Vérification des textes mathématiques par un ordinateur

Citation for published version (APA):

Document status and date:
Gepubliceerd: 01/01/1969

Document Version:
Uitgevers PDF, ook bekend als Version of Record

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Vérification des textes mathématiques par un ordinateur.

Conférence à Lille, le 21 novembre 1969,
par N.G. de Bruijn (Eindhoven).

Le problème de vérification des textes mathématiques est au fond le problème de définir un langage. Il faut que ce langage satisfasse aux propriétés suivantes:

1. Tout ce qui est correct en mathématiques peut être exprimé avec ce langage, conformément à sa grammaire, et tout ce qu'on peut dire au moyen de ce langage est correct en mathématiques.

2. On admet que les textes de mathématiques ordinales s'allongent par la traduction, mais on n'admet pas que la proportion de l'allongement tend vers l'infini à mesure qu'on fait des progrès.

3. On dispose d'un programme avec lequel un ordinateur peut trancher si un livre donné est correct ou non.

La tâche no. 1 est vraiment impossible. En premier lieu, il semble impossible de préciser ce qu'il veut dire "mathématiques". Pourtant, on peut essayer de circonscrire une partie de mathématiques, assez large pour la pratique quotidienne. Il semble que le langage AUTOMATH, développé à Eindhoven depuis 1967, satisfait à cette condition, ainsi qu'aux conditions 2 et 3.

Dans cette conférence-ci, nous considérerons principalement la définition du langage. À fin de mieux établir les difficultés principales, nous allons discuter d'abord deux langages plus simples: premièrement SEMIPAL, où on a déjà la structure de bloc et le système d'abréviation, et ensuite PAL, où on a, en outre, l'administration de catégories. On arrive à AUTOMATH en ajoutant un système de quantification en forme d'un calcul lambda généralisé.
Première phase: le langage SEMIPAL.

SEMIPAL est un système pour exprimer des notions en termes d'autres, en utilisant la structure de blocs imbriqués. Les expressions en SEMIPAL ont une structure donnée par

\[ \text{expression} ::= \text{identificateur} | \text{identificateur} \mid \text{liste d'expressions} \]

\[ \text{liste d'expressions} ::= \text{expression} \mid \text{liste d'expressions} \]

Un livre en SEMIPAL consiste d'un liste de lignes. Les lignes ont la forme

\[ \text{ligne} ::= \text{identificateur} ::= \text{explication} \]

ou

\[ \text{explication} ::= \_ | \_PN \mid \text{expression} \]

Les identificateurs des lignes diverses sont tous différents; si l'on veut, on peut choisir le numéro d'une ligne comme son identificateur.

La liste des lignes est structurée en blocs qui peuvent s'imbriquer les uns dans les autres, sans se chevaucher.

À la tête d'un bloc on trouve une ligne de la forme

\[ \text{identificateur} ::= \_ \]

En ce cas-ci, l'identificateur s'appelle variable.

L'indicateur d'une ligne est le variable à la tête du plus petit bloc contenant cette ligne, sauf pour les têtes de bloc elles-mêmes, où l'indicateur est la tête du plus petit bloc un seul excepté. Si une ligne ne se trouve dans aucun bloc, l'indicateur est le symbole 0.

La liste d'indicateurs d'une ligne est la liste des variables de têtes de blocs contenant cette ligne, dans l'ordre imposé par le livre. Encore, quand la ligne est à la tête d'un bloc, son identificateur n'appartient pas à cette liste.

Quand une expression figure dans une ligne, elle ne peut pas contenir des variables qui n'appartiennent pas à la liste d'identificateurs. D'autre part, les identificateurs non variables peuvent être utilisés partout, pourvu qu'ils sont munis d'une liste de sous-expressions dont la longueur satisfait à une certaine condition. Pour l'expression \( b(\Sigma_1, \ldots, \Sigma_2) \) utilisée dans la ligne \( \lambda \), cette condition s'exprime comme suit: Si \( x_1, \ldots, x_\xi \) est la liste
d'indicateurs de $b$, nous convenons que $\lambda \preceq r$ et que la ligne $\lambda$ se trouve dans le bloc de $x_{r-\lambda}$. Dans ce cas on peut considérer $b(E_1, \ldots, E_{r-\lambda})$ et $b(x_1, \ldots, x_{r-\lambda}, E_1, \ldots, E_{r-\lambda})$ comme identiques. On a des modifications triviales quand $r = 0$ ou $r = \lambda$.

