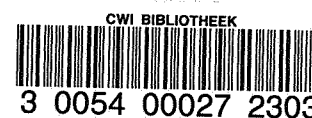




Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Heering

Variaties op het thema "stack"
Een oefening in algebraïsche specificatie



Afdeling Informatica

Notitie CS-N8502 Februari

Variations on the theme "stack"
An exercise in algebraic specification

Bibliotheek
Centrum voor Wiskunde en Informatica
Amsterdam

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

14B/134 CR 12.1 F.3.2

nsc 69D21
69F32

VARIATIES OP HET THEMA "STACK"
Een oefening in algebraïsche specificatie

Jan Heering
Centrum voor Wiskunde en Informatica
Amsterdam

Aan de hand van het datatype "stack" worden allerlei aspecten van algebraïsche specificaties geïllustreerd. Aan de orde komen onder andere constructor-functies, waarneembaar gedrag, consistentie, geparametriseerde modules, persistentie en sufficient completeness van parameters, en import van modules in andere modules.

INLEIDING

Algebraïsche specificatie van het datatype `stack` is een nuttige oefening, want - hoe eenvoudig dit datatype ook mag lijken - er zijn in kort bestek een heleboel aspecten van de algebraïsche specificatietechniek mee te illustreren zonder dat het zicht op hun essentie belemmerd wordt door overbodige details.

De semantiek van een specificatie is in het vervolg steeds de initiële algebra semantiek tenzij anders vermeld. De initiële algebra is op isomorfie na uniek en minimaal in de zin dat hij geen elementen bevat die niet op grond van de specificatie ook werkelijk vereist zijn ("no junk"). Verder zijn gesloten termen (d.w.z. termen zonder variabelen) in de initiële algebra dan en slechts dan gelijk als hun gelijkheid op grond van de in de specificatie gegeven vergelijkingen met behulp van vergelijkingenlogica bewezen kan worden ("no confusion") [1,2,3,4].

1. DE BOOLEANS

(Er is geen aparte sectie voor constanten in de signatuur. Constanten worden opgevat als functies zonder argumenten. Bij gebrek aan een maalteken wordt '#' gebruikt om in de signatuur een Cartesisch product aan te geven.)

Notitie CS-N8502
Centrum voor Wiskunde en Informatica
Postbus 4079, 1009 AB Amsterdam

```

module BOOL1
begin
    sort bool

    functions T,F: -> bool
        not: bool -> bool
        + : bool # bool -> bool    (exclusive-or)
        . : bool # bool -> bool    (conjunctie)
        v : bool # bool -> bool    (disjunctie)

    equations not(T)  = F
               not(F)  = T

               T+T = F+F = F
               T+F = F+T = T

               T.T = T
               T.F = F.T = F.F = F

               TvT = TvF = FvT = T
               FvF = F

end BOOL1

```

In tegenstelling tot gesloten termen kunnen open termen (d.w.z. termen met variabelen) in de initiële algebra van een specificatie gelijk zijn zonder dat dit met vergelijkingenlogica te bewijzen is. In de initiële algebra van BOOL1 gelden bijv. de volgende niet uit BOOL1 afleidbare open vergelijkingen (X, Y en Z zijn variabelen van soort bool):

```

not(not(X)) = X
X+not(X) = T
X+(Y+Z) = (X+Y)+Z
T.X = X
X.Y = Y.X
XvX = X
X.(Y+Z) = (X.Y)+(X.Z)

```

Er zijn echter gemakkelijk niet-minimale modellen van BOOL1 aan te geven waarin een of meer van deze vergelijkingen niet gelden ten gevolge van de aanwezigheid van "junk" die zich afwijkend gedraagt zonder met de vergelijkingen in de specificatie in conflict te komen. Bijv. aanwezigheid in het model van een element $C \neq T, F$ met $F.C = T$ en $C.F = F$ zorgt ervoor dat de conjunctie niet meer commutatief is.

