Mathematics and computers; a revolution!

Herman Geuvers (thanks to Freek Wiedijk)

Radboud University Nijmegen
(Eindhoven University of Technology)

March 7 2017
Lustrum Symposium GEWIS

overview

some mathematical history

formalized mathematics

reliability of proof assistants

a large mathematical proof: Kepler conjecture

formal proofs in computer science

some mathematical history

issues in the foundations of mathematics (beginning 20th cent.)

Cantor had developed set theory as a general language/system to do all mathematics in
various questions came up

- what is mathematical truth?
- when is a mathematical argument/proof correct?
- what is existence?
- do abstract mathematical objects exist in reality?
- can something exist when we cannot construct it?
- can we just define anything we want?
- is mathematics consistent?
- is mathematics decidable?

example of a non-constructive existence proof

theorem there exist irrational x and y with x^y rational

x rational	$x \in \mathbb{Q}$
x irrational	$x \in \mathbb{R} \setminus \mathbb{Q}$

proof

$$\left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\cdot\sqrt{2})} = \sqrt{2}^{2} = 2$$

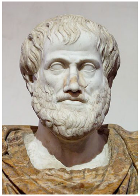
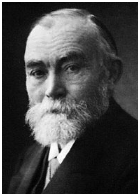
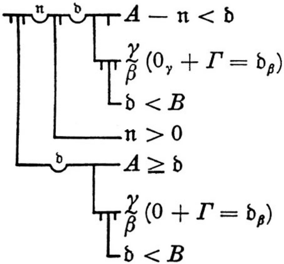
in case $\sqrt{2}^{\sqrt{2}}$ rational, take $x = y = \sqrt{2}$

in case $\sqrt{2}^{\sqrt{2}}$ irrational, take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$

QED

constructively, this is not a proof without proving whether $\sqrt{2}^{\sqrt{2}}$ is rational

(but: $\sqrt{2}^{\sqrt{2}}$ *is* irrational)

		
Aristoteles	Gottlob Frege	Begriffsschrift
fourth century before Chr.	nineteenth century	1879

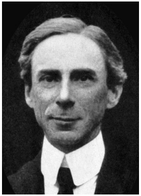
logic adrift in paradoxes: Russell's paradox

$$\{x \mid x \notin x\} \in \{x \mid x \notin x\}\,?$$

holds by definition exactly in the case that

$$\{x \mid x \notin x\} \notin \{x \mid x \notin x\}$$

is true if and only if (desda) it is false!



Bertrand Russell
1901

Frege's logic from the Begriffsschrift is inconsistent

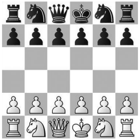
read: Apostolos Doxiadis & Christos Papadimitriou, *Logicomix*

logic adrift in paradoxes: the hypergame paradox

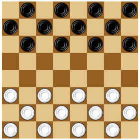
hypergame:

choose a
finite
game

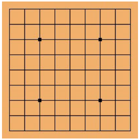
chess



checkers



go



. . hypergame?

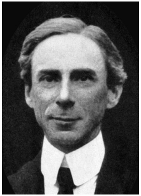
. . .

three schools

David Hilbert

formalism

'*game with symbols*'



Bertrand Russell

logicism

'*boils down to logic*'

L.E.J. Brouwer

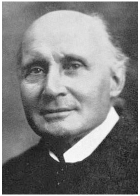
intuitionism

'*constructions in the mind*'

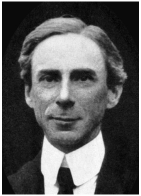
constructive mathematics

reconstruction of mathematics
coded in formal logic

three volumes
1910, 1912, 1913

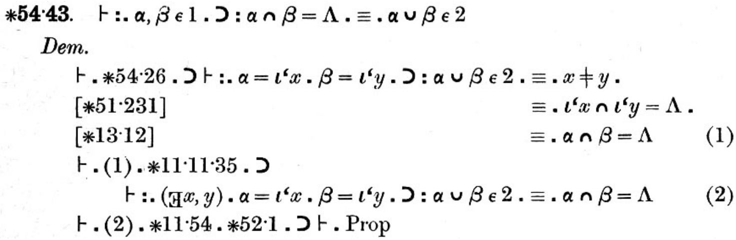


Alfred North Whitehead



Bertrand Russell

fragment of page 379:



$*54{\cdot}43.\quad \vdash :.\ \alpha, \beta \in 1 \,.\, \supset : \alpha \cap \beta = \Lambda \,.\, \equiv .\ \alpha \cup \beta \in 2$

Dem.

