Module 26 Typering en representatie

Onderwerp	Typering en interne representatie van expressies in Maple.
Voorkennis	Module 25.
Expressies	Commando's: `whattype, type, algsubs, simplify, subs, gc(), op, nops, map, indets, has;` Typen: `integer, symbol, float, string, function, numeric, equation, constant, rational, posint, negint`
Zie ook	Module 8, 29

26.1  Typering in Maple

In de vorige Modules zijn we al diverse soorten expressies tegengekomen. In deze Module gaan we in op de interne representatie van enkele van deze expressies, dat wil zeggen de manier waarop Maple de hem toevertrouwde informatie opslaat. Dit doen we om meer inzicht te verkrijgen in de wijze waarop Maple met gegevens omspringt en hoe bijvoorbeeld de substitutie werkt.

Elke expressie in Maple heeft een *type*, waarmee aangegeven wordt wat voor soort expressie dit is. Het type van een expressie kan worden opgevraagd met het commando `whattype`. We geven een paar voorbeelden:

`whattype`

`whattype(3)`	`integer`
`whattype(3.0)`	`float`
`whattype(a)`	`symbol`
`whattype("a")`	`string`
`whattype( f(a) )`	`function`
`whattype(a+3)`	`+`
`whattype(a*b)`	`*`
`whattype(a^b)`	`^`
`whattype(a*b+3=x^2)`	`=`

De diverse typen zijn gerangschikt in een hiërarchie, de zogenaamde *type-hiërarchie*. Bijvoorbeeld, een expressie van het type `integer` is automatisch ook van het type `algebraic`. We zeggen dat het type `integer` een *subtype* is van `algebraic`, en `algebraic` een *supertype* van `integer`.

Te midden van alle mogelijke typen die Maple kent is er een collectie zogenaamde *basistypen*. Tot deze categorie behoren onder andere de

typen `integer`, `fraction`, `float` (dat is een getal met een decimale punt erin), `symbol`, `string`, `function`, `+`, `*` en `^`. Men kan met `?whattype` opvragen welke hier nog meer toe behoren. De overige typen die Maple kent (dit zijn dus sub- of supertypen van een of meer basistypen) komt men te weten via `?type`. Alle typen die niet tot de basistypen behoren zijn afgeleid van deze basistypen, hetzij door combinatie, hetzij door specialisatie[57].
Bijvoorbeeld, een expressie is van het type `rational` als zij van een van de basistypen `integer` of `fraction` is (combinatie). Een expressie van het type `Array` (zie Module 8) is automatisch van het basistype `rtable`, omdat een `Array` in feite een `rtable` is die aan enkele extra eisen voldoet (specialisatie). Andere specialisaties van het basistype `rtable` zijn `Vector` en `Matrix` (zie Module 13).[58]
Basistypen kunnen dus zowel als subtype, als als supertype optreden. De procedure `whattype` geeft altijd een van de basistypen. Een expressie kan in het algemeen van verscheidene typen zijn, maar is slechts van één basistype. Om te testen of een expressie van een zeker sub- of supertype is (bijvoorbeeld om na te gaan of de expressie `3` van het type `numeric` is), dient de procedure `type`:

`type`

`a := 2:`	`type(a, integer)`	`true`
	`type(a, numeric)`	`true`
`a := p+q:`	`type(a, '+')`	`true`
`a := p=q:`	`type(a, equation)`	`true`

Let op dat bij het onderzoek of een expressie van het type `+`, `*` of `^` is, deze symbolen tussen *back-quotes* gezet moeten worden.
Bij het gebruik van `type` wordt van elk niveau waarop het eerste argument van `type` kan worden geëvalueerd, gekeken of dit van het type is dat is opgegeven als tweede argument van `type`. Dit verklaart de volgende sessie:

Voorbeeldsessie

```
> a := 2:  whattype(a);
```

integer

```
> type(a,constant); type(a,numeric);
```

true

true

[57] Het is ook mogelijk zelf eigen typen te definiëren, zie Module 28.