Omdat bewijsbaarheid in de vergelijkingenlogica precies overeenkomt met waarheid in alle modellen, zijn dus vergelijkingen die niet in alle modellen waar zijn, maar die wel in de initiële algebra gelden, niet met vergelijkingenlogica te bewijzen. Het lukt soms dergelijke vergelijkingen te bewijzen met (meervoudige) structurele inductie, maar vaak is ook dat niet voldoende en is er geen volledig bewijssysteem mogelijk.

In het geval van BOOL1 is echter wel volledigheid ten opzichte van de initiële algebra te bereiken, zelfs zonder dat structurele inductie te hulp geroepen hoeft te worden, dat wil zeggen alleen op basis van vergelijkingenlogica. De volgende specificatie BOOL2 heeft hetzelfde initiële model als BOOL1, maar is zodanig "verrijkt" dat alle vergelijkingen die gelden in het initiële model met vergelijkingenlogica afleidbaar zijn:

```

module BOOL2
  begin
    sort bool

    functions T,F: -> bool
              not: bool -> bool
              + : bool # bool -> bool
              . : bool # bool -> bool
              v : bool # bool -> bool

    variables X,Y,Z: -> bool

    equations not(F)      = T
              not(not(X)) = X
              X+F = X
              X+not(Y) = not(X+Y)
              X+Y = Y+X
              X+(Y+Z) = (X+Y)+Z
              X.not(Y) = X+(X.Y)
              X.X = X
              X.Y = Y.X
              X.(Y.Z) = (X.Y).Z
              X.(Y+Z) = (X.Y)+(X.Z)
              XvY = (X.Y)+(X+Y)

  end BOOL2

```

BOOL2 is niet zo gemakkelijk in termen van herschrijffregels te operationaliseren (compileren). Het Knuth-Bendix algoritme in zijn eenvoudige vorm [1] loopt vast op de commutativiteit van de exclusive-or en de conjunctie en er is een krachtiger versie van het algoritme nodig. Een dergelijke versie bestaat maar wordt hier verder niet besproken.

2. EEN EENVOUDIGE STACK ZONDER FOUTENSIGNALERING

```

module STACK1
begin
    sorts data, stack

    functions  $d_1, d_2, \dots, d_n: \rightarrow \text{data}$ 
                empty:  $\rightarrow \text{stack}$ 
                push:  $\text{data} \# \text{stack} \rightarrow \text{stack}$ 
                top:  $\text{stack} \rightarrow \text{data}$ 
                rest:  $\text{stack} \rightarrow \text{stack}$ 

    variables x:  $\rightarrow \text{data}$ 
                s:  $\rightarrow \text{stack}$ 

    equations top(empty)      =  $d_1$ 
                top(push(x,s)) = x
                rest(empty)    = empty
                rest(push(x,s)) = s
end STACK1

```

Opmerkingen:

- Alle gesloten termen van soort stack zijn gelijk aan een term van de vorm

$$\text{push}(d_{i_1}, \text{push}(d_{i_2}, \dots, \text{push}(d_{i_p}, \text{empty}) \dots)) \quad (p \geq 0).$$

push is de stack-constructor.

- In plaats van één pop-operatie, die topelement en nieuwe stack oplevert, zijn er twee aparte operaties, nl. top en rest.
- Van foutensignalering is nauwelijks sprake: er is geen stack-underflow conditie; d_1 is te beschouwen als default element van de stack.

3. WAARNEEMBAAR GEDRAG IN DE INITIËLE ALGEBRA VAN STACK1

In de initiële algebra van STACK1 (en trouwens ook in alle andere modellen) geldt:

```

top(push( $d_1$ , empty)) = top(empty) en
rest(push( $d_1$ , empty)) = rest(empty).

