeren. Zo'n objectconstructor zou twee objecten moeten creëeren, maar dat is niet toegelaten volgens Definitie 2.49. Daarom definiëren we de combinatorische unie enkel op het niveau van verzamelingen.

Het onderscheid maken tussen objecten van $\mathcal{B}$ en $\mathcal{C}$ is zeer belangrijk in de context van het tellen van objecten. Stel dat een object zowel in $\mathcal{B}$ als in $\mathcal{C}$ zit. Als de constructie uit (3.2) toegepast wordt op dit object, zitten er twee nieuwe verschillende objecten in $\mathcal{A}$, die dan elk apart geteld worden. Als de constructie echter niet toegepast wordt op het object, zal dit object slechts éénmaal in $\mathcal{A}$ zitten en zal het bijgevolg ook maar éénmaal geteld worden, wat niet gewenst is omdat we in feite twee verschillende combinatorische objecten uit verschillende klassen beschouwen.

Als de twee klassen al disjunct waren, komt deze unie overeen met de standaardunie uit de verzamelingenleer. In hetgeen volgt wordt verondersteld dat $\mathcal{B}$ en $\mathcal{C}$ disjunct zijn, mogelijk door toepassing van de constructie uit (3.2).

Omdat we veronderstellen dat $\mathcal{B}$ en $\mathcal{C}$ disjunct zijn, is de grootte van een object $\alpha \in \mathcal{A}$ gelijk aan diens grootte in $\mathcal{B}$ als het daarvan een element is of aan de grootte in $\mathcal{C}$ in het andere geval. De twee groottefuncties die gedefinieerd zijn op deze twee klassen kunnen namelijk verschillend zijn doordat de klassen verschillende soorten van objecten kunnen bevatten. Omdat er niets aan de grootte van het object gewijzigd wordt, geldt de additiviteitseigenschap triviaal.

In wat volgt noteren we de neutrale objecten $\square$ en $\diamondsuit$ enkel nog als het expliciet nodig is om het onderscheid te makken tussen objecten uit $\mathcal{B}$ en $\mathcal{C}$. Zoals reeds gezegd is dit onderscheid enkel belangrijk in de telwereld. Bijgevolg noteren we de neutrale objecten niet als we de objecten uit $\mathcal{B}$ en $\mathcal{C}$ beschouwen in de echte wereld.

Sequence

Definitie 3.11 De *sequence*-constructie $\mathcal{A} = \text{SEQ}(\mathcal{B})$ van een ongelabelde klasse $\mathcal{B}$ (met $B_0 = 0$) is gedefinieerd als volgt:

$$\begin{aligned} \mathcal{A} = \text{SEQ}(\mathcal{B}) &= (\{\epsilon\} + \mathcal{B} + (\mathcal{B} \times \mathcal{B}) + (\mathcal{B} \times \mathcal{B} \times \mathcal{B}) + \ldots) / \mathbf{T} \\ &= \{\epsilon\} + \mathcal{B} / \mathbf{T} + \mathcal{B}^2 / \mathbf{T} + \mathcal{B}^3 / \mathbf{T} + \ldots, \end{aligned}$$

waarbij $\mathbf{T}$ de triviale equivalentierelatie is. Of in een andere vorm:

$$\mathcal{A} = \text{SEQ}(\mathcal{B}) = \{(\beta_1, \ldots, \beta_l) \mid l \geq 0 \wedge \beta_j \in \mathcal{B} \text{ (voor } 1 \leq j \leq l)\}.$$

Voor de grootte van een object $\alpha = (\beta_1, \ldots, \beta_l) \in \mathcal{A}$ geldt dat:

$$|\alpha|_{\mathcal{A}} = |\beta_1|_{\mathcal{B}} + \ldots + |\beta_l|_{\mathcal{B}}. \qquad \blacklozenge$$

De sequence-constructor is een multi-constructor die een klasse $\mathcal{A}$ creëert die alle mogelijke samengestelde objecten bevat die bekomen worden door n (met $n \geq 0$) keer een object uit een klasse $\mathcal{B}$ te combineren tot een n-tupel (ook wel een *sequentie* genoemd). Merk op dat het lege tupel () per definitie in de resulterende klasse zit. Dit lege tupel is in feite een neutraal object dat we aanduiden met ϵ.

Omdat de sequence-constructor gedefinieerd is in termen van het cartesisch product, is de grootte van een object uit de resulterende klasse gelijk aan de groottes van de n afzonderlijke directe componenten. Bijgevolg geldt de additiviteitseigenschap ook voor de sequence-constructor.

De voorwaarde dat de klasse $\mathcal{B}$ geen object van grootte 0 mag bevatten is van belang om een bepaald type van specificaties te kunnen uitsluiten (Zie Sectie 3.3.7).

Cycle

Definitie 3.12 De *cycle*-constructie $\mathcal{A} = \text{CYC}(\mathcal{B})$ van een ongelabelde klasse $\mathcal{B}$ (met $B_0 = 0$) is gedefinieerd als volgt:

$$\begin{aligned} \mathcal{A} = \text{CYC}(\mathcal{B}) &= (\text{SEQ}(\mathcal{B}) \setminus \{\epsilon\}) / \mathbf{C} \\ &= \mathcal{B} / \mathbf{C} + \mathcal{B}^2 / \mathbf{C} + \mathcal{B}^3 / \mathbf{C} + \ldots \end{aligned}$$

waarbij $\mathbf{C}$ een equivalentierelatie tussen tupels is, gedefinieerd als volgt:

$$(\beta_1, \ldots, \beta_r)\ \mathbf{C}\ (\beta'_1, \ldots, \beta'_r)$$

als en slechts als er een circulaire shift τ van $[1 \ldots r]$ bestaat zodat voor elke j geldt dat $\beta'_j = \beta_{\tau(j)}$ (met $1 \leq j \leq r$). M.a.w. voor een bepaalde d (de *offset* genoemd) geldt dat $\beta'_j = \beta_{1+(j-1+d) \bmod r}$. Beschouw bijvoorbeeld het object $(\bigcirc_a, \bigcirc_a, \bigcirc_b)$, dan is $(\bigcirc_a, \bigcirc_b, \bigcirc_a)$ een circulaire shift ervan met offset 1. ⧫

De cycle-constructor is een multi-constructor die een klasse $\mathcal{A}$ creëert door eerst de sequence-constructor uit Definitie 3.11 toe te passen op een klasse $\mathcal{B}$ en vervolgens hier het neutrale object uit te filteren. Daarna worden equivalente objecten (cycles) samengevoegd in de overeenkomstige equivalentieklassen volgens de relatie $\mathbf{C}$. Dit wil dus zeggen dat alle objecten die een circulaire shift van elkaar zijn samenzitten in dezelfde equivalentieklasse. Het neutrale object wordt uit de sequence gefilterd omdat er per definitie geen lege cycle bestaat.

Omdat de objecten geconstrueerd worden m.b.v. de sequence-constructor en daarna alleen maar gegroepeerd worden, geldt de additiviteitseigenschap ook voor de cycle-constructor. Bovendien nemen we aan dat de klasse $\mathcal{B}$ geen object van grootte 0 bevat door het gebruik van de sequence-constructor.

Een equivalentieklasse volgens de relatie $\mathbf{C}$ wordt aangeduid m.b.v. de notatie $[\ldots]_C$. Elke equivalentieklasse telt vervolgens als één object van grootte n, waarbij n de grootte is van elk object uit die equivalentieklasse, namelijk de som van de directe componenten.

Multiset

Definitie 3.13 De *multiset*-constructie $\mathcal{A} = \text{MSET}(\mathcal{B})$ van een ongelabelde klasse $\mathcal{B}$ (met $B_0 = 0$) is gedefinieerd als volgt:

$$\begin{aligned} \mathcal{A} = \text{MSET}(\mathcal{B}) &= \text{SEQ}(\mathcal{B})\ /\mathbf{P} \\ &= \{\epsilon\} + \mathcal{B}\ /\mathbf{P} + \mathcal{B}^2\ /\mathbf{P} + \mathcal{B}^3\ /\mathbf{P} + \ldots \end{aligned}$$

waarbij $\mathbf{P}$ een equivalentierelatie tussen tupels is, gedefinieerd als volgt:

$$(\alpha_1, \ldots, \alpha_r)\ \mathbf{P}\ (\beta_1, \ldots, \beta_r)$$

als en slechts als er een willekeurige permutatie σ van $[1 \ldots r]$ bestaat zodat voor elke j geldt dat $\beta_j = \alpha_{\sigma(j)}$ (met $1 \leq j \leq r$). Beschouw bijvoorbeeld het object $(\bigcirc_a, \bigcirc_b, \bigcirc_c)$, dan is het object $(\bigcirc_c, \bigcirc_b, \bigcirc_a)$ een permutatie ervan. ⧫

De multiset-constructor is een multi-constructor die een klasse $\mathcal{A}$ creëert door ook eerst de sequence-constructor uit Definitie 3.11 toe te passen op een klasse $\mathcal{B}$ en daarna equivalente objecten (multisets) samen te voegen in de overeenkomstige equivalentieklassen volgens de relatie $\mathbf{P}$. Dit wil dus zeggen dat alle objecten die een permutatie van elkaar zijn samenzitten in dezelfde equivalentieklasse. Merk op dat een circulaire shift een speciale soort permutatie is.

Omdat de objecten geconstrueerd worden m.b.v. de sequence-constructor en daarna alleen maar gegroepeerd worden, geldt de additiviteitseigenschap ook voor de multiset-constructor. We nemen opnieuw aan dat de klasse $\mathcal{B}$ geen object van grootte 0 bevat door het gebruik van de sequence-constructor.

Een equivalentieklasse volgens de relatie $\mathbf{P}$ wordt aangeduid m.b.v. de notatie $[\ldots]_P$. Elke equivalentieklasse telt, net zoals in het geval van de cycle-constructor, vervolgens als één object van grootte n, waarbij n de grootte is van elk object uit die equivalentieklasse.

Powerset

Definitie 3.14 De *powerset*-constructie $\mathcal{A} = \text{PSET}(\mathcal{B})$ van een ongelabelde klasse $\mathcal{B}$ (met $B_0 = 0$) is gedefinieerd als volgt:

$$\mathcal{A} = \text{PSET}(\mathcal{B}) \quad = \quad \{\{\beta_1, \ldots, \beta_l\} \mid l \geq 0 \wedge \beta_j \in \mathcal{B} \text{ (voor } 1 \leq j \leq l)\}. \quad (3.4)$$

Voor de grootte van een object $\alpha = \{\beta_1, \ldots, \beta_l\} \in \mathcal{A}$ geldt dat:

$$|\alpha|_{\mathcal{A}} \quad = \quad |\beta_1|_{\mathcal{B}} + \ldots + |\beta_l|_{\mathcal{B}}. \qquad \blacklozenge$$

De powerset-constructor is een verzamelingconstructor die een klasse $\mathcal{A}$ creëert die de machtsverzameling is van een klasse $\mathcal{B}$. De powerset-constructor voegt m.a.w. alle deelverzamelingen van $\mathcal{B}$ samen in de klasse $\mathcal{A}$. Een willekeurig object van de klasse $\mathcal{A}$ is dus een verzameling.

Merk op dat de powerset-constructie fel gelijkt op de sequence-constructie. Het enige verschil is dat de sequence-constructie tupels (sequenties) oplevert en de powerset-constructie verzamelingen (sets) oplevert. Doordat volgorde en herhaling niet van belang zijn bij verzamelingen, zijn vele door (3.4) geconstrueerde objecten gelijk. Zulke gelijke objecten worden maar één keer beschouwd en dus ook maar één keer geteld.

De grootte van een object uit $\mathcal{A}$ wordt niet gedefinieerd als de cardinaliteit ervan, maar als de som van de groottes van de elementen. Hierdoor voldoet ook de powerset-constructor aan de additiviteitseigenschap.

We nemen opnieuw aan dat $\mathcal{B}$ geen objecten van grootte 0 bevat om bepaalde types van specificaties te kunnen uitsluiten (Zie Sectie 3.3.7).

Voorbeeld 3.15 *Beschouw de ongelabelde combinatorische klasse* $\mathcal{A} = \{\bigcirc_a, \bigcirc_b\}$. *De klasse* $\mathrm{SEQ}(\mathcal{A})$ *is dan gelijk aan:*

$$\begin{aligned}\mathrm{SEQ}(\mathcal{A}) \;=\; &\{\epsilon, (\bigcirc_a), (\bigcirc_b), (\bigcirc_a, \bigcirc_a), (\bigcirc_a, \bigcirc_b), (\bigcirc_b, \bigcirc_a), (\bigcirc_b, \bigcirc_b),\\ &(\bigcirc_a, \bigcirc_a, \bigcirc_a), \ldots, (\bigcirc_a, \bigcirc_b, \bigcirc_a, \bigcirc_b), \ldots,\\ &(\bigcirc_b, \bigcirc_a, \bigcirc_a, \bigcirc_b), (\bigcirc_b, \bigcirc_a, \bigcirc_b, \bigcirc_a), \ldots\}.\end{aligned}$$

$\mathrm{CYC}(\mathcal{A})$ *is gelijk aan:*

$$\begin{aligned}\mathrm{CYC}(\mathcal{A}) \;=\; &\{[\bigcirc_a]_C, [\bigcirc_b]_C, [\bigcirc_a, \bigcirc_a]_C, [\bigcirc_a, \bigcirc_b]_C, [\bigcirc_b, \bigcirc_b]_C,\\ &[\bigcirc_a, \bigcirc_a, \bigcirc_a]_C, \ldots, [\bigcirc_a, \bigcirc_b, \bigcirc_a, \bigcirc_b)]_C,\\ &[\bigcirc_b, \bigcirc_a, \bigcirc_a, \bigcirc_b]_C, \ldots\}.\end{aligned}$$

$\mathrm{MSET}(\mathcal{A})$ *is gelijk aan:*

$$\begin{aligned}\mathrm{MSET}(\mathcal{A}) \;=\; &\{\epsilon, [\bigcirc_a]_P, [\bigcirc_b]_P, [\bigcirc_a, \bigcirc_a]_P, [\bigcirc_a, \bigcirc_b]_P, [\bigcirc_b, \bigcirc_b]_P,\\ &[\bigcirc_a, \bigcirc_a, \bigcirc_a]_P, \ldots, [\bigcirc_a, \bigcirc_b, \bigcirc_a, \bigcirc_b)]_P, \ldots\}.\end{aligned}$$

Ten slotte is $\mathrm{PSET}(\mathcal{A})$ *gelijk aan:*

$$\mathrm{PSET}(\mathcal{A}) \;=\; \{\emptyset, \{\bigcirc_a\}, \{\bigcirc_b\}, \{\bigcirc_a, \bigcirc_b\}\}.$$

Merk op dat het lege object een element is van $\mathrm{MSET}(\mathcal{A})$ *wegens Definitie 3.13. Merk ook op dat* $(\bigcirc_a, \bigcirc_b, \bigcirc_a, \bigcirc_b)$, $(\bigcirc_b, \bigcirc_a, \bigcirc_a, \bigcirc_b)$ *en* $(\bigcirc_b, \bigcirc_a, \bigcirc_b, \bigcirc_a)$ *in dezelfde equivalentieklasse zitten in* $\mathrm{MSET}(\mathcal{A})$ *omdat het permutaties van elkaar zijn. In* $\mathrm{CYC}(\mathcal{A})$ *zit* $(\bigcirc_b, \bigcirc_a, \bigcirc_a, \bigcirc_b)$ *echter in een andere equivalentieklasse dan* $(\bigcirc_a, \bigcirc_b, \bigcirc_a, \bigcirc_b)$ *en* $(\bigcirc_b, \bigcirc_a, \bigcirc_b, \bigcirc_a)$, *omdat het niet kan bekomen worden door een circulaire shift uit te voeren op deze twee tupels.* ◁

Tot slot definiëren we een verzameling van specificaties waarin enkel de zes hierboven geïntroduceerde verzamelingconstructors gebruikt worden.

Definitie 3.16 De verzameling Ω_o bestaat uit alle specificaties, waarvan de producties enkel gebruikmaken van de constructors combinatorische unie, cartesisch product, sequence, cycle, multiset en powerset. ⧫

Een belangrijke deelverzameling van Ω_o is de verzameling die bestaat uit alle specificaties die enerzijds contextvrij zijn en recursief mogen zijn. Anderzijds bestaat deze verzameling uit alle *iteratieve* specificaties die enkel de combinatorische unie, het cartesisch product en de sequence-constructor als constructor gebruiken. De specificaties uit deze deelverzameling kunnen elk een formele taal beschrijven, die bestaat uit strings [21, p. 16] [23, p. 51]. Bijgevolg noemen we deze specificaties dan ook *stringspecificaties* of *stringgrammatica's*.

Definitie 3.17 Indien een ongelabelde combinatorische klasse $\mathcal{A}$ kan beschreven worden d.m.v. een specificatie uit Ω_o, noemen we $\mathcal{A}$ *constructible* (of *beschrijfbaar*). ⧫

3.3.4 Basisconstructors in het gelabelde universum

In deze sectie definiëren we vijf gelabelde basisverzamelingconstructors die we kunnen gebruiken in specificaties om gelabelde combinatorische klassen te beschrijven.

De zes basisconstructors uit het ongelabelde universum hebben elk een equivalent in het gelabelde universum. De multiset- en de powerset-constructor hebben samen één gelabelde equivalent. Bijgevolg zijn er maar vijf verzamelingconstructors in het gelabelde universum.

De gelabelde constructors werken op een analoge manier als hun ongelabelde tegenhangers. Het enige verschil is dat de gelabelde constructors ervoor moeten zorgen dat de gecreëerde objecten goed-gelabeld zijn.

Verder is het nog belangrijk om op te merken dat we de ongelabelde en gelabelde constructors nooit tezamen in één specificatie mogen gebruiken. De ongelabelde constructors zijn namelijk enkel gedefinieerd voor ongelabelde klassen (en objecten) en de gelabelde constructors enkel voor gelabelde klassen. Aangezien een combinatorische klasse nooit zowel ongelabelde als gelabelde objecten kan bevatten, heeft het dus ook geen nut om de twee soorten basisconstructors door elkaar

te gebruiken. Hierdoor kunnen we de equivalente constructors in beide universums met dezelfde naam benoemen, zonder verwarring te veroorzaken.

Gelabeld product

Een belangrijke operatie die op gelabelde objecten gedefinieerd is, is het zogenaamde *herlabelen* van objecten. Deze operatie wijst een (mogelijk) nieuw uniek label toe aan elk atoom van een gelabeld object, zonder iets aan de onderliggende structuur te veranderen. De nieuwe labels mogen gekozen worden uit de verzameling $\mathbb{N}$. We eisen echter wel dat de oorspronkelijke relatieve orde tussen labels behouden blijft in het herlabelde object. Dit wil zeggen dat als een atoom a_1 in het oorspronkelijk object een groter label bevatte dan het atoom a_2, dat a_1 in het herlabelde object nog steeds een groter label moet bevatten dan a_2.

Twee herlabelingsoperaties die we in deze thesis gebruiken zijn de reductie en de expansie:

Definitie 3.18 Een *reductie* zet een zwak-gelabeld object ω_1 van grootte n om in een goed-gelabeld object ω_2 van grootte n. De labels van ω_1 worden gereduceerd tot het interval $[1 \ldots n]$ en zó toegekend aan de atomen van ω_2 dat de oorspronkelijke relatieve orde tussen de labels behouden blijft. We noteren de reductie van een object ω met $\mathfrak{R}(\omega)$. ⧫

Een expansie doet het omgekeerde:

Definitie 3.19 Een *expansie naar een (getallen)verzameling* G zet een goed-gelabeld object ω_1 van grootte n om in een zwak-gelabeld object ω_2 van grootte n. De getallen uit G worden zó toegekend aan de atomen van ω_2 dat de oorspronkelijke relatieve orde, die aanwezig is in ω_1, tussen de labels behouden blijft. De expansie van een object ω naar de verzameling G noteren we met $\mathfrak{E}_G(\omega)$. ⧫

In deze definitie is de verzameling G een deelverzameling van $\mathbb{N}$ die n verschillende getallen bevat.

Voorbeeld 3.20 *Beschouw het zwak-gelabelde object* $(\textcircled{4}, \textcircled{8}, \textcircled{6}, \textcircled{1})$ *De reductie hiervan is gelijk aan* $(\textcircled{2}, \textcircled{4}, \textcircled{3}, \textcircled{1})$*. De expansie naar de verzameling* $\{2, 5, 7, 9\}$ *van deze reductie is gelijk aan* $(\textcircled{5}, \textcircled{9}, \textcircled{7}, \textcircled{2})$*.* ◁

Merk op dat er voor elk gelabeld object juist één reductie en oneindig veel expansies bestaan.

Gebruikmakend van deze operaties kunnen we nu het gelabeld product definiëren. Merk op dat we het ongelabelde cartesisch product niet kunnen gebruiken in het geval van gelabelde objecten. Stel dat we twee goed-gelabelde objecten zouden combineren volgens het ongelabelde cartesisch product, dan zou bijvoorbeeld het label 1 twee keer voorkomen in het verkregen object, namelijk eenmaal in elke component. Hierdoor is het verkregen object noch zwak-gelabeld, noch goed-gelabeld, want elk label mag maar één keer voorkomen in een object.

We definiëren daarom een alternatief van het ongelabelde cartesisch product, dat wel gebruikt kan worden in het gelabelde universum. De onderliggende structuur van de objecten kan nog wel gecreëerd worden m.b.v. het cartesisch product, maar daarna moeten de objecten herlabeld worden om goed-gelabelde objecten te verkrijgen.

Eerst wordt er een initieel goed-gelabeld object gecreëerd m.b.v. een objectconstructor. Meestal bestaan er echter nog andere goed-gelabelde objecten die dezelfde onderliggende structuur hebben als dit initiële object. Daarom moet dit initiële object uiteindelijk een aantal keer herlabeld worden om al deze goed-gelabelde objecten met dezelfde onderliggende structuur te verkrijgen.

Definitie 3.21 Het *gelabeld product van twee goed-gelabelde objecten* β en γ is gedefinieerd als volgt:

$$\beta \star \gamma \quad = \quad \alpha = (\beta, \gamma'),$$

waarvoor geldt dat (β, γ') goed-gelabeld is én $\mathfrak{E}_{[\ |\beta|+1\ \ldots\ |\beta|+|\gamma|\ ]}(\gamma) = \gamma'$. ♦

Deze objectconstructor creëert dus een koppel α van grootte $|\beta| + |\gamma|$, waarbij de eerste component gelijk is aan het eerste oorspronkelijke object, dat al goed-gelabeld is. Verder moet het gehele koppel, wat één gelabeld object vormt, ook goed-gelabeld zijn. Dit wil zeggen dat de tweede component een expansie naar het interval $[|\beta| + 1 \ldots |\beta| + |\gamma|]$ is van het tweede oorspronkelijke object. Merk op dat er maar één mogelijke manier is om deze expansie te creëeren.

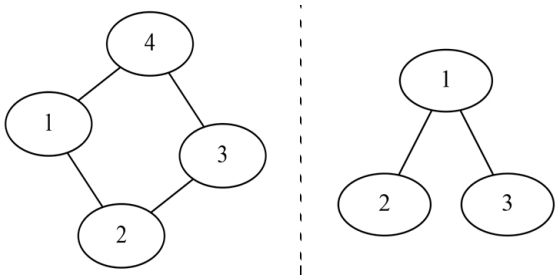
Om het verkregen initiële object α te herlabelen naar de andere goed-gelabelde objecten, definiëren we een functie *relabel* die de $|\beta| + |\gamma|$ labels van α herverdeelt over de atomen van α om telkens een ander goed-gelabeld object te verkrijgen, waarin de oorspronkelijke relatieve orde bewaard wordt. We beschouwen hierbij ook de identieke herverdeling, waardoor α zelf ook een resultaat van de functie *relabel* zal zijn. Merk op dat functie *relabel* in feite een verzamelingconstructor is.

Stel dat $|\beta| = m$ en $|\gamma| = n$. Het aantal mogelijke manieren waarop α kan herlabeld worden naar een goed-gelabeld object is gelijk aan[2]:

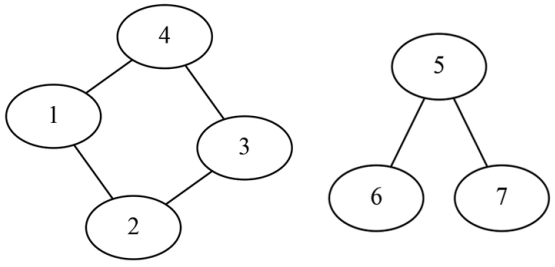
$$\binom{m+n}{m} = \frac{(m+n)!}{m! \times n!}$$

Wanneer we namelijk α willen herlabelen, moeten we telkens m unieke getallen kiezen uit het interval $[1 \ldots m+n]$ om de β-component te labelen. De n overige getallen worden daarna automatisch over de andere component verdeeld, omdat de oorspronkelijke relatieve orde bewaard moet blijven.

Voorbeeld 3.22 *Beschouw de twee gelabelde objecten β en γ uit Figuur 3.3. Het resultaat van het toepassen van de objectconstructor $\beta \star \gamma$ wordt weergegeven in Figuur 3.4. Twee van de $\binom{7}{4} = 35$ objecten die de functie relabel creëert, worden weergegeven in Figuur 3.5.* ◁

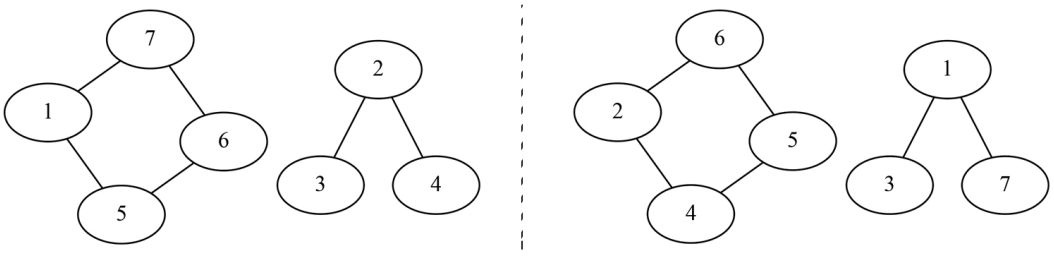


Figuur 3.3: *De gelabelde objecten β en γ uit Voorbeeld 3.22.*



Figuur 3.4: *Het resultaat van $\beta \star \gamma$ uit Voorbeeld 3.22.*

[2]Merk op dat $\binom{m+n}{m}$ gelijk is aan $\binom{m+n}{n}$.



Figuur 3.5: *Twee resultaten van de functie relabel uit Voorbeeld 3.22.*

Definitie 3.23 Het *gelabelde product* $\mathcal{A} = \mathcal{B} \star \mathcal{C}$ *van twee gelabelde klassen* $\mathcal{B}$ en $\mathcal{C}$ is gedefinieerd als volgt:

$$\mathcal{A} = \mathcal{B} \star \mathcal{C} \quad = \quad \bigcup_{\beta \in \mathcal{B}, \gamma \in \mathcal{C}} \textit{relabel}\ (\beta \star \gamma). \qquad \blacklozenge$$

Het gelabelde product van twee klassen $\mathcal{B}$ en $\mathcal{C}$ wordt dus gecreëerd door op elke combinatie van een object uit $\mathcal{B}$ en een uit $\mathcal{C}$ de objectconstructor toepassen. Op het resultaat hiervan moeten we vervolgens de functie *relabel* toepassen.

Merk op dat we in feite twee verzamelingconstructors gebruiken om het gelabelde product te definiëren. In [20] en [23] wordt het gelabelde product op een vergelijkbare manier gedefinieerd, alleen creëert hierbij de objectconstructor naast het initiële object ook nog de andere herlabelde objecten, die door onze functie *relabel* gecreëerd worden. We hebben echter gekozen voor onze aanpak om ons strikt aan Definitie 2.49 te houden. We willen namelijk dat een objectconstructor precies één object creëert.

Voorbeeld 3.24 *Beschouw de twee klassen* $\mathcal{B} = \{\textcircled{1}_a, \textcircled{1}_b\}$ *en* $\mathcal{C} = \{\textcircled{1}_a, \textcircled{1}_c\}$. *Het gelabeld product* $\mathcal{A}$ *van deze twee klassen is gelijk aan:*

$$\begin{aligned} \mathcal{A} = \mathcal{B} \star \mathcal{C} \quad = \quad & \{(\textcircled{1}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a), (\textcircled{1}_a, \textcircled{2}_c), (\textcircled{2}_a, \textcircled{1}_c), \\ & (\textcircled{1}_b, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a), (\textcircled{1}_b, \textcircled{2}_c), (\textcircled{2}_b, \textcircled{1}_c)\}. \end{aligned} \qquad \triangleleft$$

Net zoals in het ongelabelde geval kan het gelabeld product van meerdere klassen op een analoge manier gedefinieerd worden.

