Verificatie van een stuk wiskundige taal

N.G. de Bruijn - 6 juni 1967

Verificatie van een stuk wiskundige taal.

Het stuk taal bestaat uit enkele arrays:

level [0 : M] (elementen zijn integers)
middle [1 : M] (elementen zijn expressions)
right [1 : M] (elementen zijn categories).

Een expression is óf een symbool genaamd "empty blockopener"
of een symbool genaamd "primitive notion"
of een echte expressie.

Een echte expressie is óf een positive integer, óf een rij symbolen v.d. vorm

positive integer(echte expressie,...,echte expressie).

In dit laatste geval worden deze componenten aangeduid met "head" van E, 1e subexpressie van E, enz., als E een afkorting voor de expressie is.

Een "category" is één der symbolen

bool, truth, axiom, elt, set.

We willen de in onze taal opgeschreven zinnen verifiëren door ze één voor één te onderwerpen aan de procedure acceptable, nl. aan

acceptable(level [m], m, middle [m], right [m]).

Op het ogenblik dat de m-de zin onderzocht wordt dienen alle vorige reeds geaccepteerd te zijn. level [0] dient de waarde 0 te hebben.

Voorbeelden van echte expressies:

13
145(6(2,5(3)), 12(3(8)), 16(4), 14, 24).

In het laatste geval is de head 145, de eerste subexpressie is 6(2,5(3)), de derde 16(4), het aantal subexpressies is 5.

```
integer procedure numberofsubexpressions (E); expression E;
integer number of subexpressions;
begin .....................
      comment De stippeltjes geven aan numberofsubexpressions als waarde
      het aantal subexpressions van E (dit aantal kan ook nul zijn). Alleen
      aan te roepen als E een echte expressie is;
end;

expression procedure subexpression (E,i); expression E, subexpression;
integer i;
begin .....................
      comment De stippeltjes geven aan subexpression als waarde de i-de
      subexpression van E. Alleen aan te roepen als E een echte expressie
      is en h > 0, 1 ≤ i ≤ h, waarbij h het aantal subexpressies van E is;
end;

integer procedure head(E); expression E; integer head;
begin .....................
      comment De stippeltjes geven aan head als waarde de head van E.
            Alleen aan te roepen als E een echte expressie is;
end;

expression procedure build (p, t, Z); integer p, t;
expression array Z [1 : t]; expression build;
begin .....................
      comment De stippeltjes geven aan build als waarde de expressie
      p(Z[1],...,Z[t]). Alleen aan te roepen als p ≥ 1, t ≥ 1 en als
      Z[1],...,Z[t] echte expressies zijn;
end;
```

```
integer array procedure precedingleaders (n,m); integer n,m;
integer array precedingleaders [1 : m];
comment Zoekt de m direct aan plaats n voorafgaande plaatsen waar een bij
n geldig blok wordt geopend. Als er bij n zelf een blok wordt geopend is
de m-de plaats n zelf. Dus precedingleaders [k] is, als 1 ≤ k ≤ m, het
grootste getal q ≤ n met level [q] ≤ level [n] + k - m. Alleen aan te
roepen als 1 ≤ m ≤ level [n]. Werkt alleen goed als inderdaad voor alle
x geldt level [x + 1] ≤ 1 + level [x];
begin integer t,s; t := n; s := level [n];
      for k := m step -1 until 1 do
      begin
repeat: if level [t] ≥ s then goto repeat; t := t - 1;
          precedingleaders [k] := t + 1; s := s - 1
      end
end;

boolean procedure categoryimplication (p,q); category p,q;

boolean categoryimplication;
begin .....
      comment De stippeltjes bewerkstelligen dat categoryimplication true
      wordt in de volgende gevallen
```