```

Gezien vanuit soort data hebben empty en push(d_1 , empty) hetzelfde waarneembare gedrag [1]. Dat wil zeggen dat er geen context $T(\)$ van type stack $\rightarrow$ data is, zodanig dat

$$T(\text{empty}) \neq T(\text{push}(d_1, \text{empty})).$$

In de initiële algebra van STACK1 geldt echter

$$\text{push}(d_1, \text{empty}) \neq \text{empty}$$

omdat gelijkheid niet afleidbaar is uit de in de specificatie gegeven vergelijkingen. Dit heeft iets onnatuurlijks en is te verhelpen door

$$\text{push}(d_1, \text{empty}) = \text{empty} \quad (*)$$

aan STACK1 toe te voegen. In dat geval zijn gesloten termen van soort stack in de initiële algebra dan en slechts dan gelijk als zij, gezien vanuit soort data, hetzelfde waarneembare gedrag vertonen. Immers, na toevoeging van (*) is elke gesloten term van soort stack gelijk aan een term van de vorm

$$\text{push}(d_{i_1}, \text{push}(d_{i_2}, \dots, \text{push}(d_{i_p}, \text{empty}) \dots)) \quad (p \geq 0, i_p \neq 1).$$

Beschouw termen

$$t_1 = \text{push}(d_{i_1}, \dots, \text{push}(d_{i_p}, \text{empty}) \dots) \quad (p \geq 0, i_p \neq 1) \text{ en}$$

$$t_2 = \text{push}(d_{j_1}, \dots, \text{push}(d_{j_q}, \text{empty}) \dots) \quad (q \geq 0, j_q \neq 1)$$

en contexten

$$\begin{aligned} T_1 &= \text{top}() \\ T_2 &= \text{top}(\text{rest}()) \\ T_3 &= \text{top}(\text{rest}(\text{rest}())) \\ &\dots \end{aligned}$$

Neem nu aan dat $t_1 \neq t_2$, dan is $p \neq q$ of $d_{i_m} \neq d_{j_m}$ voor een of andere $m \geq 1$.

In het eerste geval (zeg $p > q$) is de ongelijkheid van t_1 en t_2 waarneembaar via T_p :

$$T_p(t_1) = d_{i_p} \neq d_1 = T_p(t_2)$$

en in het tweede geval via T_m :

$$T_m(t_1) = d_{i_m} \neq d_{j_m} = T_m(t_2).$$

Dus na toevoeging van (*) geldt

$$t_1 \neq t_2 \implies t_1 \text{ en } t_2 \text{ waarneembaar verschillend (dank zij (*))}$$

$$t_1 = t_2 \implies t_1 \text{ en } t_2 \text{ niet waarneembaar verschillend (gemakkelijk).}$$

4. MODULARISERING

Het submodule van STACK1 dat het type van de stack-elementen definieert is:

```

module DATA1
  begin
    sorts data

    functions  $d_1, d_2, \dots, d_n$ :  $\rightarrow$  data
  end DATA1

```

De initiële algebra van DATA1 bestaat uit de constanten $d_1, \dots, d_n$. Deze algebra persisteert binnen de initiële algebra van STACK1 dat wil zeggen:

- (1) STACK1 introduceert geen (nieuwe) gelijkheden tussen d_i 's ("no confusion").
- (2) Alle nieuwe termen van soort data (zoals bijv. $\text{top}(\text{push}(d_2, \text{empty}))$) zijn gelijk aan een of andere d_i ("no junk").

Sommige auteurs noemen (2) sufficient completeness en eisen voor persistentie alleen (1).

Laat nu de specificatie van de stack-elementen weg uit STACK1 en introduceer een formele parameter ITEMS:

```

module STACK1'
begin
    parameter ITEMS
    begin
        sort item

        function D: -> item
    end

    sort stack

    functions empty: -> stack
                push: item # stack -> stack
                top:  stack -> item
                rest: stack -> stack

    variables x: -> item
                s: -> stack

    equations top(empty)      = D
                top(push(x,s)) = x
                rest(empty)    = empty
                rest(push(x,s)) = s
    end STACK1'

```

STACK1' is incompleet in die zin dat hij alleen interessant is in combinatie met een nadere specificatie van ITEMS. Dat STACK1' ook zonder nadere invulling van ITEMS een initiële algebra heeft is toeval. In het algemeen hebben geparametriseerde specificaties geen initiële algebra, omdat er geen termen van elke soort hoeven te zijn zolang de parameter niet gebonden is. Bovendien is de initiële algebra van geparametriseerde modules, zelfs als hij bestaat, als semantiek inadequaar. Het is immers niet zo interessant wat een dergelijk module op zich is, als wel wat het na completering worden kan. Hoe dit ook zij, er geldt

```

STACK1 = STACK1' with ITEMS
           bound by [item -> data, D -> d1] to DATA1.