Definitie 3.25 Zij $r_1, \ldots, r_n$ n goed-gelabelde objecten (met $n > 2$). Het gelabeld product $r_1 \star \ldots \star r_n$ van deze objecten is gedefinieerd als volgt:

$$r_1 \star \ldots \star r_n \quad = \quad r = (r_1, r'_2, \ldots, r'_n)$$

waarvoor geldt dat het object $(r_1, r'_2, \ldots, r'_n)$ goed-gelabeld is én

$$\mathfrak{E}_{[\ (\sum_{j=1}^{i-1}|r_j|)+1\ \ldots\ \sum_{j=1}^{i}|r_j|\ ]}(r_i) = r'_i$$

voor elke $i \geq 2$. ♦

Het gelabeld product van n goed-gelabelde objecten creëert dus één n-tupel. De eerste component hiervan is een kopie van het eerste object, omdat dit object al goed-gelabeld is. De andere componenten moeten echter nog zó herlabeld worden dat het gehele object (het n-tupel) ook goed-gelabeld is. De ondergrens van het expansie-interval van de i-de component is gelijk aan de som van de groottes van de vorige $i-1$ componenten, vermeerderd met 1. De bovengrens van dit interval is gelijk aan de som van de groottes van de vorige $i-1$ componenten, vermeerderd met de grootte van de i-de component.

Deze objectconstructor creëert o.b.v. n objecten één initieel goed-gelabeld object r. De constructie van dit initiële object gebeurt altijd op dezelfde manier. De eerste n natuurlijke getallen worden gebruikt als labels voor de eerste component (van grootte n), zodat deze goed-gelabeld is. De volgende m natuurlijke getallen worden gebruikt als labels van de tweede component, enzovoort. De labels worden opnieuw zó aan de atomen van de componenten toegekend dat de oorspronkelijke orde erin bewaard blijft.

Naast het initieel gecreëerde object r bestaan er meestal nog andere goed-gelabelde objecten die dezelfde onderliggende structuur als r hebben. Het enige verschil tussen deze objecten is de manier waarop ze gelabeld zijn. Daarom definiëren we opnieuw een functie *relabel* die, vertrekkende van het initiële object r, de andere goed-gelabelde objecten met dezelfde onderliggende structuur construeert. Deze functie moet nu echter de labels van r herverdelen over meer dan twee componenten.

Stel dat $|r_i| = m_i$ (voor elke $1 \leq i \leq n$) en $\sum_{i=1}^{n} m_i = m$ (m is dus de grootte van het object r). Het aantal mogelijke manieren waarop r kan herlabeld worden is gelijk aan:

$$\binom{m}{m_1, \ldots, m_n} = \frac{m!}{m_1! \times \ldots \times m_n!}$$

Inderdaad, om de eerste component te herlabelen moeten we m_1 labels kiezen uit het totaal aantal labels (m). Om de tweede component te herlabelen, moeten we er m_2 kiezen uit $m - m_1$ labels, enzovoort. Om de laatste component te herlabelen,

hebben we nog maar keuze uit de overige m_n labels, omdat de relatieve orde ín de objecten moet bewaard blijven.

Definitie 3.26 Het gelabeld product $\mathcal{R} = \mathcal{R}_{(1)} \star \ldots \star \mathcal{R}_{(n)}$ van n gelabelde klassen $\mathcal{R}_{(1)} \ldots \mathcal{R}_{(n)}$ is gedefinieerd als volgt:

$$\mathcal{R} = \mathcal{R}_{(1)} \star \ldots \star \mathcal{R}_{(n)} \quad = \quad \bigcup_{r_1 \in \mathcal{R}_{(1)}, \ldots, r_n \in \mathcal{R}_{(n)}} \textit{relabel}\ (r_1 \star \ldots \star r_n). \qquad \blacklozenge$$

Combinatorische unie

De combinatorische unie in het gelabelde universum wordt analoog gedefinieerd aan die in het ongelabelde universum, maar in dit geval wordt er gebruikgemaakt van het gelabeld product.

Definitie 3.27 Zij $\mathcal{E}_\square = \{\square\}$ en $\mathcal{E}_\diamondsuit = \{\diamondsuit\}$ twee gelabelde neutrale klassen. De combinatorische unie $\mathcal{A} = \mathcal{B} + \mathcal{C}$ van twee gelabelde klassen $\mathcal{B}$ en $\mathcal{C}$ wordt gedefinieerd als volgt:

$$\mathcal{A} = \mathcal{B} + \mathcal{C} \quad = \quad (\mathcal{B} \star \mathcal{E}_\square) \cup (\mathcal{C} \star \mathcal{E}_\diamondsuit). \tag{3.5}$$

$\blacklozenge$

Het relabelen van het gecreëerde object gebeurt in dit geval triviaal, omdat zowel $\square$ en $\diamondsuit$ geen label moeten krijgen.

Net zoals in het geval van de ongelabelde combinatorische unie beschouwen we de neutrale objecten $\square$ en $\diamondsuit$ enkel in de telwereld.

Sequence

Definitie 3.28 Ook de gelabelde sequence-constructor wordt analoog gedefinieerd aan die in het ongelabelde universum, opnieuw gebruikmakend van het gelabeld product.

$$\mathcal{A} = \mathrm{SEQ}(\mathcal{B}) \quad = \quad (\{\epsilon\} + \mathcal{B} + (\mathcal{B} \star \mathcal{B}) + (\mathcal{B} \star \mathcal{B} \star \mathcal{B}) + \ldots)\ /\mathbf{T}$$

waarbij $\mathbf{T}$ opnieuw de triviale equivalentierelatie is. We veronderstellen opnieuw dat $B_0 = 0$. $\blacklozenge$

Cycle

Definitie 3.29 De gelabelde cycle-constructor wordt opnieuw analoog gedefinieerd aan die in het ongelabelde universum.

$$\mathcal{A} = \mathrm{CYC}(\mathcal{B}) \quad = \quad (\mathrm{SEQ}(\mathcal{B}) \setminus \{\epsilon\}) \,/\mathbf{C}$$

waarbij $\mathbf{C}$ dezelfde equivalentierelatie is als in Definitie 3.12, die objecten in dezelfde equivalentieklasse groepeert als en slechts als ze circulaire shifts van elkaar zijn. Omdat er gebruikgemaakt wordt van de sequence-constructor wordt er opnieuw verondersteld dat $B_0 = 0$. ⧫

Set

Definitie 3.30 De multiset- en de powerset-constructor hebben samen één equivalent in het gelabelde universum. Deze wordt de *set-constructor* genoemd en is een multi-constructor die analoog gedefinieerd wordt aan de multiset-constructor.

$$\mathcal{A} = \mathrm{SET}(\mathcal{B}) \quad = \quad \mathrm{SEQ}(\mathcal{B}) \,/\mathbf{P}$$

waarbij $\mathbf{P}$ dezelfde equivalentierelatie is als in Definitie 3.13, die objecten in dezelfde equivalentieklasse groepeert als en slechts als ze permutaties van elkaar zijn. Omdat er gebruikgemaakt wordt van de sequence-constructor wordt er opnieuw verondersteld dat $B_0 = 0$. ⧫

De reden dat de multiset- en de powerset-constructor één equivalent hebben, is omdat er geen notie van *multiset* bestaat in het gelabelde universum. In het gelabelde universum kan er namelijk geen herhaling optreden in een goed-gelabeld tupel, omdat twee willekeurige componenten van een dergelijk tupel altijd een verschillend label met zich meedragen en dus nooit gelijk aan elkaar kunnen zijn.

Voorbeeld 3.31 *Beschouw de gelabelde klasse* $\mathcal{B} = \{\textcircled{1}_a, \textcircled{1}_b\}$ *uit Voorbeeld 3.24. Dan zijn de klassen* $\mathrm{SEQ}(\mathcal{B}), \mathrm{SET}(\mathcal{B})$ *en* $\mathrm{CYC}(\mathcal{B})$ *gelijk aan:*

$$\begin{aligned}\mathrm{SEQ}(\mathcal{B}) \quad = \quad & \{\epsilon, (\textcircled{1}_a), (\textcircled{1}_b), (\textcircled{1}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a), (\textcircled{1}_a, \textcircled{2}_b),\\ & (\textcircled{2}_a, \textcircled{1}_b), (\textcircled{1}_b, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a), (\textcircled{1}_b, \textcircled{2}_b)\\ & (\textcircled{2}_b, \textcircled{1}_b), (\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_a), (\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_a), \ldots\\ & (\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), (\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_b), \ldots\}.\end{aligned}$$

$$\begin{aligned}\mathrm{SET}(\mathcal{B}) \;=\; \{&\epsilon, [\textcircled{1}_a]_P, [\textcircled{1}_b]_P, [\textcircled{1}_a, \textcircled{2}_a]_P, [\textcircled{1}_a, \textcircled{2}_b]_P,\\ &[\textcircled{1}_b, \textcircled{2}_a]_P, [\textcircled{1}_b, \textcircled{2}_b]_P, [\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_a]_P,\\ &[\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b]_P, [\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_b]_P, \ldots\}.\end{aligned}$$

$$\begin{aligned}\mathrm{CYC}(\mathcal{B}) \;=\; \{&[\textcircled{1}_a]_C, [\textcircled{1}_b]_C, [\textcircled{1}_a, \textcircled{2}_a]_C, [\textcircled{1}_a, \textcircled{2}_b]_C, [\textcircled{1}_b, \textcircled{2}_a]_C,\\ &[\textcircled{1}_b, \textcircled{2}_b]_C, [\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_a]_C, [\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_a]_C,\\ &[\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b]_C, [\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_b]_C, \ldots\}.\end{aligned}$$ ◁

Net zoals in het ongelabelde geval kunnen we een verzameling van specificaties definiëren waarin enkel de vijf hierboven geïntroduceerde verzamelingconstructors gebruikt worden.

Definitie 3.32 De verzameling Ω_g bestaat uit alle gelabelde specificaties, waarvan de producties enkel gebruikmaken van de constructors combinatorische unie, gelabeld product, sequence, set en cycle. ⧫

Net zoals in het ongelabelde universum kunnen we een deelverzameling van Ω_g beschouwen die bestaat uit alle stringspecificaties. Dit zijn alle specificaties uit Ω_g die contextvrij zijn en recursief mogen zijn óf die iteratief zijn en enkel de combinatorische unie, het gelabelde product en de gelabelde sequence-constructor als constructors gebruiken.

De term "constructible" kunnen we ook beschouwen in het gelabelde universum.

Definitie 3.33 Indien een gelabelde combinatorische klasse $\mathcal{A}$ kan beschreven worden d.m.v. een specificatie uit Ω_g, noemen we $\mathcal{A}$ *constructible*. ⧫

3.3.5 Beperkende constructors

Het is verder nog nuttig om op te merken dat het mogelijk is om een predicaat te definiëren bij de sequence-, cycle-, multiset- en set-constructor om het aantal componenten van de tupels in de resulterende combinatorische klasse te kunnen beperken.

Definitie 3.34 Zij Φ één van de constructors SEQ, CYC, MSET, SET, $\mathcal{A}$ een combinatorische klasse en Δ een predicaat over de natuurlijke getallen. We definiëren $\Phi_\Delta(\mathcal{A})$ dan als de combinatorische klasse, geconstrueerd door Φ, waarvan het aantal componenten van de tupels voldoet aan Δ. De constructor Φ noemen we een *beperkende* constructor. ⧫

De volgende predicaten zijn toegelaten: $= k$, $< k$, $\leq k$, $> k$, $\geq k$. Deze zorgen ervoor dat de tupels van de resulterende klasse respectievelijk exact k, minder dan k, minder dan of evenveel als k, meer dan k of meer dan of evenveel als k componenten bevatten.

Voorbeeld 3.35 *Beschouw de beperkende constructor*

$$\mathcal{A} \;=\; \mathrm{SEQ}_k(\mathcal{B})$$

Deze constructor creëert, op basis van een klasse $\mathcal{B}$, de klasse $\mathcal{A}$ die alleen sequenties bevat met exact k componenten. We kunnen dit op andere manier noteren als volgt:

$$\mathcal{A} \;=\; \underbrace{\mathcal{B} \times \ldots \times \mathcal{B}}_{k}.$$

Bijgevolg korten we deze beperkende constructor af met $\mathcal{B}^k$. In de andere vier gevallen gebruiken we een analoge afkorting. ◁

3.3.6 Specificaties in SPNF

In deze sectie beschrijven we hoe de specificaties die we gebruiken om combinatorische klassen te beschrijven kunnen omgezet worden naar SPNF (zie Sectie 2.5.4). Herinner dat er volgens Definitie 2.53 hoogstens één verzamelingconstructor mag voorkomen in de producties van een specificatie. Daarom zijn de producties in de specificaties uit Ω_o van de vorm

$$\begin{aligned}
A &\to \epsilon;\\
A &\to a;\\
A &\to B;\\
A &\to B + C;\\
A &\to B \times C;\\
A &\to \mathrm{PSET}(B);
\end{aligned}$$

$$\begin{aligned} A &\rightarrow \text{SEQ}(B); \\ A &\rightarrow \text{CYC}(B) \text{ of} \\ A &\rightarrow \text{MSET}(B), \end{aligned}$$

waarbij a een terminal is en A, B en C nonterminals zijn. Merk op dat in de laatste drie producties ook een beperkende constructor kan gebruikt worden.

De producties in de specificaties uit Ω_g zijn van de vorm:

$$\begin{aligned} A &\rightarrow \epsilon; \\ A &\rightarrow a; \\ A &\rightarrow B; \\ A &\rightarrow B + C; \\ A &\rightarrow B \star C; \\ A &\rightarrow \text{SEQ}(B); \\ A &\rightarrow \text{CYC}(B) \text{ of} \\ A &\rightarrow \text{SET}(B), \end{aligned}$$

waarbij a opnieuw een terminal is en A, B en C nonterminals zijn. In de laatste drie producties kan opnieuw de beperkende variant gebruikt worden.

Merk op dat de constructors in de bovenstaande producties allemaal ariteit 1 of 2 hebben, omdat we geen constructors met een ariteit groter dan 2 hebben ingevoerd. Daarom staan de specificaties uit Ω_o en Ω_g automatisch in CNF. Het enige waar we dus nog voor moeten zorgen opdat de specificaties in SPNF staan, is dat er voor elke nonterminal uit de specificatie juist één productie gedefinieerd is.

Om dit te verwezenlijken, kunnen we gebruikmaken van de compacte vorm van specificaties, die beschreven werd in Sectie 2.5.4. Hiermee kunnen we de unie van alle producties van eenzelfde nonterminal nemen. Deze unie beschouwen we vervolgens als de combinatorische unie. Hierdoor verkrijgen we een langere productie, die mogelijk meerdere constructors bevat. In dat geval splitsen we de productie op in deelproducties door nieuwe nonterminals, die we nog niet gebruikt hebben, in te voeren voor deze deelproducties.

We spreken af dat we de splitsing uitvoeren door eerst de deelproducties met een terminal af te zonderen in nieuwe producties. Vervolgens isoleren we de deelproducties met een (beperkende) constructor van ariteit 1. Daarna zonderen we de deelproducties met een cartesisch of gelabeld product af.

Op die manier verkrijgen we één lange productie, waarin enkel nog gebruikgemaakt wordt van de combinatorische unie als constructor. Indien er in deze productie meer dan één combinatorische unie gebruikt wordt, moet ze nogmaals opgesplitst worden. Dit doen we door telkens een deelproductie die bestaat uit één unie van twee klassen af te zonderen in een nieuwe productie en hiervoor een nieuwe nonterminal in te voeren. Deze stap blijven we herhalen totdat er geen enkele productie meer overschiet die meer dan één constructor gebruikt.

Voorbeeld 3.36 *Beschouw een specificatie S uit Ω_o die o.a. de volgende zes producties bevat:*

$$\begin{aligned} U &\rightarrow \epsilon \\ U &\rightarrow A \times B \\ U &\rightarrow C + D \\ U &\rightarrow \mathrm{CYC}(E) \\ U &\rightarrow f \\ U &\rightarrow \mathrm{SEQ}(G) \end{aligned}$$

Door deze producties in de compacte vorm te zetten, krijgen we de productie

$$U \rightarrow \epsilon + A \times B + C + D + \mathrm{CYC}(E) + f + \mathrm{SEQ}(G).$$

Om te beginnen splitsen we deze productie op door eerst de deelproducties ϵ en f in een nieuwe productie te zetten. Hiervoor voeren we de nieuwe nonterminals H en I in:

$$\begin{aligned} U &\rightarrow H + A \times B + C + D + \mathrm{CYC}(E) + I + \mathrm{SEQ}(G) \\ H &\rightarrow \epsilon \\ I &\rightarrow f \end{aligned}$$

We doen hetzelfde voor de deelproducties $\mathrm{CYC}(E)$ en $\mathrm{SEQ}(G)$ door de nieuwe nonterminals J en K in te voeren:

$$\begin{aligned} U &\rightarrow H + A \times B + C + D + J + I + K \\ H &\rightarrow \epsilon \\ I &\rightarrow f \\ J &\rightarrow \mathrm{CYC}(E) \\ K &\rightarrow \mathrm{SEQ}(G) \end{aligned}$$

Vervolgens splitsen we de eerste productie op door de deelproductie $A \times B$ af te zonderen. Hiervoor voeren we de nieuwe nonterminal L in.

$$\begin{aligned} U &\rightarrow H + L + C + D + J + I + K \\ H &\rightarrow \epsilon \\ I &\rightarrow f \\ J &\rightarrow \mathrm{CYC}(E) \\ K &\rightarrow \mathrm{SEQ}(G) \\ L &\rightarrow A \times B \end{aligned}$$

In de eerste productie wordt nu enkel nog gebruikgemaakt van de combinatorische unie als constructor. Deze productie splitsen we op door de deelproducties $H + L$, $C + D$ en $J + I$ af te zonderen in producties voor de nieuwe nonterminals M, N en O.

$$\begin{aligned} U &\rightarrow M + N + O + K \\ H &\rightarrow \epsilon \\ I &\rightarrow f \\ J &\rightarrow \mathrm{CYC}(E) \\ K &\rightarrow \mathrm{SEQ}(G) \\ L &\rightarrow A \times B \\ M &\rightarrow H + L \\ N &\rightarrow C + D \\ O &\rightarrow J + I \end{aligned}$$

Ten slotte splitsen we de eerste productie nogmaals op door de deelproducties $M+N$ en $O + K$ af te zonderen. Hiervoor voeren we de nonterminals P en Q in. Zo hebben we de producties voor U omgevormd tot producties die voldoen aan de eisen van SPNF.

$$\begin{aligned} U &\rightarrow P + Q \\ H &\rightarrow \epsilon \\ I &\rightarrow f \\ J &\rightarrow \mathrm{CYC}(E) \\ K &\rightarrow \mathrm{SEQ}(G) \\ L &\rightarrow A \times B \\ M &\rightarrow H + L \end{aligned}$$

$$\begin{aligned} N &\to C + D \\ O &\to J + I \\ P &\to M + N \\ Q &\to O + K \end{aligned}$$ ◁

3.3.7 Well-defined specificaties

In deze sectie wordt de belangrijke notie van *well-defined* specificaties beschreven. Dit zijn namelijk de specificaties waartoe we ons beperken bij het beschrijven van combinatorische klassen, omdat specificaties die niet well-defined zijn geen combinatorische klasse kunnen beschrijven. We veronderstellen in deze sectie dat de beschouwde specificaties in SPNF staan. Voorts beschouwen we in deze sectie zowel ongelabelde als gelabelde specificaties.

Definitie 3.37 De *valuatie* van een nonterminal U van een specificatie $(T, N, S, P, |\cdot|)$, genoteerd met $\text{val}(U)$, is gedefinieerd als de grootte van het kleinste object dat afgeleid kan worden uit U. Het is mogelijk dat $\text{val}(U)$ oneindig is. ⧫

We kunnen ook de valuatie van een terminal van een grammatica beschouwen. Deze stellen we dan gelijk aan de grootte van deze terminal.

Definitie 3.38 Een specificatie $(T, N, S, P, |\cdot|)$ is *well-defined*[3] als en slechts als ze voldoet aan de volgende eigenschappen:

1. elke nonterminal $U \in N$ heeft een eindige valuatie;
2. voor elke nonterminal $U \in N$ en elk natuurlijk getal $n \in \mathbb{N}$ geldt dat de deelverzameling $\mathcal{U}_n$ van objecten van grootte n die afgeleid worden uit U eindig is. ⧫

Een specificatie die niet well-defined is, is in de praktijk niet nuttig. Als een nonterminal geen eindig object oplevert, kunnen we niets afleiden uit deze nonterminal. De afleiding zou dan oneindig lang doorgaan en dat is uiteraard niet gewenst.

Als anderzijds een bepaalde nonterminal oneindig veel objecten van een bepaalde grootte kan afleiden, komen we in de problemen bij het tellen van objecten. Stel

[3] In de PhD-thesis van Paul Zimmermann [20, Sectie 1.4] worden well-defined specificaties "des spécifications *bien fondées*" genoemd.

dat we verder een berekening willen uitvoeren met het aantal objecten van een bepaalde grootte. Als dit aantal gelijk is aan oneindig, heeft het geen nut om de berekening uit te voeren, omdat oneindigheid in de praktijk niet gebruikt kan worden.

Voorbeeld 3.39 *Beschouw de volgende ongelabelde specificatie S:*

$$S \begin{cases} A \to \mathrm{SEQ}(B) \\ B \to \mathrm{SEQ}(Z) \\ Z \to \{a\} \end{cases}$$

Herinner dat we de verzameling van alle objecten die afgeleid worden uit een nonterminal U aanduiden met dezelfde naam in kalligrafisch schrift.

Wegens de definitie van de sequence-operator bevat $\mathcal{B}$ het neutrale object ϵ van grootte 0. Het gevolg hiervan is dat $\mathcal{A}$ elke sequentie van de vorm $\epsilon^k = (\epsilon, \epsilon, \ldots, \epsilon)$ (met $k \geq 0$) bevat. Elke ϵ^k is een object van grootte 0, waardoor $A_0 = \infty$. Bijgevolg is de specificatie S niet well-defined. ◁

Definitie 3.40 De verzameling Ω_o^W bestaat uit alle ongelabelde specificaties uit Ω_o die well-defined zijn. ⧫

Definitie 3.41 De verzameling Ω_g^W bestaat uit alle gelabelde specificaties uit Ω_g die well-defined zijn. ⧫

Voor de verzamelingen Ω_o^W en Ω_g^W gelden respectievelijk de volgende belangrijke eigenschapen:

Stelling 3.42 *Alle specificaties uit Ω_o^W beschrijven een ongelabelde combinatorische klasse.*

Stelling 3.43 *Alle specificaties uit Ω_g^W beschrijven een gelabelde combinatorische klasse.*

Bewijs: Het bewijs bestaat uit twee delen. Voor elk van de twee condities die voorkomen in de definitie van een combinatorische klasse (zie Definitie 3.1) moeten we bewijzen dat deze conditie geldt wanneer de klasse beschreven wordt d.m.v. een well-defined specificatie.

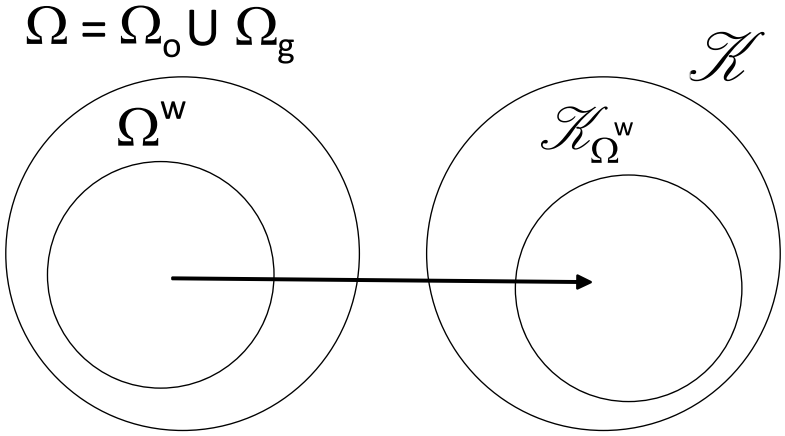
(i) Omdat de valuatie van elke nonterminal van een well-defined specificatie per definitie eindig is (eerste eigenschap van Definitie 3.38), kan er uit elk nonterminal

U minstens één *minimaal* object afgeleid worden. De nonterminals die afhankelijk zijn van U moeten op een gegeven moment één of meerdere minimale objecten gebruiken om hun objecten te creëeren. Als dit niet het geval zou zijn, zouden deze nonterminals enkel oneindige objecten kunnen creëeren en dat is uitgesloten omdat de valuatie van elke nonterminal eindig is.

Alle objecten worden dus gecreëerd door een constructor die een eindig aantal minimale objecten gebruikt. Omdat voor elke ingevoerde constructor de grootte-functie additief is, hebben alle gecreëerde objecten een eindige grootte. Bijgevolg geldt de eerste conditie van Definitie 3.1.

(ii) Wegens de tweede eigenschap van Definitie 3.38 is voor elke nonterminal van een well-defined specificatie de deelverzameling met objecten van grootte n (met $n \in \mathbb{N}$) die afgeleid worden uit dit nonterminal eindig. In het bijzonder geldt dit ook voor het startsymbool. Elke deelverzameling met objecten van grootte n (met $n \in \mathbb{N}$) van de taal die aanvaard wordt door een well-defined specificatie is dus eindig. Deze tweede eigenschap komt m.a.w. rechtstreeks overeen met de tweede conditie van Definitie 3.1. □

Stelling 3.42 en 3.43 wordt schematisch weergegeven in Figuur 3.6. Hierin is $\mathcal{K}$ de verzameling van alle combinatorische klassen en $\mathcal{K}_{\Omega^W}$ de deelverzameling van alle combinatorische klassen die kunnen beschreven worden d.m.v. een well-defined specifatie.



Figuur 3.6: *Het verband tussen de verzamelingen Ω_o^W en Ω_g^W en de verzameling van de combinatorische klassen $\mathcal{K}$.*

In hetgeen volgt beschouwen we enkel well-defined specificaties, omdat zij altijd een combinatorische klasse beschrijven. We beschouwen m.a.w. enkel de constructible combinatorische klassen die beschrijfbaar zijn d.m.v. een well-defined specificatie. Daarom volgt nu een belangrijke stelling:

Stelling 3.45 *Het is beslisbaar of een specificatie well-defined is of niet.*

In de rest van deze sectie geven we een bewijs van deze stelling. Het bewijs steunt op twee lemma's die elk één van de twee condities uit de definitie van een well-defined specificatie voor hun rekening nemen.

Lemma 3.46 *De valuatie van elke nonterminal van een specificatie is berekenbaar [21, p. 16].*

Bewijs: Algoritme 1 berekent de valuatie van elk symbool (atomen en nonterminals) van een *ongelabelde* specificatie S. Dit gebeurt door een ééndimensionale dynamisch veranderende tabel v bij te houden. Het aantal rijen van v is gelijk aan de som van het aantal atomen en het aantal nonterminals van S. De uiteindelijke waarden na afloop van het algoritme komen overeen met de valuatie van elk symbool van S.

Waarom het algoritme stopt en correct is, staat beschreven in [21, p. 17]. In essentie komt het erop neer dat de waarden van v in elke iteratie afnemen. Na k iteraties bevat v de grootte van de kleinste objecten die in maximum k iteraties kunnen afgeleid worden uit de symbolen van S.

Bij een productie van de vorm $U \to R_1 + R_2$ kan U alle objecten afleiden die ofwel uit R_1, ofwel uit R_2 afgeleid worden. Het kleinste object dat U kan afleiden is dus een object dat R_1 of R_2 kan afleiden. Bijgevolg is de valuatie die bij een combinatorische unie hoort gelijk aan het minimum van de valuaties van de gebruikte nonterminals.

In het geval van het cartesisch product $U \to R_1 \times R_2$ kan het kleinste object dat uit U kan worden afgeleid, bekomen worden door de minimale objecten te combineren die respectievelijk uit R_1 en R_2 afgeleid worden. Door de additiviteitseigenschap is de valuatie die bij het cartesisch product hoort gelijk aan de som van de valuaties van de gebruikte nonterminals.

Algorithm 1 Een algoritme dat de valuatie van elk symbool van een specificatie S berekent.

Algoritme Valuatie:
Invoer: een specificatie S die in SPNF staat

{Begin initialisatie}
for each atoom a van S **do**
  $\mathrm{v}[a] = |a|$
for each nonterminal U van S **do**
  $\mathrm{v}[U] = \infty$
{Einde initialisatie}

repeat
  for each productie voor een nonterminal U in S **do**
    if van de vorm $U \to a$ waarbij a een atoom is **then**
      $\mathrm{v}[U] = \mathrm{v}[a]$
    if van de vorm $U \to V$ waarbij V een nonterminal is **then**
      $\mathrm{v}[U] = \mathrm{v}[V]$
    if van de vorm $U \to R_1 + R_2$ **then**
      $\mathrm{v}[U] = \min(\mathrm{v}[R_1], \mathrm{v}[R_2])$
    if van de vorm $U \to R_1 \times R_2$ **then**
      $\mathrm{v}[U] = \mathrm{v}[R_1] + \mathrm{v}[R_2]$
    if van de vorm $U \to \Phi(R)$ waarbij $\Phi \in \{\mathrm{SEQ}, \mathrm{PSET}, \mathrm{MSET}\}$ **then**
      $\mathrm{v}[U] = 0$
    if van de vorm $U \to \mathrm{CYC}(R)$ **then**
      $\mathrm{v}[U] = \mathrm{v}[R]$
until v is onveranderd

In het geval van de sequence- en de multiset-constructor is het kleinste object dat kan afgeleid worden per definitie een neutraal object ϵ. In het geval van de powerset-constructor is dat de lege verzameling $\emptyset$ die we als neutraal object beschouwen. De valuatie die bij deze drie constructors hoort is bijgevolg gelijk aan 0.