[58] De alledaagse Maple-gebruiker zal vrijwel nooit met het basistype `rtable` in aanraking komen, tenzij zij een eigen Array-achtig subtype zou willen definiëren.

```
> type(a,rational); type(a,algebraic);
```

true

true

```
> type(a,float); type(a,even);
```

false

true

```
> f := x -> x^2;
```

$$f := x \rightarrow x^2$$

```
> whattype(f);
```

symbol

```
> whattype(eval(f));
```

procedure

```
> type(f,symbol);
```

true

```
> type(f,procedure);
```

true

```
> whattype(f(x));
```

'^'

g is nog niet gedefinieerd:

```
> whattype(g(x));
```

function

```
> whattype(Pi);
```

symbol

```
> type(Pi, numeric); type(Pi, constant);
```

false

true

Toelichting

De argumenten van `whattype` worden volgens de normale regels geëvalueerd. Een procedurenaam evalueert dus tot een naam (zie §25.2, blz. 392), en namen zijn (meestal) van het basistype `symbol`. Na toepassing van `eval` blijkt dat een volledige evaluatie van `f` in een procedure resulteert. Procedures zijn van het basistype `procedure`. De procedure `type` controleert voor elk van deze evaluaties van `f` of deze van het als tweede argument opgegeven type zijn (`symbol` respectievelijk `procedure`).

Maple-constanten zoals `Pi` zijn enigszins problematisch. Omdat de typen `numeric` en `constant` geen basistypen zijn, reageert Maple op `whattype(Pi);` slechts met `symbol`. Dat het getal(!) π door Maple niet als `numeric` wordt beschouwd zou men kunnen betreuren; het

heeft te maken met de wijze waarop formules waarin π voorkomt worden vereenvoudigd. Als men bijvoorbeeld van een actuele parameter in een procedures wil testen of het een getal is (zie §29.4) moet men hiermee rekening houden: men zou in plaats van `a::numeric` moeten opgeven: `a::{constant,numeric}` ◇

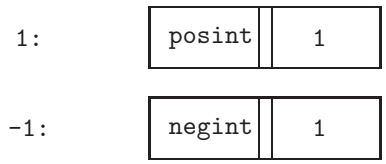
26.2 Representatie van expressies

In het algemeen wordt een object opgeslagen als een *paar* of *tripel* bestaande uit het type van het object, eventueel de naam van het object en een lijst met hierin de overige informatie die nodig is om het object vast te leggen. Deze overige informatie noemen we de *waarde* van het object. Schematisch geven we dit weer als in figuur 60.

FIGUUR 60. Representatie van een expressie

We zullen nu een aantal voorbeelden bespreken van de representatie van objecten van een aantal typen. Deze voorbeelden zijn verre van uitputtend, maar representatief voor de wijze waarop de toevertrouwde informatie door Maple wordt opgeslagen.

Gehele getallen (Integers). Wat betreft het type `integer` maakt Maple onderscheid tussen positieve en negatieve getallen. Zie bijvoorbeeld figuur 61 voor de representatie van de getallen -1 en $+1$.

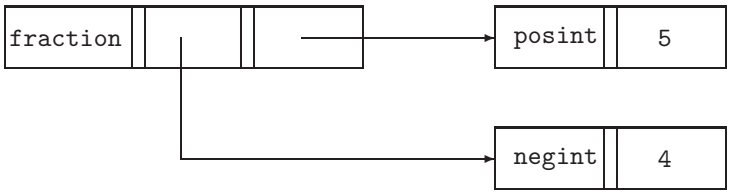


FIGUUR 61. Representatie van de gehele getallen 1 en -1

Het naamgedeelte ontbreekt hier.

