Hoofdstuk 2

Iteratie, Recursie en Inductie

▷ SCHAUM §1.8: Mathematical Induction, ook §11.3

▷ SCHAUM §3.6: Recursively Defined Functions

Er zijn slechts enkele passages in SCHAUM aan het belangrijke begrip recursie gewijd. We geven in dit hoofdstuk een aantal voorbeelden.

2.1 Fibonacci getallen

▷ SCHAUM p.54 Fibonacci Sequence

▷ mathworld.wolfram.com/FibonacciNumber.html

De reeks F_n —gedefinieerd door $F_0 = 0$, $F_1 = 1$, en $F_n = F_{n-1} + F_{n-2}$ voor $n \geq 2$— heeft als eerste termen $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$ en staat bekend als de reeks van Fibonacci. Deze reeks is zo beroemd dat er een tijdschrift naar genoemd is. In de Informatica komt de reeks voor bij de analyse van (de complexiteit van) algoritmen, in de Biologie bijvoorbeeld bij de plaatsing van bloemblaadjes.

Een *iteratieve* manier om Fibonacci getallen uit te rekenen wordt gegeven door de volgende C++ functie (uit de oplossingen van het college Programmeermethoden).

```
int fibo (int n) {     // n-de Fibonacci-getal (n > 0 geheel)
  int fiboN, fiboN1, fiboN2;  // misschien is type long beter
  int termnummer;
  if ( n <= 2 )
    return 1;  // eerste en tweede Fibonacci-getal zijn 1
  else {
    fiboN1 = 1; fiboN2 = 1;
    for (termnummer = 3; termnummer <= n; termnummer++) {
      fiboN = fiboN1 + fiboN2; // som van de vorige twee,
      fiboN2 = fiboN1;         // en doorschuiven maar
      fiboN1 = fiboN;
    }  // for
```

```
    return fiboN;
  }  // if
}  // fibo
```

Er bestaat ook een *recursieve* methode om de Fibonacci getallen uit te rekenen. De methode is nogal voor de hand liggend, omdat de definitie precies gevolgd wordt. Echter, de methode is inefficiënt omdat veel waarden herhaald berekend worden. Een C++ functie ziet er bijvoorbeeld als volgt uit.

```
long fiborec (int n) {
// bereken n-de Fibonacci-getal recursief; niet handig
  if ( n <= 1 )
    return n;
  else
    return fiborec (n-1) + fiborec (n-2);
}  // fiborec
```

Een *expliciete formule* voor de getallen van Fibonacci wordt gegeven door de formule van Binet: $F_n = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right\}$. Omdat $|\frac{1-\sqrt{5}}{2}| < 1$ wordt die bijdrage aan de waarde kleiner voor grotere n, en vormt de eerste term een goede benadering voor F_n.

Het getal $\phi = \frac{1+\sqrt{5}}{2}$ uit de formule staat bekend als de *gulden snede*, en is een oplossing van de vergelijking $x^2 - x - 1 = 0$. In het algemeen is de vergelijking $x^2 - Ax - B$ geassocieerd aan de recurrente betrekking $f(n+2) = Af(n+1) + Bf(n)$. Deze vormt de basis voor het vinden van een expliciete formule. Dit valt buiten de collegestof.

Er bestaan veel relaties tussen de Fibonacci getallen onderling, maar ook met andere reeksen. Voorbeelden van gelijkheden:

1 $\sum_{k=1}^{n} F_k = F_{n+2} - 1$

2 $\sum_{k=1}^{n} F_k^2 = F_n F_{n+1}$

3 $F_{2n} = F_{n+1}^2 - F_{n-1}^2$, $F_{2n+1} = F_{n+1}^2 + F_n^2$

4 $F_{2n} = \sum_{k=0}^{n} \binom{n}{k} F_k$

Dit soort eigenschappen lenen zich nogal eens voor een *inductief bewijs*, omdat de getallen recursief gedefinieerd zijn. Als voorbeeld bewijzen we de *formule van Cassini*.