**Exemple.** Nous prenons le texte qui s'écrit normalement comme

\[
\begin{align*}
\text{non} & : \text{notion primitive} \\
\text{et} (x,y) & : \text{notion primitive} \\
\text{ou} (x,y) & : \text{non (et(non(x), non(y)))} \\
\text{impl} (x,y) & : \text{ou (non(x), y)} \\
\text{equiv} (x,y) & : \text{et (impl(x,y), impl(y,x))} \\
\text{outroi} (x,y,z) & : \text{ou (ou(x,y), z)} \\
\text{tralaia} (x,y) & : \text{outroi (et(x,y), non(impl(x,y)), non(impl(y,x)))}
\end{align*}
\]

Ce texte se lit en SEMIPAL comme suit:

| x := | |
| non := PN |
| y := | |
| et := PN |
| ou := non (et(non, non(y))) |
| impl := ou (non,y) |
| equiv := et (impl, impl(y,x)) |
| z := | |
| outroi := ou (ou, z) |
| tralaia := outroi (et, non(impl), non(impl(y,x))) |

Dans la première colonne on trouve les indicateurs.

**Remarque.** Sans changer l'effet, on peut remplacer, par exemple, la définition de "equiv" par

\[
\begin{align*}
eviv & := \text{et (impl(y), impl(y,x))}, \\
ou & := \text{equiv := et (impl(x,y), impl(y,x))}.
\end{align*}
\]
Deuxième phase. Le langage PAL.

En PAL chaque expression a une catégorie, sauf les expressions qui sont des catégories elles-mêmes. Dans le cas dernier, on écrit au lieu de la catégorie le symbole type. On a le droit d'introduire des catégories nouvelles, soit comme des variables, soit comme des notions primitives, soit comme des expressions. Il y a quelques restrictions que nous donnons en forme abrégée:

1. Une tête de bloc doit avoir une catégorie qui est valable dans le bloc où cette tête de bloc est insérée.

2. La catégorie d'une notion primitive doit être une catégorie valable au même lieu.

3. Quand on écrit des expressions comme b(Σ₁, ..., Σᵣ), on demande (quand x₁, ..., xᵣ est la liste d'indicateurs de b) que Σᵢ a la catégorie de xᵣ₋ᵢ₊₂, que Σ₂ a la catégorie de xᵣ₋ᵢ₊₂ avec substitution de Σ₁ pour xᵣ₋ᵢ, etc.

Exemple.  

<table>
<thead>
<tr>
<th>type</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>réel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exemple.

0 := nat  
a := réel  
n := nat  
.............
.............

a := pot := réel  
0 := x := type  
x := y := type  
y := produit cartésien := PN type  
0 := u := produit cartésien (real, nat)  
u .............

On peut écrire des assertions en PAL, et par suite, on peut donner certaines démonstrations. Pour ça on convient qu'une assertion correspond à une catégorie. Alors, faire l'assertion veut dire: exprimer qu'on a une expression avec cette catégorie. Par exemple si l'on veut introduire l'égalité de nombres réels comme notion primitive, on écrit
D'ailleurs, on peut écrire l'axiome $\varphi = \psi$ comme

$$v := \text{PN \ EGAL (p,q)},$$

et le théorème $\varphi = \psi$ comme

$$w := \text{EGAL (p,q)},$$

ou ... est une expression qui peut être considérée comme la démonstration du théorème. Cette expression contient, soit implicitement, soit explicitement, toute information nécessaire pour comprendre la démonstration. Le théorème n'est pas énoncé avant que la démonstration soit terminée.

Si l'on veut, on peut considérer les propositions comme objets mathématiques:

$$\text{bool} := \text{PN \ type}$$

et on peut postuler que chaque proposition a son catégorie d'affirmation:

$$b := \text{bool}$$

$$\text{TRUE} := \text{PN \ type}$$

Alors, si $c$ est une proposition quelconque, on peut affirmer $c$ (soit comme hypothèse, soit comme axiome, soit comme théorème) par une ligne

$$... := \text{TRUE (c)}.$$

C'est ainsi qu'en PAL on peut décrire la structure de blocs en mathématiques où les têtes de blocs sont des variables aussi bien que des hypothèses.