De cycle-constructor kan geen neutraal object creëeren. Een lege cycle bestaat immers niet. De cycle-constructor $U \to \mathrm{CYC}(R)$ bouwt de objecten op m.b.v. de sequence-constructor. Daarom hebben we bij de definitie van de cycle-constructor telkens het lege object ϵ, dat per definitie in het resultaat van de sequence-constructor zit, genegeerd. Het kleinste object dat op die manier kan bekomen worden is een tupel dat één component bevat, namelijk het kleinste object uit R. Bijgevolg is de valuatie die bij de cycle-constructor hoort gelijk aan de valuatie van de gebruikte nonterminal.

Het algoritme voor de valuatie van een *gelabelde* specificatie verschilt slechts in twee onderdelen van Algoritme 1. Enerzijds moet het cartesisch product vervangen worden door het gelabelde product in de vierde if-test. De valuatie ervan blijft echter nog steeds de som van de valuaties van de gebruikte nonterminals.

Anderzijds moeten in de voorlaatste if-test de constructors PSET en MSET vervangen worden door SET, omdat deze twee constructors één equivalent hebben in het gelabelde universum. De valuatie ervan blijft nog steeds 0. □

Voorbeeld 3.48 *Beschouw de volgende ongelabelde specificatie S:*

$$S \begin{cases} A \to B \times A \\ B \to b \end{cases}$$

De valuatie van de nonterminal A is gelijk aan ∞. Intuïtief is het duidelijk dat A geen object met een eindige grootte kan afleiden, omdat A oneindig lang vervangen kan worden door zijn productie $B \times A$. Er is m.a.w. geen "stopconditie" om een eindig object uit A te verkrijgen. Bijgevolg is S dan ook geen well-defined specificatie. ◁

Hiermee kunnen we de eerste conditie uit de definitie van een well-defined specificatie beslissen. De volgende stap bestaat uit het elimineren van specificaties met een nonterminal U, waarvoor geldt dat $U_n = \infty$. Hiervoor voeren we nu een belangrijke soort specificatie in.

Definitie 3.49 Een *gereduceerde ongelabelde* specificatie is een specificatie $(T, N, S, P, |\cdot|)$ waarin:

1. de valuaties $\text{val}(U)$ van alle nonterminals $U \in N$ eindig zijn;
2. geen enkele productie voorkomt van de vorm $U \to \Phi(V)$ met $\Phi \in \{\text{SEQ}, \text{CYC}, \text{MSET}, \text{PSET}\}$ én $\text{val}(V) = 0$. ⧫

In het gelabelde universum kunnen we een analoge definitie beschouwen.

Definitie 3.50 Een *gereduceerde gelabelde* specificatie is een specificatie $(T, N, S, P, |\cdot|)$ waarin:

1. de valuaties $\text{val}(U)$ van alle nonterminals $U \in N$ eindig zijn;
2. geen enkele productie voorkomt van de vorm $U \to \Phi(V)$ met $\Phi \in \{\text{SEQ}, \text{CYC}, \text{SET}\}$ én $\text{val}(V) = 0$. ⧫

M.b.v. Algoritme 1 is het eenvoudig na te gaan of een specificatie gereduceerd is. Indien een specificatie niet gereduceerd is, kan ze niet well-defined zijn o.w.v. twee redenen. Ten eerste, als de valuatie van een nonterminal van een specificatie niet eindig is, voldoet ze sowieso niet aan de definitie van well-definedness.

Ten tweede, in het geval dat een ongelabelde specificatie een productie van de vorm $U \to \text{SEQ}(V)$, $U \to \text{MSET}(V)$ of $U \to \text{CYC}(V)$, waarvoor geldt dat de valuatie van de nonterminal V gelijk is aan 0, krijgen we een situatie zoals in Voorbeeld 3.39. Er worden namelijk oneindig veel sequenties van de vorm $(\epsilon, \ldots, \epsilon)$ gecreëerd o.w.v. het gebruik van de sequence-constructie. Al deze sequenties hebben grootte 0, waardoor er oneindig veel objecten van grootte 0 in de resulterende verzameling zitten. Bijgevolg voldoet een dergelijke specificatie niet aan de definitie van well-definedness.

Hetzelfde geldt voor een gelabelde specificatie die een productie van de vorm $U \to \text{SEQ}(V)$, $U \to \text{SET}(V)$ of $U \to \text{CYC}(V)$ bevat.

Voor een ongelabelde specificatie die een productie van de vorm $U \to \text{PSET}(V)$ bevat, waarvoor geldt dat $\text{val}(V) = 0$ geldt een analoge redenering. Deze constructor creëert in dit geval namelijk oneindig veel verzamelingen van de vorm $\{\epsilon, \ldots, \epsilon\}$, die allemaal gelijk zijn aan de verzameling $\{\epsilon\}$ en bijgevolg grootte 0 hebben. Hierdoor bevat de resulterende verzameling opnieuw oneindig veel objecten van grootte 0, waardoor de specificatie niet voldoet aan de definitie van well-definedness.

Dit is dan ook de reden waarom we bij de definitie van alle voorgenoemde constructors hebben aangenomen dat de gebruikte klasse geen enkel object van grootte 0 bevat.

Lemma 3.51 *Voor een bepaald natuurlijk getal $n \in \mathbb{N}$ en een nonterminal U van een gereduceerde ongelabelde specificatie S, geldt dat $U_n = \infty$ als en slechts als er een cyclus $(U^{(0)}, U^{(1)}, \ldots, U^{(k)} = U^{(0)})$ van nonterminals bestaat, zodat de nonterminal $U^{(j)}$ voorkomt in de productie die $U^{(j-1)}$ definieert (voor $1 \leq j \leq k$), én voor* alle *productgebaseerde producties, $U^{(j-1)} \to U^{(j)} \times V$ of $U^{(j-1)} \to V \times U^{(j)}$ hiervan, geldt dat* $\text{val}(V) = 0$ *[21, p. 17].*

Bewijs: Het bewijs van dit Lemma staat beschreven in [20, p. 32 - 33]. Hierin wordt bewezen dat het controleren van circulariteit van een specificatie (i.e. onderzoeken of $U_n = \infty$ voor een bepaalde n) gereduceerd kan worden tot cyclusdetectie in de overeenkomstige afhankelijkheidsgraaf. □

In het gelabelde universum kunnen we een analoog Lemma beschouwen, waarbij het cartesisch product vervangen wordt door het gelabelde product.

Indien een specificatie gereduceerd is, kan ze toch nog oneindig veel objecten van een bepaalde grootte objecten afleiden. Dit kan alleen als de specificatie aan een bepaalde voorwaarde voldoet. Ze moet enerzijds recursief zijn, dit wil zeggen dat er een cyclus van nonterminals bestaat, waarbij elke nonterminal afhangt van de volgende nonterminal in deze cyclus ($U^{(j-1)}$ wordt telkens gedefinieerd a.d.h.v. $U^{(j)}$). Als anderzijds voor elke productgebaseerde productie in deze cyclus, $U^{(j-1)} \to U^{(j)} \times V$ of $U^{(j-1)} \to V \times U^{(j)}$, geldt dat de valuatie van de nonterminal dat niet in de cyclus zit (V in dit geval), gelijk is aan 0, kunnen we besluiten dat de gereduceerde specificatie in kwestie niet well-defined is.

De tweede conditie kan als volgt beschouwd worden. Als we productgebaseerde producties in een cyclus blijven uitvoeren en het nonterminal dat niet in deze cyclus zit, blijft neutrale objecten aan deze producten toevoegen, dan worden er oneindig veel objecten van een bepaalde grootte n gecreëerd. Sommige objecten die uit de

productgebaseerde producties komen, "groeien" als het ware niet omdat er alleen maar objecten van grootte 0 aan toegevoegd worden. Echter zijn de objecten in elke stap wel verschillend van elkaar door het toevoegen van nieuwe componenten. Doordat dit zich oneindig herhaalt, verkrijgen we oneindig veel objecten van een bepaalde grootte.

Ter verduidelijking volgt nu een voorbeeld van de afleiding van objecten uit een ongelabelde gereduceerde specificatie, waarin de conditie uit Lemma 3.51 van toepassing is. Deze afleiding toont aan waarom er oneindig veel objecten van een bepaalde grootte afgeleid kunnen worden uit de specificatie.

Voorbeeld 3.53 *Beschouw de volgende ongelabelde specificatie S:*

$$S \begin{cases} A & \to & B \times C \\ B & \to & D \times C \\ C & \to & \mathrm{SEQ}(F) \\ D & \to & E + F \\ E & \to & \mathrm{CYC}(A) \\ F & \to & \{f\} \end{cases}$$

Hierin is de grootte van het atoom f gelijk aan 1. De valuaties van alle nonterminals van S worden weergegeven in Tabel 3.1. De afhankelijkheidsgraaf van S wordt afgebeeld in Figuur 3.7.

De specificatie S is gereduceerd want enerzijds zijn de valuaties van alle nonterminals eindig. Dit zien we in Tabel 3.1. Anderzijds zijn er twee producties van de vorm $U \to \Phi(V)$, waarvoor geldt dat $\mathrm{val}(V) \neq 0$. Deze producties zijn $C \to \mathrm{SEQ}(F)$ en $E \to \mathrm{CYC}(A)$. In Tabel 3.1 zien we dat de valuatie van A en F niet gelijk zijn aan 0.

De specificatie S is ook recursief omdat er een cyclus voorkomt in de afhankelijkheidsgraaf van S, namelijk (A, B, D, E, A). De conditie uit Lemma 3.51 is van toepassing voor deze specificatie, want:

- *Elke nonterminal uit de bovengenoemde cyclus komt voor in de definitie van zijn voorganger. A hangt namelijk af van B, B van D, D van E en E van A.*
- *Voor elke productgebaseerde productie voor een nonterminal uit de bovenstaande cyclus, geldt dat de valuatie van de nonterminal die niet in de cyclus zit, gelijk is aan 0. De twee productgebaseerde producties $A \to B \times C$ en $B \to D \times C$ zijn namelijk gedefinieerd voor een nonterminal uit de cyclus.*

Verder is de valuatie van het nonterminal C, dat niet voorkomt in deze cyclus, gelijk aan 0.

Nu rest ons enkel nog uit te leggen waarom S oneindig veel objecten van een bepaalde grootte n creëert. Om te beginnen zien we dat D het object f met grootte 1 afleidt door de combinatorische unie. De productie van de nonterminal B koppelt dit object f o.a. met een neutraal object ϵ, waardoor we het object (f, ϵ) verkrijgen. Dit object is in essentie niet gelijk aan het object f, maar de groottes ervan komen wel overeen.

Vervolgens zal de productie van de nonterminal A het object (f, ϵ) koppelen met o.a. een ander neutraal object ϵ'. Hierdoor verkrijgen we het object $((f, \epsilon), \epsilon')$ dat opnieuw grootte 1 heeft. De productie van de nonterminal E zal daarna dit object gebruiken om een cycle met één component te maken. Deze cycle kunnen we voorstellen door $((f, \epsilon), \epsilon')$ (merk op dat dit een representant van een equivalentieklasse is). De cycle heeft echter opnieuw grootte 1.

Daarna zal de productie van de nonterminal D dit object ook afleiden door de combinatorische unie. De productie van de nonterminal B koppelt dit object opnieuw met o.a. het neutrale object ϵ, waardoor we het object $(((f, \epsilon), \epsilon'), \epsilon)$ verkrijgen, dat ook grootte 1 heeft.

De productie van de nonterminal A koppelt dit object vervolgens opnieuw met o.a. het neutrale object ϵ'. Hierdoor bekomen we het object $((((f, \epsilon), \epsilon'), \epsilon), \epsilon')$, dat opnieuw grootte 1 heeft.

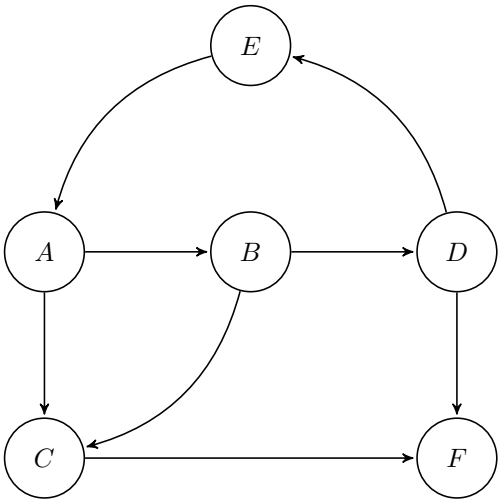
Als we deze afleiding tot in het oneindige zouden doorzetten, zien we dat de productgebaseerde producties in dit geval oneindig veel keer een neutraal object koppelen met een eerder bekomen object. Hierdoor wijzigt er echter niets aan de grootte van dit object. De specificatie S creëert m.a.w. oneindig veel objecten van grootte 1. Bijgevolg is S niet well-defined.

We kunnen besluiten dat indien een specificatie gereduceerd is en voldoet aan de conditie uit Lemma 3.51, dat deze specificatie niet well-defined is.

Omgekeerd, als er in een cyclus minstens één product voorkomt waarvan de valuaties van beide symbolen niet gelijk zijn aan 0, dan kunnen er niet oneindig veel objecten van een bepaalde grootte gegenereerd worden. In dat geval zijn de groottes van de gegenereerde objecten in elke cyclus namelijk verschillend van de groottes van de objecten van de vorige cyclus, omdat de objecten in elke cyclus gegroeid zijn wanneer ze terug bij het begin aankomen. Een dergelijk product vermijdt als het ware het doorgeven van objecten die even groot zijn. ◁

U	val(U)
A	1
B	1
C	0
D	1
E	1
F	1

Tabel 3.1: *De valuaties van de nonterminals uit Voorbeeld 3.53*



Figuur 3.7: *De afhankelijkheidsgraaf van de specificatie S uit Voorbeeld 3.53.*

Nu kunnen we het bewijs van Stelling 3.45 voltooien en daadwerkelijk een algoritme [21, p. 17 - 18] opstellen om de well-definedness van een specificatie te beslissen. Dit algoritme wordt weergegeven in Algoritme 2.
Als de drie testen van Algoritme 2 slagen, dan kunnen we concluderen dat de specificatie S well-defined is.

Algorithm 2 Een algoritme dat bepaalt of een gegeven specificatie well-defined is of niet.

Algoritme Well-definedness:
Invoer: een specificatie S

1. [**valuatie**] Pas het algoritme *Valuatie* op S toe om de valuaties van alle nonterminals van S te berekenen. Als er minstens één valuatie gelijk is aan ∞, dan is S niet well-defined.

2. [**reductie**] Controleer of S gereduceerd is. Als dit niet het geval is, dan is S niet well-defined.

3. [**circulariteit**] Voer de cyclusdetectie-test uit Lemma 3.51 uit op de gereduceerde specificatie S om na te gaan of er oneindig veel objecten van dezelfde grootte gegenereerd kunnen worden. Als dit het geval is, dan is S niet well-defined.

3.3.8 Standaardvorm

In deze sectie wordt een algemene standaardvorm [22, p. 7 - 9] van well-defined gelabelde specificaties beschreven. M.b.v. deze standaardvorm is het eenvoudiger om gelabelde combinatorische objecten te tellen en random te genereren. Deze standaardvorm is een veralgemening van de Chomsky-normaalvorm, waarbij elke productie van de specificatie gebruikmaakt van constructors met ariteit 1 of 2. Elke well-defined specificatie uit Ω_g kan gereduceerd worden naar deze standaardvorm.

In deze sectie beschouwen we enkel gelabelde specificaties. Verder veronderstellen we dat de gebruikte specificaties in SPNF staan. Om het onderscheid tussen SPNF en de standaardvorm goed weer te geven, duiden we de standaardvorm in wat volgt aan met *STDF*.

De standaardvorm die we hier invoeren, steunt op een speciale constructor die van groot belang is in het gelabelde universum, namelijk de zogenaamde *pointing operator* (of *pointing constructor*). Daarom geven we eerst de definitie van deze nieuwe constructor. Vervolgens beschrijven we hoe de specificaties uit Ω_g kunnen gereduceerd worden naar specificaties die de pointing operator gebruiken.

Pointing operator

Definitie 3.54 Zij $\mathcal{A}$ een gelabelde klasse. De *pointing* van $\mathcal{A}$, genoteerd als $\Theta\mathcal{A}$, wordt geconstrueerd als volgt:

$$\Theta\mathcal{A} \;=\; \bigcup_{n=1}^{\infty} (\mathcal{A}_n \star [1 \ldots n]) \qquad \blacklozenge$$

Herinner dat de objecten van $\mathcal{A}$ per definitie goed-gelabeld zijn. De pointing operator is een verzamelingconstructor met ariteit 1 die op basis van één gelabelde klasse $\mathcal{A}$ een nieuwe klasse (de *gepointe klasse* genoemd) construeert, waarin de objecten van $\mathcal{A}$ *gepoint* worden. Dit gebeurt door elk atoom van elk object uit $\mathcal{A}$ eenmaal te accentueren. Er wordt namelijk telkens een koppel gecreëerd waarvan de eerste component een object uit $\mathcal{A}$ is. De tweede component is een neutraal object, dat een label van een atoom van dit object voorstelt.

Merk op dat de objecten van grootte 0 uit $\mathcal{A}$ niet gepoint worden. Dit is namelijk onmogelijk omdat ze geen enkel atoom (en dus ook geen enkel label) bevatten. De objecten van grootte 0 worden m.a.w. genegeerd tijdens het creëeren van de klasse $\Theta\mathcal{A}$. De klasse $\Theta\mathcal{A}$ bevat dus ook geen objecten van grootte 0.

Voorbeeld 3.55 *Beschouw de combinatorische klasse* $\mathcal{A} = \{(\textcircled{1}_a, \textcircled{2}_b), (\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), (\textcircled{1}_b)\}$. *De pointing van* $\mathcal{A}$ *is gelijk aan:*

$$\begin{aligned}\Theta\mathcal{A} \;=\; & \{((\textcircled{1}_a, \textcircled{2}_b), 1), ((\textcircled{1}_a, \textcircled{2}_b), 2),\\ & ((\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), 1), ((\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), 2), ((\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), 3),\\ & ((\textcircled{1}_b), 1)\}.\end{aligned}$$ ◁

We kunnen ook de omgekeerde weg, depointen, beschouwen.

Definitie 3.56 Zij $\Theta\mathcal{A}$ een gepointe klasse. De *gedepointe* klasse $\mathcal{A}$ wordt bekomen door bij elk object uit $\Theta\mathcal{A}$ het neutrale object dat de pointing aangeeft, weg te laten. ♦

Merk op dat meerdere gepointe objecten gedepoint kunnen worden naar hetzelfde object. Omdat we echter verzamelingen van objecten beschouwen, maakt dit niet uit doordat we dubbels negeren. Zo is bijvoorbeeld het gedepointe object van de objecten $((\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), 1)$ en $((\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b), 2)$ telkens gelijk aan $(\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_b)$. Dit object wordt echter maar één keer beschouwd in de gedepointe klasse.

Reductie naar standaardvorm (STDF)

De bedoeling van de reductie naar de standaardvorm is om specificaties te verkrijgen, waarin enkel gebruikgemaakt wordt van de combinatorische unie, het cartesisch product en de pointing operator als constructors. Als we ons beperken tot deze drie constructors, wordt het tellen en random genereren van combinatorische objecten namelijk veel eenvoudiger. In deze sectie beschrijven we daarom hoe de producties die de gelabelde SEQ-, CYC- of SET-constructor gebruiken, kunnen omgezet worden naar producties die enkel deze drie constructors gebruiken. De andere producties staan al in de juiste vorm en blijven daarom ongewijzigd.

De producties van een specificatie in de standaardvorm mogen enkel één van de volgende vormen hebben:

$$
\begin{aligned}
A &\rightarrow \epsilon; \\
A &\rightarrow a;
\end{aligned}
$$

$$
\begin{aligned}
A &\rightarrow B + C; \\
A &\rightarrow \Theta B + C; \\
A &\rightarrow B + \Theta C; \\
A &\rightarrow \Theta B + \Theta C;
\end{aligned}
$$

$$
\begin{aligned}
A &\rightarrow B \star C; \\
A &\rightarrow \Theta B \star C; \\
A &\rightarrow B \star \Theta C; \\
A &\rightarrow \Theta B \star \Theta C;
\end{aligned}
$$

$$
\begin{aligned}
\Theta A &\rightarrow B \star C; \\
\Theta A &\rightarrow \Theta B \star C; \\
\Theta A &\rightarrow B \star \Theta C \text{ of} \\
\Theta A &\rightarrow \Theta B \star \Theta C
\end{aligned}
$$

waarbij a een terminal is, ϵ een neutraal object is en A, B en C nonterminals zijn. Merk op dat sommige van deze producties twee constructors bevatten en dus volgens Definitie 2.53 moeten opgesplitst worden. De mogelijke vormen van een productie uit een specificatie die in STDF staat, zijn dus:

$$\begin{aligned} A &\rightarrow \epsilon; \\ A &\rightarrow a; \\ A &\rightarrow B + C; \\ A &\rightarrow B \star C; \\ A &\rightarrow \Theta B; \\ \Theta A &\rightarrow B \end{aligned}$$

In deze sectie splitsen we de producties echter nog niet op om de leesbaarheid te verhogen.

Hieronder beschrijven we hoe de producties $A \rightarrow \mathrm{SEQ}(B)$, $A \rightarrow \mathrm{SET}(B)$ en $A \rightarrow \mathrm{CYC}(B)$ kunnen omgevormd worden zodat ze in één van de bovenstaande vormen staan.

Voor de gelabelde sequence-constructor geldt de volgende implicatie:

$$B \rightarrow \mathrm{SEQ}(A) \quad \Rightarrow \quad B \rightarrow \epsilon + A \star B \tag{3.6}$$

Maar aangezien deze laatste productie meer dan één constructor bevat, moeten we ze opsplitsen (na deze sectie). Zo verkrijgen we de producties

$$\begin{aligned} B &\rightarrow E + C \\ C &\rightarrow A \star B \\ E &\rightarrow \epsilon \end{aligned}$$

Waarom Implicatie (3.6) geldt, is intuïtief eenvoudig in te zien. Het recursieve deel $A \star B$ zorgt er in elke stap voor dat sequenties in de huidige voorlopige klasse $\mathcal{B}$ aan de linkerkant (eerste component van de nieuwe sequentie) uitgebreid worden met objecten uit de klasse $\mathcal{A}$. Hierdoor kunnen we m.a.w. sequenties met $k+1$ componenten bekomen uit sequenties met k componenten (k startend vanaf 0). Het linkse deel zorgt ervoor dat de klasse $\mathcal{B}$ het lege object bevat, wat per definitie in het resultaat van de SEQ-constructor zit.

Voorbeeld 3.57 *Beschouw de gelabelde klasse* $\mathcal{A} = \{\textcircled{1}_a, \textcircled{1}_b\}$. $\mathcal{B} = \text{SEQ}(\mathcal{A})$ *is dan gelijk aan:*

$$\begin{aligned}
\mathcal{B} = \text{SEQ}(\mathcal{A}) \;=\; & \{\epsilon, \\
& (\textcircled{1}_a), (\textcircled{1}_b), \\
& (\textcircled{1}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a), (\textcircled{1}_a, \textcircled{2}_b), (\textcircled{2}_a, \textcircled{1}_b), \\
& (\textcircled{1}_b, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a), (\textcircled{1}_b, \textcircled{2}_b), (\textcircled{2}_b, \textcircled{1}_b), \\
& (\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_a), (\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a, \textcircled{3}_a), \\
& (\textcircled{2}_a, \textcircled{3}_a, \textcircled{1}_a), (\textcircled{3}_a, \textcircled{1}_a, \textcircled{2}_a), (\textcircled{3}_a, \textcircled{2}_a, \textcircled{1}_a) \\
& \ldots \\
& (\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}_a), (\textcircled{1}_b, \textcircled{3}_a, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a, \textcircled{3}_a), \\
& (\textcircled{2}_b, \textcircled{3}_a, \textcircled{1}_a), (\textcircled{3}_b, \textcircled{1}_a, \textcircled{2}_a), (\textcircled{3}_b, \textcircled{2}_a, \textcircled{1}_a) \\
& \ldots\}.
\end{aligned}$$

Als we de productie $\mathcal{B} \to \epsilon + \mathcal{A} \star \mathcal{B}$ *op een iteratieve manier uitwerken, krijgen we de volgende resultaten:*

$$\begin{aligned}
\mathcal{B}^{(0)} &= \emptyset \\
\\
\mathcal{B}^{(1)} &= \epsilon + \mathcal{A} \star \mathcal{B}^{(0)} \\
&= \{\epsilon\} \\
\\
\mathcal{B}^{(2)} &= \epsilon + \mathcal{A} \star \mathcal{B}^{(1)} \\
&= \{\epsilon, (\textcircled{1}_a, \epsilon), (\textcircled{1}_b, \epsilon)\} \\
&= \{\epsilon, (\textcircled{1}_a), (\textcircled{1}_b)\} \\
\\
\mathcal{B}^{(3)} &= \epsilon + \mathcal{A} \star \mathcal{B}^{(2)} \\
&= \{\epsilon, (\textcircled{1}_a), (\textcircled{1}_b), \\
&\quad (\textcircled{1}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a), (\textcircled{1}_a, \textcircled{2}_b), (\textcircled{2}_a, \textcircled{1}_b), \\
&\quad (\textcircled{1}_b, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a), (\textcircled{1}_b, \textcircled{2}_b), (\textcircled{2}_b, \textcircled{1}_b)\} \\
\\
\mathcal{B}^{(4)} &= \epsilon + \mathcal{A} \star \mathcal{B}^{(3)} \\
&= \{\epsilon, (\textcircled{1}_a), (\textcircled{1}_b), \\
&\quad (\textcircled{1}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a), (\textcircled{1}_a, \textcircled{2}_b), (\textcircled{2}_a, \textcircled{1}_b), \\
&\quad (\textcircled{1}_b, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a), (\textcircled{1}_b, \textcircled{2}_b), (\textcircled{2}_b, \textcircled{1}_b),
\end{aligned}$$

$$(\textcircled{1}_a, \textcircled{2}_a, \textcircled{3}_a), (\textcircled{1}_a, \textcircled{3}_a, \textcircled{2}_a), (\textcircled{2}_a, \textcircled{1}_a, \textcircled{3}_a), \ldots$$
$$(\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}_a), (\textcircled{1}_b, \textcircled{3}_a, \textcircled{2}_a), (\textcircled{2}_b, \textcircled{1}_a, \textcircled{3}_a), \ldots\} \quad \triangleleft$$

Als we deze constructie tot in het oneindige zouden voortzetten, zien we dat beide constructies hetzelfde resultaat opleveren.

Voor de gelabelde set-constructor geldt de volgende implicatie:

$$B \to \mathrm{SET}(A) \quad \Rightarrow \quad \Theta B \to B \star \Theta A \tag{3.7}$$

Deze laatste productie bevat opnieuw meer dan één constructor. Daarom splitsen we ze na deze sectie op in de producties

$$\begin{aligned} \Theta B &\to D \\ D &\to B \star C \\ C &\to \Theta A \end{aligned}$$

Beschouw een gepoint object α uit de klasse $\Theta\mathcal{B} = \Theta\,\mathrm{SET}(\mathcal{A})$. Het object α is m.a.w. de representant van de equivalentieklasse, die alle permutaties van α bevat. Bovendien is α een gepoint object, wat wil zeggen dat één atoom uit één component van α geaccentueerd is. Voor de eenvoud stellen we dat α een representant is waarbij deze component achteraan staat (dit kan doordat we een equivalentieklasse van permutaties beschouwen).

Stel dat we uit de representant α de gepointe component afzonderen. Dan verkrijgen we in feite twee (sub)objecten, namelijk een kleiner tupel β en de gepointe component γ. Het is eenvoudig om in te zien dat β een object (equivalentieklasse) is uit de klasse $\mathcal{B} = \mathrm{SET}(\mathcal{A})$, omdat de reductie van elke component ervan in $\mathcal{A}$ zit én omdat de volgorde van de componenten niet belangrijk is. Verder is de component γ een object, waarvan de reductie in de klasse $\Theta\mathcal{A}$ zit.

Wanneer we de objecten $\beta \in \mathcal{B}$ en $\gamma \in \Theta\mathcal{A}$ koppelen volgens het gelabelde product $\mathcal{B} \star \Theta\mathcal{A}$, verkrijgen we dus het object $\alpha \in \Theta\mathcal{B}$. Bijgevolg geldt Implicatie (3.7).