2.1 Lemma. $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$ $(n \geq 1)$

Bewijs. Met volledige inductie.
basis. $n = 1$: $F_0F_2 - F_1^2 = 0 \cdot 1 - 1^2 = -1 = (-1)^1$.
$n = 2$: $F_1F_3 - F_2^2 = 1 \cdot 2 - 1^2 = 1 = (-1)^2$.
inductiestap. Voor $n \geq 2$ tonen we de formule voor $n+1$ aan. In onze oplossing vervangen we F_{n+2} en één F_{n+1} uit het kwadraat door de formule van de Fibonacci reeks. $F_nF_{n+2} - F_{n+1}^2 = F_n(F_{n+1} + F_n) - (F_n + F_{n-1})F_{n+1} = F_nF_{n+1} + F_n^2 - F_nF_{n+1} - F_{n-1}F_{n+1} = -(F_{n-1}F_{n+1} - F_n^2)$. Volgens de *inductieveronderstelling* is dit gelijk aan $-(-1)^n = (-1)^{n+1}$. □

Ook de eerste twee gegeven formules hierboven hebben een bewijs met inductie. De andere twee lijken efficiënter aan te tonen met andere methoden (zie Opgaven).

2.2 Recursieve Definitie van Talen

MU Puzzle. In het boek *Gödel, Escher, Bach: an eternal golden braid* geeft Douglas Hofstadter een inmiddels beroemde puzzel. Als voorbeeld van een 'formeel systeem' geeft hij de taal L van strings over $\{M, I, U\}$ die gevonden worden volgens een aantal regels.

Allereerst is MI een element van L. Verder kunnen nieuwe strings gevonden worden uit oude op vier manieren. Als een string op I eindigt mogen we er een U achter zetten. Als een string met M begint mogen we het stuk erachter verdubbelen. Als een string drie I's achter elkaar heeft mogen we deze vervangen door U. Als een string twee opeenvolgende U's heeft mogen we deze verwijderen.

De puzzel is nu om te bepalen of MU op deze manier gemaakt kan worden.

- $MI \in L$
- als $xI \in L$ dan $xIU \in L$
  als $Mx \in L$ dan $Mxx \in L$
  als $xIIIy \in L$ dan $xUy \in L$
  als $xUUy \in L$ dan $xy \in L$
- L bevat geen andere elementen

We passen de regels op systematische manier toe. MI levert MIU en MII. MIU levert $MIUIU$. MII levert $MIIU$ en $MIIII = MI^4$. $MIUIU$ levert $MIUIUIUIU$. $MIIU$ levert $MIIUIIU$. MI^4 levert MI^4U, MI^8, MUI en MIU. Dat laatste woord hadden we reeds eerder gevonden.

Alle strings uit L lijken met een M te beginnen. Dat is te bewijzen met behulp van *inductie naar de opbouw van L*.

2.2 Lemma. *Elk woord in L begint met een M.*

Bewijs. Met inductie naar de opbouw van L.

Basis. Het gegeven woord voldoet aan het lemma. Immers MI begint met de letter M.

Inductiestap. Als een woord in de taal aan het lemma voldoet, dan voldoen ook alle woorden die uit dat woord geconstrueerd worden. We moeten daarvoor vier gevallen nagaan, overeenkomstig de inductieve regels van L.

Als xI met een M begint, moet kennelijk x met een M beginnen, en dus xIU ook.
Het woord Mxx begint altijd met een M.
Voor $xIIIy$ en $xUUy$ geldt dezelfde redenatie als bij xI. Als het woord met een M begint, dan begint x met een M, en dus ook xUy respectievelijk xy. □

Het antwoord op de vraag van Hofstadter volgt uit een wat ingewikkeldere eigenschap van de woorden uit L. Een eigenschap die misschien niet voor de hand ligt, maar die zich net zo makkelijk laat bewijzen.

