Module 25 Evaluatie van expressies

Onderwerp	Het evaluatiemechanisme.
Voorkennis	Module 1, 4.
Expressies	`eval, ' ', evaln`
Zie ook	Module 7, 28, 30.

25.1  Evaluatie van expressies

Beschouw de volgende Maple-sessie:

```
x := 3;
```

Hier staat dat aan de variabele met de *naam* `x` de *waarde* `3` wordt toegekend. De naam van een variabele kan worden opgevat als het etiket van een doosje waarin iets kan worden opgeborgen. De operator `:=` zorgt dat het rechterlid hiervan in het doosje wordt opgeborgen. Wanneer de variabele wordt opgevraagd, wordt het doosje uitgepakt.

Voorbeeldsessie

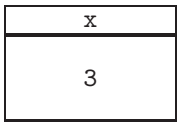
```
> x := 3;
```

$$x := 3$$

```
> x;
```

$$3$$

We kunnen dit schematisch weergeven als in figuur 53.



FIGUUR 53. Variabele `x` met waarde 3: 'doosjes'-voorstelling

Een andere manier om dit te illustreren, die meer aansluit bij de interne representatie van Maple-objecten, is als een gerichte graaf. Dat is voor dit eenvoudige voorbeeld in figuur 54 gedaan. Vanuit een knooppunt (blokje) met de naam `x` loopt een pijl naar een knooppunt met de waarde `3` van `x`.

FIGUUR 54. De gerichte-graaf-voorstelling van variabele `x` met waarde 3

Wanneer een doosje nog leeg is (bijvoorbeeld in het geval van een variabele waaraan nog niet iets is toegekend) wordt de naam van het doosje teruggegeven:

Voorbeeldsessie

```
> y;
```

$$y$$

Het uitpakken van doosjes noemen we *evaluatie*. We zeggen dat `x` evalueert tot `3` en dat `y` evalueert tot (haar eigen naam) `y`.
Natuurlijk kan het zo zijn dat het rechterlid van de `:=` operator zelf weer een doosje is. In dit geval wordt, vóór de uitvoering van de `:=` operator, eerst dit doosje uitgepakt. Bij wijze van voorbeeld zullen we twee gevallen onderscheiden.

Geval 1 **Het doosje rechts van de `:=` is niet leeg.** Dit is het geval in het volgende voorbeeld:

Voorbeeldsessie

```
> x := 3;
```

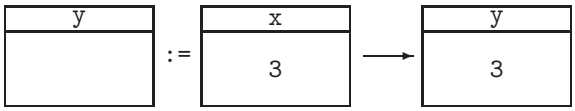
$$x := 3$$

```
> y := x;
```

$$y := 3$$

```
> y;
```

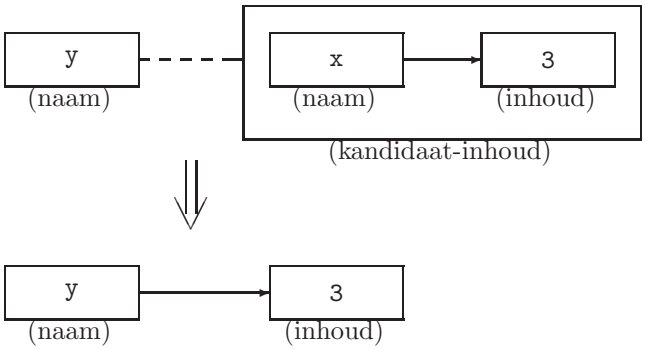
$$3$$



FIGUUR 55. Toekenning van een variabele met een waarde: 'doosjes'-voorstelling

Bij de toekenning `y := x` wordt eerst het doosje met de naam `x` uitgepakt. De inhoud hiervan wordt in het doosje met de naam `y`

gestopt. Zie figuur 55 voor de 'doosjes'-voorstelling en figuur 56 voor de graafvoorstelling.



FIGUUR 56. Toekenning van een variabele met een waarde: gerichte-graaf-voorstelling

Er wordt in figuur 56 als het ware vanuit `y` een pijl getrokken naar `x` (zie het stippellijntje), maar omdat je via de naam `x` een pijl kunt volgen naar de waarde van `x`, wordt de pijl vanuit `y` naar de waarde van `x` doorgetrokken.