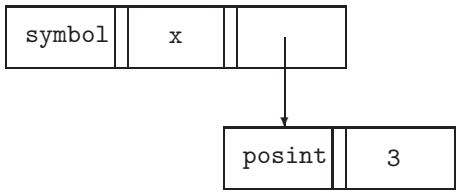
Rationale getallen. Een rationaal getal wordt weergegeven als een object van het type `fraction`. In het waardegedeelte hiervan

staan verwijzingen naar de teller en de noemer. Hierbij is de noemer altijd een positief geheel getal. Als voorbeeld is de representatie van het getal $-\frac{4}{5}$ weergegeven in figuur 62. Ook hierbij ontbreekt het naamgedeelte.



FIGUUR 62. Representatie van het rationale getal $-\frac{4}{5}$

Variabelen. Een variabele wordt in Maple gerepresenteerd als een object van het type `symbol`[59]. Uiteraard heeft een variabele een naam, zodat, in tegenstelling tot de twee voorbeelden hierboven, in de representatie van een variabele wél een naamgedeelte voorkomt. Het waardegedeelte van een variabele bevat nooit de waarde zelf, maar een *verwijzing* naar een object met een type en een waarde. Het resultaat van een toekenning `x := 3` is weergegeven in figuur 63.



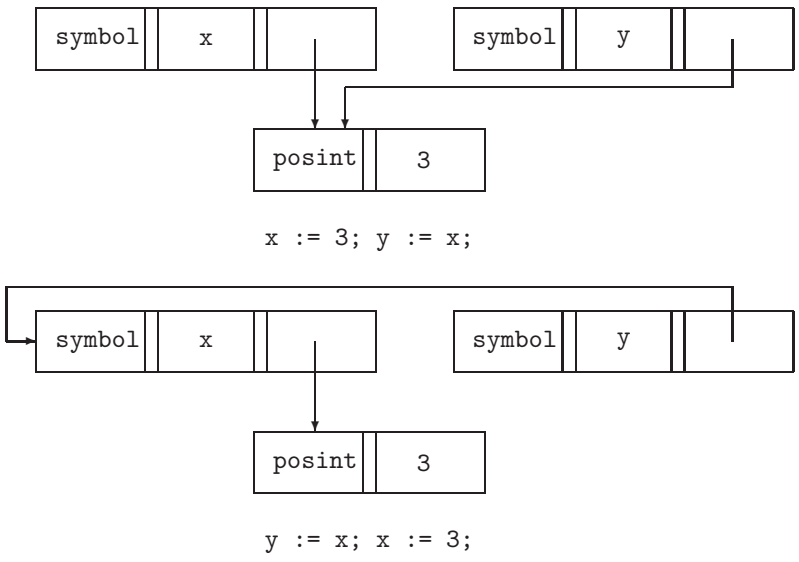
FIGUUR 63. Representatie van de variabele `x` met waarde 3

In dit geval levert `whattype(x)` het type `integer`[60] (en niet `symbol`). Dit is weer een voorbeeld van het feit dat het argument van de procedure `whattype` (en overigens van vrijwel elke procedure) eerst geheel wordt geëvalueerd.

[59]Dat is niet helemaal waar: een variabele kan ook nog van het type `indexed` zijn, zie Module 8.De typen `symbol` en `indexed` zijn beide basistypen, maar vallen onder het supertype `name`.

[60]Want `posint` is een subtype van het basistype `integer`.

In figuur 64 geven we nog de representaties van twee Maple-sessies uit §25.1.