2.3 Lemma. *Elk woord in L heeft een aantal letters I dat geen drievoud is.*

Bewijs. Met inductie naar de opbouw van L.

Basis. Het gegeven woord voldoet aan het lemma. Immers MI heeft één letter I.

Inductiestap. Als een woord in de taal aan het lemma voldoet, dan voldoen ook alle woorden die daaruit geconstrueerd worden. We moeten daarvoor vier gevallen nagaan, overeenkomstig de inductieve regels van L.

Het woord xIU heeft evenveel I's als xI, als dit laatste aantal geen drievoud is, dan dat van het nieuwe woord ook niet.
Het woord Mxx heeft tweemaal zovel I's als Mx. Dit aantal kan geen drievoud worden.
Het aantal I's in xUy is drie minder dan in $xIIIy$ en kan ook geen drievoud zijn als dat voor $xIIIy$ ook niet het geval is.
Voor $xUUy$ geldt dezelfde redenatie als bij xI. □

Expressies. ▷ SCHAUM *Algebraic expressions and Polish notation*, p.238.

Expressies zijn er in soorten en maten. We hebben reeds expressies gezien in de verzamelingeleer (met operatoren $\cap$, $\cup$ en ^{c}), kennen expressies voor getallen (met $+$ en $\cdot$ bijvoorbeeld), Boolese expressies voor logische uitdrukkingen, en in Hoofdstuk 4.1 leren we *reguliere expressies* kennen die (formele) talen vastleggen met operatoren $\cap$, $\cdot$ en $*$.

Het is belangrijk om bij expressies de vorm (*syntaxis*) en de betekenis (*semantiek*) uit elkaar te houden. Zo is '$3/(4-4)$' qua vorm een keurige rekenkundige expressie maar is de betekenis ongedefinieerd, terwijl de betekenis van een expressie als '$1+2+3+4$' duidelijk is, terwijl de vorm verduidelijking nodig heeft door het plaatsen van extra haakjes als in '$(1+2)+(3+4)$' of het maken van associatie afspraken (de richting waarin de operatoren worden toegepast).

In het algemeen worden expressies opgebouwd uit basis-ingrediënten, de *constanten*, en *operatoren* om ze samen te stellen.

Als alle operatoren binair zijn (dwz. met twee argumenten) gebruiken we meestal expressies in infix-notatie, de operator staat tussen zijn argumenten (operanden). Haakjes dienen om de onderdelen te groeperen.

- Elke constante is een expressie.
- Als x en y expressies zijn, en @ een (binaire) operator, dan is ook $(x@y)$ een expressie.

We krijgen zo *volledig gehaakte* expressies, waarvan in de praktijk veel haakjes niet worden geschreven. Men laat met de buitenste haakjes weg, en de haakjes die volgen uit voorrangsregels en associatieregels. Voorangsregels bepalen in welke volgorde verschillende operatoren ge-evalueerd moeten worden. Associatieregels geven aan of bij een reeks van operatoren van gelijke prioriteit we eerst de linker of de rechter gaan evalueren. Meestal betekent de expressie '$1+2*3+4$' dan '$((1+(2*3))+4)$' omdat $*$ de hoogste prioriteit heeft, en $+$ vervolgens links-associatief is.

Alternatief is de *Poolse notatie*, of prefix notatie. Hier staat de operator vóór al zijn argumenten. Deze notatie werkt voor operatoren van verschillende ariteit, en heeft geen haakjes nodig en geen verdere regels om dubbelzinnigheid te voorkomen.

- Elke constante is een expressie.
- Als f een n-aire operator is, en $x_1,\ldots,x_n$ expressies, dan is $fx_1\cdots x_n$ een expressie.

Bij *omgekeerde Poolse notatie*, of postfix expressie, staat de operator ná zijn argumenten.