We kunnen nu nog steeds de inhoud van doosje `x` veranderen:

Voorbeeldsessie

```
> x := 3;
```

$$x := 3$$

```
> y := x;
```

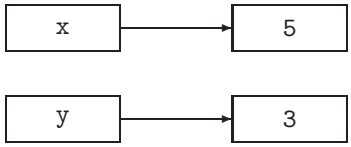
$$y := 3$$

```
> x := 5;
```

$$x := 5$$

```
> y;
```

$$3$$



FIGUUR 57. Gerichte-graaf-voorstelling van het resultaat van de bovenstaande voorbeeldsessie

De inhoud van y blijft dan wat hij was. In de graafrepresentatie geven we het resultaat hiervan weer in figuur 57. De pijl vanuit de naam x wordt als het ware verlegd naar een andere inhoud. De pijl vanuit y blijft staan.

Geval 2 **Het doosje rechts van de := is leeg.** Beschouw het volgende voorbeeld:

Voorbeeldsessie

```
> y := x;
```

$$y := x$$

```
> x := 3;
```

$$x := 3$$

```
> y;
```

$$3$$

Een nieuwe waarde voor x:

```
> x := 5:  y;
```

$$5$$

De waarde van y is ook veranderd.
Nu kennen we expliciet een waarde toe aan y:

```
> y := x;
```

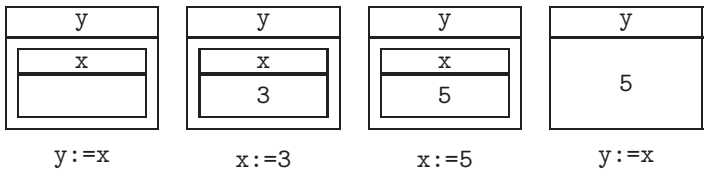
$$y := 5$$

Een nieuwe waarde voor x:

```
> x := 7:  y;
```

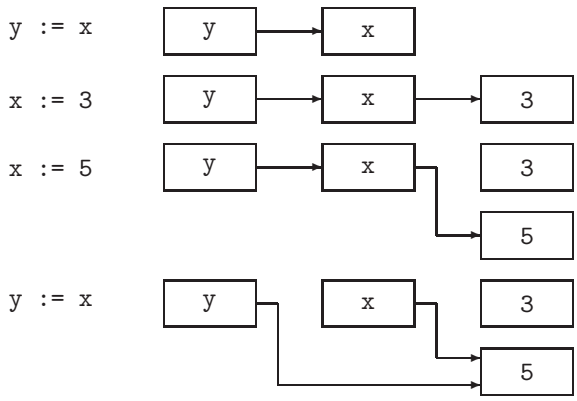
$$5$$

Nu is y onveranderd.



FIGUUR 58. 'Doosjes'-voorstelling van de bovenstaande voorbeeldsessie

Ook nu wordt in de eerste toekenning (y := x) eerst het doosje met de naam x uitgepakt. Hierin zit echter niets, zodat x evalueert tot haar eigen naam. Dan wordt het doosje x tot inhoud van y gemaakt. Wanneer we nu in het doosje x het getal 3 stoppen, en we vragen y op, dan wordt het doosje y uitgepakt. Hierin zit een ander doosje, namelijk x. Dit wordt vervolgens ook uitgepakt, totdat Maple iets



FIGUUR 59. Gerichte-graaf-voorstelling van de voorbeeldsessie op blz. 390

tegenkomt wat niet verder kan worden uitgepakt. Als we nu het doosje `x` van een andere inhoud voorzien, verandert ook de waarde waartoe `y` evalueert. Immers, het uitpakken van het doosje `y` levert het doosje `x`. Dit doosje kan verder worden uitgepakt, en dat levert dan de waarde `5`. Bij het tweede statement `y := x` wordt wederom het doosje met de naam `x` uitgepakt en de gevonden waarde wordt aan `y` toegekend. Zo is er expliciet een nieuwe inhoud in het doosje `y` gestopt, de inhoud van `y` verandert nu niet meer bij een verandering van de inhoud van `x`. In figuur 58 en 59 wordt dit tot uitdrukking gebracht.
Om de waarde van `y` te vinden wordt de pijl vanuit het knooppunt met de naam `y` gevolgd. Via het knooppunt met de naam `x` komt men dan terecht bij de waarde `3`. Als de pijl vanuit `x` wordt verlegd, verandert ook het resultaat van het aflopen van de pijlen vanuit `y`. Pas wanneer de pijl vanuit `y` wordt verlegd, hangt de waarde van `y` niet meer af van die van `x`.