```

STACK1' laat zich ook combineren met bijv. de natuurlijke getallen:

```

module NAT
begin
    sort N

    functions 0: -> N
                S: N -> N
end NAT

```

Dan specificiceert

STACK1 with ITEMS bound by [item $\rightarrow$ N, D $\rightarrow$ 0] to NAT

stacks van natuurlijke getallen met default-element 0.

5. STACK MET POP-OPERATIE

```

module STACK2
begin
  sorts data, stack, tuple

  functions d1, d2, ..., dn:  $\rightarrow$  data
    empty:  $\rightarrow$  stack
    push:  data # stack  $\rightarrow$  stack
    pop:   stack  $\rightarrow$  tuple
    pair:  data # stack  $\rightarrow$  tuple
    first: tuple  $\rightarrow$  data
    second: tuple  $\rightarrow$  stack

  variables x:  $\rightarrow$  data
             s:  $\rightarrow$  stack

  equations push(d1,empty)    = empty           --(1)
              pop(empty)        = pair(d1,empty)   --(2)
              pop(push(x,s))     = pair(x,s)        --(3)
              first(pair(x,s))   = x                --(4)
              second(pair(x,s))  = s                --(5)

end STACK2

```

Opmerkingen:

- Deze stack heeft een pop-operatie die een tuple bestaande uit top-element en nieuwe stack oplevert. Deze combineert de eigenschappen van de functies top en rest van STACK1.
- Rudimentaire foutensignalering als bij STACK1. Voor vergelijking (1) zie par. 3. Vergelijking (2) is eigenlijk overbodig, immers
 $\text{pop}(\text{empty}) = \text{pop}(\text{push}(d_1, \text{empty})) = \text{pair}(d_1, \text{empty})$ volgens (1), resp. (3).

6. STACK MET FOUTENSIGNALERING (MISLUKTE POGING)

```

module STACK3
begin
    sorts data, stack

    functions %, d1, d2, ..., dn: -> data
                E, empty: -> stack
                push:    data # stack -> stack
                top:     stack -> data
                rest:    stack -> stack

    variables x: -> data
                s: -> stack

    equations push(%,s)    = E                --(1)
                push(x,E)   = E                --(2)
                top(E)       = %                --(3)
                top(empty)   = %                --(4)
                top(push(x,s)) = x              --(5)
                rest(E)      = E                --(6)
                rest(empty)  = E                --(7)
                rest(push(x,s)) = s            --(8)

end STACK3

```

Opmerkingen:

- Als STACK1, maar nu met een poging tot serieuze foutensignalering.
% en E zijn de foutsignalen van soort data resp. stack.
- Hoe ziet de initiële algebra van STACK3 eruit? Er geldt:

$$\begin{aligned}
 x &= \text{top}(\text{push}(x,E)) = \text{top}(E) = \% \quad \text{volgens (5), resp. (2), resp. (3)} \\
 s &= \text{rest}(\text{push}(\%,s)) = \text{rest}(E) = E \quad \text{volgens (8), resp. (1), resp. (6)}
 \end{aligned}$$

De initiële algebra (en trouwens op isomorfie na alle andere modellen) hebben dus slechts twee elementen, nl. % van soort data en E van soort stack. Met andere woorden STACK3 is inconsistent. DATA1 persisteert volstrekt niet binnen STACK3, want er is maximale "confusion" tussen de d_i's. Dit kan niet de bedoeling zijn.