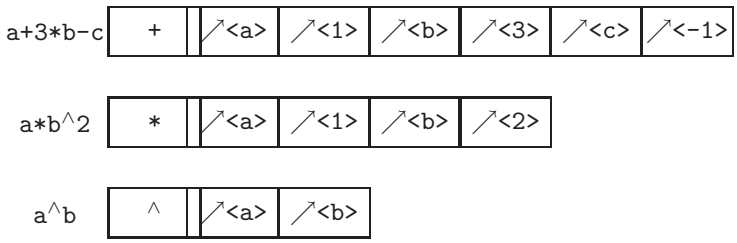


FIGUUR 64. Voorbeeldsessies van blz. 388 en van blz. 389

Het verschil tussen beide representaties is het gevolg van het feit dat wat rechts van de toekenningsoperator `:=` staat éérst wordt geëvalueerd vóórdat de feitelijke toekenning plaatsvindt. Daarom resulteert bij de eerste sessie de toekenning `y:=x` in `y:=3`. Bij de tweede sessie heeft `x` geen waarde als het commando `y:=x` wordt gegeven.

De volledige evaluatie van een van de variabelen `x` of `y` kan nu worden beschouwd als het volledig aflopen van de pijlen in de schema's van figuur 64.
Het feit dat in het waardegedeelte van de representatie van variabelen wordt verwezen naar de representatie van het object dat aan die variabele is toegekend, heeft te maken met efficiënt geheugenbeheer. Elk object hoeft nu maar één keer te worden opgeslagen.
De representatie van een object in Maple kunnen we aldus opvatten als een zogenoemde *Gerichte, Acyclische Graaf* (GAG) (Engels: *Directed Acyclic Graph (DAG)*).

In figuur 65 geven we nu nog drie voorbeelden van representaties van enkele expressies. Hierin betekent een uitdrukking als ↗`<a>` dat *verwezen wordt* naar de representatie van `a` enzovoort.



FIGUUR 65. Representatie van drie expressies

De waarde van een som wordt dus gerepresenteerd als een lijst bestaande uit paren. Het eerste element van zo'n paar verwijst naar een expressie, het tweede naar de factor vóór die expressie. Zo is de representatie van de expressie `a+3*b-c` van het type `+`, met drie operanden, namelijk (de paren): (↗`<a>`,↗`<1>`), (↗`<b>`,↗`<3>`) en (↗`<c>`,↗`<-1>`). De expressie `a+3*b-c` wordt dus opgeslagen als `1*a+3*b+(-1)*c`.
Analoge verhalen gaan op voor producten en machten; zo wordt `a*b^2` opgeslagen als `(a^1)*(b^2)`.

Als laatste voorbeeld hebben we in figuur 66 de volledig uitgewerkte representatie van de expressie `b*d+a-c` gegeven. Deze expressie is van het type `+` en heeft de operanden `b*d` (type `*`), `a*1` en `c*(-1)`.

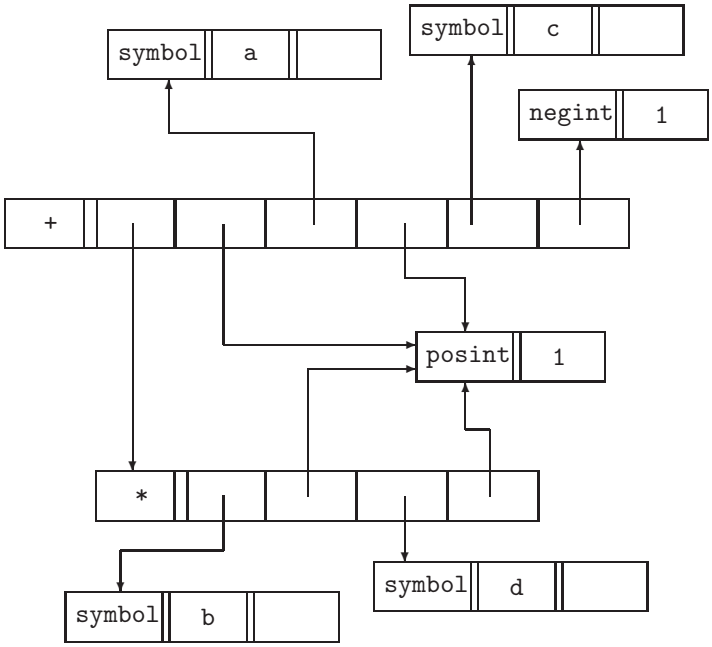
26.3 Substitutie

`subs`

We kunnen nu de werking van het `subs`-commando (zie ook Module 4) beter begrijpen. Bij de uitvoering van de opdracht

```
subs(x=z, p)
```