In principe mag links en rechts van de toekenningsoperator `:=` dezelfde naam (variabele) voorkomen. Dit gaat alleen goed als deze variabele een waarde heeft (geval 1):

Voorbeeldsessie

```
> x := 7:
```

Hier heeft `x` een waarde, en dan gaat het goed:

```
> x := x + 1;
```

$$x := 8$$

```
> x := 'x':
```

Maar als `x` geen waarde heeft:

```
> x := x + 1;
```

```
Error, recursive assignment
```

Toelichting

In de toekenning `x := x+1` wordt het rechterlid éérst volledig geëvalueerd. Als `x` de waarde 7 heeft, dan wordt dat dus 8, en deze waarde kan zonder bezwaar toegekend aan de variabele `x` (die daardoor dus een nieuwe waarde krijgt).
Als `x` daarentegen géén waarde heeft, dan evalueert het rechterlid tot `x+1`, en als we deze 'waarde' aan `x` willen toekennen, dan weigert Maple dat.

25.2 Volledige, gedeeltelijke en verhinderde evaluatie

Het volledig uitpakken van doosjes (of het volledig aflopen van pijlen) noemen we *volledige evaluatie*. We kunnen de afzonderlijke stappen in de evaluatie ook apart doen met het commando `eval`:

`eval`

Voorbeeldsessie

```
> y := x:      x := 3:
> eval(y,1);
```

$$x$$

```
> eval(y,2);
```

$$3$$

```
> y;
```

$$3$$

De procedure `eval` pakt zoveel doosjes uit (of volgt precies evenveel pijlen) als opgegeven in het tweede argument.[53] Wordt helemaal geen tweede argument gegeven, dan zorgt `eval` voor een volledige evaluatie. Voorbeelden van '`eval`-achtige' expressies zijn we al tegengekomen in Module 3. De procedures `evalf` en `evalc` doen hetzelfde

`evalc`, `evalf`

[53] Merk op dat er geen verwarring kan ontstaan met het `eval`-commando zoals het in §4.5 is gebruikt, omdat daar als tweede argument een *vergelijking* moet worden ingevuld.

als `eval` zonder tweede argument, maar interpreteren intussen alle getallen als reële respectievelijk als complexe getallen.

`' '` Het gebruik van (gewone) quotes (`' '`) verhindert (éénmalig) het uitpakken van doosjes of het aflopen van pijlen.[54]

Voorbeeldsessie

```
> x := 3;
```

$$x := 3$$

```
> 'x';
```

$$x$$

In deze sessie evalueert `'x'` tot haar eigen naam `x`. Dit verklaart ook waarom we met een commando als `x := 'x'` eerdere toekenningen aan `x` ongedaan kunnen maken.

`evaln` Een alternatief voor het gebruik van quotes is het commando `evaln` (dat dus eigenlijk precies het omgekeerde doet van `eval`). Er is een subtiel verschil met de quotes als de variabele een (ongedefinieerde) *functie*-aanroep (zie §6.9) of een *geïndiceerde* variabele is (zie Module 8).

Voorbeeldsessie

```
> T := [eerste,tweede,derde,vierde,vijfde]:  i := 3:
> T[i];
```

$$derde$$

```
> 'T[i]';
```

$$T_i$$

```
> evaln(T[i]);
```

$$T_3$$

`T` is een *lijst*; het derde element ervan wordt aangegeven met `T[3]`; het getal 3 is de *index*. We zien in het bovenstaande voorbeeld dat door het gebruik van quotes de naam van de lijst én die van de index beide ongeëvalueerd blijven, terwijl bij het gebruik van `evaln` de index wél wordt geëvalueerd.
Bij een *functie*-aanroep van de vorm `f(x)` is het verschil tussen `'f(x)'` en `evaln(f(x))` vergelijkbaar.

[54] De (gewone) quotes niet verwarren met back-quotes! Deze laatste worden gebruikt om namen te maken die afwijkende karakters bezitten, zoals leestekens (zie §2.3).