p	q
bool	bool
truth	bool
axiom	bool
truth	truth
axiom	truth
set	set
set	elt
elt	elt
bool	axiom
truth	axiom

```
      en false in alle andere gevallen
end;
```

```
procedure compatible (l,n,E,p); integer n,l; expression E, category p;
comment Onderzoekt of de regel met level l, indentifier n, middle E,
category p geschreven zou mogen worden. Alleen aan te roepen als E een
echte expressie is;
begin integer s,m,N; s := head(E); m := numberofsubexpressions (E);
      if s ≥ n or l + m < level [s] then goto fout;
      for N := n-1 step -1 until s do
      if level [N] + m < level [s] then goto fout;
      if categoryimplication (right [s], p) = false then goto fout;
      if m = 0 then goto end;
      begin integer h; integer array a[1 : m]; a := precedingleaders (n,m);
            for h := 1 step 1 until m do
            begin integer u; u := a[h]; compatible (l,n,subexpression (E,h),u);
                  if middle [u] ≠ emptyblockopener then
                  begin expression H; if h = 1 then
                        begin H := a[h]; goto verder
                        end;
                        begin expression array W[1 : h - 1];
                              integer j;
                              for j := 1 step 1 until h - 1 do W[j] :=
                              subexpression (E,j); H := build (a[h],h-1,W);
verder:                       checkidentity (H, subexpression (E,h))
                        end;
                  end;
            end;
      end;
end;
end;
```

```
procedure acceptable (l,n,E,p); integer n,l; expression E; category p;
comment beoordeelt aanvaardbaarheid van regel n met niveau l, midden E,
categorie p;
begin if l > level [n-1] + 1 then goto fout;
      if l < 0                then goto fout;
      if l ≤ level [n-1] then
      begin if E = primitivenotion then goto OK;
            compatible (l,n,E,p); goto OK
      end;
      if l = level [n-1] + 1 then
      begin if E = emptyblockopener then goto OK;
            compatible (l,n,E,bool);
            if p = true then goto OK else goto fout;
      end;
OK: print ⊧ accepted ⊨ ; goto end;
fout: print ⊧ rejected ⊨ ;
end:
end;

expression procedure applydefinition (E); expression E, applydefinition;
comment Als E een echte expressie is, wordt een nieuwe expressie bepaald
door E met gebruikmaking van de bij head(E) te vinden expressie uit te
werken.
begin integer t,q,l,j; expression array H[1 : t]; expression Q; integer array
      b [1 : t];
      t := numberofsubexpressions (E); q := head (E);
      if t = 0 then
      begin if middle [q] = emptyblockopener or middle [q] = primitivenotion
            then
            begin applydefinition := E; goto end
            end;
            applydefinition := middle [q]; goto end;
      end;
```

```
        b := precedingleaders (q,t); l := level b[1];
        if middle [q] = primitivenotion then
        begin applydefinition := E goto end
        end;
        if middle [q] := emptyblockopener then
        begin applydefinition := subexpression (E,t); goto end
        end;
        for j:= 1 step 1 until t do H[j] := subexpression (E,j);
        applydefinition := substitute(middle [q],t,H,l);
end:
end;
```

```
expression procedure substitute(M,t,H,l); expression M, integer t,l;
expression array H[1 : t];
comment Laat M een op plaats k geldige echte expressie zijn, laat t > 1,
en laat n1,...,nt de t bij plaats k behorende precedingleaders voorstellen.
Het level van n1 neet l. Laat H1,...,Ht echte expressies zijn.  De procedure
substitute beschrijft het effect op M van substitutie van H1,...,Ht op plaats
n1,...,nt;
begin integer s,r,h,i,e;
      s := head(M);
      h := numberofsubexpressions (M);
      r := level[s] - h - 1
      if r>0 then e := r + h else e := h;
      if e = 0 then
      begin substitute := M; goto end
      end;
      begin expression array Z [1 : e];
            for i := 1 step 1 until e - h do Z[ i] := H[i];
            for i := e - h + 1 step 1 until e do Z[i] := subexpression (M,i-e+h);
            comment als r = 0 resp. h = 0 doen deze regels niets;
            substitute := build(s,e,Z);
      end;
end;
end;
```

```
procedure checkidentity (E,F); expression E,F,G;
begin if head(E) < head(F) then
      begin G := F; F := E; E := G
      end;
      if head(E) > head(F) then
      begin G := applydefinition (E);
            checkidentity (G,F);
            goto end
      end;
      if head E = head F then
      begin integer k,l,h,j;
            k := numberofsubexpressions (E);
            l := numberofsubexpressions (F);
            if k > l then
            begin k := l; l := k; k := h; G := F; F := E, E := G
            end;
            if k < l then
            begin expression array Q[1 : l]; integer array a[1 : l];
                  comment Dit deel van het programma zorgt ervoor dat E er
                  aan de voorkant l-k subexpressies bij krijgt, elk uit een
                  identifier bestaande, nl. a[1],...,a[l-k];
                  a := precedingleaders (head(E),l);
                  for j := 1 step 1 until l-k do Q[j] := a[j];
                  for j := l - k + 1 step 1 until l do Q[j] := subexpression
                  (E, j - l + k);
                  E := build (head(E),l,Q)
            end;
```

```
                for j:=1 step 1 until k do
                begin expression P,Q;
                      P := subexpression (E,j);
                      Q := subexpression (F,j);
                      checkidentity (P,Q)
                end;
          end;
    end:
end;
```