zoekt Maple de GAG van `p` af tot het een exemplaar van de deelgraaf behorende bij de representatie van de expressie `x` tegenkomt. Daarna vervangt Maple de verwijzing(en) naar deze deelgraaf door verwijzing(en) naar de GAG behorende bij de representatie van de expressie `z`. De expressie `x` moet dus wel een representatie hebben die voorkomt als (samenhangende) deelgraaf van `p`. In figuur 67 geven we aan wat dit in een eenvoudig geval betekent voor de interne representatie.



FIGUUR 66. Representatie van de expressie `b*d+a-c`

Omdat de GAG van `x*y` niet als deelgraaf in de GAG van `x^2*y-x` voorkomt werkt het volgende dus niet:

```
subs(x*y=s, x^2*y-x);
```

`algsubs` Voor dit soort gevallen kunnen we gebruikmaken van de procedure `algsubs`:

```
algsubs(x*y=s, x^2*y-x);
```

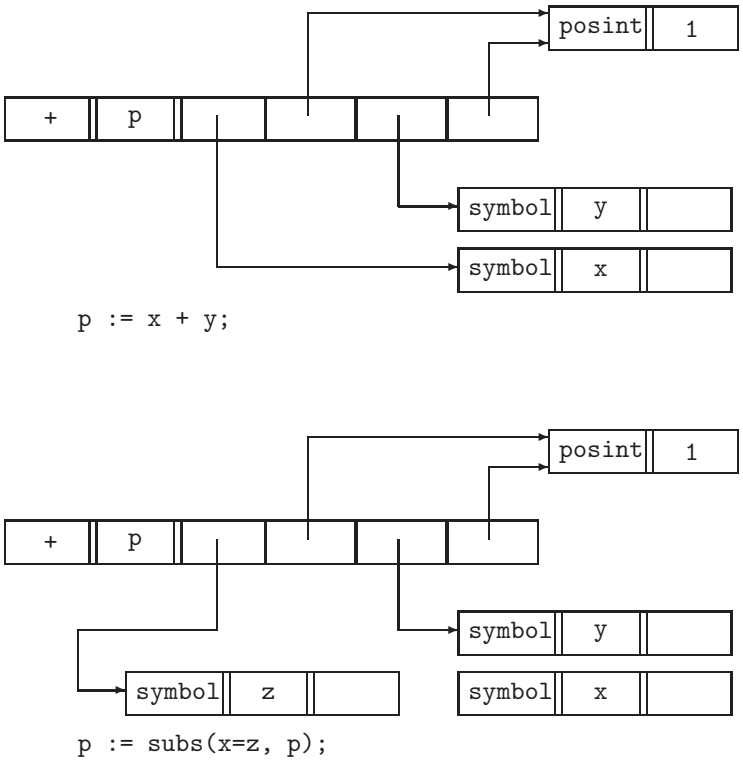
Aan de hand van de figuur 66 is nu te begrijpen dat het commando

```
subs(1=ppp, b*d+a-c);
```

resulteert in de expressie $b^{ppp}d^{ppp}ppp+a\,ppp-c$. Door dit commando wordt de `posint` 1 vervangen door het `symbol` ppp.