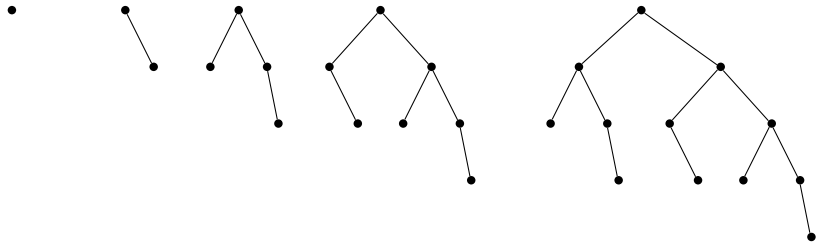
Bij elke expressie hoort een boom (en omgekeerd). De boom bij een constante is een enkele knoop, gelabeld met die constante. De boom bij '$fx_1\cdots x_n$' is recursief gedefinieerd als de boom met wortel een knoop met label f en als subbomen de bomen behorend bij de expressies $x_1,\ldots,x_n$. In ▷ SCHAUM blz. 238 staat weliswaar een voorbeeld, maar geen (recursieve) definitie.

2.3 AVL bomen.

▷ SCHAUM Ch. 10 voor binaire bomen.

Een *AVL boom* is een binaire (zoek-)boom waarvan in elke knoop de linker en rechter deelboom maximaal één in hoogte verschillen. Om de efficiëntie van deze datastructuur te bepalen is het noodzakelijk om de relatie tussen de hoogte van zo'n boom en het aantal knopen te kennen. Voor de 'ongunstigste' AVL boom (kleinste aantal knopen bij een gegeven hoogte) geldt dat in elke knoop de deelbomen zelf ook ongunstig zijn, en één in hoogte verschillen.

We definiëren deze bomen recursief als volgt. T_0 is de boom zonder knopen, T_1 is de binaire boom met één knoop. Voor $n \geq 2$ is T_n de binaire boom waarvan de wortel deelbomen T_{n-2} en T_{n-1} heeft.



Deze bomen leveren ons de Fibonacci getallen weer op.

2.4 Lemma. *T_n heeft hoogte n en bevat $F_{n+2}-1$ knopen.*

Bewijs. Beide beweringen worden met inductie bewezen.

Voor de hoogte is gezond verstand voldoende; we maken ons er snel van af. De hoogte van T_0 en T_1 is respectievelijk 0 en 1. De hoogte van boom T_n is één meer dan (het maximum van) de hoogtes van T_{n-2} en T_{n-1} omdat dit de deelbomen van de wortel zijn, en dus gelijk aan $n = (n-1)+1$.

Zij t_n het aantal knopen van T_n. Er geldt natuurlijk dat $t_0 = 0$ en $t_1 = 1$. Omdat $T-n$ uit een wortel en de subbomen T_{n-2} en T_{n-1} bestaat, volgt de recursieve relatie $t_n = 1+t_{n-2}+t_{n-1}$ voor $n \geq 2$. Dit lijkt op de formule van Fibonacci.

We laten met inductie zien dat $t_n = F_{n+2}-1$. Omdat T_0 en T_1 als basis gevallen gedefinieerd zijn starten we onze inductie weer met twee basis gevallen.

basis. $t_0 = 0 = 1-1 = F_2-1$; $t_1 = 1 = 2-1 = F_3-1$. ok!

inductiestap. Neem aan dat de bewering klopt tot en met n. We tonen daarmee de bewering voor $n+1$ aan. Gebruik de recursieve vergelijkingen voor zowel t_n als F_n. $t_{n+1} = 1+t_n+t_{n-1}$. Dit is volgens de inductieveronderstelling gelijk aan $1+(F_{n+2}-1)+(F_{n+1}-1) = (F_{n+2}+F_{n+1})-1 = F_{n+3}-1$. Hetgeen bewezen moest worden. □

We veranderen van variabele. Het lemma zegt dat een boom van hoogte (diepte) d tenminste $n = F_{d+2} - 1$ knopen bevat. Maximaal zijn dat er $n = 2^d - 1$.
De relatie tussen hoogte d en aantal knopen n wordt daarmee: $\frac{1}{\sqrt{5}}\phi^{d+2} \leq n+1 \leq 2^d$. Neem $\log_2$ rechts en links; enerzijds vinden we $d \geq \log_2(n+1)$, anderzijds $d \leq (\log_2(n+1) + \frac{1}{2}\log_2 5)/\log_2 \phi - 2$.
Afgezien van wat ongemakkelijke constanten groeit d evenredig aan $\log_2 n$, dus $d \sim \log_2 n$.