$\vdash .\ *54{\cdot}26 \,.\, \supset \vdash :.\ \alpha = \iota`x \,.\, \beta = \iota`y \,.\, \supset : \alpha \cup \beta \in 2 \,.\, \equiv .\ x \neq y \,.$

$[*51{\cdot}231] \qquad \equiv .\ \iota`x \cap \iota`y = \Lambda \,.$

$[*13{\cdot}12] \qquad \equiv .\ \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash .\ (1) \,.\, *11{\cdot}11{\cdot}35 \,.\, \supset$

$\vdash :.\ (\exists x, y) \,.\, \alpha = \iota`x \,.\, \beta = \iota`y \,.\, \supset : \alpha \cup \beta \in 2 \,.\, \equiv .\ \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash .\ (2) \,.\, *11{\cdot}54 \,.\, *52{\cdot}1 \,.\, \supset \vdash .\ \mathrm{Prop}$

Curry-Howard isomorphism


Haskell Curry


William Howard

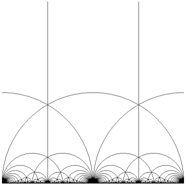
proofs	correspond with	computer programs
constructive mathematics		functional programming language

- a proof is an object (term) of a well-defined formal language
- when you prove constructively that something exists, you have a computer program to compute it.

N.G. de Bruijn

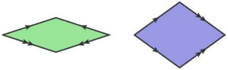
Nicolaas Govert ('Dick') de Bruijn
Den Haag 1918 – Nuenen 2012

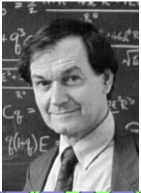
professor in mathematics at the
Eindhoven University of Technology

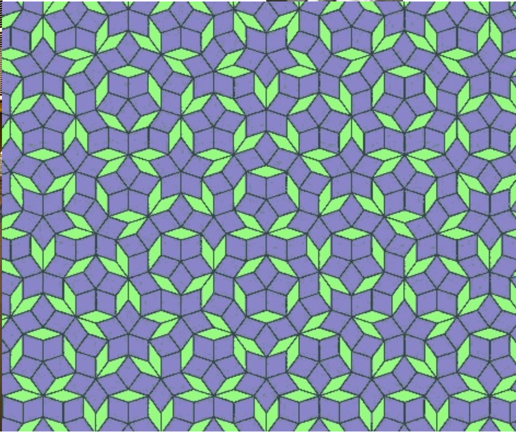


- modular forms
- BEST-theorem
  (de Bruijn, van Aardenne-Ehrenfest, Smith, Tutte)
  formula for the number of Euler-cycles in a graph
- asymptotic analysis

Penrose-tilings and quasicrystals







AUTOMATH

N.G. de Bruijn :

1967 : mathematical language AUTOMATH
use the computer to encode mathematics fully formally

```
        U0              := INN(1,U)                              : NAT
        T1              := 1TOP(1,U)                             : LESSIS(U0,1)
        T2              := ORE2(MORE(U0,1),IS(U0,1),SATZ24(U0),
                           SATZ10D(U0,1,T1))                     : IS(U0,1)
        TH1             := TRIS(1TO(1),U,OUTN(1,U0,T1),10,
                           ISOUTINN(1,U),ISOUTNI(1,U0,T1,1,
                           LESSISI2(1,1,REFIS(NAT,1)),T2))       : IS"E"(1TO(1),U,10)

-SINGLET

      @ 2               := PL(1,1)                               : NAT
        [X:NAT]

+PAIR

        [L:LESSIS(X,2)][N:NIS(X,2)]
        T1              := SATZ26(1,X,ORE1(LESS(X,2),IS(X,2),L,N)
                           )                                     : LE
        T2              := ORE2(MORE(X,1),IS(X,1),SATZ24(X),
                           SATZ10D(X,1,T1))                      : IS
    L @ TH1             := TH2"L.OR"(IS(X,1),IS(X,2),[T:NIS(X,2)]
                           T2(T))                                : OR
      @ TH2             := TH1"E.NOTIS"(NAT,<1>SUC,1,2,<1>AX3,
                           SATZ4E(1))                            : NI
```

encoding a complete mathematics book

Grundlagen der Analysis
mathematics book of 158 pages

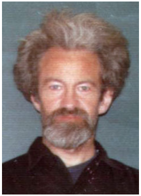
complete precise definition of
$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$
plus all operations on these sets

Edmund Landau
1929

Checking Landau's 'Grundlagen'
in the AUTOMATH system

PhD. Thesis

compete formal version
approximately 10,000 lines AUTOMATH



Bert Jutting
1976

Automath approach to formalising mathematics

- sets are types and formulas are types

  a formula is represented as the type of its proofs

- $t : A$ (term t is of type A) can be read as
  - t is an object in the set A (if A represents as set)
  - t is a proof of formula A (if A represents as formula)
- proof-checking is type-checking:
  to verify whether t is a correct proof, we type-check it.
- proof-checking is decidable, proof-finding is not