Exemple. Nous introduirons la notion de conjonction de deux propositions comme notion primitive et nous exprimerons quelques axiomes et un théorème; après ça nous indiquerons comment on peut appliquer ce résultat dans une situation donnée.
Maintenant, supposons que les lignes suivantes sont disponibles:

\[
\begin{align*}
0 & \quad u := \quad \ldots \quad \text{bool} \\
0 & \quad v := \quad \ldots \quad \text{bool} \\
0 & \quad w := \quad \ldots \quad \text{TRUE}(\text{and}(u,v))
\end{align*}
\]

alors nous pouvons écrire

\[
0 \quad z := \quad \text{theorem}(u,v,w) \quad \text{TRUE}(\text{and}(v,u))
\]

3. Troisième phase: le langage AUTOMATH.

AUTOMATH est plus général que PAL; chaque texte en PAL est aussi valable en AUTOMATH. La différence entre les deux est qu'en AUTOMATH on se sert du calcul lambda de Church, ou plutôt d'une extension de ce calcul.

Si \( \xi \) est une catégorie quelconque, une application qui donne, à tout \( x \) du type \( \xi \), une valeur \( f(x) \) du type \( \eta(x) \), est écrite comme

\[
[x,\xi] f(x),
\]

et on dit que son type est

\[
[x,\xi] \eta(x).
\]
Une telle application peut être introduite comme une variable:

\[
p := \quad [x, \xi] \cdot \eta(x)
\]

ou comme notion primitive

\[
q := PN \quad [x, \xi] \cdot \eta(x).
\]

En outre, si l'expression \( \Sigma \) a le type \([x, \xi] \cdot \eta(x)\), et si l'expression \( \Lambda \) a le type \( \xi \), on a le droit de former

\[\{\Lambda\} \cdot \Sigma\]

Du reste, si \( \Sigma \) a la forme \([x, \xi] \cdot \Gamma\) (où \( \Gamma \) est une expression qui contient \( x \)), il est entendu que \( \{\Lambda\} \cdot \Sigma \) peut être remplacé par l'expression obtenue par substitution de \( \Lambda \) pour \( x \) dans \( \Gamma \).

**Exemple.**

\[
f := [x, \xi] \cdot \text{plus} \quad (x, x) \quad [x, \xi] \quad \text{réel}
\]

\[
a := \ldots \quad \text{réel}
\]

\[
b := \{a\} f \quad \text{réel}
\]

Ainsi \( b \) et plus \( (a, a) \) sont interchangeables (égal par définition).

Les règles d'opération pour \([\cdot]\) et \(\{\}\) sont intuitivement claires. Pourtant, cette matière n'est pas simple. En particulier, la substitution peut être assez compliquée à cause des variables cachées.

Dans les pages suivantes on trouve un exemple d'un texte un peu plus long, écrit en AUTOMATH. On verra la possibilité de donner des définitions assez compliquées et partagées en étapes.

Sans doute, le lecteur remarquera que ce texte est difficile à écrire et difficile à lire. C'est vrai, mais dans le sens de langages de programmation – on peut le comparer avec un langage de machine, et nous espérons qu'on réussisse à construire des langages artificiels plus simples, lesquels peuvent être écrits et lus plus facilement, et lesquels peuvent être traduits par un ordinateur en notre "langage de machine".
1. Quelques fondements de mathématiques qui suffisent
pour le théorème sur limites.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0</td>
<td>bool : = PN</td>
<td></td>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>0</td>
<td>b1 : =</td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>b1</td>
<td>TRUE : = PN</td>
<td></td>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>0</td>
<td>ksi : =</td>
<td></td>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>ksi</td>
<td>nonempty : = PN</td>
<td></td>
<td>bool</td>
<td>ksi</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>ksi</td>
<td>iks : =</td>
<td></td>
<td>ksi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>iks</td>
<td>then 2 : = PN</td>
<td></td>
<td>TRUE(nonempty)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>ksi</td>
<td>P : =</td>
<td></td>
<td>[u,ksi] bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.9</td>
<td>P</td>
<td>exists : = PN</td>
<td></td>
<td>bool</td>
<td>ksi</td>
<td></td>
</tr>
<tr>
<td>1.10</td>
<td>P</td>
<td>v : =</td>
<td></td>
<td>TRUE({v} P)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.11</td>
<td>v</td>
<td>ass 1 : =</td>
<td></td>
<td>TRUE(exists)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.12</td>
<td>ass 1</td>
<td>then 13* : = PN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.13</td>
<td>P</td>
<td>ALL : = [u,ksi] TRUE({u} P)</td>
<td></td>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.14</td>
<td>P</td>
<td>all : = nonempty(ALL)</td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.15</td>
<td>0</td>
<td>b2 : =</td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.16</td>
<td>b2</td>
<td>b3 : =</td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.17</td>
<td>b3</td>
<td>IMPL : = [u,TRUE(b2)] TRUE(b3)</td>
<td></td>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.18</td>
<td>b3</td>
<td>impl : = nonempty(IMPL)</td>
<td></td>
<td>bool</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Définition de limite d'une suite.