Voorbeeld 3.58 *Beschouw de klassen $\mathcal{A}$ en* $\mathrm{SEQ}(\mathcal{A})$ *uit Voorbeeld 3.57. Deze laatste klasse bevat o.a. de objecten $(\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}_a)$ en $(\textcircled{1}_b, \textcircled{3}_a, \textcircled{2}_a)$. Deze objecten zijn permutaties van elkaar en zitten daarom samen in dezelfde equivalentieklasse in de klasse $\mathcal{B} = \mathrm{SET}(\mathcal{A})$.*

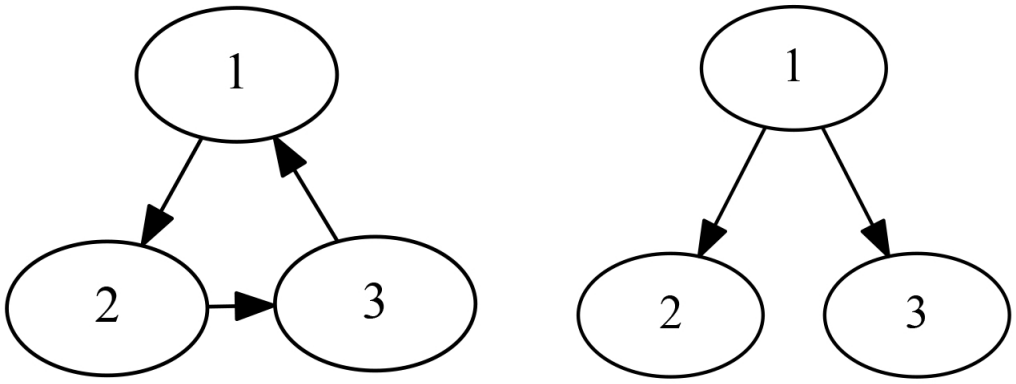
Voor dit voorbeeld beschouwen we het object $(\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}_a)$ *als representant van de equivalentieklasse* $[\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}_a]$ *die zes objecten bevat. In dit object pointen we vervolgens de laatste component* $\textcircled{3}_a$*. Het verkregen gepointe object* $\alpha = (\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}^{\bullet}_a)$ *kiezen we als representant van de equivalentieklasse* $[\textcircled{1}_b, \textcircled{2}_a, \textcircled{3}^{\bullet}_a]$ *uit* $\Theta B = \Theta\,\mathrm{SET}(\mathcal{A})$*.*

Als we uit het object α *de component* $\gamma = \textcircled{3}^{\bullet}_a$ *afzonderen, verkrijgen we het kleinere object* $\beta = (\textcircled{1}_b, \textcircled{2}_a)$*. Het object* β *kunnen we vervolgens zien als een representant van de equivalentieklasse* $[\textcircled{1}_b, \textcircled{2}_a]$ *die in* $\mathcal{B}$ *zit.*

Verder zien we dat de reductie van de gepointe component γ *gelijk is aan* $\textcircled{1}^{\bullet}_a$*. Deze reductie is op zijn beurt een object uit* $\Theta\mathcal{A}$*.*

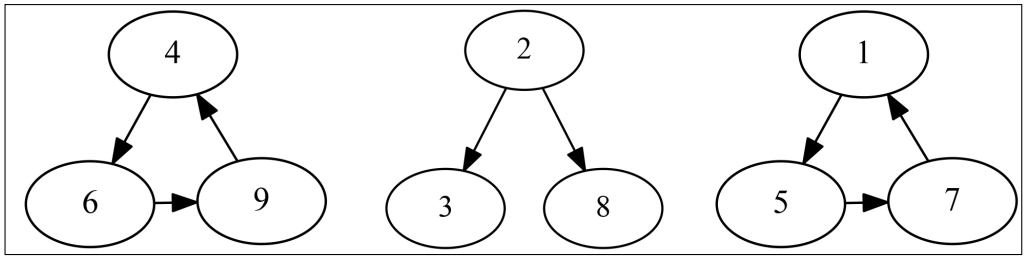
Door de twee objecten β *en* γ *samen te voegen d.m.v. het gelabelde product* $\mathcal{B} \star \Theta\mathcal{A}$ *verkrijgen we dus een object uit* $\Theta\mathcal{B}$*.* ◁

Voorbeeld 3.59 *Beschouw de klasse* $\mathcal{A}$ *die bestaat uit de twee objecten die zijn afgebeeld in Figuur 3.8. De klasse* $\mathcal{B} = \mathrm{SET}(\mathcal{A})$ *bevat o.a. een equivalentieklasse, waarvan we het object dat is afgebeeld in Figuur 3.9 beschouwen als representant.*

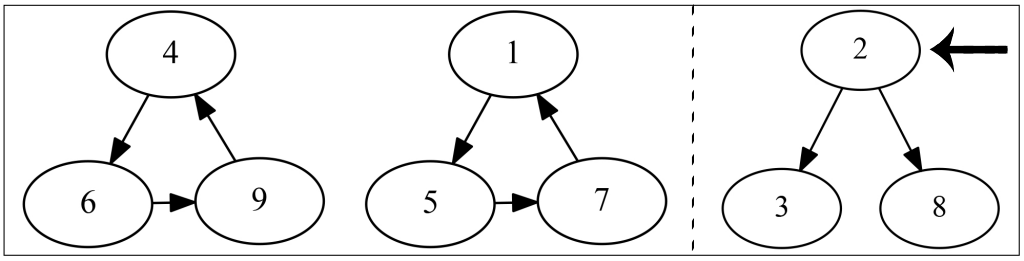


Figuur 3.8: *De twee objecten uit de klasse* $\mathcal{A}$ *uit Voorbeeld 3.59.*

Als we in dit object het eerste atoom van de tweede component pointen en vervolgens het gepointe object zó cyclisch permuteren dat de gepointe component achteraan staat, verkrijgen we het object α *uit Figuur 3.10. Het object* α *is een object uit de klasse* $\Theta\mathcal{B}$*. Uit dit object kunnen we, net zoals in het vorige voorbeeld, de gepointe component afzonderen. In Figuur 3.10 wordt dit aangegeven door de verticale stippellijn.*



Figuur 3.9: *Een object uit de klasse* $\mathcal{B} = \mathrm{SET}(\mathcal{A})$ *uit Voorbeeld 3.59.*



Figuur 3.10: *Een object uit de klasse* $\Theta\mathcal{B}$ *uit Voorbeeld 3.59.*

Het object β *dat voor de stippellijn staat is een tupel dat kan gezien worden als een representant van een equivalentieklasse uit* $\mathcal{B} = \mathrm{SET}(\mathcal{A})$*, omdat de reducties van beide componenten in* $\mathcal{A}$ *zitten én omdat de volgorde van de componenten niet van belang is. Het object* γ *dat achter de stippellijn staat is een object waarvan de reductie in* $\Theta\mathcal{A}$ *zit.*

Als we β *en* γ *combineren m.b.v. het gelabelde product* $\mathcal{B} \star \Theta\mathcal{A}$*, verkrijgen we m.a.w. een object uit de klasse* $\Theta\mathcal{B}$*.*

Voor de gelabelde cycle-constructor geldt de volgende implicatie:

$$B \to \mathrm{CYC}(A) \quad \Rightarrow \quad \Theta B \to \mathrm{SEQ}(A) \star \Theta A \tag{3.8}$$

Aangezien deze laatste productie meer dan één constructor bevat én de sequence-constructor gebruikt, moeten we ze weer opsplitsen. Zo verkrijgen we de producties

$$\begin{aligned} \Theta B &\to G \\ G &\to C \star F \\ C &\to E + D \end{aligned}$$

$$\begin{aligned} D &\rightarrow A \star C \\ E &\rightarrow \epsilon \\ F &\rightarrow \Theta A \end{aligned}$$

Hierbij zijn de derde en vierde productie de STDF-producties van $C \rightarrow \mathrm{SEQ}(A)$. Om Implicatie (3.8) in te zien, kunnen we een analoge redenering volgen als in het geval van de set-constructor.

Een gepoint object α uit $\Theta\mathcal{B} = \Theta\,\mathrm{CYC}(\mathcal{A})$ kan namelijk gezien worden als een gepointe representant van een equivalentieklasse, die alle circulaire shifts van α bevat. We stellen opnieuw dat α de representant is waarbij de gepointe component achteraan staat (dit kan doordat we een equivalentieklasse van circulaire shifts beschouwen).

We kunnen opnieuw uit α de gepointe component afzonderen om twee subobjecten over te houden. Het eerste subobject is een tupel β uit $\mathrm{SEQ}(\mathcal{A})$, omdat de reductie van elke component ervan in $\mathcal{A}$ zit én de volgorde van deze componenten belangrijk is. De reductie van het tweede subobject γ (de gepointe component) zit in de klasse $\Theta\mathcal{A}$.

Wanneer we de objecten $\beta \in \mathrm{SEQ}(\mathcal{A})$ en $\gamma \in \Theta\mathcal{A}$ koppelen volgens het gelabelde product $\mathrm{SEQ}(\mathcal{A}) \star \Theta\mathcal{A}$, verkrijgen we dus het object $\alpha \in \Theta\mathcal{B}$. Bijgevolg geldt Implicatie (3.8).

Voorbeeld 3.60 *We kunnen een analoge redenering volgen als in Voorbeeld 3.58. Beschouw namelijk opnieuw de klassen $\mathcal{A}$ en $\mathrm{SEQ}(\mathcal{A})$ uit Voorbeeld 3.57. Deze laatste klasse bevat o.a. de objecten $(①_b, ②_a, ③_a)$ en $(②_a, ③_a, ①_b)$. Omdat deze objecten circulaire shifts van elkaar zijn, zitten ze samen in dezelfde equivalentieklasse in $\mathcal{B} = \mathrm{CYC}(\mathcal{A})$.*

We kiezen het gepointe object $(①_b, ②_a, ③^\bullet_a)$ als representant van deze equivalentieklasse $[①_b, ②_a, ③^\bullet_a]$ uit $\Theta B = \Theta\,\mathrm{CYC}(\mathcal{A})$. Als we hieruit de component $③^\bullet_a$ afzonderen, krijgen we het object $(①_b, ②_a)$. Dit tupel is een object uit de klasse $\mathrm{SEQ}(\mathcal{A})$ (zie Voorbeeld 3.57). De reductie van het gepointe object $③^\bullet_a$ is gelijk aan $①^\bullet_a$ en is opnieuw een object uit $\Theta\mathcal{A}$. Door deze twee objecten samen te voegen d.m.v. het gelabelde product $\mathrm{SEQ}(\mathcal{A}) \star \Theta\mathcal{A}$ verkrijgen we dus een object uit $\Theta\mathcal{B}$. ◁

3.4 Voorbeelden van combinatorische klassen en specificaties

In deze sectie worden een aantal voorbeelden gegeven hoe een beschrijfbare combinatorische klasse formeel beschreven kan worden m.b.v. een specificatie.

Voorbeeld 3.61 *Beschouw de verzameling $\mathcal{W}$ van alle binaire strings over het alfabet $\{0,1\}$. Dus*

$$\mathcal{W} \;=\; \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}.$$

We stellen de grootte van een binaire string $\omega \in \mathcal{W}$ gelijk aan het aantal symbolen ervan. We zien dat $\mathcal{W}$ een combinatorische klasse is, want de grootte van elke binaire string is een natuurlijk getal en het aantal binaire strings van een bepaalde grootte n is eindig voor elke waarde van n.

In de telwereld kunnen we $\mathcal{W}$ als een ongelabelde combinatorische klasse voorstellen als volgt:

$$\begin{aligned}\mathcal{W} \;=\; \{&\epsilon, (\bigcirc_0), (\bigcirc_1), (\bigcirc_0, \bigcirc_0), (\bigcirc_0, \bigcirc_1), (\bigcirc_1, \bigcirc_0), (\bigcirc_1, \bigcirc_1),\\ &(\bigcirc_0, \bigcirc_0, \bigcirc_0), (\bigcirc_0, \bigcirc_0, \bigcirc_1), \ldots\}.\end{aligned}$$

We zien dat een object uit $\mathcal{W}$ een sequentie is, waarbij de componenten gelijk zijn aan atomen die de markering `0` of `1` dragen. Bijgevolg kan $\mathcal{W}$ beschreven worden d.m.v. de volgende iteratieve ongelabelde specificatie S_1:

$$S_1 \;\begin{cases} W \to \mathrm{SEQ}(L) \\ L \to A \;+\; B \\ A \to \bigcirc_0 \\ B \to \bigcirc_1 \end{cases}$$

Er bestaat ook een recursieve contextvrije specificatie S_2 die de klasse $\mathcal{W}$ beschrijft:

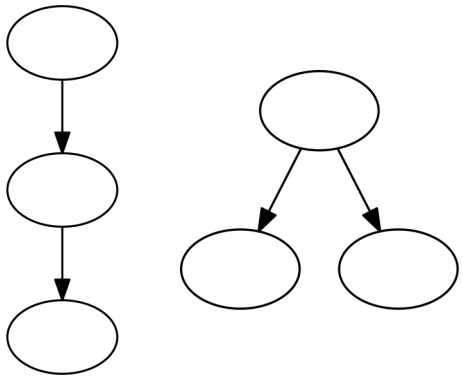
$$S_2 \;\begin{cases} W \to V \;+\; \epsilon \\ V \to W \;\times\; L \\ L \to A \;+\; B \\ A \to \bigcirc_0 \\ B \to \bigcirc_1 \end{cases}$$

◁

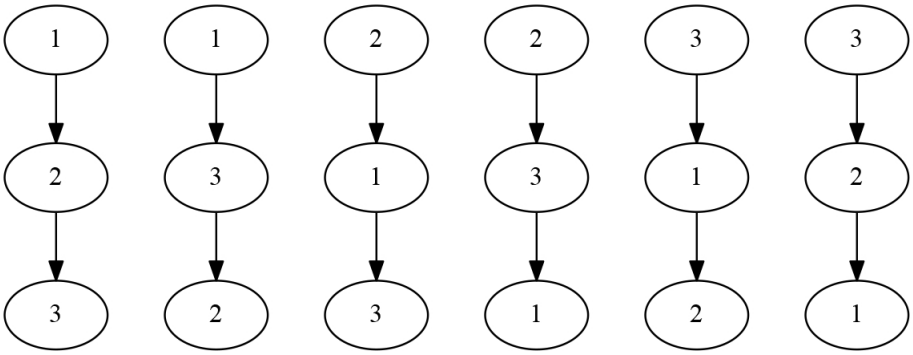
Voorbeeld 3.62 *In dit voorbeeld beschouwen we zogenaamde* vlakke bomen *[23, p. 31 - 32]. Dit zijn bomen waarbij de volgorde van de kinderen van een node belangrijk is. Deze bomen zijn als het ware ingebed* (Eng: embedded) *in het vlak. De twee ongelabelde vlakke bomen van grootte 3 worden weergegeven in Figuur 3.11.*

In het gelabelde universum kunnen we ook vlakke bomen beschouwen. Voor elke ongelabelde vlakke boom van grootte n zijn er $n!$ gelabelde vlakke bomen, die dezelfde onderliggende structuur hebben [23, p. 126]. Dit komt doordat elk label uit het interval $[1 \dots n]$ bij elke node van een gelabelde vlakke boom kan staan. Hierdoor zijn alle permutaties van het interval $[1 \dots n]$ een geldige labelling.

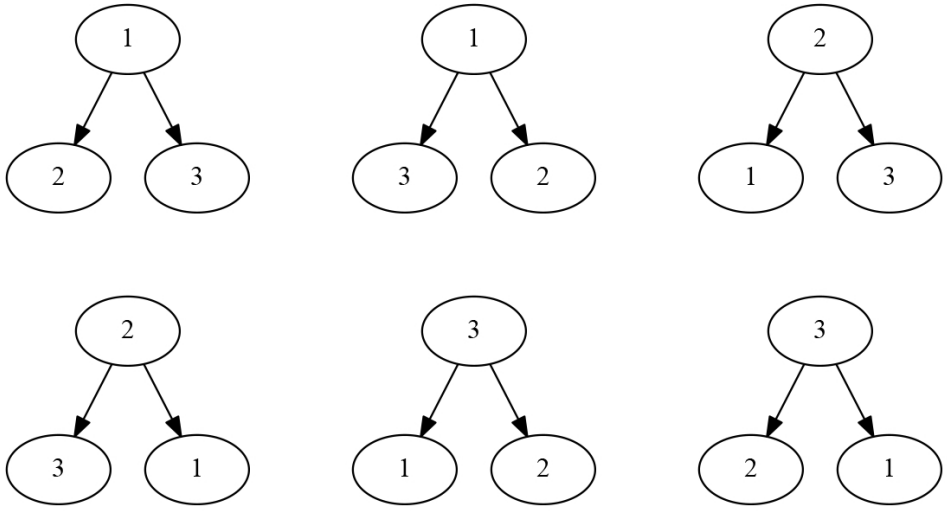
De $2 \times 3! = 12$ gelabelde vlakke bomen van grootte 3 worden afgebeeld in Figuur 3.12 en 3.13.



Figuur 3.11: *De twee ongelabelde vlakke bomen van grootte 3 uit Voorbeeld 3.62.*



Figuur 3.12: *Zes van de twaalf gelabelde vlakke bomen van grootte 3 uit Voorbeeld 3.62.*



Figuur 3.13: *De andere zes gelabelde vlakke bomen van grootte 3 uit Voorbeeld 3.62.*

In een vlakke boom wordt elke node gekoppeld met een aantal kinderen (mogelijk 0). In de ongelabelde combinatorische wereld kunnen we dit modelleren door een atoom m.b.v. het cartesisch product te combineren met een sequentie van atomen. Elk atoom uit deze sequentie kan vervolgens opnieuw gecombineerd worden met een sequentie van atomen, enzovoort. We kunnen ongelabelde vlakke bomen m.a.w. modelleren d.m.v. de volgende recursieve ongelabelde specificatie G_{pb}:

$$G_{pb} \begin{cases} G \to Z \times H \\ H \to \mathrm{SEQ}(G) \\ Z \to \bigcirc. \end{cases}$$

In het gelabelde universum kunnen we een analoge specificatie G_{pb} definiëren, die gebruikmaakt van het gelabelde product en een gelabeld atoom:

$$G_{pb} \begin{cases} G \to Z \star H \\ H \to \mathrm{SEQ}(G) \\ Z \to \textcircled{1}. \end{cases}$$

Uit de ongelabelde specificatie G_{pb} kunnen o.a. de volgende objecten afgeleid worden:

$$(\bigcirc, (\bigcirc, (\bigcirc))) \quad \textit{en} \quad (\bigcirc, (\bigcirc, \bigcirc))$$

Merk op dat we het lege object ϵ weggelaten hebben uit deze objecten om de leesbaarheid te verhogen. Het eerste object komt overeen met de abstracte combinatorische

voorstelling van de eerste vlakke boom uit Figuur 3.11. Het tweede object komt op zijn beurt overeen met de combinatorische voorstelling van de tweede vlakke boom uit Figuur 3.11.

De gelabelde vlakke bomen uit Figuur 3.12 en Figuur 3.13 kunnen op een analoge manier voorgesteld worden. Het enige verschil is dat de atomen in de combinatorische voorstelling een label moeten dragen. ◁

Hoofdstuk 4

Tellen van combinatorische objecten

In dit hoofdstuk gaan we dieper in op het tellen van het aantal objecten van een bepaalde grootte uit een combinatorische klasse. Elke combinatorische klasse kan voorgesteld worden d.m.v. een zogenaamde counting sequence. Dit is een rij van getallen die aangeeft hoeveel objecten van elke grootte de klasse bevat.

Het doel is om de counting sequence van een bepaalde combinatorische klasse te berekenen. Deze counting sequence wordt vervolgens in het volgende hoofdstuk gebruikt bij het random genereren van combinatorische objecten.

In dit hoofdstuk beschrijven we twee methodes om de counting sequence van een combinatorische klasse te berekenen. De eerste methode steunt op speciale veeltermen, waarin de counting sequence gecodeerd zit. De tweede methode steunt op wiskundige gelijkheden waarmee de n-de term van de counting sequence van een klasse kan berekend worden o.b.v. de counting sequences van de klassen die gebruikt worden om deze klasse op te bouwen.

De begrippen, theorema's en bewijzen die in dit hoofdstuk beschreven worden, komen voornamelijk uit [23].

4.1 Begrippen en notaties

We starten dit hoofdstuk met een aantal veelgebruikte begrippen en notaties omtrent het tellen van objecten uit een combinatorische klasse.

4.1.1 Counting sequence

Definitie 4.1 De *counting sequence* van een combinatorische klasse $\mathcal{A}$ is de sequentie van natuurlijke getallen $(A_n)_{n\geq 0}$, waarbij A_n het aantal objecten van grootte n uit $\mathcal{A}$ is. ⧫

Kortom, de counting sequence van een klasse is een oneindige sequentie van getallen, waarbij het i-de getal het aantal objecten van grootte i uit deze klasse voorstelt (met i startend vanaf 0).

Voorbeeld 4.2 *Beschouw de ongelabelde combinatorische klasse $\mathcal{W}$ uit Voorbeeld 3.61 De counting sequence van $\mathcal{W}$ is gelijk aan:*

$$(W_n) \quad = \quad 1, 2, 4, 8, \ldots$$

of:

$$W_n \quad = \quad 2^n.$$

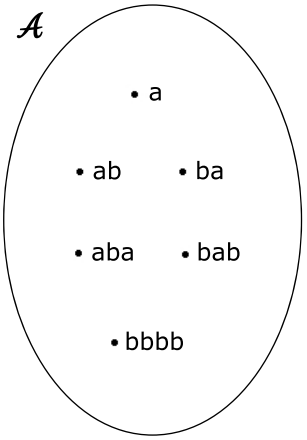
Inderdaad, om een binaire string van grootte n te bekomen zijn er op elke positie telkens twee mogelijkheden (0 of 1). Deze mogelijkheden moeten allemaal met elkaar vermenigvuldigd worden om het aantal binaire strings van grootte n te bekomen. ◁

Definitie 4.3 Twee combinatorische klassen $\mathcal{A}$ en $\mathcal{B}$ zijn *combinatorisch isomorf*, genoteerd met $\mathcal{A} \cong \mathcal{B}$, als en slechts als hun counting sequences gelijk zijn. ⧫

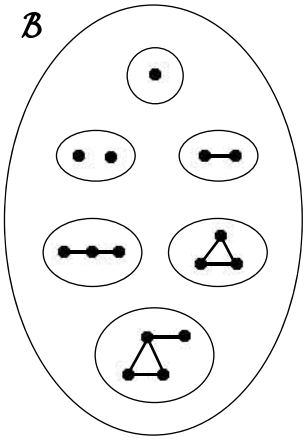
Twee combinatorische klassen zijn dus combinatorisch isomorf als ze voor elke $n \in \mathbb{N}$ evenveel objecten bevatten van grootte n.

Stel dat twee combinatorische klassen $\mathcal{A}$ en $\mathcal{B}$ combinatorisch isomorf zijn, maar dat $\mathcal{A}$ moeilijker te beschrijven is dan $\mathcal{B}$ (bv. doordat er alleen een recursieve specificatie voor $\mathcal{A}$ bestaat). Als we enkel geïnteresseerd zijn in het tellen van objecten is het eenvoudiger om met $\mathcal{B}$ verder te werken, omdat het aantal objecten van elke grootte toch gelijk is in beide klassen.

Voorbeeld 4.4 *Beschouw de twee combinatorische klassen $\mathcal{A}$ en $\mathcal{B}$ die weergegeven worden in Figuur 4.1 en 4.2. De klasse $\mathcal{A}$ bevat zes strings en de klasse $\mathcal{B}$ bevat zes grafen. We stellen de grootte van een graaf gelijk aan het aantal nodes ervan.*



Figuur 4.1: *De klasse $\mathcal{A}$ uit Voorbeeld 4.4*



Figuur 4.2: *De klasse $\mathcal{B}$ uit Voorbeeld 4.4*

Opmerking: De grafen van $\mathcal{B}$ worden in Figuur 4.2 telkens afgebeeld in een aparte gesloten kromme in het vlak om ze duidelijk van elkaar te onderscheiden. Dit is van belang voor de tweede graaf, die bestaat uit twee losse nodes zonder verbinding.

De klassen $\mathcal{A}$ en $\mathcal{B}$ zijn combinatorisch isomorf, omdat hun counting sequences (A_n) en (B_n) aan elkaar gelijk zijn. We zien namelijk in Figuur 4.1 en 4.2 dat de beide counting sequences gelijk zijn aan:

$$0, 1, 2, 2, 1, 0, 0, \ldots$$

$\mathcal{A}$ en $\mathcal{B}$ bevatten m.a.w. allebei nul objecten van grootte 0, één van grootte 1, twee van grootte 2, twee van grootte 3 en één van grootte 4. ◁

4.1.2 Generating functions

Ordinary generating functions (ongelabeld universum)

Definitie 4.5 De *ordinary generating function (OGF)* $A(z)$ van (de counting sequence van) een *ongelabelde* combinatorische klasse $\mathcal{A}$ is gedefinieerd als volgt:

$$A(z) = \sum_{n=0}^{\infty} A_n z^n. \tag{4.1}$$

♦

De OGF van een ongelabelde klasse $\mathcal{A}$ is dus een oneindige veelterm in een variabele z met als coëfficiënten de counting sequence van $\mathcal{A}$. Dit wil zeggen dat de coëfficiënt van de n-de term gelijk is het aan het aantal objecten van grootte n van $\mathcal{A}$, met andere woorden:

$$[z^n]A(z) \quad = \quad A_n.$$

De OGF brengt m.a.w. de counting sequence van de gebruikte combinatorische klasse voort en kan daarom gezien worden als een gereduceerde representatie van deze klasse. Een OGF bevat namelijk enkel informatie over het aantal objecten van iedere grootte en niets meer.

Opmerking: de OGF van een klasse wordt aangeduid met dezelfde letter als deze klasse, maar dan in gewoon (niet-kalligrafisch) schrift.

De OGF van een ongelabelde klasse $\mathcal{A}$ kan ook in een meer combinatorische vorm genoteerd worden als volgt:

$$A(z) \quad = \quad \sum_{\alpha \in \mathcal{A}} z^{|\alpha|}. \tag{4.2}$$

In deze alternatieve formule komt de term van z^n evenveel keer voor als het aantal objecten van grootte n in $\mathcal{A}$. Dit is precies hetzelfde idee als in formule (4.1). De twee formules zijn dus equivalent, wat ook blijkt uit het volgende voorbeeld.

Voorbeeld 4.6 *Zij $\mathcal{A}$ een combinatorische klasse, waarvan de counting sequence begint met:*

$$(A_n) \quad = \quad 1, 1, 2, 4, 0, 0, \ldots$$

Volgens formule (4.1) is de OGF van $\mathcal{A}$ gelijk aan:

$$\begin{aligned} A(z) &= 1z^0 + 1z^1 + 2z^2 + 4z^3 + 0z^4 + 0z^5 + \ldots \\ &= 1z^0 + 1z^1 + 2z^2 + 4z^3. \end{aligned}$$

Volgens formule (4.2) is de OGF van $\mathcal{A}$ gelijk aan:

$$\begin{aligned} A(z) &= z^0 + z^1 + z^2 + z^2 + z^3 + z^3 + z^3 + z^3 \\ &= 1z^0 + 1z^1 + 2z^2 + 4z^3. \end{aligned}$$

De twee formules geven in dit voorbeeld inderdaad precies dezelfde uitkomst. ◁

Voorbeeld 4.7 *Als we de counting sequence van de klasse $\mathcal{W}$ uit Voorbeeld 3.61 invullen in formule (4.1), zien we dat de OGF van $\mathcal{W}$ gelijk is aan:*

$$\begin{aligned} W(z) &= \sum_{n=0}^{\infty} 2^n z^n \\ &= 1z^0 + 2z^1 + 4z^2 + 8z^3 + \ldots \end{aligned}$$ ◁

Merk op dat de OGF's van twee combinatorisch isomorfe klassen identiek zijn, omdat hun counting sequences identiek zijn. Alle coëfficiënten van beide OGF's zijn m.a.w. onderling gelijk aan elkaar. Zo zijn bijvoorbeeld de OGF's van de klassen $\mathcal{A}$ en $\mathcal{B}$ uit Voorbeeld 4.4 beide gelijk aan:

$$\begin{aligned} A(z) &= B(z) \\ &= 0z^0 + 1z^1 + 2z^2 + 2z^3 + 1z^4 + 0z^5 + 0z^6 + \ldots \\ &= 0z^0 + 1z^1 + 2z^2 + 2z^3 + 1z^4. \end{aligned}$$

Exponential generating functions (gelabeld universum)

In het gelabelde universum spreekt met niet van *ordinary* generating functions, maar van *exponential* generating functions:

Definitie 4.8 De *exponential generating function (EGF)* $\hat{A}(z)$ van (de counting sequence van) een *gelabelde* combinatorische klasse $\mathcal{A}$ is gedefinieerd als volgt:

$$\hat{A}(z) = \sum_{n=0}^{\infty} \frac{A_n}{n!} z^n. \tag{4.3}$$ ⧫

De EGF van een gelabelde klasse $\mathcal{A}$ is dus ook een oneindige veelterm in een variabele z. Alleen wordt in de EGF niet de counting sequence van $\mathcal{A}$ gecodeerd, maar wel telkens het quotiënt van (één getal uit) de counting sequence en het natuurlijke getal $n!$. Dit wil zeggen dat de coefficiënt van de n-de term van de EGF niet gelijk is aan het aantal objecten van grootte n uit $\mathcal{A}$, maar aan dit aantal gedeeld door $n!$, of met andere woorden:

$$[z^n]\hat{A}(z) = \frac{A_n}{n!}.$$

Of

$$A_n \quad = \quad n! \times [z^n]\hat{A}(z).$$

Opmerking: de EGF van een klasse wordt aangeduid met dezelfde letter als deze klasse, opnieuw in niet-kalligrafisch schrift, met een hoedje erop.