2.4 Boom-wandelingen

▷ SCHAUM §10.5: Traversing binary trees.
De 'algoritmen' om bomen te bewandelen zijn *recursieve* functies. De *basis* van de recursie wordt niet expliciet gegeven bij SCHAUM: in elke ordening hoeft er niets gedaan te worden in het geval van een lege boom. De recursie is als beschreven: de ordening van de hele boom wordt uitgedrukt in de ordeningen van de deelbomen van de wortel.

Functies. Op dezelfde manier als de wandelingen (ordeningen) kunnen verschillende recursieve functies op bomen gedefinieerd worden. We geven twee voorbeelden.

Het *aantal knopen* van de lege boom is nul; anders is het één plus het aantal knopen van de linker plus het aantal knopen van de rechter deelboom van de wortel.

De *hoogte* van de lege boom is nul; anders is het één plus het maximum van de hoogtes van de linker en de rechter deelboom van de wortel.

▷ SCHAUM §9.4: Rooted trees
De waarde van een expressie kan recursief gedefinieerd worden als opgebouwd uit de waardes van deelexpressies. Het is inzichtelijker om dit te doen aan de hand van de *expressie boom*, ▷ SCHAUM blz. 238.

Basis: de waarde van een *blad* is het getal opgeslagen (of de waarde geassocieerd met de variabele in het blad); inductie: de waarde van een *interne knoop* is de uitkomst van de operator van de knoop toegepast op de waarden van de kinderen van de knoop.

Omdat de waarde van een knoop noodzakelijkerwijs pas berekend kan worden als de waardes van de kinderen bekend zijn, heet dit wel de *postorde evaluatie* van de boom, vgl. ▷ SCHAUM § 10.5 voor binaire bomen.

Spelevaluaties. Een bijzonder geval van de evaluatie van bomen is het evalueren van de 'waarde' van een bepaalde stand in een spel (zoals schaken). Zo'n

spel definieert impliciet een boom. Knopen zijn de mogelijke toestanden van het spel (borden of standen) en de kinderen van een knoop zijn de mogelijke vervolg toestanden.

Bij het programmeren van complexe spelen moet de spelboom op een bepaalde diepte afgeknot worden omdat er teveel standen bestaan; het is onbegonnen werk ze allemaal door te rekenen.

Van sommige spelen kan volledig berekend worden of in een bepaald beginstand de eerste dan wel de tweede speler kan winnen (bij optimaal spel), als volgt. Markeer de bladeren (eindstanden) al naar gelang ze gewonnen dan wel verloren staan. Elke overige knoop is 'gewonnen' als er een 'verloren' kind is (de speler kan de bijbehorende zet doen, de tegenstander verliest) en 'verloren' als elk kind 'gewonnen' is (de speler kan winst van de tegenstander niet afwenden).

2.5 Co-grafen

▷ SCHAUM Ch. 8: Graph Theory

Op een aantal plaatsen in de Grafen theorie worden eigenschappen met inductie bewezen.

Theorem 8.6 (over bomen, aantallen takken en cycles) wordt behandeld in Problem 8.14. In de uitwerking wordt gebruik gemaakt van inductie naar het aantal knopen van de graaf.