26.4 Garbage collection

In figuur 67 zagen we dat niet meer wordt verwezen naar `x`. In principe zorgt Maple er zelf voor dat het deel van het geheugen waar `x` is opgeslagen weer kan worden gebruikt om iets anders in op te slaan. We



FIGUUR 67. Representatie van een substitutiecommando

noemen dit *garbage collection*. Garbage collection vindt niet voortdurend plaats, maar op regelmatige tijdstippen. Soms, vooral bij lange Maple-sessies, is het echter verstandig om Maple hiermee wat tot spoed te manen door het zelf geven van de opdracht `gc();`

`gc()`

26.5 De procedure `simplify`

Voor Maple zijn expressies pas gelijk als ze dezelfde GAG hebben. Dit verklaart waarom $x^2 - x - x(x-1)$ niet onmiddellijk tot 0 evalueert. Ondanks dat de expressies $x^2 - x$ en $x(x-1)$ wiskundig aan elkaar gelijk zijn, hebben ze een verschillende GAG in Maple. Het Maple-commando `simplify(expressie)` draagt er zorg voor dat de GAG van `expressie` in een standaardvorm wordt gebracht. Daarom levert `simplify(x^2-x-x(x-1))` wel 0 op.

`simplify`

Naast de opdracht `simplify` kent Maple ook nog een *automatische* simplificatie van expressies. Dit mechanisme zorgt er bijvoorbeeld

voor dat een uitdrukking als `0*x` onmiddellijk en onontkoombaar wordt vervangen door `0`. Daarnaast zijn er nog vele andere omzettingen, die alle hetzelfde doel dienen: het voorkomen van uitdrukkingen die nodeloos ingewikkeld zijn.

26.6 Operanden; de procedure `map`

We beschouwen nogmaals de expressie `p := a + 3*b - c`. Deze expressie werd in figuur 65 gerepresenteerd als een rij *paren*. De paren in deze lijst noemen we de *operanden* van de expressie `p`. Deze hele rij kunnen we opvragen met de procedure `op`; de n^e operande krijgen we met `op(n, p)`.

`op`

Het *aantal* operanden van een expressie kunnen we opvragen met de opdracht `nops`.

`nops`

De procedures `op` en `nops` kunnen worden gebruikt om enig inzicht te verkrijgen in de structuur van de GAG van een expressie. Echter, niet de gehele GAG kan hiermee worden gereconstrueerd. Zo vinden we bijvoorbeeld voor de expressie `a + 3*b - c` de operanden `a`, `3*b` en `-c`. Niet duidelijk wordt dat de operande `a` in feite `1*a` zou moeten zijn, dus van het type `product`. Dit komt omdat de uitdrukking `1*a`, die het resultaat is van `op(1,p)`, door automatische simplificatie direct wordt omgezet naar `a`.

Voorbeeldopgave

- Bepaal van de expressie $x + y(z + 3)$ het aantal operanden en welke dit zijn. Geef bovendien de tweede operande.
- Onderzoek de operanden van de *function* `f(a,b,c)`.

Voorbeeldsessie

Een algebraïsche expressie:

```
> p := x + y * (z+3);
```

$$p := x + y\,(z + 3)$$

```
> aantal_operanden := nops(p);
```

$$aantal\_operanden := 2$$

```
> de_operanden := op(p);
```

$$de\_operanden := x,\ y\,(z + 3)$$

```
> tweede_operande := op(2,p);
```

$$tweede\_operande := y\,(z + 3)$$

```
> op(0,p);
```

$$\text{`+`}$$

Een expressie van het type `function`:

```
> op( f(a,b,c) );
```

$$a,\ b,\ c$$

```
> op( 1, f(a,b,c) );
```

$$a$$

```
> op( 2, f(a,b,c) );
```

$$b$$

```
> op( 0, f(a,b,c) );
```

$$f$$

Toelichting

In het algemeen heeft `op(0,p)` ook betekenis. Als `p` (in geëvalueerde vorm) een expressie is, dan is de 'nulde operande' van `p` het (basis-) type; bij een functie (-aanroep) is het de naam van de functie. ◇

Soms is het gewenst om alle operanden van een expressie *afzonderlijk* aan de een of andere bewerking te onderwerpen. Hiervoor is de procedure `map` bedoeld. We demonstreren het gebruik ervan in de volgende