De EGF van een gelabelde klasse $\mathcal{A}$ kan (net zoals in het ongelabelde geval) ook in een meer combinatorische vorm genoteerd worden als volgt:

$$\hat{A}(z) \quad = \quad \sum_{\alpha \in \mathcal{A}} \frac{z^{|\alpha|}}{|\alpha|!}. \tag{4.4}$$

Opnieuw kan eenvoudig ingezien worden dat formule (4.3) en (4.4) equivalent zijn.

4.2 Berekening van de counting sequence van een klasse

In deze sectie beschrijven we hoe (de n-de term van) de counting sequence van een constructible combinatorische klasse berekend kan worden. We reiken twee methodes aan die we hiervoor kunnen gebruiken. Beide methodes werken met een well-defined specificatie die de combinatorische klasse in kwestie beschrijft.

De eerste methode maakt gebruik van generating functions. Bij elke nieuwe klasse die wordt geconstrueerd m.b.v. de in Hoofdstuk 3 ingevoerde constructors hoort een generating function. Deze generating function kan bepaald worden a.d.h.v. de generating functions van de gebruikte klassen. Deze methode beschrijven we zowel in het ongelabelde universum als in het gelabelde universum.

Voor de tweede methode maken we gebruik van de standaardvorm (STDF) van gelabelde specificaties, die geïntroduceerd werd in Sectie 3.3.8. Elke gelabelde specificatie kan omgezet worden naar een specificatie waarin slechts drie constructors gebruikt worden. Voor elk van deze drie constructos kunnen we de counting sequence van de resulterende klasse berekenen m.b.v. wiskundige gelijkheden.

4.2.1 Admissible constructors

In Sectie 3.3.3 en 3.3.4 hebben we respectievelijk zes en vijf verzamelingconstructors uit enerzijds het ongelabelde en anderzijds het gelabelde universum ingevoerd die we in deze thesis gebruiken om combinatorische klassen te construeren. Herinner dat een verzamelingconstructor Φ_v één nieuwe klasse $\mathcal{A}$ creëert op basis van m klassen $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$. Voor zulk een verzamelingconstructor voeren we nu een belangrijke eigenschap in die ons helpt bij de berekening van de counting sequence van $\mathcal{A}$.

Definitie 4.9 Zij Φ_v een verzamelingconstructor die op basis van m combinatorische klassen één nieuwe klasse creëert, of met andere woorden:

$$\mathcal{A} \quad = \quad \Phi_v(\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}).$$

We noemen Φ_v een *admissible* constructor als en slechts als de counting sequence (A_n) van $\mathcal{A}$ enkel afhangt van de counting sequences $(B_n^{(1)}), \ldots, (B_n^{(m)})$ van $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$. ♦

De zes verzamelingconstructors uit het ongelabelde universum: het cartesisch product, de combinatorische unie, de sequence-, cycle-, multiset- en powerset-constructor zijn allemaal admissible constructors [23].

De vijf verzamelingconstructors uit het gelabelde universum: het gelabelde product, de combinatorische unie, de sequence-, cycle- en set-constructor zijn ook allemaal admissible constructors [23].

Elke admissible constructor voldoet aan een belangrijke eigenschap:

Stelling 4.10 *Zij Φ_v een* ongelabelde *admissible constructor, gedefinieerd als volgt:*

$$\mathcal{A} \quad = \quad \Phi_v(\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}),$$

waarbij $\mathcal{A}$ en $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ ongelabelde *combinatorische klassen zijn. Dan bestaat er een overeenkomstige operator Ψ die werkt met de overeenkomstige* OGF's*, of met andere woorden:*

$$A(z) \quad = \quad \Psi(B^{(1)}(z), \ldots, B^{(m)}(z)).$$

In het gelabelde universum kunnen we een analoge eigenschap definiëren:

Stelling 4.11 *Zij* Φ_v *een* gelabelde *admissible constructor, gedefinieerd als volgt:*

$$\mathcal{A} \;=\; \Phi_v(\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}),$$

waarbij $\mathcal{A}$ *en* $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ gelabelde *combinatorische klassen zijn. Dan bestaat er een overeenkomstige operator* Ψ *die werkt met de overeenkomstige* EGF's*, of met andere woorden:*

$$\hat{A}(z) \;=\; \Psi(\hat{B}^{(1)}(z), \ldots, \hat{B}^{(m)}(z)).$$

Dit betekent dus dat we een combinatie van combinatorische klassen kunnen *vertalen* naar een combinatie van de overeenkomstige generating functions. Wanneer we enkel geïnteresseerd zijn in het tellen van het aantal objecten van grootte n uit de nieuwe klasse $\mathcal{A}$ volstaat het om de generating functions van de klassen $\mathcal{B}_1, \ldots, \mathcal{B}_k$ te combineren volgens Ψ. De coëfficiënten van de bekomen generating function geven dan, zoals eerder vermeld, het aantal objecten van iedere grootte n uit $\mathcal{A}$ weer. Op die manier moeten we niet eerst alle objecten van deze nieuwe klasse construeren (wat tijdrovend kan zijn) om ze daarna te tellen.

In de volgende twee secties bewijzen we de twee bovenstaande stellingen door voor elke in Hoofdstuk 3 ingevoerde basisconstructor Φ de *vertaalregel* naar de operator Ψ te beschrijven. Dit doen we zowel voor de ongelabelde constructors (Sectie 4.2.2) als voor de gelabelde constructors (Sectie 4.2.3).

4.2.2 Vertaalregels ongelabelde basisconstructors

Basisklassen

Ook bij de twee ongelabelde basisklassen $\mathcal{E}$ en $\mathcal{Z}$ die gebruikt kunnen worden als argumenten van de zes ongelabelde basisconstructors hoort telkens een OGF. Deze worden beschreven in deze sectie.

Neutrale klasse

Theorema 4.12 *Voor de OGF van een ongelabelde neutrale klasse* $\mathcal{E} = \{\epsilon\}$ *geldt dat:*

$$\begin{aligned} E(z) &= 1z^0 \\ &= 1. \end{aligned}$$

$\Diamond$

De neutrale klasse $\mathcal{E}$ bevat namelijk één object van grootte 0. Als we dit vervolgens invullen in formule (4.1) of (4.2) krijgen we de veelterm $1z^0$, die afgekort wordt tot 1.

Atomaire klasse

Theorema 4.13 *Voor de OGF van een ongelabelde atomaire klasse* $\mathcal{Z} = \{\bigcirc\}$ *geldt dat:*

$$\begin{aligned} Z(z) &= 0z^0 + 1z^1 \\ &= z. \end{aligned}$$ ◊

De atomaire klasse $\mathcal{Z}$ bevat namelijk nul objecten van grootte 0 en één van grootte 1. Als we dit vervolgens invullen in formule (4.1) of (4.2) krijgen we de veelterm $0z^0 + 1z^1$, die afgekort wordt tot z.

Basisconstructors

In deze sectie beschrijven we hoe de zes ongelabelde basisconstructors kunnen vertaald worden naar de overeenkomstige operator die werkt met de OGF's. Voor de bewijzen van de verschillende theorema's verwijzen we naar Appendix A.1.

Cartesisch product

Theorema 4.14 *Voor de OGF van het cartesisch product* $\mathcal{A} = \mathcal{B} \times \mathcal{C}$ *geldt dat:*

$$A(z) = B(z) \times C(z).$$ ◊

De OGF's van de klassen $\mathcal{B}$ en $\mathcal{C}$ moeten in dit geval met elkaar vermenigvuldigd worden (merk op dat dit een vermenigvuldiging van twee veeltermen betekent). De intuïtie achter deze vermenigvuldiging is als volgt: we kunnen een object van grootte n uit $\mathcal{A}$ bekomen door alle combinaties van een object van $\mathcal{B}_i$ met een object van $\mathcal{C}_j$ te beschouwen, waarbij $i + j = n$.

Bij een veeltermvermenigvuldiging van $f(z)$ met $g(z)$ wordt elke term van $f(z)$ vermenigvuldigd met elke term van $g(z)$ en van het geheel wordt vervolgens de som genomen.

In het geval van OGF's levert dit alle combinaties van groottes op, omdat alle termen $c_1 z^i$ en $c_2 z^j$ met $i + j = n$ combineren tot $c_1 c_2 z^n$. De som van al deze

nieuwe coëfficiënten $c_1 c_2$ geeft uiteindelijk de coëfficiënt van z^n in $A(z)$ weer en wegens Definitie 4.5 dus het aantal objecten van grootte n van $\mathcal{A}$. Merk op dat er voor elke grootte altijd een overeenkomstige coëfficiënt in $A(z)$ bestaat, die mogelijk gelijk is aan 0.

Combinatorische unie

Theorema 4.15 *Voor de OGF van de combinatorische unie $\mathcal{A} = \mathcal{B} + \mathcal{C}$ geldt dat:*

$$A(z) = B(z) + C(z). \qquad \Diamond$$

In het geval van de combinatorische unie moeten de OGF's van de twee gebruikte klassen $\mathcal{B}$ en $\mathcal{C}$ dus opgeteld worden (merk op dat dit een optelling van veeltermen betekent).

Dit komt doordat de combinatorische unie van twee klassen gedefinieerd is als de disjuncte unie ervan. Tijdens het creëeren van de disjuncte kopieën van $\mathcal{B}$ en $\mathcal{C}$ wordt er namelijk niets gewijzigd aan de grootte van de objecten uit deze klassen. Daarom is het aantal objecten van een bepaalde grootte uit $\mathcal{A}$ gelijk aan de som van het aantal objecten van deze grootte uit $\mathcal{B}$ en het aantal objecten van deze grootte uit $\mathcal{C}$.

Ter illustratie volgt nu een voorbeeld van de berekening van de OGF van een klasse die resulteert uit de combinatorische unie en het cartesisch product. Hieruit zal duidelijker worden waarom de OGF's in het geval van het cartesisch product vermenigvuldigd moeten worden.

Voorbeeld 4.16 *Zij $\mathcal{B}$ een ongelabelde combinatorische klasse die één object van grootte 0 en twee van grootte 3 bevat en zij $\mathcal{C}$ een ongelabelde combinatorische klasse die één object van grootte 0 en twee van grootte 1 bevat. Dit kan genoteerd worden m.b.v. de volgende OGF's:*

$$\begin{aligned} B(z) &= 1z^0 + 2z^3, \\ C(z) &= 1z^0 + 2z^1. \end{aligned}$$

We zien intuïtief dat de combinatorische unie van deze klassen, $\mathcal{A} = \mathcal{B}+\mathcal{C}$, bestaat uit twee objecten van grootte 0, twee van grootte 1, nul van grootte 2 en twee van grootte 3. Als we de OGF's van $\mathcal{B}$ en $\mathcal{C}$ optellen, moeten we dit resultaat ook

bekomen:

$$\begin{aligned} A(z) &= B(z) + C(z) \\ &= 1z^0 + 2z^3 + 1z^0 + 2z^1 \\ &= 2z^0 + 2z^1 + 2z^3. \end{aligned}$$

Dat is inderdaad het geval, want $[z^0]A(z) = 2$, $[z^1]A(z) = 2$, $[z^2]A(z) = 0$ *en* $[z^3]A(z) = 2$.

Voor het cartesisch product van deze klassen, $\mathcal{A} = \mathcal{B} \times \mathcal{C}$*, kunnen we een analoge redenering toepassen. Stel dat we alleen de objecten van grootte 4 uit de klasse* $\mathcal{A}$ *beschouwen (voor de andere groottes geldt de uitleg uiteraard ook). Deze objecten worden bekomen door een object van grootte 0 van* $\mathcal{B}$ *te koppelen met een object van grootte 4 van* $\mathcal{C}$ *(idem voor twee objecten uit respectievelijk* $\mathcal{B}_1$ *en* $\mathcal{C}_3$*,* $\mathcal{B}_2$ *en* $\mathcal{C}_2$*,* $\mathcal{B}_3$ *en* $\mathcal{C}_1$*,* $\mathcal{B}_4$ *en* $\mathcal{C}_0$*). We zien intuïtief dat dit alleen resultaat oplevert als we de twee objecten van* $\mathcal{B}_3$ *koppelen met de twee objecten van* $\mathcal{C}_1$*. Bijgevolg zitten er dus vier objecten van grootte 4 in* $\mathcal{A}$*.*

Als we de OGF's van $\mathcal{B}$ *en* $\mathcal{C}$ *met elkaar vermenigvuldigen, moeten we dit resultaat ook bekomen:*

$$\begin{aligned} A(z) &= B(z) \times C(z) \\ &= (1z^0 + 2z^3) \times (1z^0 + 2z^1) \\ &= 1z^0 \times 1z^0 + 1z^0 \times 2z^1 + 2z^3 \times 1z^0 + 2z^3 \times 2z^1 \\ &= 1z^0 + 2z^1 + 2z^3 + 4z^4. \end{aligned}$$

Uit deze berekening zien we dus ook dat $\mathcal{A}$ *vier objecten van grootte 4 bevat (want* $[z^4]A(z) = 4$*), wat hierboven ook al gevonden werd.* ◁

Sequence

Theorema 4.17 *Voor de OGF van de sequence-constructor* $\mathcal{A} = \text{SEQ}(\mathcal{B})$ *(met de aanname dat* $B_0 = 0$ *wegens Definitie 3.11) geldt dat:*

$$A(z) = \frac{1}{1 - B(z)} \qquad \Diamond$$

Powerset

Theorema 4.18 *Voor de OGF van de powerset-constructor* $\mathcal{A} = \text{PSET}(\mathcal{B})$ *(met de aanname dat* $B_0 = 0$ *wegens Definitie 3.14) geldt dat:*

$$A(z) = \begin{cases} \prod_{n=1}^{\infty} (1+z^n)^{B_n} \\ \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} \times B(z^k)\right) \end{cases} \qquad \Diamond$$

Multiset

Theorema 4.19 *Voor de OGF van de multiset-constructor* $\mathcal{A} = \text{MSET}(\mathcal{B})$ *(met* $B_0 = 0$ *wegens Definitie 3.13) geldt dat:*

$$A(z) = \begin{cases} \prod_{n=1}^{\infty} (1-z^n)^{-B_n} \\ \exp\left(\sum_{k=1}^{\infty} \frac{1}{k} \times B(z^k)\right) \end{cases} \qquad \Diamond$$

Cycle

Theorema 4.20 *Voor de OGF van de cycle-constructor* $\mathcal{A} = \text{CYC}(\mathcal{B})$ *(met* $B_0 = 0$ *wegens Definitie 3.12) geldt dat:*

$$A(z) = \sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \times \log\left(\frac{1}{1-B(z^k)}\right)$$

waarbij $\varphi(k)$ *de Euler-indicator is, die gedefinieerd is als het aantal getallen in het interval* $[1..k]$ *dat relatief priem is t.o.v. k [23, p. 730].* $\Diamond$

Het bewijs van dit theorema is te vinden in [23, p. 729 - 730].

4.2.3 Vertaalregels gelabelde basisconstructors

Basisklassen

Net zoals bij de ongelabelde basisklassen, hoort er ook een generating function (in dit geval een EGF) bij elke gelabelde basisklasse.

Neutrale klasse

Theorema 4.21 *Voor de EGF van een gelabelde neutrale klasse* $\mathcal{E} = \{\epsilon\}$ *geldt dat:*

$$\begin{aligned} \hat{E}(z) &= \frac{1}{0!}z^0 \\ &= 1. \end{aligned}$$ ◊

De neutrale klasse $\mathcal{E}$ bevat namelijk één object van grootte 0. Als we dit vervolgens invullen in formule (4.3) of (4.4) krijgen we de veelterm $\frac{1}{1}z^0$, die afgekort wordt tot 1.

Atomaire klasse

Theorema 4.22 *Voor de EGF van een gelabelde atomaire klasse* $\mathcal{Z} = \{\textcircled{1}\}$ *geldt dat:*

$$\begin{aligned} \hat{Z}(z) &= \frac{0}{0!}z^0 + \frac{1}{1!}z^1 \\ &= z. \end{aligned}$$ ◊

De atomaire klasse $\mathcal{Z}$ bevat namelijk nul objecten van grootte 0 en één van grootte 1. Als we dit vervolgens invullen in formule (4.3) of (4.4) krijgen we de veelterm $\frac{0}{1}z^0 + \frac{1}{1}z^1$, die afgekort wordt tot z.

Basisconstructors

In deze sectie beschrijven we hoe de vijf gelabelde basisconstructors kunnen vertaald worden naar de overeenkomstige operator die werkt met de EGF's. Voor de bewijzen van de verschillende theorema's verwijzen we naar [21, p. 23].

Gelabeld product

Theorema 4.23 *Voor de EGF van het gelabelde product $\mathcal{A} = \mathcal{B} \star \mathcal{C}$ geldt dat:*

$$\hat{A}(z) \quad = \quad \hat{B}(z) \times \hat{C}(z). \qquad \Diamond$$

Voorbeeld 4.24 *Beschouw de twee gelabelde klassen $\mathcal{B}$ en $\mathcal{C}$ uit Voorbeeld 3.24. De EGF van beide klassen is gelijk aan $2z$. Als we deze EGF van $\mathcal{B}$ vermenigvuldigen met die van $\mathcal{C}$ (en dus de veelterm $2z$ kwadrateren), verkrijgen we EGF $4z^2$. Als we hieruit vervolgens de coëfficiënt van de tweede term halen en vermenigvuldigen met $2!$, verkrijgen we het aantal objecten van grootte 2 uit de klasse $\mathcal{A} = \mathcal{B} \star \mathcal{C}$. We zien dat $\mathcal{A}$ acht objecten van grootte 2 bevat, wat we ook al zagen in Voorbeeld 3.24.* $\triangleleft$

Combinatorische unie

Theorema 4.25 *Voor de EGF van de combinatorische unie $\mathcal{A} = \mathcal{B} + \mathcal{C}$ geldt dat:*

$$\hat{A}(z) \quad = \quad \hat{B}(z) + \hat{C}(z). \qquad \Diamond$$

Net zoals in het ongelabelde geval moeten de EGF's van de beide gebruikte klassen opgeteld worden omdat de combinatorische unie gedefinieerd is als de disjuncte unie.

Sequence

Theorema 4.26 *Voor de EGF van de sequence-constructor $\mathcal{A} = \mathrm{SEQ}(\mathcal{B})$ (met de aanname dat $B_0 = 0$ wegens Definitie 3.28) geldt dat:*

$$\hat{A}(z) \quad = \quad \frac{1}{1 - \hat{B}(z)} \qquad \Diamond$$

Set

Theorema 4.27 *Voor de EGF van de set-constructor $\mathcal{A} = \mathrm{SET}(\mathcal{B})$ (met de aanname dat $B_0 = 0$ wegens Definitie 3.30) geldt dat:*

$$\hat{A}(z) \quad = \quad e^{\hat{B}(z)} \qquad \Diamond$$

Cycle

Theorema 4.28 *Voor de EGF van de cycle-constructor* $\mathcal{A} = \mathrm{CYC}(\mathcal{B})$ *(met* $B_0 = 0$ *wegens Definitie 3.29) geldt dat:*

$$\hat{A}(z) \;=\; \log\left(\frac{1}{1-\hat{B}(z)}\right). \qquad \Diamond$$

Pointing operator Ook de gelabelde pointing operator, die we geïntroduceerd hebben in Sectie 3.3.8 is een admissible constructor [23, p. 137].

Theorema 4.29 *Voor de EGF van de pointing operator* $\mathcal{A} = \Theta(\mathcal{B})$ *geldt dat:*

$$\hat{A}(z) \;=\; z \times \hat{B}'(z)$$

waarbij $\hat{B}'(z)$ *de eerste afgeleide is van de veelterm* $\hat{B}(z)$. $\Diamond$

In het geval van de pointing operator wordt de resulterende EGF dus bekomen door de EGF van de gebruikte klasse af te leiden en daarna te vermenigvuldigen met de veelterm z.

4.2.4 Methode 1: Berekening via generating functions

In deze sectie beschrijven we een eerste methode voor de berekening (van de n-de term) van de counting sequence van een constructible combinatorische klasse $\mathcal{A}$. Omdat we een constructible klasse beschouwen, beschikken we over een specificatie S (in SPNF) die deze klasse beschrijft. In het geval van een ongelabelde klasse is dat een specificatie uit Ω_o en in het geval van een gelabelde klasse is dat een specificatie uit Ω_g.

Herinner dat de klasse $\mathcal{A}$ overeenkomt met de verzameling van alle objecten die afgeleid worden uit het startsymbool van S. Voor elke basisconstructor die kan voorkomen in S hebben we in de voorgaande secties een vertaalregel ingevoerd waarmee we de generating function van de resulterende klasse kunnen bepalen.

De eerste stap van de deze methode bestaat erin om de generating function van $\mathcal{A}$ te bepalen m.b.v. de ingevoerde vertaalregels. Indien $\mathcal{A}$ in haar specificatie beschreven wordt o.b.v. simpelere (hulp)klassen, moeten we eerst de generating functions van deze klassen bepalen, vooraleer we die van $\mathcal{A}$ kunnen bepalen.

Omdat de counting sequence van $\mathcal{A}$ gecodeerd zit in de zojuist bepaalde generating function, bestaat de tweede en tevens laatste stap van deze methode erin om de coëfficiënt van de n-de term van de generating function van $\mathcal{A}$ te bepalen. Dit is echter typisch een uitgebreide en complexe taak. Er bestaat een handmatige methode om dit te doen die men *complexe analyse* noemt, maar de werkwijze van deze methode is te complex en te uitgebreid om te behandelen in deze thesis. Daarom verwijzen we naar [23, Deel B] voor de geïnteresseerde lezer.

Gelukkig voor ons bestaan er softwarepakketten die op een automatische manier de coëfficiënt van de n-de term van een generating function kunnen bepalen. Deze pakketten maken gebruik van de zogenaamde *Taylor-expansie* om een willekeurige reële functie in een bepaald punt (meestal $z = 0$) te benaderen door diens *Taylorpolynoom* [10, p. 124 - 139].

De Taylorpolynoom $t(z)$ (ook wel Taylorreeks genoemd) van een oneindig afleidbare reële functie $f(z)$ rond een bepaald punt $z = a$ wordt berekend m.b.v. de volgende formule [9]:

$$t(z) \quad = \quad \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} \times (x-a)^n.$$

Of indien $a = 0$ (dan noemt men de reeks een *McLaurinreeks*):

$$t(z) \quad = \quad \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} \times x^n.$$

Hierbij stelt de factor $f^{(n)}(a)$ de n-de afgeleide van de functie f in het punt a voor.

Voorbeelden van softwarepakketten die op een automatische manier een generating function (wat in essentie een reële functie is) kunnen benaderen door diens Taylorpolynoom zijn `Matlab` [13, 14], `Maple` [11] en `Wolfram Alpha` [28]. In deze thesis maken we gebruik van `Matlab` om de Taylorpolynoom van een generating function te bepalen.

Wanneer we eenmaal de Taylorpolynoom van een generating function van een klasse $\mathcal{A}$ hebben bepaald, volstaat het om de coëfficiënt van de n-de term eruit te bepalen. Indien we een ongelabelde klasse en diens OGF beschouwen, is deze coëfficiënt namelijk gelijk aan de n-de term uit de counting sequence van $\mathcal{A}$. Indien we een gelabelde klasse en diens EGF beschouwen, is deze coëfficiënt, vermenigvuldigd met $n!$ gelijk aan de n-de term uit de counting sequence van $\mathcal{A}$.

Voorbeeld 4.30 *Beschouw de volgende specificatie S voor een ongelabelde klasse $\mathcal{U}$:*

$$S \begin{cases} U \to \text{SEQ}(I) \\ I \to C + H \\ H \to F + G \\ G \to D \times E \\ F \to A \times B \\ A \to \bigcirc_a \\ B \to \bigcirc_b \\ C \to \bigcirc_c \\ D \to \bigcirc_d \\ E \to \bigcirc_e \end{cases}$$

De klasse $\mathcal{U}$ bevat dus sequenties met objecten uit $\mathcal{I}$, waarbij $\mathcal{I}$ de objecten $(\bigcirc_a, \bigcirc_b)$, $\bigcirc_c$ en $(\bigcirc_d, \bigcirc_e)$ bevat.

Stel dat we het aantal objecten van grootte 4 uit $\mathcal{U}$ willen berekenen. We willen m.a.w. $[z^4]U(z)$ berekenen. Laten we eerst de OGF's van alle klassen die beschreven worden door S bepalen m.b.v. de ingevoerde vertaalregels:

$$\begin{aligned} A(z) &= z \\ B(z) &= z \\ C(z) &= z \\ D(z) &= z \\ E(z) &= z \end{aligned}$$

$$\begin{aligned} F(z) &= A(z) \times B(z) \\ &= z^2 \end{aligned}$$

$$\begin{aligned} G(z) &= D(z) \times E(z) \\ &= z^2 \end{aligned}$$

$$\begin{aligned} H(z) &= F(z) + G(z) \\ &= 2z^2 \end{aligned}$$

$$\begin{aligned} I(z) &= C(z) + H(z) \\ &= z + 2z^2 \end{aligned}$$

$$\begin{aligned} U(z) &= \frac{1}{1-(z+2z^2)} \\ &= \frac{1}{1-z-2z^2} \end{aligned}$$

De OGF van $\mathcal{U}$ is een reële functie waar we de coëfficiënt van de vijfde term niet op een triviale manier kunnen bepalen. Daarom berekenen we m.b.v. `Matlab` *de Taylorpolynoom van $U(z)$ en verkrijgen:*

$$U(z) = 1 + z + 3z^2 + 5z^3 + 11z^4 + 21z^5 + \ldots$$

We zien dat de counting sequence van $\mathcal{U}$ gelijk is aan

$$(U_n) = 1, 1, 3, 5, 11, 21, \ldots$$

Hieruit leiden we af dat $\mathcal{U}$ 11 objecten van grootte 4 bevat. ◁

4.2.5 Methode 2: Berekening via wiskundige gelijkheden

In deze sectie beschrijven we een tweede methode voor de berekening van de counting sequence van een constructible gelabelde klasse $\mathcal{A}$. We focussen in deze sectie enkel op het gelabelde universum. Verder veronderstellen we dat de specificatie S die $\mathcal{A}$ beschrijft in STDF staat (zie Sectie 3.3.8).

Omdat S in STDF staat, maakt ze gebruik van slechts (hoogstens) drie gelabelde constructors, namelijk de combinatorische unie, het gelabelde product en de pointing operator. We weten uit Sectie 4.2.1 dat deze drie constructors admissible zijn. We weten dus ook dat de counting sequence van de resulterende klasse per definitie enkel afhangt van de counting sequence van de gebruikte klassen.

In deze sectie leggen we voor elk van de drie vernoemde constructors een wiskundige gelijkheid vast waarmee we de n-de term van de counting sequence van

de resulterende klasse kunnen berekenen o.b.v. de counting sequences van de gebruikte klassen. De tweede methode bestaat er dan in om voor elke constructor van de gegeven specificatie de juiste wiskundige gelijkheid toe te passen. Op die manier kunnen we voor elke nonterminal van de gegeven specificatie de counting sequence berekenen tot en met een bepaalde grootte n.

Combinatorische unie

Theorema 4.31 *Voor de combinatorische unie $\mathcal{A} = \mathcal{B} + \mathcal{C}$ van twee gelabelde klassen $\mathcal{B}$ en $\mathcal{C}$ geldt de volgende gelijkheid voor de counting sequence van $\mathcal{A}$:*

$$A_n = B_n + C_n. \qquad \Diamond$$

Herinner dat de combinatorische unie de disjuncte unie van twee klassen oplevert, zonder dat er iets aan de grootte van de objecten wordt gewijzigd. Hierdoor worden objecten die zowel in $\mathcal{B}$ en $\mathcal{C}$ zitten dubbel geteld. Bijgevolg is het aantal objecten van grootte n uit $\mathcal{A}$ gelijk aan de som van het aantal objecten van grootte n uit $\mathcal{B}$ en het aantal objecten van grootte n uit $\mathcal{C}$.

De standaardunie (zonder het maken van disjuncte kopieën) is geen admissible constructor, want daarvoor wordt de counting sequence als volgt bekomen:

$$\begin{aligned} A_n &= \text{card}(\mathcal{B}_n \cup \mathcal{C}_n) \\ &= \text{card}(\mathcal{B}_n) + \text{card}(\mathcal{C}_n) - \text{card}(\mathcal{B}_n \cap \mathcal{C}_n) \\ &= B_n + C_n - \text{card}(\mathcal{B}_n \cap \mathcal{C}_n). \end{aligned}$$

Hierbij hangt de counting sequence van $\mathcal{A}$ dus ook telkens af van de doorsnede van beide gebruikte subklassen met objecten van grootte n. Deze doorsnede is in essentie een nieuwe combinatorische klasse, die niet gebruikt wordt in de constructie. Bijgevolg is de standaardunie niet admissible.