Theorem 8.8 (Euler) wordt bewezen in Problem 8.18 door op te merken dat elke vlakke graaf uit een kleinere ontstaat door één van twee mogelijke operaties. Ofwel er wordt een tak toegevoegd naar een nieuwe knoop, ofwel er wordt een tweetal bestaande knopen verbonden. De bewering is dat in beide gevallen de waarde $V - E + R$ niet wijzigt. Dit is inductie naar de opbouw van de (vlakke) graaf.

Co-grafen We definiëren een tweetal bewerkingen op grafen. Gegeven zijn twee disjuncte grafen $G_1 = (V_1, E_1)$ en $G_2 = (V_2, E_2)$, dwz. ze hebben geen knopen gemeenschappelijk; $V_1 \cap V_2 = \varnothing$.

Uit dit tweetal maken we de *vereniging* $G_\cup = G_1 \cup G_2$ door beide grafen samen te voegen: $G_\cup = (V_1 \cup V_2, E_1 \cup E_2)$. De *som* $G_+ = G_1 + G_2$ is de graaf die gebaseerd is op de vereniging, maar bovendien alle lijnen tussen V_1 en V_2 bevat: $G_+ = (V_1 \cup V_2, E_1 \cup E_2 \cup (V_1 * V_2))$, waarbij in het algemeen $V * W = \{\, \{v, w\} \mid v \in V, w \in W, v \neq w \}$.

De familie van *complement reducible graphs* wordt recursief gedefinieerd als volgt:

1 De graaf met één knoop is een co-graaf.

2 Als G_1 en G_2 disjuncte co-grafen zijn, dan zijn $G_1 \cup G_2$ en $G_1 + G_2$ co-grafen.

Een (geïnduceerde) *deelgraaf* van een graaf onstaat door uit die graaf een aantal knopen weg te laten, met de bijbehorende lijnen. De lijnen die aanwezig zijn tussen de overgebleven knopen zijn precies de lijnen die in de oorspronkelijke graaf aanwezig zijn. Formeel: als $G = (V,E)$ en $W \subseteq V$ dan is de deelgraaf G_W (de restrictie van G tot knopen in W) gedefinieerd als $G_W = (W, E \cap (W * W))$.

Tenslotte is P_4 een *pad graaf,* bestaande uit vier knopen verbonden door vier opvolgende lijnen: $P_4 = (\ \{0,1,2,3\}, \{(0,1),(1,2),(2,3)\}\ )$.

2.5 Lemma. *Een co-graaf heeft geen geïnduceerde deelgraaf isomorf met P_4.*

Bewijs. Door middel van *structurele inductie,* dus naar de opbouw van de graaf.

basis. De graaf met één knoop heeft P_4 niet als deelgraaf.

inductie. We nemen aan dat de disjuncte co-grafen G_1 en G_2 aan de bewering voldoen, en dus geen P_4 als (geïnduceerde) deelgraaf hebben.

We laten zien dat ook de *som* $G_+ = G_1 + G_2$ voldoet. Als G_+ minder dan vier knopen bevat is voldaan aan de bewering. Anders kijken we naar elk mogelijk viertal knopen Elk viertal knopen dat ofwel geheel in G_1 ligt, ofwel geheel in G_2, is niet isomorf met P_4 volgens de inductieveronderstelling. Kijk dus naar vier knopen, één uit de ene graaf en drie uit de andere, of twee uit elk van beide grafen. In het eerste geval $(1,3)$ ontstaat in de deelgraaf van de som een knoop met graad drie; deze is niet in P_4 te vinden. In het tweede geval $(2,2)$ ontstaan vier knopen elk met graad tenminste twee; deze zijn niet in P_4 te vinden.

Conclusie is dat ook 'gemengde' knopen niet tot een deelgraaf P_4 leiden.