25.3 Evaluatieregels

In vele gevallen worden expressies volledig geëvalueerd zodra ze worden gebruikt, tenzij we uitdrukkelijk aangeven dat we dat niet willen. Ook bij het *aanroepen* van een procedure worden de argumenten (dus de expressies tussen de haakjes achter de naam van de procedure) eerst geëvalueerd vóórdat de procedure ermee 'aan het werk gaat'! Bijvoorbeeld:

Voorbeeldsessie

```
> x := Pi;
```

$$x := \pi$$

```
> sin(x);
```

$$0$$

Voordat hier de `sin` wordt berekend, wordt eerst `x` geëvalueerd.

Evaluatie van met `->` gedefinieerde functies. In Module 7 hebben we al gezien dat een expressie die bij de definitie van een procedure rechts van de `->`-operator staat juist *niet* wordt geëvalueerd. Door de toekennning `df := x -> diff(f(x),x)` wordt de expressie `diff(f(x),x)` niet geëvalueerd. Daarom resulteert de sessie

```
df := x -> diff(f(x),x); y := 3; df(y);
```

in een foutmelding. Eerst wordt namelijk in de uitdrukking `df(y)` de actuele parameter `y` van de procedure `df` geëvalueerd (en wel tot de waarde `3`). Pas daarna wordt elk voorkomen van de corresponderende formele parameter (namelijk de dummy `x`) vervangen door de waarde van de actuele parameter, waardoor de body wordt:

```
diff(f(3),3)
```

en dat geeft de foutmelding "wrong number (or type) of parameters in function diff."

Evaluatie in het `subs`-commando. Een ander voorbeeld van een situatie waarin de evaluatie van de argumenten wel eens aanleiding tot verwarring kan geven is de procedure `subs`.

Voorbeeldsessie

```
> x := 'x':  p := x^2 +3*x + 2;
```

$$p := x^2 + 3\,x + 2$$

```
> x := 4:  eval(p,1);
```

$$x^2 + 3\,x + 2$$

```
> eval(p,2);
```

$$30$$

```
> subs( x=qqq, p );
```

$$30$$

```
> subs( 'x'=qqq, p );
```

$$30$$

```
> subs( x=qqq, eval(p,1) );
```

$$x^2 + 3\,x + 2$$

```
> subs( 'x'=qqq, eval(p,1) );
```

$$qqq^2 + 3\,qqq + 2$$

```
> x := 2:  subs( x=qqq, eval(p,1) );
```

$$x^{qqq} + 3\,x + qqq$$

Toelichting

Ook als `x` een waarde heeft, geeft de eerste stap in de evaluatie `eval(p,1)` de expressie die we het eerst aan `p` hadden toegekend; pas in de tweede stap wordt de waarde van `x` ingevuld. De eerste twee `subs`-commando's werken beide niet, omdat de daarin voorkomende `p` eerst volledig wordt geëvalueerd tot de waarde 30. Maar ook het daarna volgende `subs`-commando werkt niet omdat eerst de argumenten worden geëvalueerd, hetgeen resulteert in

```
subs( 4=qqq, x^2 + 3*x + 2 );
```

Als we hier `'x'` in plaats van `x` gebruiken wordt de evaluatie van `x` in de uitdrukking `'x'=qqq` voorkomen, en dan lukt het wel.
De laatste regel van dit voorbeeld demonstreert trouwens dat de substitutie van een *getal* in iets anders best mogelijk is, en soms tot onverwachte resultaten kan leiden als we vergeten zouden zijn dat `x` een waarde heeft. ◇

In het algemeen vindt in Maple altijd volledige evaluatie plaats, behalve in de volgende gevallen:

- Namen tussen (gewone) quotes;
- De expressie rechts van het `->`-symbool (zie §7.3);
- De *eerste* naam in een expressie gevormd met het concatenatiesymbool `||` . Zie opgave 25.2.

||

– Namen van tabellen (`table`s, zie §8.7) en van procedures (zie Module 28) (evenals de niet meer gebruikte: arrays, vectoren, matrices[55]);
- De argumenten (actuele parameters) van de procedures `eval`, `evaln`, `evalf`, `traperror` (zie Module 28) en `seq` (zie Module 8)
- Zogenaamde lokale variabelen binnen procedures (zie Module 28).