Gelabeld product

Theorema 4.32 *Voor het gelabeld product $\mathcal{A} = \mathcal{B} \star \mathcal{C}$ van twee gelabelde klassen $\mathcal{B}$ en $\mathcal{C}$ geldt de volgende gelijkheid voor de counting sequence van $\mathcal{A}$:*

$$A_n = \sum_{n_1+n_2=n} \binom{n}{n_1} \times B_{n_1} \times C_{n_2}.$$

Of

$$A_n \;=\; \sum_{k=0}^{n} \binom{n}{k} \times B_k \times C_{n-k}. \qquad \Diamond$$

De intuïtie hierachter is dat een object van grootte n uit $\mathcal{A}$ kan bekomen worden door een object van grootte k (met $k \leq n$) uit $\mathcal{B}$ te combineren met een object van grootte $n-k$ uit $\mathcal{C}$. Samen vormen zij een koppel van grootte n, want $k+n-k = n$.

Verder moeten we ook alle relabellings van dit koppel in rekening brengen. In Sectie 3.3.4 hebben we gezien dat het aantal relabellings van een object van grootte n met één component van grootte k en één van grootte $n-k$ gelijk is aan de *binomiaalcoëfficiënt* van n en k.

Pointing operator

Theorema 4.33 *Voor de pointing $\mathcal{A} = \Theta\mathcal{B}$ van een gelabelde klasse $\mathcal{B}$ geldt de volgende gelijkheid voor de counting sequence van $\mathcal{A}$:*

$$A_n \;=\; n \times B_n. \qquad \Diamond$$

De intuïtie hierachter is dat elk object van grootte n uit $\mathcal{B}$ aanleiding geeft tot n gepointe objecten van deze grootte, omdat elk van de n atomen gepoint kan worden. De gepointe objecten hebben dezelfde grootte als het oorspronkelijke object, omdat er tijdens het pointen niets gewijzigd wordt aan de grootte van een object.

We kunnen ook de omgekeerde weg beschouwen.

Theorema 4.34 *Zij $\mathcal{B}$ een gepointe klasse en $\mathcal{A}$ de klasse die we verkrijgen door $\mathcal{B}$ te depointen, dan geldt de volgende gelijkheid voor de counting sequence van $\mathcal{A}$:*

$$A_n \;=\; \frac{B_n}{n}. \qquad (4.5)$$
$$\Diamond$$

Dit resultaat komt voort uit het feit dat de n gepointe objecten van grootte n, waarvan alleen het neutrale object verschilt dat aangeeft welk atoom gepoint wordt, gedepoint worden naar één object van grootte n.

Hoofdstuk 5

Uniforme random generatie van combinatorische objecten

In dit hoofdstuk wordt een methode uitgelegd om één object van een bepaalde grootte n uit een constructible combinatorische klasse $\mathcal{A}$ random te genereren (afleiden) op een uniforme manier. We focussen in dit hoofdstuk enkel op het gelabelde universum. Verder veronderstellen we dat de specificatie die $\mathcal{A}$ beschrijft in STDF staat én well-defined is. Indien de specificatie nog niet in STDF staat, kan ze hiernaar omgezet worden m.b.v. de methode die werd beschreven in Sectie 3.3.8.

Een naïeve aanpak om één object van grootte n uit een klasse $\mathcal{A}$ uniform random te genereren bestaat erin om eerst alle objecten van grootte n uit $\mathcal{A}$ te genereren en vervolgens hier random één object uit te kiezen. Deze aanpak is echter niet efficiënt omdat het genereren van alle objecten bijzonder tijdrovend kan zijn. Stel bijvoorbeeld dat de klasse $\mathcal{A}$ miljoenen objecten van grootte n bevat. Het genereren van al deze objecten zou immens veel tijd in beslag nemen. Daarom presenteren we in dit hoofdstuk een efficiëntere manier die het random te genereren object stap voor stap opbouwt uit kleinere objecten.

5.1 Inleiding

In Sectie 3.3.8 hebben we gezien dat een specificatie die in STDF staat zes mogelijke vormen van producties kan gebruiken, namelijk

$$\begin{aligned} A &\rightarrow \epsilon \\ A &\rightarrow a \\ A &\rightarrow B + C \end{aligned}$$

$$\begin{aligned} A &\rightarrow B \star C \\ A &\rightarrow \Theta B \\ \Theta A &\rightarrow B \end{aligned}$$

waarbij A, B en C nonterminals zijn, ϵ een neutraal object is en a een atoom. De eerste soort productie beschrijft dus een neutrale klasse en de tweede een atomaire klasse.

Voor elk van deze soorten producties definiëren we een *procedure* die op een uniforme random manier één object uit het resultaat van de constructor van zulk een productie genereert. We noteren de procedure van de productie die bij het nonterminal A hoort met gA. De invoer van een dergelijke procedure is de gewenste grootte n.

De procedures die in dit hoofdstuk bescheven worden, komen voornamelijk uit [22, p. 9 - 11].

Het genereren van een willekeurig object α uit een combinatorische klasse $\mathcal{A}$ gebeurt door, vertrekkende vanaf het startsymbool van de bijhorende specificatie, de opeenvolgende producties te beschouwen. Voor elke constructor die voorkomt in deze producties wordt de overeenkomstige procedure uitgevoerd. Hierbij wordt gebruikgemaakt van de resultaten van de procedures die bij de argumenten van de constructor horen.

Stel bijvoorbeeld dat we een (afgekorte) productie beschouwen van de vorm $\mathcal{A} \star (\mathcal{B} + \mathcal{C})$. Dan wordt eerst de procedure uitgevoerd dat bij de combinatorische unie hoort om een random object uit de klasse $\mathcal{B} + \mathcal{C}$ te genereren. Dit object wordt samen met een object uit de klasse $\mathcal{A}$ gebruikt in de procedure die een random object uit het resultaat van het gelabelde product genereert.

De procedures die in Sectie 5.2 beschreven worden, veronderstellen dat de counting sequences tot en met een bepaalde grootte n van de gebruikte combinatorische klassen reeds berekend zijn. Hiervoor kan gebruikgemaakt worden van de methodes die beschreven werden in Sectie 4.2.4 en 4.2.5.

Er valt op te merken dat de random generatie procedures alleen de *vorm* van de objecten genereert. De labels die de atomen van een object krijgen, negeren we tijdelijk tijdens het opbouwen van het resulterende object. Zo verkrijgen we na afloop een niet-goedgelabeld object α van grootte n, waarvan elk atoom het label 1 draagt. Om uiteindelijk een goedgelabeld object te verkrijgen, genereren we een

random permutatie van het interval $[1 \ldots n]$. Vervolgens kennen we aan elk atoom van α in volgorde één label uit deze random permutatie toe.

Om de vormen van de objecten te creëeren, gebruiken we niet de "gewone" counting sequence waarden, maar de *genormaliseerde* counting sequence waarden. Deze komen namelijk overeen met het aantal ongelabelde tegenhangers van een gelabeld object, waarin alleen de vorm van het object belangrijk is.

Indien A_n de n-de waarde uit de counting sequence van $\mathcal{A}$ voorstelt, dan is de n-de genormaliseerde counting sequence waarde gelijk aan

$$a_n = \frac{A_n}{n!}.$$

De n-de genormaliseerde waarde van een counting sequence (A_n) duiden we aan met de overeenkomstige kleine letter a_n.

Het random genereren van objecten uit een combinatorische klasse gebeurt op een *uniforme* manier. Dit wil zeggen dat de kans dat een willekeurig object van een bepaalde grootte n uit deze klasse gegenereerd wordt even groot is als de kans dat een willekeurig ander object van grootte n uit deze klasse gegenereerd wordt. Hiervoor wordt er verondersteld dat er een methode beschikbaar is die uniform een random reëel getal uit het gesloten interval $[0, 1]$ genereert. De random generatie van objecten is vervolgens volledig afhankelijk van dit getal.

5.2 Random generatie procedures

In deze sectie worden de zes random generatie procedures besproken in het geval van een gelabelde neutrale en atomaire klasse, de combinatorische unie, het gelabeld product en de pointing operator.

Neutrale klasse

Algoritme 3 genereert random het enige object ϵ uit een gelabelde neutrale klasse $\mathcal{E} = \{\epsilon\}$.

De procedure controleert eerst of de invoer gelijk is aan 0. Indien dit zo is, wordt het enige object van grootte 0 van $\mathcal{E}$ teruggegeven. De kans dat $\mathcal{E}$ een object van grootte 0 bevat is m.a.w. gelijk aan 1. Als de invoer niet gelijk is aan 0 wordt

Algorithm 3 Een procedure die uniform random een object uit een gelabelde neutrale klasse genereert.

```
Procedure Random generatie neutrale klasse:
int n = invoer;
if n = 0 then
    return ϵ;
```

er niets teruggeven, omdat de klasse $\mathcal{E}$ geen enkel object bevat met een grootte verschillend van 0.

Atomaire klasse

Algoritme 4 genereert random het enige object ① uit een gelabelde atomaire klasse $\mathcal{Z} = \{①\}$.

Algorithm 4 Een procedure die uniform random een object uit een gelabelde atomaire klasse genereert.

```
Procedure Random generatie atomaire klasse:
int n = invoer;
if n = 1 then
    return ①;
```

Het algoritme controleert eerst of de invoer gelijk is aan 1. Indien dit zo is, wordt het enige object van grootte 1 van $\mathcal{Z}$ teruggegeven. De kans dat $\mathcal{Z}$ een object van grootte 1 bevat is m.a.w. gelijk aan 1. Als de invoer niet gelijk is aan 1 wordt er niets teruggeven, omdat de klasse $\mathcal{Z}$ geen enkel object bevat met een grootte verschillend van 1.

Combinatorische unie

Zij $\mathcal{C} = \mathcal{A} + \mathcal{B}$ de combinatorische unie van twee gelabelde klassen $\mathcal{A}$ en $\mathcal{B}$. Algoritme 5 genereert random een object uit $\mathcal{C}$. Hierbij stellen gA en gB de algoritmes voor die respectievelijk random een object van grootte n uit $\mathcal{A}$ en $\mathcal{B}$ genereren.

De kans dat een object van grootte n van $\mathcal{C} = \mathcal{A} + \mathcal{B}$ uit $\mathcal{A}$ komt is gelijk aan het quotiënt van de n-de genormaliseerde waarde uit de counting sequence van $\mathcal{A}$ en de n-de genormaliseerde waarde uit de counting sequence van $\mathcal{C}$.

Algorithm 5 Een procedure die uniform random een object uit het resultaat van een combinatorische unie genereert.

Procedure Random generatie combinatorische unie:
int n = invoer;
double U = uniform random reëel getal in interval $[0, 1]$;
if $U < (a_n \,/\, c_n)$ **then**
  return gA(n);
else
  return gB(n);

Om de keuze te maken of het te genereren object uit $\mathcal{A}$ gehaald wordt, genereert het algoritme eerst random een uniform reëel getal uit het interval $[0, 1]$. Op basis van de vergelijking tussen dit getal en de kans $a_n \,/\, c_n$ wordt vervolgens een random generatie procedure toegepast op de klasse $\mathcal{A}$ of $\mathcal{B}$.

Gelabeld product

Zij $\mathcal{C} = \mathcal{A} \star \mathcal{B}$ het gelabeld product van twee gelabelde klassen $\mathcal{A}$ en $\mathcal{B}$. Algoritme 6 genereert random een object uit $\mathcal{C}$. Hierbij stellen gA en gB de algoritmes voor die respectievelijk random een object van grootte n uit $\mathcal{A}$ en $\mathcal{B}$ genereren.

Algorithm 6 Een procedure die uniform random een object uit het resultaat van een gelabeld product genereert.

Procedure Random generatie gelabeld product:
int n = invoer; k = 0;
double S = $(a_0 \times b_n) \,/\, c_n$;
double U = uniform random reëel getal in interval $[0, 1]$;
while $U > S$ **do**
  $k = k + 1$;
  $S = S + (a_k \times b_{n-k}) \,/\, c_n$;
return(gA(k), gB(n - k));

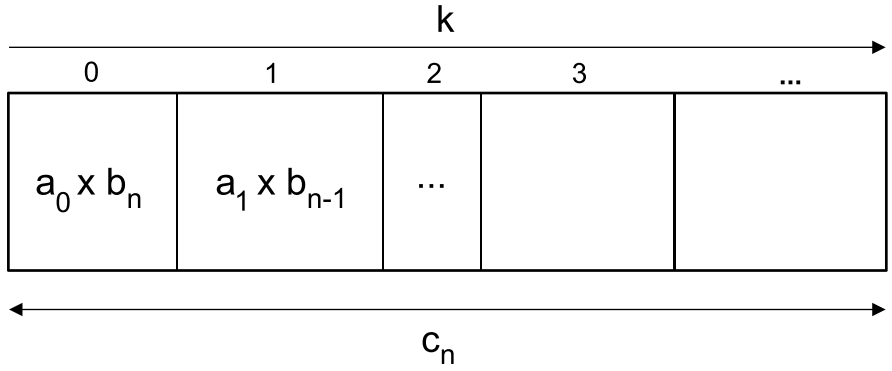
De kans dat een object van grootte n uit $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ een $\mathcal{A}$-component heeft van grootte k en een $\mathcal{B}$-component van grootte $n - k$ is gelijk aan:

$$\frac{a_k \times b_{n-k}}{c_n}.$$

Om een keuze te maken voor de waarde van k genereert het algoritme random een uniform reëel getal uit het interval $[0, 1]$. Het algoritme start vervolgens bij $k = 0$ en incrementeert deze waarde telkens door het getal $(a_k \times b_{n-k})$ / c_n te vergelijken met het uniform random gegenereerde getal totdat de juiste waarde voor k bereikt wordt. Ten slotte wordt een random generatie procedure toegepast op de klasse $\mathcal{A}$ en $\mathcal{B}$ m.b.v. de juiste parameters k en $n - k$.

Het principe dat hier gebruikt wordt om de juiste waarde voor k te vinden wordt geïllustreerd in Figuur 5.1. Merk op dat hierin het delen door c_n weggelaten werd om de eenvoudigere schaal $[0, c_n]$ te verkrijgen.

Het bepalen van de juiste waarde voor k komt in de illustratie overeen met het kiezen van een "hokje". Hiervoor geldt dat hoe groter het "hokje" is, hoe meer kans het maakt om gekozen te worden. In de zin van aantallen van objecten wil dit zeggen dat hoe meer objecten er zijn van grootte k in $\mathcal{A}$ én grootte $n - k$ in $\mathcal{B}$, hoe groter de kans is dat deze waarde gekozen wordt voor k.



Figuur 5.1: *Illustratie van het bepalen van de geschikte waarde voor k in Algoritme 6.*

Pointing operator

Zij $\mathcal{B} = \Theta\mathcal{A}$ de gepointe klasse van een gelabelde klasse $\mathcal{A}$. Algoritme 7 genereert random een object uit $\mathcal{B}$. Hierbij stelt gA het algoritme voor dat random een object van grootte n uit $\mathcal{A}$ genereert. Verder voert de functie $point(\alpha, k)$ de pointing van het object α met het getal k uit. Deze functie accentueert m.a.w. het k-de atoom van α.

Algorithm 7 Een procedure die uniform random een object uit het resultaat van de pointing operator genereert.

Procedure Random generatie pointing operator:
int n = invoer;
double U = uniform random reëel getal in interval $[0, 1]$;
return(point(gA(n), $1 + \lfloor n \times U \rfloor$))

De procedure start met random een object uit $\mathcal{A}$ te genereren. Vervolgens wordt dit object gepoint met een getal d.m.v. de functie $point(\alpha, k)$. Het getal waarmee het object gepoint wordt, wordt op een uniforme manier random bepaald door eerst een random reëel getal uit het interval $[0, 1]$ te bepalen. Vervolgens verkrijgen we door de constructie $1 + \lfloor n \times U \rfloor$ een getal uit het interval $[1, n]$, waarbij elk van de n getallen een even grote kans heeft om gekozen te worden.

Voor de omgekeerde weg, het depointen van een object, kunnen we ook een procedure opstellen. Hiervoor hebben we een functie nodig die de pointing van een eerder gecreëerd gepoint object verwijderd. Deze functie noemen we *erase* en hiervoor geldt dat

$$erase(point(\alpha, k)) \quad = \quad \alpha.$$

Stel dat de klasse $\mathcal{A}$ impliciet gedefinieerd is als $\Theta A = B$. De procedure die random een object uit $\mathcal{A}$ genereert, wordt weergegeven in Algoritme 8. Hierbij moet getest worden of $\mathcal{A}$ een object van grootte 0 bevat. Indien dit zo is, wordt het neutrale object ϵ teruggegeven, omdat een object van grootte 0 geen atomen bevat en dus ook niet kan gedepoint worden.

Algorithm 8 Een procedure die uniform random een object uit de klasse $\mathcal{A}$ genereert indien $\mathcal{A}$ gedefinieerd is als $\Theta\mathcal{A} = \mathcal{B}$.

Procedure Random generatie depointing:
int n = invoer;
if $n = 0 \wedge a_0 \neq 0$ **then**
  return ϵ;
if $n \geq 1$ **then**
  return(erase(gB(n));

Stelling 5.1 *De worst-case tijdscomplexiteit van het uniform random genereren van combinatorische objecten m.b.v. de beschreven procedures is* $O(n^2)$ *aritmetische operaties.*

Voor een bewijs van deze stelling verwijzen we naar [22, p. 11].

Hoofdstuk 6

Implementatie

Samen met het schrijven van deze thesistekst hebben we ook een implementatie gemaakt van het random generatie algoritme. De implementatie werd geprogrammeerd in Java [16] en bundelt de drie algoritmes (well-definednesscontrole, counting sequence berekening en uniforme random generatie) uit de vorige hoofdstukken samen. Bij het implementeren hebben we ons beperkt tot het gelabelde universum.

Als invoer verwacht het programma een grammatica G uit Ω_g die in SPNF staat, samen met een natuurlijk getal n. Het programma controleert eerst of G well-defined is m.b.v. Algoritme 2. Indien dat zo is, wordt G omgezet naar een grammatica G' die in STDF staat. Vervolgens wordt voor elke nonterminal van G' diens counting sequence berekend tot en met het n-de getal. Ten slotte worden de algoritmes uit Hoofdstuk 5 toegepast om één random object van grootte n uit de combinatorische klasse $\mathcal{L}(G)$ te genereren.

6.1 Model

6.1.1 Nonterminals

Om een nonterminal van een grammatica te modelleren hebben we een klasse `NonTerminal` gecreëerd. Een object van dit type bevat o.a. een naam en een tabel van het type `BigInteger` [17] die de counting sequence van de nonterminal voorstelt. We hebben voor het type `BigInteger` gekozen omdat hierbij geen limiet staat op de waarde die ze kan aannemen, i.t.t. een gewone `int`-variabele.

6.1.2 Producties

Om de producties van een grammatica te modelleren, hebben we een abstracte basisklasse `Production` gecreëerd, waarvan alle soorten producties overerven. In een gelabelde grammatica die in SPNF staat, zijn er acht vormen van producties mogelijk, namelijk:

- $U \to \epsilon$
- $U \to a$
- $U \to V$

- $U \to V + W$
- $U \to V \star W$

- $U \to \mathrm{SEQ}(V)$
- $U \to \mathrm{SET}(V)$
- $U \to \mathrm{CYC}(V)$

De eerste drie soorten producties noemen we "simpele" producties. Hiervoor hebben we respectievelijk drie klassen `NeutralObjectProduction`, `AtomProduction` en `NonTerminalProduction` gecreëerd die rechtstreeks overerven van `Production`.

De volgende twee soorten producties zijn binaire producties. Hiervoor hebben we een extra abstractielaag toegevoegd door een abstracte klasse `BinaryConstructorProduction` te creëeren die overerft van `Production`. De klassen `UnionProduction` en `ProductProduction` erven hier vervolgens van over. Een object van het type `BinaryConstructorProduction` bevat twee nonterminals die de argumenten van de constructor voorstellen.

Analoog hebben we voor de laatste drie soorten unaire producties een abstracte klasse `UnaryConstructorProduction` ingevoerd, waar de klassen `SequenceProduction`, `SetProduction` en `CycleProduction` van overerven. Deze klassen bevatten één nonterminal die het enige argument van de constructor voorstelt.

Omdat de invoergrammatica in SPNF staat, hoort er bij elke nonterminal precies één productie. Hierdoor kunnen we elk object van het type `NonTerminal` linken met één object van het type `Production`.

In een grammatica die in STDF staat, kunnen ook producties van de vorm

$$\begin{aligned} U &\rightarrow \Theta V \text{ of} \\ \Theta U &\rightarrow V \end{aligned}$$

voorkomen, waarbij U en V nonterminals zijn. Voor een productie van de vorm $U \rightarrow \Theta V$ hebben we een klasse `PointingProduction` gecreëerd, die overerft van de klasse `UnaryConstructorProduction` omdat de pointing operator een unaire constructor is.

Voor een productie van de vorm $\Theta U \rightarrow V$ maken we gebruik van een extra veld in de klasse `NonTerminal` om aan te geven dat de nonterminal U gepoint is. Vervolgens modelleren we de productie ervan met een object van het type `NonTerminalProduction`.

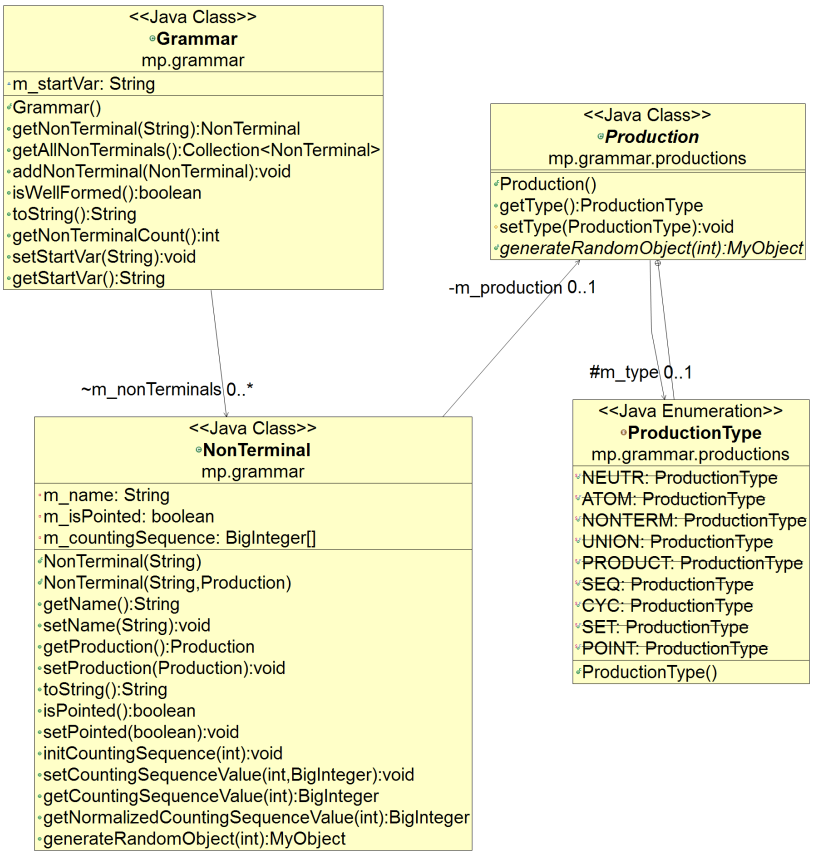
6.1.3 Grammatica's

Een grammatica wordt gemodelleerd door de klasse `Grammar`. Deze bevat een hashmap [18] die de naam van elke nonterminal afbeeldt op een object van het type `NonTerminal`. Op die manier kunnen we alle producties van de grammatica opslaan, omdat elk `NonTerminal`-object één object van het type `Productie` bevat.

Het klassendiagram omtrent de klassen die een nonterminal of een grammatica voorstellen wordt afgebeeld in Figuur 6.1. De klassendiagrammen van de klassen die producties modelleren worden weergegeven in Figuur 6.2, 6.3 en 6.4. Figuur 6.2 geeft de "simpele" producties weer, Figuur 6.3 de binaire en Figuur 6.4 de unaire.

6.1.4 Objecten

Tijdens het random genereren kunnen er drie soorten objecten gecreëerd worden. Ten eerste kan een object van het type `NeutralObjectProduction` één neutraal object creëeren. Een object van het type `AtomProduction` kan één atoom creëeren. De andere producties kunnen ten slotte een tupel creëeren. Een tupel bevat een aantal componenten die op hun beurt een neutraal object, een atoom of opnieuw



Figuur 6.1: *Het klassendiagram van de klassen die een nonterminal of een grammatica voorstellen.*

een tupel kunnen zijn. In het laatste geval beschouwen we m.a.w. geneste tupels.

Om al deze soorten objecten te modelleren hebben we een abstracte basisklasse `MyObject` gecreëerd, waar drie klassen `NeutralObject`, `AtomObject` en `TupleObject` van overerven. Het klassendiagram van de klassen die we hiervoor gebruiken, wordt weergegeven in Figuur 6.5.

6.2 Werking

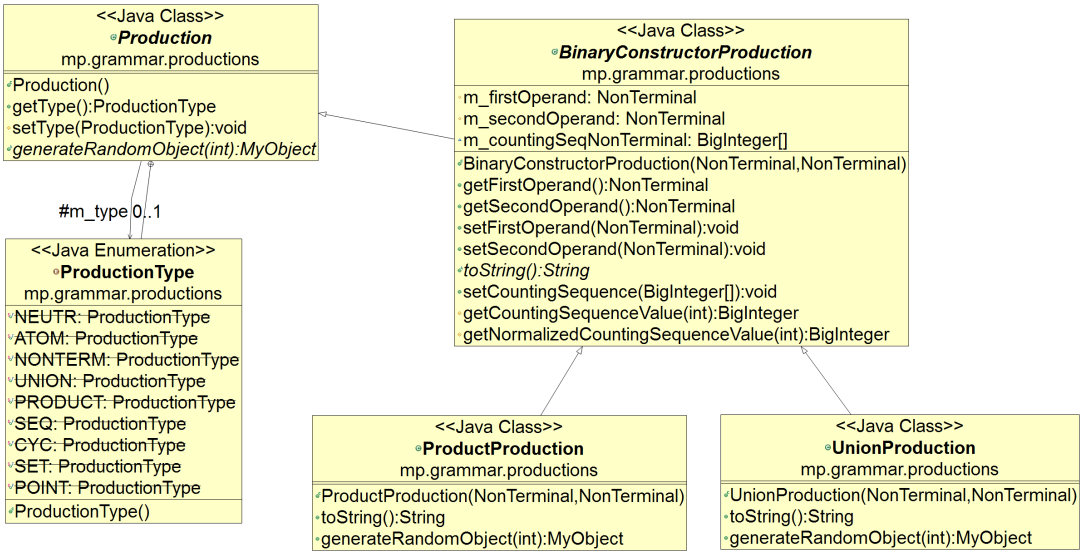
De werking van de implementatie kan, zoals reeds gezegd, ruwweg in vier grote delen opgesplitst worden. De eerste stap bestaat erin om te controleren of de invoergrammatica well-defined is. Dit gebeurt in een object van het type `WellDefinedChecker`.

Figuur 6.2: *Het klassendiagram van de "simpele" producties.*

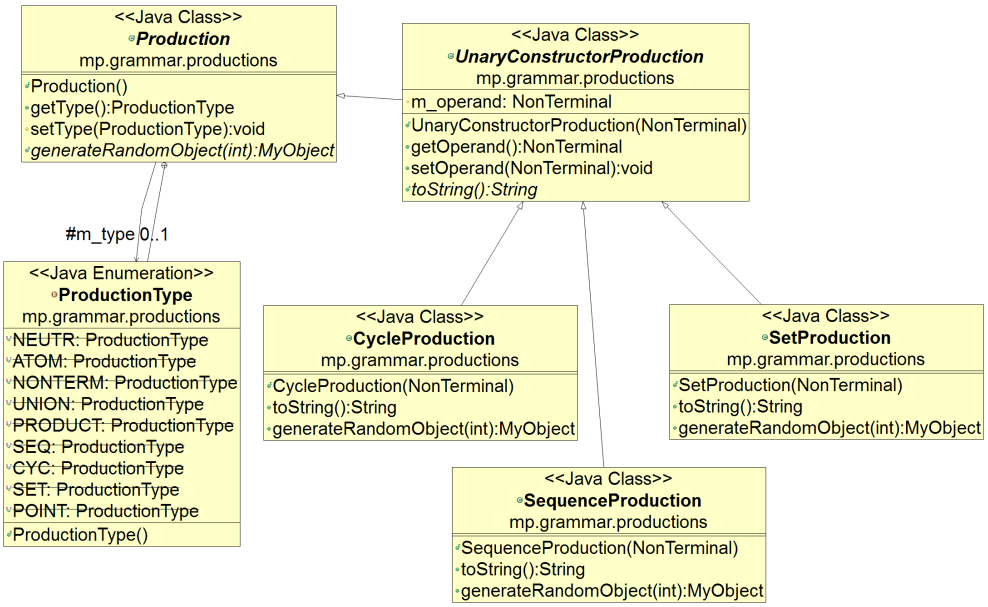
In de volgende stap wordt de invoergrammatica gereduceerd naar een grammatica die in STDF staat. Dit gebeurt in een object van het type `STDFConverter`. Tijdens de derde stap wordt in een object van het type `CountingSequenceCalculator` de counting sequence van elke nonterminal van de STDF-grammatica berekend.