Nu moet ook nog beredeneerd worden dat de *vereniging* $G_1 \cup G_2$ geen deelgraaf P_4 bevat. Dit gaat op gelijksoortige wijze. De oorspronkelijke grafen hebben geen P_4 als deelgraaf, en er ontstaat geen P_4 in de vereniging omdat 'gemengde' knopen geen samenhangende graaf opleveren. □

Het omgekeerde geldt ook: elke graaf zonder (geïnduceerde) P_4 is een co-graaf. Ook dit kan met inductie bewezen worden. Het is hiervoor nodig om te laten zien dat een graaf zonder P_4 ofwel zelf niet samenhangend is, ofwel dat zijn complement dat is. (Het complement van een graaf is de graaf die ontstaat door het complement van de lijn-relatie te bekijken. Het complement van een co-graaf is weer een co-graaf.)

2.6 Associatieve Operaties

Blader terug naar Definitie 1.5. Daar vinden we beschreven wat de *machten* van een relatie R zijn, de samenstelling van een aantal kopieen van R: $R^n = \underbrace{R \circ R \ldots \circ R}$ (n maal).

Bekijk nu algemeen de binaire bewerking $\circ$ op verzameling A, dwz. voor $x, y \in A$ is ook $x \circ y$ weer een element van A. Ook dan kunnen we een element met zichzelf nemen $x \circ x$, of zelfs meerdere malen, en de *macht* van x definiëren. Het standaard voorbeeld is natuurlijk vermenigvuldigen, we zagen het bij samenstelling van relaties, en zien het later weer bij *concatenatie* van strings, en concatenatie van talen.

We moeten ons realizeren dat deze definitie niet altijd netjes de betekenis vastlegt. $x^n = \underbrace{x \circ x \ldots \circ x}$ (n maal), staat hier dan $x^4 = x \circ (x \circ (x \circ x))$ of $x^4 = ((x \circ x) \circ x) \circ x$ of misschien zelfs $x^4 = (x \circ x) \circ (x \circ x)$? Hier kunnen verschillende dingen uitkomen, tenminste als we geen eigenschappen (associativiteit) van $\circ$ veronderstellen. Een nette definitie, waardoor misverstanden uitgesloten zijn, gebruikt inductie.

2.6 Definitie. Laat $\circ : A \times A \to A$ een binaire bewerking op A zijn, en x een element van A. De n-de macht van x, $n \geq 1$, is $x^1 = x$ voor $n = 1$, en voor, $x^n = x^{n-1} \circ x$. □

Op deze manier geldt $x^4 = ((x \circ x) \circ x) \circ x$, terwijl $x^2 \circ x^2 = (x \circ x) \circ (x \circ x)$, iets wat niet hetzelfde hoeft te zijn als het voorgaande.

Bij een *associatieve* relatie geldt onze intuitie dat het zetten van haakjes de uitkomst niet beïnvloedt. (Het volgende resultaat kan ik niet meer vinden inde derde druk van SCHAUM.

2.7 Lemma. *Laat $\circ : A \times A \to A$ een* associatieve *binaire bewerking op A zijn. Dan heeft elke uitdrukking met operatie $\circ$ een waarde die alleen afhangt van de argumenten (en hun volgorde) en niet van de nesting van de haakjes.*

> Bewijs. We laten zien, met inductie naar de opbouw van de formule, dat de expressie $\alpha(x_1, x_2, \ldots, x_n)$ gelijk is aan $(\ldots((x_1 \circ x_2) \circ x_3)\ldots) \circ x_n$.
>
> *Basis.* Voor de expressie x_1 is de bewering vanzelfsprekend waar.
>
> *Inductiestap.* De expressie $\alpha(x_1, x_2, \ldots, x_{n+1})$ moet van de vorm $\alpha_1 \circ \alpha_2$ zijn, met argumenten $(x_1, \ldots, x_i)$ respectievelijk $(x_{i+1}, \ldots, x_{n+1})$.
>
> Pas nu de inductiehypothese toe op α_2, deze expressie is gelijk aan $\alpha_3(x_{i+1}, \ldots, x_n) \circ x_{n+1}$. Dus, vanwege associativiteit $\alpha = \alpha_1 \circ (\alpha_3 \circ x_{n+1}) = (\alpha_1 \circ \alpha_3) \circ x_{n+1}$.
>
> Tenslotte passen we de inductiehypothese toe op het stuk $\alpha_1 \circ \alpha_3$ binnen haakjes. Door hier de juiste vorm in te vullen wordt de hele expressie van juiste vorm. □

Heel vaak komen we de volgende bewering tegen, met een karakteristiek bewijs. Deze bewering is eigenlijk een speciaal geval van bovenstaande (en het bewijs kan overgeslagen worden).