`map`

Voorbeeldsessie

```
> functie := x -> 1/x:
> formule := a+p*q+b+c+sin(t)+1/d;
```

$$formule := a + p\,q + b + c + \sin(t) + \frac{1}{d}$$

```
> functie(formule);
```

$$\frac{1}{a + p\,q + b + c + \sin(t) + \frac{1}{d}}$$

```
> map(functie,formule);
```

$$\frac{1}{a} + \frac{1}{p\,q} + \frac{1}{b} + \frac{1}{c} + \frac{1}{\sin(t)} + d$$

```
> map( x->1/x, formule );
```

$$\frac{1}{a} + \frac{1}{p\,q} + \frac{1}{b} + \frac{1}{c} + \frac{1}{\sin(t)} + d$$

```
> map( whattype, formule );
```

$$3\,symbol + \text{`*`} + function + \text{`^`}$$

```
> map(type,formule,symbol);
```

$$3\,true + 3\,false$$

```
> indets(formule);
```

$$\{\sin(t),\ a,\ p,\ q,\ b,\ c,\ t,\ d\}$$

```
> has( formule, t );
```

true

```
> has( formule, p*q );
```

true

```
> has( formule, a + p*q );
```

false

Toelichting

De expressie `formule` heeft de operanden `a`, `p*q`, `b`, `c`, `sin(t)` en `1/d`. Let op het verschil in resultaat wanneer we de procedure `functie` op de hele `formule` of met behulp van `map` op elke operande afzonderlijk laten werken.
Enigszins merkwaardige, maar in dit geval goed te interpreteren, resultaten krijgen we als we procedures als `whattype` of `type` op de expressie laten werken. Zo levert `whattype`, toegepast op elk van de operanden van `formule` afzonderlijk:

$$symbol + \text{`}*\text{`} + symbol + symbol + function + \text{`}\wedge\text{`},$$

en dat vereenvoudigt tot het getoonde resultaat. Zie Module 8 voor een uitgebreidere bespreking van `map` met praktisch bruikbare toepassingen.

Ten slotte zijn in deze voorbeeldsessie nog twee functies gebruikt die soms kunnen helpen de structuur van een expressie te verhelderen.

`indets` Met `indets` verkrijgt men de verzameling (let op de accolades) van alle onbekenden in een expressie.

`has` Het commando `has` kan worden gebruikt om te onderzoeken of een bepaalde *subexpressie* in een formule voorkomt. Ook hier moeten we enige zorgvuldigheid betrachten (vergelijk §26.3). Om bijvoorbeeld te kunnen constateren dat $a + pq$ in de formule voorkomt, hadden we kunnen vragen:

```
type(formule,'+') and has(formule,p*q) and
has(formule,a)
```

Opgave 26.1

Bepaal de GAG van de expressie $x^3y(x+3) + 4xy$.
Controleer het antwoord zoveel mogelijk door gebruik te maken van `whattype`, `nops` en `op`.

Opgave 26.2

Voer de volgende sessie uit:

```
whattype( eval(sin) );
whattype( sin(x) );
```

Raadpleeg `?function` en verklaar de sessie.

Opgave 26.3

(a) Bepaal het type van de expressies $1/x$, $2/x$, $2/3$.
Is Maple consequent in zijn typering? (Bepaal ook de operanden.)

(b) Verklaar het resultaat van de substitutie

```
subs( 1=-1, -1/3 );
```

(c) Verklaar het resultaat van de substitutie

```
subs( 1=-1, x^2 - x + 1/x - 1/3 ).
```

Opgave 26.4

Transformeer de expressie $x + y + z$ naar $d + z$. Waarom werkt

```
subs(x+y=d, x+y+z)
```

niet? (Vergelijk §4.2, vereenvoudigen met nevenvoorwaarde.)