In de laatste stap gebeurt het random genereren zelf. Hiervoor hebben we geen aparte klasse gecreëerd. Het random genereren wordt namelijk uitgevoerd door de verschillende type `Production`-objecten.

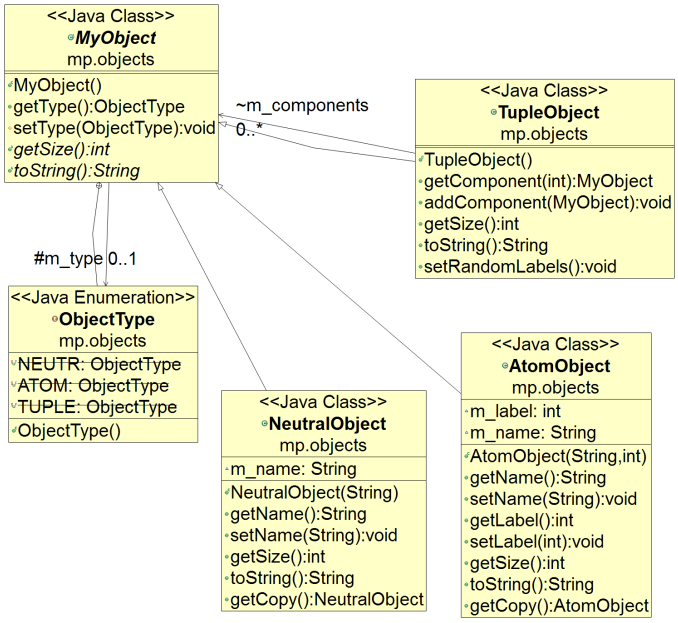
Deze vier stappen samen hebben we gebundeld in één klasse, genaamd `RandomStringGenerator`. Het klassendiagram hiervan wordt weergegeven in Figuur 6.6.



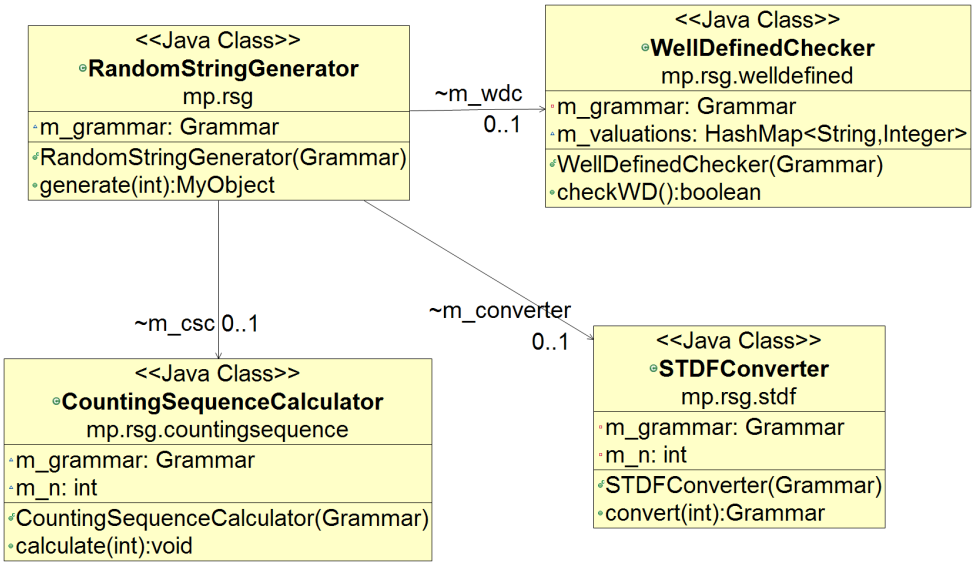
Figuur 6.3: *Het klassendiagram van de binaire producties.*



Figuur 6.4: *Het klassendiagram van de unaire producties.*



Figuur 6.5: *Het klassendiagram van de klassen die een object voorstellen.*



Figuur 6.6: *Het klassendiagram van de klassen die het random generatie algoritme uitvoeren.*

Voorbeeld 6.1 *Als lopend voorbeeld voor deze sectie gebruiken we de gelabelde grammatica G_{pb} uit Voorbeeld 3.62, die we hier herhalen voor de eenvoud:*

$$G_{pb} \begin{cases} G \to Z \star H \\ H \to \mathrm{SEQ}(G) \\ Z \to \textcircled{1} \end{cases}$$

◁

6.2.1 Well-definedness controle

Een object van het type `WellDefinedChecker` voert de drie stappen uit Algoritme 2 uit. Om te beginnen worden de valuaties van de nonterminals uit de invoergrammatica berekend m.b.v. Algoritme 1. Hierbij valt op te merken dat we de valuaties van de nonterminals initialiseren met de waarde -1 i.p.v. ∞, omdat dit in de praktijk niet mogelijk is. Een valuatie van -1 kan vervolgens als "foute waarde" beschouwd worden i.p.v. ∞, omdat een nonterminal geen objecten met een negatieve grootte kan afleiden. Na afloop moet er dus gecontroleerd of er nergens meer een waarde gelijk is aan -1, anders is de grammatica niet well-defined.

Voorbeeld 6.2 *Beschouw de grammatica G_{pb} uit Voorbeeld 6.1. De uiteindelijke waarden in de valuatietabel worden weergegeven in Tabel 6.1. We zien dat de eerste stap uit Algoritme 2 slaagt, omdat er geen enkele valuatie gelijk is aan -1.*

G	1
H	0
Z	1

Tabel 6.1: *De valuaties van de nonterminals uit de grammatica G_{pb} uit Voorbeeld 6.1*

Vervolgens wordt er gecontroleerd of de invoergrammatica gereduceerd is door Definitie 3.50 toe te passen. Voor elke productie die een unaire constructor (sequence, set of cycle) gebruikt, moet er gecontroleerd worden of de valuatie van het argument niet gelijk is aan 0.

Voorbeeld 6.3 *De enige productie van de grammatica G_{pb} die een unaire constructor gebruikt is*

$$H \quad \to \quad \mathrm{SEQ}(G).$$

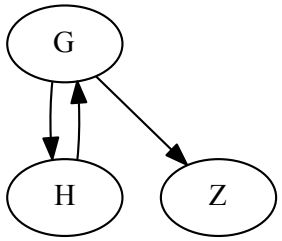
In het vorige voorbeeld hebben we gezien dat de valuatie van het argument G gelijk is aan 1. Bijgevolg kunnen we besluiten dat G_{pb} een gereduceerde grammatica is.$\triangleleft$

Ten slotte wordt de afhankelijkheidsgraaf van de grammatica bepaald. Hieruit worden alle cycles bepaald en voor elke cycle wordt gecontroleerd of de conditie uit Lemma 3.51 geldt. Indien deze conditie geldt voor minstens één cycle, is de grammatica niet well-defined.

Voorbeeld 6.4 *De afhankelijkheidsgraaf van G_{pb} wordt afgebeeld in Figuur 6.7. We zien dat de graaf twee cycles bevat, namelijk (G, H, G) en (H, G, H). In beide cycles wordt de productgebaseerde productie*

$$G \rightarrow Z \star H$$ $\triangleleft$

gebruikt. De nonterminal hiervan die niet in beide cycles zit, is Z. In Voorbeeld 6.2 hebben we gezien dat de valuatie hiervan gelijk is aan 1. Bijgevolg worden er niet oneindig veel objecten van een bepaalde grootte gecreëerd door beide cycles. We zien m.a.w. dat de conditie uit Lemma 3.51 niet geldt. Hieruit kunnen we besluiten dat de grammatica G_{pb} well-defined is.



Figuur 6.7: *De afhankelijkheidsgraaf van de grammatica G_{pb} uit Voorbeeld 6.1.*

6.2.2 Reductie naar STDF en berekening van de counting sequence

Tijdens de tweede stap van het algoritme wordt de invoergrammatica geconverteerd naar een grammatica die in STDF staat, zoals beschreven werd in Sectie 3.3.8.

Hiermee is het namelijk eenvoudiger om de counting sequence te berekenen.

Een productie van de vorm $B \to \text{SEQ}(A)$ wordt omgezet naar

$$\begin{aligned} B &\to E + C \\ E &\to \epsilon \\ C &\to A \star B \end{aligned}$$

Voor de n-de waarde van de counting sequence van $\mathcal{B}$, $\mathcal{E}$ en $\mathcal{C}$ weten we uit Sectie 4.2.5 dat

$$\begin{aligned} B_n &= E_n + C_n \\ (E_n) &= 1, 0, 0, 0, \ldots \\ C_n &= \sum_{k=0}^{n} \binom{n}{k} \times A_k \times B_{n-k} \end{aligned}$$

Een productie van de vorm $B \to \text{SET}(A)$ wordt omgezet naar

$$\begin{aligned} \Theta B &\to D \\ D &\to B \star C \\ C &\to \Theta A \end{aligned}$$

Uit Sectie 4.2.5 weten we voor de n-de waarde van de counting sequence van Θ $\mathcal{B}$, $\mathcal{B}$, $\mathcal{D}$ en $\mathcal{C}$ dat

$$\begin{aligned} \Theta B_n &= D_n \\ B_n &= \frac{\Theta B_n}{n} \\ D_n &= \sum_{k=0}^{n} \binom{n}{k} \times B_k \times C_{n-k} \\ C_n &= n \times A_n \end{aligned}$$

Een productie van de vorm $B \to \text{CYC}(A)$ wordt omgezet naar

$$\begin{aligned} \Theta B &\to G \\ G &\to C \star F \\ C &\to E + D \\ D &\to A \star C \end{aligned}$$

$$\begin{aligned} E &\rightarrow \epsilon \\ F &\rightarrow \Theta A \end{aligned}$$

Hierbij zijn de derde en vierde productie de STDF-producties van $C \rightarrow \mathrm{SEQ}(A)$. Hiervoor kunnen we de bovenstaande formules gebruiken. Hieruit en uit Sectie 4.2.5 volgt dus:

$$\begin{aligned} \Theta B_n &= G_n \\ B_n &= \frac{\Theta B_n}{n} \\ G_n &= \sum_{k=0}^{n} \binom{n}{k} \times C_k \times F_{n-k} \\ C_n &= E_n + D_n \\ D_n &= \sum_{k=0}^{n} \binom{n}{k} \times A_k \times C_{n-k} \\ (E_n) &= 1, 0, 0, 0, \ldots \\ F_n &= n \times A_n \end{aligned}$$

De 0-de tot en met de n-de waarde uit de counting sequence van elke nonterminal van de STDF-grammatica wordt geïnitialiseerd op 0. Vervolgens starten we met de 0-de waarde uit de counting sequence van elke nonterminal van de grammatica te berekenen m.b.v. de bovenstaande formules. Daarna doen we hetzelfde voor de 1-ste waarde en alle opeenvolgende waarden tot we n bereikt hebben. Na afloop hebben we dus de eerste n waarden uit de counting sequence van elke nonterminal berekend.

Hierbij is het belangrijk om op te merken dat in het geval van een recursieve grammatica de n-de waarde van een bepaalde nonterminal kan afhangen van de n-de waarde van een andere nonterminal en omgekeerd. Hierdoor is het mogelijk dat de n-de waarden uit de counting sequences nog niet vastliggen na één iteratie over de nonterminals. Daarom moeten we blijven itereren over de nonterminals en telkens de bijhorende formule berekenen totdat er in een hele iteratie niets meer verandert.

Voorbeeld 6.5 *Beschouw opnieuw de gelabelde well-defined grammatica G_{pb} uit Voorbeeld 6.1. De grammatica G'_{pb} die we verkrijgen na de reductie naar STDF is de volgende:*

$$G'_{pb} \begin{cases} G \to Z \star H \\ H \to C + E \\ C \to G \star H \\ E \to \epsilon \\ Z \to \textcircled{1} \end{cases}$$

De formules die moeten berekend worden om de n-de waarde uit de counting sequence van elke nonterminal van G'_{pb} zijn de volgende:

$$\begin{aligned} G_n &= \sum_{k=0}^{n} \binom{n}{k} \times Z_k \times H_{n-k} \\ H_n &= C_n + E_n \\ C_n &= \sum_{k=0}^{n} \binom{n}{k} \times G_k \times H_{n-k} \\ (E_n) &= 1, 0, 0, 0, \ldots \\ (Z_n) &= 0, 1, 0, 0, \ldots \end{aligned}$$ ◁

Stel dat we de counting sequences willen berekenen tot en met grootte 3. We merkten dat de 0-de waarde van de counting sequences vastligt na drie iteraties over de nonterminals. De 1-ste waarde ligt vast na vier iteraties, de 2-de na drie iteraties en de 3-de ook na drie iteraties. De uiteindelijke waarden van de counting sequences worden weergegeven in Tabel 6.2.

n	0	1	2	3
(G_n)	0	1	2	12
(H_n)	1	1	4	30
(C_n)	0	1	4	30
(E_n)	1	0	0	0
(Z_n)	0	1	0	0

Tabel 6.2: *De counting sequences tot en met grootte 3 van de nonterminals van de grammatica G'_{pb} uit Voorbeeld 6.5*

6.2.3 Uniforme random generatie

De laatste stap van het algoritme bestaat uit het effectief genereren van objecten. Hiervoor hebben we de procedures uit Sectie 5.2 geïmplementeerd. Al deze procedures leveren één object van het type `MyObject` op.

Tijdens het opbouwen van het resulterende object geven we elk atoom tijdelijk het label 1. Hierdoor verkrijgen we na afloop dus een niet-goedgelabeld object van grootte n. Om het uiteindelijke object goedgelabeld te maken, genereren we een random permutatie van het interval $[1 \ldots n]$ en kennen we elk getal in volgorde toe aan één atoom van het gecreëerde object.

Om het uniforme random reële getal U te berekenen, gebruiken we de ingebouwde functie $Math.random()$ [19] van Java.

Voorbeeld 6.6 *Stel dat we een object ω van grootte 3 uit G'_{pb} uniform random willen genereren. De stappen die ons programma volgt zijn de volgende. We starten met de startvariabele van G'_{pb}, namelijk G. De productie die bij deze nonterminal hoort is*

$$G \rightarrow Z \star H$$

Hiervoor wordt de procedure gebruikt die bij het gelabelde product hoort. Voor de waarde van k is er in dit geval maar één mogelijkheid, ongeacht de waarde van U, omdat het "hokje" van $Z_1 H_2$ 100% groot is t.o.v. het "hokje" G_n. Hierdoor neemt k de waarde 1 aan. Om een object van grootte 3 uit $\mathcal{G}$ te bekomen, hebben we m.a.w. enkel de keuze om een object van grootte 1 uit $\mathcal{Z}$ te combineren met een object van grootte 2 uit $\mathcal{H}$ om een tupel t_1 met twee componenten te bekomen.

De klasse $\mathcal{Z}$ is een atomaire klasse, waardoor het enige atoom (1) uit $\mathcal{Z}$ 100% kans heeft om gekozen te worden. De eerste component van t_1 is m.a.w. het atoom (1).

De tweede component van t_1 is een object van grootte 2 uit $\mathcal{H}$. De productie die bij het nonterminal H hoort, is

$$H \rightarrow C + E$$

In Voorbeeld 6.5 hebben we echter gezien dat E_2 gelijk is aan 0. Waardoor er 100% kans is dat er een object van grootte 2 uit $\mathcal{C}$ wordt gegenereerd, ongeacht de waarde

van U. De productie die bij de nonterminal C hoort, is

$$C \rightarrow G \star H$$

De 0-de waarde uit de counting sequence van $\mathcal{G}$ is gelijk aan 0, hierdoor heeft het "hokje" van G_0H_2 een relatieve grootte van 0% t.o.v. het hokje H_2. Zowel de genormaliseerde waarde van G_1 als die van H_1 is gelijk aan 1. De genormaliseerde waarde van C_2 is gelijk aan

$$\begin{aligned} c_2 &= \frac{4}{2!} \\ &= 2. \end{aligned}$$

Bijgevolg heeft het "hokje" van G_1H_1 50% kans om gekozen te worden. Het "hokje" G_2H_0 heeft exact dezelfde kans. Stel dat voor U de waarde 0.17142 bepaald wordt, dan vervolgen we de random generatie dus door een object van grootte 1 uit $\mathcal{G}$ te koppelen met een object van grootte 1 uit $\mathcal{H}$ tot het tupel t_2.

Om een object uit $\mathcal{G}_1$ te genereren, voeren we de procedure uit die bij het gelabelde product hoort, want de productie van de nonterminal G is

$$G \rightarrow Z \star H$$

Opnieuw is er maar één mogelijk voor de waarde van k omdat het "hokje" Z_1H_0 100% kans heeft om gekozen te worden, ongeacht de waarde van U. Hierdoor wordt opnieuw het atoom $\textcircled{1}$ uit de atomaire klasse $\mathcal{Z}$ gegenereerd als eerste component van t_2.

Om een object uit $\mathcal{H}_1$ te genereren, volgen we de productie van H,

$$H \rightarrow C + E.$$

Omdat $\mathcal{E}_1$ gelijk is aan 0, komt dit object uit $\mathcal{C}_1$ (de kans dat het "hokje" van C gekozen wordt, is namelijk opnieuw 100%). Om het object uit $\mathcal{C}_1$ te genereren volgen we de productie

$$C \rightarrow G \star H.$$

Voor de waarde van k is er in dit geval slechts één mogelijkheid, namelijk 1. Dit komt doordat het "hokje" van G_1H_0 100% kans heeft om gekozen te worden. Vervolgens wordt er dus een object uit $\mathcal{G}_1$ gekoppeld met een object uit $\mathcal{H}_0$ tot het tuple t_3.

Om het object uit $\mathcal{G}_1$ te genereren, moeten we de procedure die bij het gelabelde product

$$G \;\rightarrow\; Z \star H$$

uitvoeren. Het getal k neemt hier sowieso de waarde 1 aan omdat het "hokje" Z_1H_0 100% kans heeft om gekozen te worden. De procedure van de atomaire klasse $\mathcal{Z}$ genereert vervolgens opnieuw het atoom $\textcircled{1}$. De eerste component van t_3 is m.a.w. dit atoom. De tweede component komt, zoals eerder vermeld, uit $\mathcal{H}_0$. We volgen m.a.w. de productie

$$H \;\rightarrow\; C + E.$$

Uit Voorbeeld 6.5 weten we dat C_0 gelijk is aan 0 en E_0 aan 1. Er is dus een kans van 100% dat er een object van grootte 0 uit $\mathcal{E}$ gegenereerd wordt. Omdat $\mathcal{E}$ een neutrale klasse is, levert dit het neutrale object ϵ op. Hier stopt vervolgens de uitvoering van het algoritme. Op dit moment weten we dat

$$\begin{aligned} t_1 &= (\textcircled{1}, t_2) \\ t_2 &= (\textcircled{1}, t_3) \\ t_3 &= (\textcircled{1}, \epsilon) \end{aligned}$$

Als we deze resultaten samenvoegen, verkrijgen we uiteindelijk het object ω dat het random generatie algoritme genereert. Dit object is dus gelijk aan $(\textcircled{1}, (\textcircled{1}, (\textcircled{1}, \epsilon)))$. Merk nogmaals op dat dit object niet goedgelabeld is. Om het object goedgelabeld te maken, genereren we een random permutatie van de verzameling $\{1, 2, 3\}$ en kennen we de labels eruit in volgorde toe aan de atomen van ω.

Stel dat de random permutatie $\{2, 1, 3\}$ gegenereerd wordt, dan is ω uiteindelijk gelijk aan $(\textcircled{2}_a, (\textcircled{1}_a, (\textcircled{3}_a, \epsilon)))$. ◁

Hoofdstuk 7

Experimenten

In dit hoofdstuk beschrijven we twee experimenten die we hebben uitgevoerd met onze implementatie, samen met de resultaten ervan.

7.1 Performantie

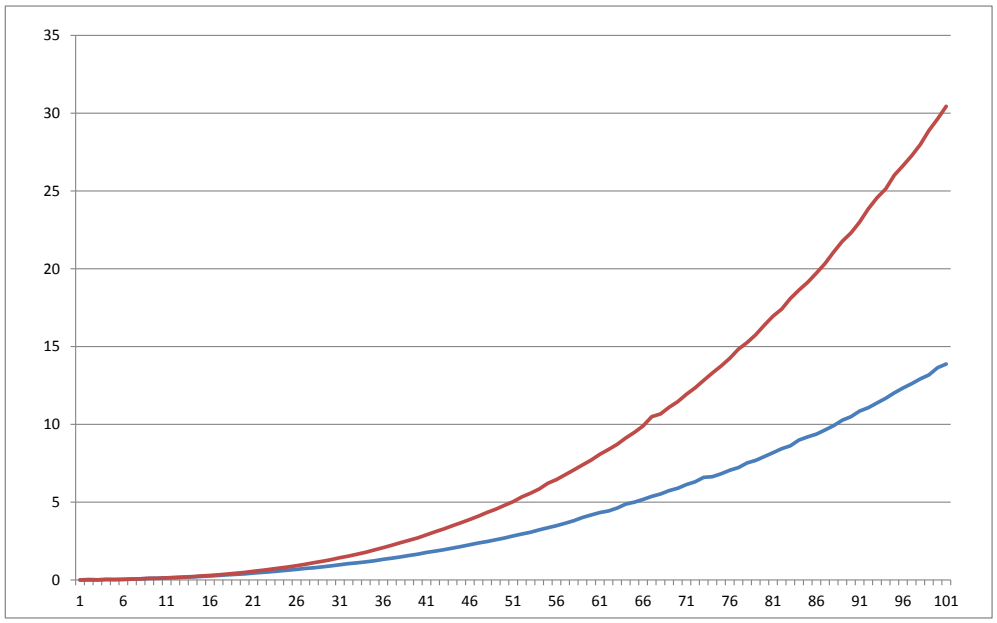
Een eerste experiment dat we hebben uitgevoerd met onze implementatie is een performantietest. Ten eerste hebben we getest hoe snel de uitvoeringstijd stijgt wanneer we telkens grotere object genereren. Voorts hebben we getest of er een verschil is in de uitvoeringstijd wanneer we een object uit een grammatica met weinig nonterminals genereren t.o.v. wanneer we een object genereren uit een "grotere" grammatica.

De grammatica's die we voor dit experiment gebruikt hebben, zijn enerzijds de grammatica G'_{pb} uit Voorbeeld 6.5. Deze grammatica bevat vijf nonterminals. De andere grammatica die we gebruikt hebben is een grammatica uit [5, Sectie 3] die alle *bigeconnecteerde outerplanaire grafen* [7, 8] genereert. Deze grammatica hebben we omgezet naar SPNF en bevat in deze vorm 54 nonterminals.

Voor dit experiment hebben we voor elke waarde van n, gaande van 0 t.e.m. 100 telkens 10000 objecten van grootte n uit beide grammatica's gegenereerd. Vervolgens hebben we voor elke waarde van n de totale uitvoeringstijd in milliseconden gedeeld door 10000 om zó de gemiddelde uitvoeringstijd per object van grootte n te verkrijgen. We nemen enkel de tijd van het random genereren in acht.

De verhouding tussen de grootte n en de uitvoeringstijd wordt weergegeven in de grafiek in Figuur 7.1. Op de horizontale as staan de invoergroottes n en op de

verticale as de gemiddelde uitvoeringstijd in milliseconden van ons programma om één object van grootte n te genereren. De blauwe curve hoort bij de grammatica G'_{pb}. De rode curve hoort bij de andere grammatica.



Figuur 7.1: *De verhoudingen tussen de uitvoeringstijd van het random generatie algoritme en de invoergrootte n.*

Op de grafiek zien we duidelijk de kwadratische tijdscomplexiteit van het random generatie algoritme. Verder zien we ook een significant verschil in uitvoeringstijd tussen de grammatica met 54 nonterminals en die met 5 nonterminals. Dit komt doordat er bij de grammatica met 54 nonterminals meer producties moeten verwerkt worden per object dat gegenereerd wordt. Dit zorgt ervoor dat de afleidingsbomen van de gegenereerde objecten aanzienlijk groter is dan in het geval van de grammatica met 5 nonterminals. Bijgevolg duurt het dan ook langer om deze afleidingsbomen te creëeren.

We kunnen m.a.w. besluiten dat de uitvoeringstijd van het random generatie algoritme enerzijds afhangt van de invoergrootte n en anderzijds van het aantal gebruikte nonterminals van de beschouwde grammatica.

7.2 Ambiguïteit

Het tweede experiment dat we hebben uitgevoerd is een vergelijking van de uitvoering van de implementatie met een niet-ambigue grammatica t.o.v. de uitvoering met een ambigue grammatica. Hiervoor hebben we de volgende contextvrije stringgrammatica's G_{h1} en G_{h2} gebruikt:

$$G_{h1} \begin{cases} H \to M + E \\ M \to K \star L \\ K \to HO \star H \\ L \to HT \star H \\ HO \to (\\ HT \to) \\ E \to \epsilon \end{cases}$$

$$G_{h2} \begin{cases} H \to S + O \\ S \to R \star HT \\ R \to HO \star H \\ O \to P + Q \\ P \to H \star H \\ Q \to HO \star HT \\ HO \to (\\ HT \to) \end{cases}$$

Hierin zijn `'('` en `')'` twee atomen van grootte 1 en ϵ een neutraal object dat de lege string voorstelt. De grammatica G_{h1} is de combinatorische SPNF-versie van de contextvrije grammatica

$$H \quad \to \quad (H)H + \epsilon$$

en G_{h2} is de combinatorische SPNF-versie van de contextvrije grammatica

$$H \quad \to \quad (H) + HH + ()$$

De grammatica's G_{h1} en G_{h2} beschrijven allebei dezelfde formele taal $\mathcal{LH}$, namelijk de taal die bestaat uit alle goedgeneste haakjesparen. Voorbeelden van strings uit $\mathcal{LH}$ zijn `'()'`, `'(())'` en `'(()()())'`.

De grammatica G_{h2} is echter ambigu, omdat bijvoorbeeld de string van grootte 6 `'()()()'` op twee manieren kan afgeleid worden uit G_{h2}. De twee afleidingsbomen hiervan worden weergegeven in Figuur 7.2 en 7.3. De grammatica G_{h1} daarentegen is niet ambigu [6, p. 160].

In totaal bevat $\mathcal{LH}$ vijf strings van grootte 6, namelijk:

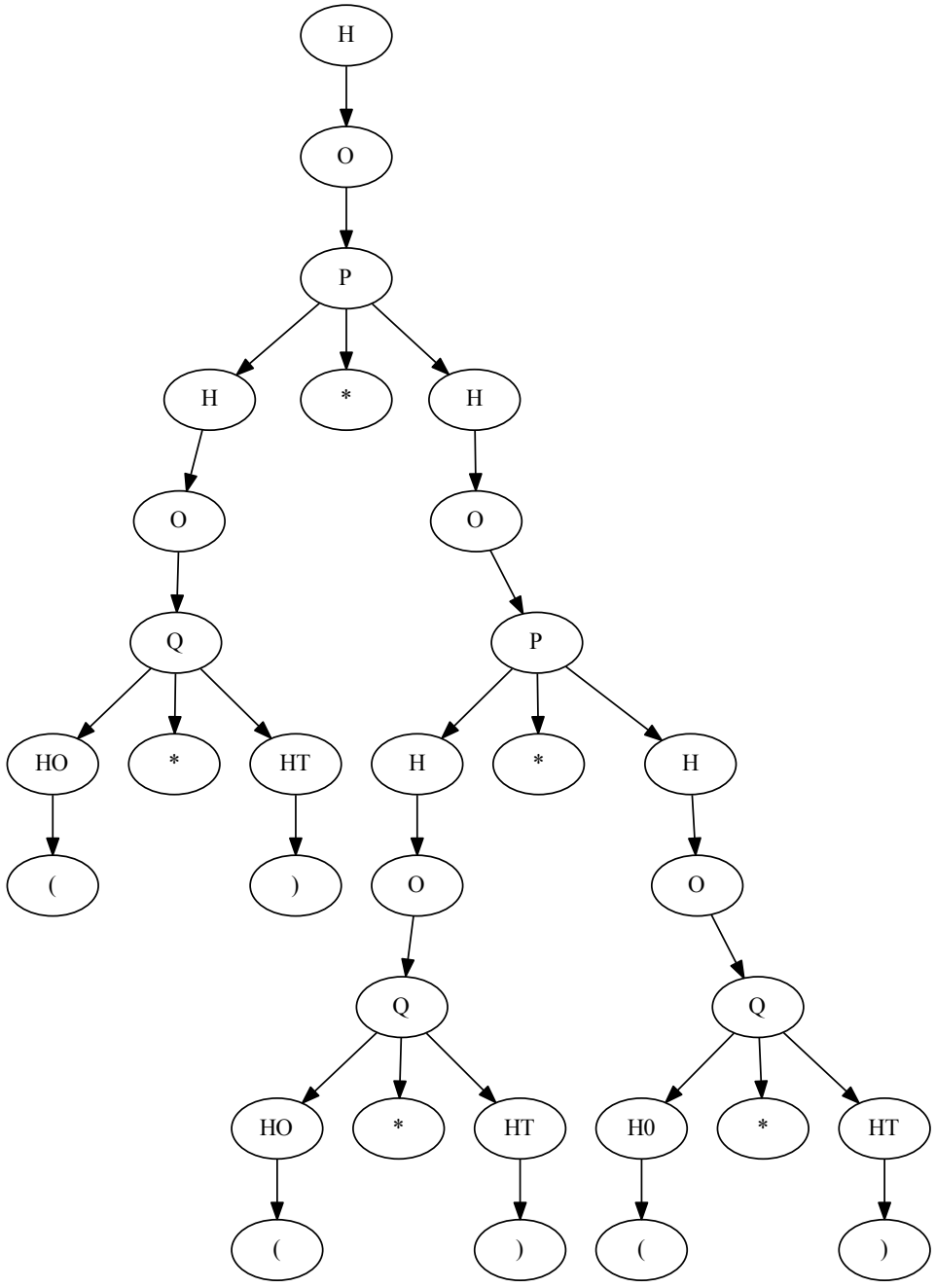
- `((()))`
- `()(())`
- `()()()`
- `(()())`
- `(())()`

Het tweede experiment bestaat erin om 10 miljoen keer een string van grootte 6 uit zowel G_{h1} als G_{h2} te genereren en vervolgens na te kijken hoeveel keer elke string gegenereerd werd. In Figuur 7.4 staat voor elk van de vijf bovenstaande strings grafisch weergegeven hoeveel keer hij gegenereerd werd uit G_{h1}. In Figuur 7.5 staan de resultaten voor de ambigue grammatica G_{h2}. Tussen haakjes staat telkens de relatieve hoeveelheid dat een string gegenereerd werd t.o.v. het totaal aantal gegenereerde strings (10 miljoen).