2.8 Lemma. *Laat* $\circ : A \times A \to A$ *een* associatieve *binaire bewerking op A zijn en x een element van A. Voor alle* $m, n \in \mathbb{N}$ *geldt:* $x^{m+n} = x^m \circ x^n$.

Bewijs. We tonen de bewering aan met inductie naar n bij een willekeurige waarde van m. Zoals boven is ε de eenheid van $\circ$.

basis. $x^{m+0} = x^m = x^m \circ \varepsilon = x^m \circ x^0$.

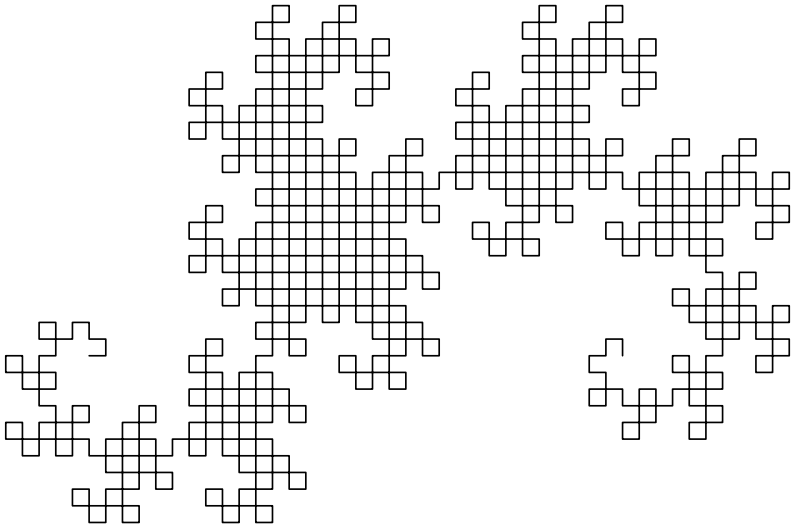
inductiestap. Neem aan dat de bewering geldt voor een willekeurige n. We hebben de volgende reeks gelijkheden, tot in de details uitgeschreven: $x^{m+(n+1)} \stackrel{a}{=} x^{(m+n)+1} \stackrel{d}{=} x^{m+n} \circ x \stackrel{h}{=} \left(x^m \circ x^n\right) \circ x \stackrel{a}{=} x^m \circ \left(x^n \circ x\right) \stackrel{d}{=} x^m \circ x^{n+1}$.

De gelijkheden aangegeven met *a* gelden vanwege associativiteit (van optelling en van $\circ$), *h* geldt vanwege de inductieaanname, en bij *d* hebben we de definitie van macht gebruikt.

Omdat m willekeurig gekozen is, geldt de bewering voor alle m. □

2.7 The Dragon Curve

▷ www.cs.unh.edu/~charpov/Programming/L-systems/



```
%!PS
/order 10 def

/START { X } def
/X {
  dup 0 ne
  {1 sub 4 {dup} repeat - F X + + F Y -}
  if pop
} def

/Y {
  dup 0 ne
  {1 sub 4 {dup} repeat + F X - - F Y +}
  if pop
} def

/F {
  0 eq { 10 0 rlineto } if
} bind def

/- { -45 rotate } bind def
/+ { 45 rotate } bind def

1 setlinejoin  1 setlinecap

newpath
220 180 moveto
50 order { 2 sqrt div } repeat dup scale
90 rotate
order START
stroke
showpage
```