7. STACK MET FOUTENSIGNALERING (TWEDE EN GESLAAGDE POGING)

De vergelijkingen (5) en (8) in STACK3 zijn te algemeen. Ze moeten vervangen worden door zoiets als

$$\begin{aligned}
 \text{top}(\text{push}(x,s)) &= x & \text{als } s \neq E & \quad \text{resp.} \\
 \text{rest}(\text{push}(x,s)) &= s & \text{als } x \neq \% &
 \end{aligned}$$

Een in die zin verbeterde versie van STACK3 valt op natuurlijke wijze uiteen in twee modules, nl. BOOL3 en STACK4', die hieronder volgen.

```

module BOOL3
begin
    import BOOL1

    function if: bool # * # * -> *

    variables x,y: -> *

    equations if(T,x,y) = x
               if(F,x,y) = y
end BOOL3

```

Opmerkingen:

- BOOL3 is hetzelfde als BOOL1 maar met een if-functie toegevoegd.
- Voor * mag elke soort gesubstitueerd worden. De if-functie is dus poly-morf. Vooralsnog is er binnen BOOL3 alleen

```
ifbool: bool # bool # bool -> bool (* = bool),
```

maar als BOOL3 in een ander module geïmporteerd wordt komen er net zoveel if's bij als er nieuwe soorten in het module zijn.

De (geparametriseerde) specificatie van de stack ziet er als volgt uit:

```

module STACK4
begin
    parameter ITEMS
    begin
        sort item

        functions %: -> item
                error: item -> bool

        equation error(%) = T
    end

    import BOOL3

    sort stack

    functions E, empty: -> stack
            ERROR:      stack -> bool
            push:       item # stack -> stack
            top:        stack -> item
            rest:       stack -> stack

    variables x: -> item
            s, t: -> stack

    equations ERROR(E)      = T
            ERROR(empty) = F
            ERROR(push(x,s)) = error(x) v ERROR(s)

            push(%,s)      = E                --(1)
            push(x,E)      = E                --(2)
            top(E)         = %                --(3)
            top(empty)     = %                --(4)
            top(push(x,s)) = if(ERROR(s),%,x) --(5)
            rest(E)        = E                --(6)
            rest(empty)    = E                --(7)
            rest(push(x,s)) = if(error(x),E,s) --(8)

end STACK4

```

Als parameter kan bijv. fungeren:

```

module DATA2
begin
    import BOOL3

    sort data:

    functions %, d1, d2, ..., dn: -> data
                error: data -> bool

    equations error(%)      = T
                error(d1)   = F
                error(d2)   = F
                ...
                error(dn)   = F
end DATA2

```

Opmerkingen:

- De bedoelde verbeterde versie van STACK3 is

```

STACK4 = STACK4' with ITEMS
          bound by [item -> data, % -> %, error -> error] to DATA2.

```

- Vergelijk i.h.b. (5') en (8') van STACK4' met (5) en (8) van STACK3.

Er geldt nu:

(A) In de initiële algebra van STACK4 is elke gesloten term van soort stack gelijk aan E òf aan een term van de vorm

$$\text{push}(d_{i_1}, \text{push}(d_{i_2}, \dots, \text{push}(d_{i_p}, \text{empty}) \dots)) \quad (p \geq 0).$$

(B) De initiële algebra van DATA2 persisteert (par. 4) in de initiële algebra van STACK4. In het bijzonder geldt

(1) $T \neq F$, $d_i \neq d_j$ ($i \neq j$) en $d_i \neq \%$ (persistentie in enge zin)

(2) Alle termen van soort bool zijn gelijk aan T of F en alle termen van soort data zijn gelijk aan % of aan een of andere d_i (sufficient completeness).

REFERENTIES

- [1] J.A. Bergstra & J.W. Klop, "Algebraïsch Programmeren", Cursusmateriaal PAO Cursus "Moderne Technieken in Software Engineering", CWI, Amsterdam, 1984.
- [2] A.T. Berztiss & S. Thatte, "Specification and implementation of abstract data types", in: M.C. Yovits (Ed.), Advances in Computers, Vol. 22, 1983, pp. 295-353.
- [3] R.M. Burstall & J.A. Goguen, "Algebras, Theories and Freeness: An Introduction for Computer Scientists", in: M. Broy & G. Schmidt (eds.), Theoretical Foundations of Programming Methodology, Reidel, 1982, pp. 329-348.
- [4] H.A. Klaeren, Algebraische Spezifikation, Springer-Verlag, 1983.

ONTVANGEN 0 1 MAART 1985