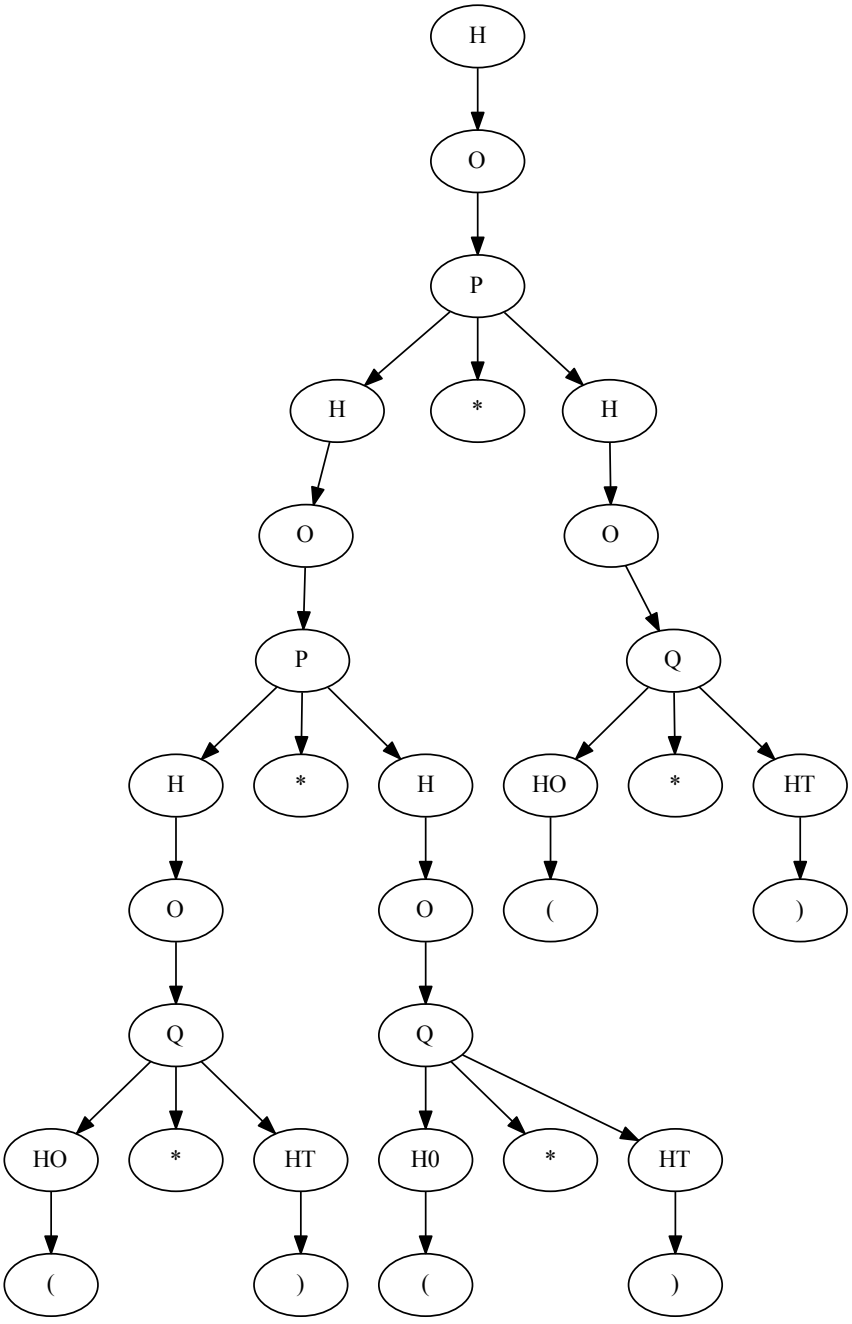
We zien dat in het geval van de grammatica G_{h1} elke string ongeveer evenveel keer gegenereerd wordt. Hieruit kunnen we besluiten dat het random generatie algoritme de objecten op een uniforme manier genereert.

In het geval van de ambigue grammatica G_{h2} zien we echter dat de string `'()()()'` dubbel zo veel keer gegenereerd wordt als alle andere strings. Dit komt doordat het random generatie algoritme, zoals reeds gezegd, in feite de afleidingsbomen van objecten creëert. We hebben hierboven gezien dat de string `'()()()'` twee afleidingsbomen heeft en dus dubbel zo veel als alle andere strings. Daarom is de kans dat deze string gegenereerd wordt dan ook dubbel zo groot als de kans dat eender welke andere string van grootte 6 gegenereerd wordt.

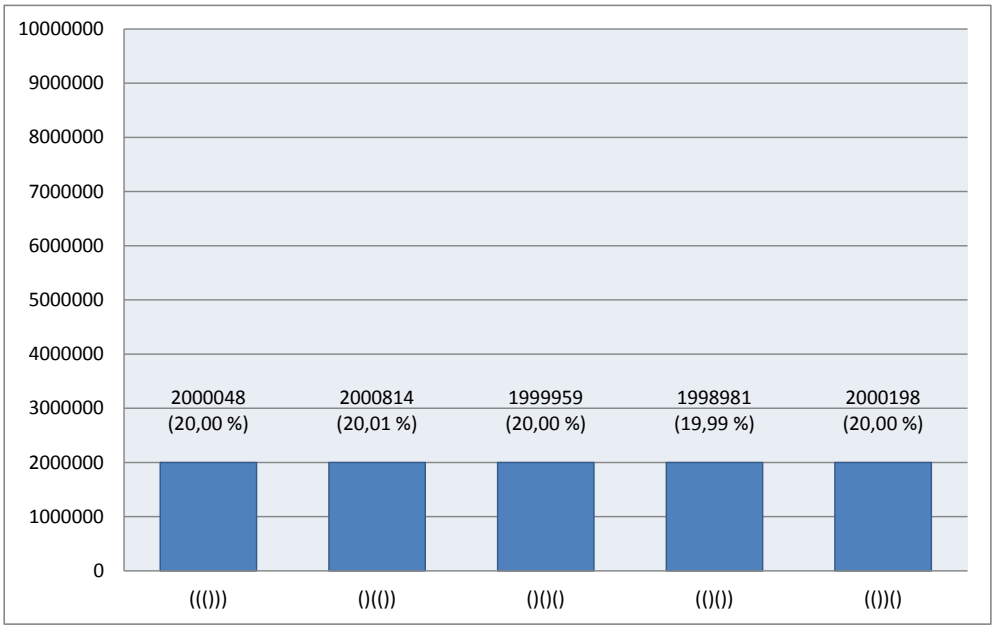
In essentie is het aantal keer dat een object, dat meerdere afleidingsbomen heeft, gegenereerd wordt een veelvoud van het aantal keer dat de andere objecten gegenereerd worden. Dit veelvoud komt overeen met de relatieve hoeveelheid afleidingsbomen dat een object, dat meerdere keren gegenereerd wordt, heeft t.o.v. de andere objecten.



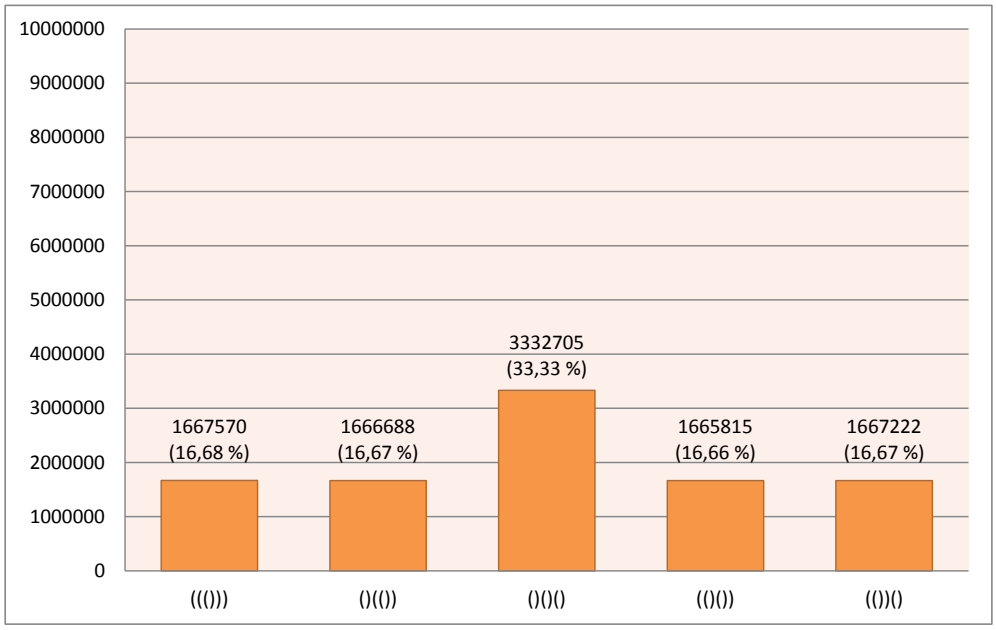
Figuur 7.2: *De eerste afleidingsboom van de afleiding van de string '()()()' uit de grammatica G_{h2}.*



Figuur 7.3: *De tweede afleidingsboom van de afleiding van de string '()()()' uit de grammatica G_{h2}.*



Figuur 7.4: *De strings van grootte 6 die uit G_{h1} kunnen gegenereerd worden en het aantal keer dat ze effectief gegenereerd worden.*



Figuur 7.5: *De strings van grootte 6 die uit G_{h2} kunnen gegenereerd worden en het aantal keer dat ze effectief gegenereerd worden.*

M.b.v. het random generatie algoritme lijkt het er dus op dat we kunnen bepalen of een grammatica ambigu is door gewoonweg een aantal strings van een bepaalde grootte te genereren dat groot genoeg is om de relatieve hoeveelheden met elkaar te kunnen vergelijken. De moeilijkheid is echter om de grootte van de te genereren objecten te bepalen omdat hier bij een willekeurige grammatica geen bovengrens op bestaat. Moest deze bovengrens wel bepaald kunnen worden, is het beslisbaar of een grammatica ambigu is. In Sectie 2.5.3 hebben we echter gezien dat dit probleem niet beslisbaar is.

Hoofdstuk 8

Conclusie

Het doel van deze thesis was het onderzoeken hoe strings uit een formele taal op een uniforme random manier gegenereerd kunnen worden. Hiervoor hebben we gebruikgemaakt van en tak van de wiskunde die de "combinatoriek" genoemd wordt.

Eerst hebben we combinatorische klassen grondig bestudeerd om het nut ervan in te zien. Vervolgens hebben we uitgelegd hoe zulke klassen formeel beschreven kunnen worden a.d.h.v. specificaties, wat in feite grammatica's zijn. Deze specificaties maken gebruik van basisconstructors die op basis van elementaire bouwstenen nieuwe klassen van objecten creëeren.

Hierbij hebben we een onderscheid gemaakt tussen ongelabelde en gelabelde combinatorische klassen. Beide soorten klassen worden op een verschillende manier beschreven. Voorts gebeurt het tellen van objecten uit deze soorten klassen ook op een andere manier.

Daarna hebben we de counting sequences en de generating functions van combinatorische klassen bestudeerd. Deze worden gebruikt bij het tellen van het aantal objecten van een bepaalde grootte uit een combinatorische klasse. De generating functions zijn in feite gereduceerde representaties van combinatorische klassen als we enkel deze aantallen beschouwen. Omdat deze generating functions nauw samenhangen met combinatorische klassen hebben we het verband ertussen dan ook grondig bestudeerd. Zo hebben we een aantal wiskundige technieken beschreven die gebruikt worden om de specificatie van een combinatorische klasse te vertalen naar de bijhorende generating function.

Vervolgens hebben we een methode beschreven die toelaat om het aantal objecten van een bepaalde grootte n uit een combinatorische klasse te berekenen, vertrekkende van de bijhorende specificatie die in SPNF staat. Deze methode heeft een

worst-case tijdscomplexiteit van $O(|N| \times n^2)$, waarbij N het aantal nonterminals van de gebruikte specificatie is. Deze methode telt in feite de afleidingsbomen van de objecten i.p.v. de objecten zelf. Daarom zullen er problemen opduiken bij het tellen van objecten die afgeleid worden uit een ambigue specificatie.

Daarna hebben we een methode beschreven om een combinatorisch object van grootte n uit een combinatorische klasse op een uniforme random te genereren m.b.v. de bijhorende specificatie die naar STDF is omgezet. Voor elke constructor die kan voorkomen in deze specificatie hebben we een procedure opgesteld die op een uniforme random manier één object uit het resultaat van deze constructor genereert. Deze procedures worden vervolgens hiërarchisch gecombineerd om uiteindelijk één object uit de beschouwde combinatorische klasse te genereren. Deze random generatie methode heeft een worst-case tijdscomplexiteit van $O(n^2)$.

Random genereratie steunt op het berekenen van kansen, die berekend worden op basis van aantallen van objecten van een bepaalde grootte. We zagen dus dat de beide beschreven methoden (tellen en random generatie) nauw samenhangen.

Omdat een gelabelde combinatorische klasse geschikt is om een formele taal voor te stellen, hebben we de beschreven methodes voor het tellen en random genereren kunnen toepassen op formele talen. Het generieke random generatie framework hebben we m.a.w. toegepast om het specificieke probleem van het uniform random genereren van een string aan te pakken.

Ten slotte hebben we onze implementatie van het random generatie algoritme besproken en hiermee een aantal experimenten uitgevoerd. Ten eerste hebben we de performantie van de implementatie bestudeerd. Hieruit volgde duidelijk dat het random generatie algoritme een tijdscomplexiteit heeft van $O(n^2)$. Verder hebben we bestudeerd of de random generatie effectief op een uniforme manier gebeurt. Dit blijkt zo te zijn voor niet-ambigue grammatica's. Elk object van een bepaalde grootte heeft namelijk evenveel kans om gegenereerd te worden. Er treedt echter een probleem op wanneer we een ambigue gramamtica beschouwen. Hierbij zagen we dat het aantal keer dat bepaalde objecten gegenereerd werden een veelvoud is van het aantal keer dat andere objecten van dezelfde grootte gegenereerd werden. Omdat het random generatie algoritme in feite de afleidingsbomen van de beschouwde objecten creëert, komt dit veelvoud overeen met de relatieve hoeveelheid afleidingsbomen dat een object, dat meerdere keren kan afgeleid worden, heeft t.o.v. de andere objecten.

Bibliografie

[1] Alan Tucker. *Applied Combinatorics*. John Wiley and Sons, 2nd edition, 1984.

[2] D. B. Arnold and M. Ronan Sleep. Uniform Random Generation of Balanced Parenthesis Strings. *ACM Trans. Program. Lang. Syst.*, 2(1):122–128, 1980.

[3] Bruce McKenzie. Generating Strings at Random from a Context Free Grammar, October 1997.

[4] Noam Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In *Computer programming and formal systems*, pages 118–161. North-Holland Publishing, Amsterdam, 1963.

[5] Cristophe Costa Floréncio, Jan Ramon, Jonny Daenen, Jan Van den Bussche, and Dries Van Dyck. Context-free graph grammars as a language-bias mechanism for graph pattern mining. In: Blockeel, H., Borgwardt, K., Yan, X. (eds.) 7th International Workshop on Mining and Learning with Graphs, Extended Abstracts (2009).

[6] Eric Lehman, F. Thomson Leighton, and Albert R. Meyer. Mathematics for Computer Science, 2013. revised Thursday 10th January, 2013.

[7] Eric W. Weisstein. Biconnected Graph - From MathWorld – A Wolfram Web Resource.

[8] Eric W. Weisstein. Outplanar Graph - From MathWorld – A Wolfram Web Resource.

[9] Eric W. Weisstein. Taylor Series - From MathWorld – A Wolfram Web Resource.

[10] D. W. Jordan and P. Smith. *Mathematical Techniques*. Oxford university press, 4th edition, 2008.

[11] Maplesoft. Maple - The Essential Tool for Mathematics and Modeling. http://www.maplesoft.com/products/Maple/.

[12] Marc Gyssens. Eindige automaten, 2008. cursus Universiteit Hasselt.

[13] MathWorks. Matlab - The Language of Technical Computing. http://www.mathworks.com/products/matlab/.

[14] MathWorks. Taylor series expansion. http://www.mathworks.com/help/symbolic/taylor.html.

[15] Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2006.

[16] Oracle. Java SE Technical Documentation. http://docs.oracle.com/javase/.

[17] Oracle. Java SE Technical Documentation – Class BigInteger. http://docs.oracle.com/javase/1.4.2/docs/api/java/math/BigInteger.html.

[18] Oracle. Java SE Technical Documentation – Class HashMap. http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html.

[19] Oracle. Java SE Technical Documentation – Class Math. http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#random().

[20] Paul Zimmermann. *Séries génératrices et analyse automatique d'algorithmes*. PhD thesis, L'école polytechniqe Palaiseau, 1991.

[21] Philippe Flajolet, Bruno Salvy, and Paul Zimmermann. Automatic Average-Case Analysis of Algorithms. *Theor. Comput. Sci.*, 79(1):37–109, 1991.

[22] Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A Calculus of Random Generation. In *ESA*, pages 169–180, 1993.

[23] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, electronic edition, 2009.

[24] Richard P. Stanley. Enumerative Combinatorics Volume 1 (version of 15 July 2011). 2nd edition.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3th edition, 2009.

[26] Timothy Hickey and Jacques Cohen. Uniform Random Generation of Strings in a Context-Free Language. *SIAM J. Comput.*, 12(4):645–655, 1983.

[27] Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A Quasi-Polynomial-Time Algorithm for Sampling Words from a Context-Free Language. *Inf. Comput.*, 134(1):59–74, 1997.

[28] Wolfram Alpha LLC. WolframAlpha - computational knowledge engine. `http://www.wolframalpha.com`.

[29] Yinhuo Zhang. Verzamelingenleer, 2008. cursus Universiteit Hasselt.

Appendices

Bijlage A

Bewijzen vertaalregels

In dit hoofdstuk bewijzen we een aantal theorema's uit Sectie 4.2.2.

A.1 Ongelabelde basisconstructors

A.1.1 Cartesisch product

Bewijs: De formule uit Theorema 4.14 kan afgeleid worden a.d.h.v. formule (3.1) en (4.2) [23, p. 28]:

$$\begin{aligned}
A(z) &= \sum_{\alpha\in\mathcal{A}} z^{|\alpha|} \\
&= \sum_{(\beta,\gamma)\in(\mathcal{B}\times\mathcal{C})} z^{|\beta|+|\gamma|} \\
&= \sum_{(\beta,\gamma)\in(\mathcal{B}\times\mathcal{C})} z^{|\beta|}\times z^{|\gamma|} \\
&= \sum_{\beta\in\mathcal{B}}\left(\sum_{\gamma\in\mathcal{C}}\left(z^{|\beta|}\times z^{|\gamma|}\right)\right) \\
&= \sum_{\beta\in\mathcal{B}}\left(z^{|\beta|}\times\sum_{\gamma\in\mathcal{C}} z^{|\gamma|}\right) \\
&= \left(\sum_{\beta\in\mathcal{B}} z^{|\beta|}\right)\times\left(\sum_{\gamma\in\mathcal{C}} z^{|\gamma|}\right) \\
&= B(z)\times C(z).
\end{aligned}$$

□

A.1.2 Combinatorische unie

Bewijs: Omdat de combinatorische unie gedefinieerd is als een disjuncte unie volgt de formule uit Theorema 4.15 rechtstreeks uit formule (3.3) en (4.2) [23, p. 28]:

$$\begin{aligned} A(z) &= \sum_{\alpha \in \mathcal{A}} z^{|\alpha|_{\mathcal{A}}} \\ &= \sum_{\alpha \in \mathcal{B}} z^{|\alpha|_{\mathcal{B}}} + \sum_{\alpha \in \mathcal{C}} z^{|\alpha|_{\mathcal{C}}} \\ &= B(z) + C(z) \end{aligned}$$ □

A.1.3 Sequence

Bewijs: De formule uit Theorema 4.17 volgt uit de constructie van de combinatorische unie en het cartesisch product. In Definitie 3.11 werd de ongelabelde sequence-constructor namelijk gedefinieerd als volgt (we negeren hier het gebruik van de triviale equivalentierelatie omdat deze geen rol speelt bij het tellen van objecten):

$$\mathcal{A} = \mathrm{SEQ}(\mathcal{B}) \quad = \quad \{\epsilon\} + \mathcal{B} + (\mathcal{B} \times \mathcal{B}) + (\mathcal{B} \times \mathcal{B} \times \mathcal{B}) + \ldots$$

We zien vervolgens a.d.h.v. Theorema 4.15 en 4.14 dat de OGF van $\mathcal{A}$ gelijk is aan:

$$\begin{aligned} A(z) &= 1 + B(z) + B(z)^2 + B(z)^3 + \ldots \\ &= \frac{1}{1 - B(z)} \end{aligned}$$

waarbij de oneindige geometrische som convergeert in de zin van formele machtreeksen, omdat $[z^0]B(z) = 0$ per aanname [23, p. 28]. □

A.1.4 Powerset

Bewijs: Zij $\mathcal{A} = \mathrm{PSET}(\mathcal{B})$ en $\mathcal{B}$ is eindig (voor het oneindige geval verwijzen we naar [23, p. 29]). Dan geldt het volgende combinatorisch isomorfisme [23, p. 28]:

$$\mathcal{A} = \mathrm{PSET}(\mathcal{B}) \quad \cong \quad \prod_{\beta \in \mathcal{B}} (\{\epsilon\} + \{\beta\})$$

waarbij ϵ een neutraal object is. Dit isomorfisme kan intuïtief opgevat worden als volgt: bij het creëeren van een willekeurige deelverzameling van een bepaalde klasse $\mathcal{B}$ moeten we voor elk object $\beta \in \mathcal{B}$ kiezen of het in deze deelverzameling zit of niet. Als we een object niet kiezen, kunnen we het eigenlijk vervangen door het neutrale object ϵ. Vandaar dus de combinatorische unie $(\{\epsilon\} + \{\beta\})$.

Met deze combinatorisch isomorfe klasse is het eenvoudiger om de OGF van $\mathcal{A}$ af te leiden, omdat nu enkel gebruik wordt gemaakt van de combinatorische unie en het cartesisch product. Uit Theorema 4.15 en 4.14 volgt dan:

$$A(z) = \prod_{\beta \in \mathcal{B}} (1 + z^{|\beta|})$$

waarbij $1 + z^{|\beta|}$ de OGF van $\{\epsilon\} + \{\beta\}$ is wegens formule (4.2) en Theorema 4.12. Deze vergelijking kunnen we herschrijven zodat we telkens elke subklasse met objecten van grootte n apart beschouwen (de subklasse $\mathcal{B}_0$ moet echter niet beschouwd worden omdat deze per aanname leeg is):

$$\begin{aligned} A(z) &= \prod_{n=1}^{\infty} \left(\prod_{\beta \in \mathcal{B}_n} (1 + z^{|\beta|}) \right) \\ &= \prod_{n=1}^{\infty} \left(\prod_{\beta \in \mathcal{B}_n} (1 + z^{n}) \right). \end{aligned}$$

Het middelste product wordt voor elke subklasse $\mathcal{B}_n$ evenveel keer uitgevoerd als er elementen zijn in zo'n subklasse (namelijk B_n). We krijgen daarom:

$$A(z) = \prod_{n=1}^{\infty} (1 + z^n)^{B_n}.$$

Dit kunnen we nog verder uitwerken door gebruik te maken van de *exp-log- transformatie* $A(z) = \exp(\log(A(z)))$ (in deze Sectie wordt het logaritme met grondtal e gebruikt):

$$A(z) = \exp\left(\log\left(\prod_{n=1}^{\infty} (1 + z^n)^{B_n} \right) \right).$$

Volgens de rekenregels van logaritmen krijgen we dan:

$$\begin{aligned} A(z) &= \exp\left(\sum_{n=1}^{\infty} \log(1+z^n)^{B_n}\right) \\ &= \exp\left(\sum_{n=1}^{\infty} B_n \times \log(1+z^n)\right). \end{aligned}$$

Vervolgens kunnen we het logaritme uitwerken m.b.v. de gelijkheid: $\log(1+u) = \frac{u}{1} - \frac{u^2}{2} + \frac{u^3}{3} - \ldots$:

$$A(z) = \exp\left(\sum_{n=1}^{\infty} B_n \times \sum_{k=1}^{\infty} (-1)^{k-1} \times \frac{z^{nk}}{k}\right).$$

Dit kunnen we verder uitwerken als volgt:

$$\begin{aligned} A(z) &= \exp\left(\sum_{n=1}^{\infty}\left(\sum_{k=1}^{\infty} \frac{B_n \times z^{nk} \times (-1)^{k-1}}{k}\right)\right) \\ &= \exp\left(\sum_{k=1}^{\infty}\left(\sum_{n=1}^{\infty} \frac{B_n \times z^{nk} \times (-1)^{k-1}}{k}\right)\right) \\ &= \exp\left(\sum_{k=1}^{\infty}\left(\frac{(-1)^{k-1}}{k} \times \sum_{n=1}^{\infty} B_n \times (z^k)^n\right)\right). \end{aligned}$$

Als we een aantal waarden voor k invullen in deze formule, dan krijgen we:

$$\begin{aligned} A(z) = \exp\Bigg(&\frac{(-1)^0}{1} \times \sum_{n=1}^{\infty} B_n z^n + \frac{(-1)^1}{2} \times \sum_{n=1}^{\infty} B_n (z^2)^n + \\ &\frac{(-1)^2}{3} \times \sum_{n=1}^{\infty} B_n (z^3)^n + \ldots\Bigg). \end{aligned}$$

Dit resultaat kan volgens formule (4.1) verder verkort geschreven worden als volgt:

$$A(z) = \exp\left(B(z) - \frac{B(z^2)}{2} + \frac{B(z^3)}{3} - \ldots\right).$$

In een algemene vorm staat hier m.a.w. de volgende formule:

$$A(z) \quad = \quad \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} \times B(z^k)\right). \qquad \square$$

A.1.5 Multiset

Bewijs: In het geval van de multiset-constructor kunnen we nagaan dat het volgende combinatorisch isomorfisme geldt [23, p. 29]:

$$\mathcal{A} = \text{MSET}(\mathcal{B}) \quad \cong \quad \prod_{\beta \in \mathcal{B}} \text{SEQ}(\{\beta\}). \tag{A.1}$$

De intuïtie achter dit isomorfisme is gelijkaardig aan dat van de powerset-constructor. Alleen moeten we in dit geval voor elk object $\beta \in \mathcal{B}$ alle mogelijke sequenties ervan creëeren en vervolgens alle bekomen sequenties combineren m.b.v. het cartesisch product. Hierbij is het belangrijk dat de elementen van elke sequentie samenblijven (en dus niet gemengd worden) in het cartesisch product om te voorkomen dat er dubbele objecten gecreëerd worden.

Met behulp van het combinatorisch isomorfisme in formule (A.1) kunnen we gebruikmaken van Theorema 4.14 en 4.17 om de OGF van de multiset-constructor af te leiden:

$$A(z) \quad = \quad \prod_{\beta \in \mathcal{B}} (1 - z^{|\beta|})^{-1}.$$

De rest van het bewijs verloopt volledig analoog met het bewijs van de powerset-constructor. $\square$

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Uniforme Random Generatie van Strings

Richting: **master in de informatica-databases**
Jaar: **2013**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vandeput, Raf

Datum: **13/06/2013**