In sommige gevallen wil men dat een of meer van de actuele parameters van een procedure juist *niet* wordt geëvalueerd. In dat geval kan men evaluatie dus voorkómen door ze tussen gewone quotes te zetten of `evaln` te gebruiken. Zie §28.7 voor meer details.

Evaluatie tot de laatste naam. De manier waarop tables worden geëvalueerd staat bekend als *evaluatie tot de laatste naam* ("*last name evaluation*"). Dit betekent dat namen van tables worden geëvalueerd tot de *naam* van de variabele waaraan oorspronkelijk de table was toegekend. Het komt er in feite op neer dat de *laatste* stap van de volledige evaluatie niet wordt uitgevoerd.

Voorbeeldsessie

```
> T1 := table( [(a)=1, (b)=x^2, (y)=z] );
```

$$T1 := \mathrm{table}([b = x^2,\ a = 1,\ y = z])$$

```
> T2 := T1:    T3 := T2:    T4 := T3;
```

$$T4 := T1$$

```
> T4[b] := 1:    eval(T1);
```

$$\mathrm{table}([b = 1,\ a = 1,\ y = z])$$

```
> T := 'T': T := table([(1)=a,(2)=b] ):
> a :=5:  b := 6:
> eval(T);
```

$$\mathrm{table}([1 = a,\ 2 = b])$$

```
> T[1];
```

$$5$$

```
> map(eval,T);
```

$$\mathrm{table}([1 = 5,\ 2 = 6])$$

Toelichting

De naam `T4` evalueert eerst tot `T3`, dan tot `T2` en dan tot `T1`. Aan

[55] Arrays, Vectoren en Matrices (hoofdletter!) worden echter wél altijd volledig geëvalueerd.

deze laatste naam in de rij `T4,T3,T2,T1` was oorspronkelijk de table toegekend. Vandaar de term 'last name evaluation'.
We zien ook dat door een verandering van de waarde van `T4[b]` tegelijk de waarde van het origineel `T1[b]` is gewijzigd. Dit betekent dat `T4,T3,T2,T1` allemaal namen zijn voor één en hetzelfde object.[56]
Daarom kunnen `T4`,`T3` en `T2` ook niet zonder meer worden beschouwd als onafhankelijke kopieën van `T1`. Het *kopiëren* van tables kan daarom het beste gebeuren met de procedure `copy`. Dit is vergelijkbaar met het maken van kopieën van Vectoren en Matrices (en ook van Arrays) met een `Copy`-commando, zoals besproken in §13.5.
In het voorbeeld is bovendien te zien dat door `eval(T)` de *elementen* van `T` slechts tot hun naam worden geëvalueerd. Volledige evaluatie van elk element afzonderlijk moet worden afgedwongen met `map`. ◇

`copy`

Opgave 25.1

Wat gaat er mis in de volgende sessie en waarom?

```
x := 3; int(sin(x),x=0..Pi);
```

Opgave 25.2

De sessie

```
y := 3; z := y; x||y||z;
```

resulteert in `x33`. Ga na dat hieruit blijkt dat de tweede en derde naam in de expressie `x||y||z` eerst volledig evalueren voordat de concatenatie wordt uitgevoerd. Bedenk ook een voorbeeld waaruit blijkt dat de eerste naam niet eerst volledig evalueert.

Opgave 25.3

Zie ook Module 5.

(a) Voer de volgende sessie uit:

```
opl := solve({x-3}, {x});
assign(opl[1]);
opl := solve({x-2}, {x});
assign(opl[1]);
```

Waar gaat het mis en waarom?

[56] In programmeertalen als Pascal en C noemen we `T4,T3,T2,T1` *pointers* naar dit object.

(b) We proberen de ellende van het vorige onderdeel te verhelpen op de volgende wijze:

```
restart; # nieuwe sessie:
opl := solve({x-3}, {x});
assign(opl[1]);
opl := solve({'x'-2}, {'x'});
assign(opl[1]);
```

Waarom werkt ook dit niet?

(c) We beginnen weer met dezelfde drie statements:

```
restart; # nieuwe sessie:
opl := solve({x-3}, {x});
assign(opl[1]);
opl := solve({'x'-2}, {'x'});
```

We proberen nu het resultaat van de tweede versie van `opl` toe te kennen aan `x`. Waarom werkt

```
x := subs( opl, 'x' );
```

niet, maar

```
x := subs(solve({'x'-2}, {'x'}), 'x' );
```

wel?