<table>
<thead>
<tr>
<th>No</th>
<th>Expression</th>
<th>Domaine</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>0 real := PN</td>
<td>type</td>
</tr>
<tr>
<td>2.2</td>
<td>0 r1 := real</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>r1 r2 := real</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>r2 distance := PN</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>r2 less := PN</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>0 null := PN</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>0 nat := PN</td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td>0 sequence := [x,nat]real</td>
<td></td>
</tr>
<tr>
<td>2.9</td>
<td>0 1 := PN</td>
<td></td>
</tr>
<tr>
<td>2.10</td>
<td>0 k1 := nat</td>
<td></td>
</tr>
<tr>
<td>2.11</td>
<td>k1 k2 := nat</td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td>k2 lessnat := PN</td>
<td></td>
</tr>
<tr>
<td>2.13</td>
<td>0 a := sequence</td>
<td></td>
</tr>
<tr>
<td>2.14</td>
<td>a 1 := real</td>
<td></td>
</tr>
<tr>
<td>2.15</td>
<td>1 eps := real</td>
<td></td>
</tr>
<tr>
<td>2.16</td>
<td>eps m0 := nat</td>
<td></td>
</tr>
<tr>
<td>2.17</td>
<td>m0 m := nat</td>
<td></td>
</tr>
<tr>
<td>2.18</td>
<td>m w := impl(lessnat(m0,m), less(distance(m a,1),eps))</td>
<td></td>
</tr>
<tr>
<td>2.19</td>
<td>m0 z := all(nat,[s,nat]w(s))</td>
<td></td>
</tr>
<tr>
<td>2.20</td>
<td>eps y := exists(nat,[t,nat]z(t))</td>
<td></td>
</tr>
<tr>
<td>2.21</td>
<td>eps q := impl(less(null,eps),y)</td>
<td></td>
</tr>
<tr>
<td>2.22</td>
<td>l lim := all(real,[t,real]q(t))</td>
<td></td>
</tr>
<tr>
<td>2.23</td>
<td>a conv := exists(real,[t,real]lim(t))</td>
<td></td>
</tr>
</tbody>
</table>

Note: The table represents the definitions and types of various entities in a formal context, with specific assignments and constraints specified.
3. Un lemme introduit comme axiome.

\[ x^1 := \text{real} \]
\[ y^1 := \text{real} \]
\[ \text{if} := \text{TRUE(less(null,y1))} \]
\[ \text{lemma} := \text{PN} \]
\[ \text{TRUE(less(distance(x1,x1),y1))} \]

4. Chaque suite constante est convergente.

\[ c := \text{real} \]
\[ p := [x,nat] c \]
\[ \text{delta} := \text{real} \]
\[ \text{abbrev 1} := \text{TRUE(less(null,delta))} \]
\[ \text{ass} := \text{abbrev 1} \]
\[ n_0 := \text{nat} \]
\[ n := \text{nat} \]
\[ \text{abbrev 2} := \text{TRUE(less(distance([n]p,c),delta))} \]
\[ \text{lessnat}(n_0,n) := \text{bool} \]
\[ w(p,c,delta,n_0,n) := \text{bool} \]
\[ \text{then 2}([u,TRUE(a)] \text{abbrev 2},[u,TRUE(a)] \text{then}) := \text{TRUE(aa)} \]
\[ \text{then 2}([s,nat] TRUE(aa(s)), [s,nat] b(s)) := \text{TRUE(z(p,c,delta,n_0))} \]
\[ \text{then 2}([x,abbrev 1] \text{TRUE}(y(p,c,delta)), [x,abbrev 1]d(x)) := \text{TRUE(y(p,c,delta))} \]
\[ \text{true}(2([s,real] TRUE(q(p,c,s)), [s,real] e(s)) := \text{TRUE(lim(p,c))} \]
\[ \text{true}(2([s,real] lim(p,s),c,f) := \text{TRUE(conv(p))} \]
Références:

1. N.G. de Bruijn: AUTOMATH, a language for mathematics.
   Report 68-WSK-05(1968) Technological University

2. N.G. de Bruijn: The mathematical language AUTOMATH, its usage,
   and some of its extensions.
   To be published in the Proceedings of the Symposium