Coq

type theory

modern version of AUTOMATH and formalized mathematics

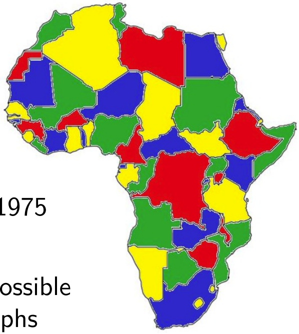
Gérard Huet

Thierry Coquand

Christine Paulin

pCIC = predicative Calculus of Inductive Constructions

the Coq proof assistant
INRIA, France
system used most at Nijmegen



- Francis Guthry, 1852
  question
- Percy Heawood, 1890
  proof of the **five color theorem**
- Kenneth Appel & Wolfgang Haken, 1975
  proof of the **four color theorem**

  computer computes gigantic set of possible colorings of a whole catalogue of graphs

  complicated computer program
  very long computation

  computer does *not* check the proof !
- Neil Robertson, Daniel Sanders, Paul Seymour, Robin Thomas, 1997
  polished version of the proof and the program

‘mathematical components’ project of INRIA-Microsoft

Coq version of the four color theorem

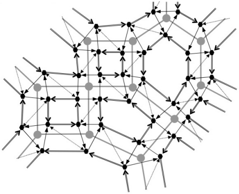
- formal version of the proof
- program correctness
  Coq as functional programming language

new technologies :

- mathematical language Ssreflect

  Feit-Thompson theorem

- hypermaps



Georges Gonthier
2005

Assia Mahboubi

example of a formal Coq proof

```
Lemma no_minSimple_odd_group (gT : minSimpleOddGroupType) : False.
Proof.
have [/forall_inP | [S [T [_ W W1 W2 defW pairST]]]] := FTtypeP_pair_cases gT.
  exact/negP/not_all_FTtype1.
have xdefW: W2 \x W1 = W by rewrite dprodC.
have pairTS := typeP_pair_sym xdefW pairST.
pose p := #|W2|; pose q := #|W1|.
have p'q: q != p.
  have [[[ctiW _ _] _ _ _ _] /mulG_sub[sW1W sW2W]] := (pairST, dprodW defW).
  have [cycW _ _] := ctiW; apply: contraTneq (cycW) => eq_pq.
  rewrite (cyclic_dprod defW) ?(cyclicS _ cycW) // -/q eq_pq.
  by rewrite /coprime gcdnn -trivg_card1; have [] := cycTI_nontrivial ctiW.
without loss{p'q} ltqp: S T W1 W2 defW xdefW pairST pairTS @p @q / q < p.
  move=> IH_ST; rewrite neq_ltn in p'q.
  by case/orP: p'q; [apply: (IH_ST S T) | apply: (IH_ST T S)].
have [[_ maxS maxT] _ _ _ _] := pairST.
have [[U StypeP] [V TtypeP]] := (typeP_pairW pairST, typeP_pairW pairTS).
have Stype2: FTtype S == 2 := FTtypeP_max_typeII maxS StypeP ltqp.
have Ttype2: FTtype T == 2 := FTtypeP_min_typeII maxS maxT StypeP TtypeP ltqp.
have /mmax_exists[L maxNU_L]: 'N(U) \proper setT.
  have [[_ ntU _ _] cUU _ _ _] := compl_of_typeII maxS StypeP Stype2.
  by rewrite mFT_norm_proper // mFT_sol_proper abelian_sol.
have /mmax_exists[M maxNV_M]: 'N(V) \proper setT.
  have [[_ ntV _ _] cVV _ _ _] := compl_of_typeII maxT TtypeP Ttype2.
  by rewrite mFT_norm_proper // mFT_sol_proper abelian_sol.
have [[maxL sNU_L] [maxM sNV_M]] := (setIdP maxNU_L, setIdP maxNV_M).
have [frobL sUH _] := FTtypeII_support_facts maxS StypeP Stype2 pairST maxNU_L.
have [frobM _ _] := FTtypeII_support_facts maxT TtypeP Ttype2 pairTS maxNV_M.
```

etcetera

HOL Light

LCF tradition (Milner):
LCF → HOL → HOL Light
Stanford, US → Cambridge, UK → Portland, US
Based on: higher order logic

John Harrison
proves correctness of floating point hardware at Intel
formalises mathematics in his spare time

very simple and elegant system
easy to extend (add your own tactics)
not user friendly

example of a formal HOL Light proof

$$\vec{w} \neq \vec{0} \;\wedge$$
$$\vec{u} \cdot \vec{w} = \vec{v} \cdot \vec{w} \;\Rightarrow$$
$$\angle(\vec{u} \times \vec{w}, \vec{u} \times \vec{w}) = \angle(\vec{u}, \vec{v})$$

John Harrison

```
let VECTOR_ANGLE_DOUBLECROSS = prove
 (`!u v w.
       ~(w = vec 0) /\ u dot w = &0 /\ v dot w = &0
       ==> vector_angle (u cross w) (v cross w) = vector_angle u v`,
  REPEAT GEN_TAC THEN ASM_CASES_TAC `u:real^3 = vec 0` THENL
   [ASM_REWRITE_TAC[vector_angle; CROSS_0]; ALL_TAC] THEN
  ASM_CASES_TAC `v:real^3 = vec 0` THENL
   [ASM_REWRITE_TAC[vector_angle; CROSS_0]; ALL_TAC] THEN
  STRIP_TAC THEN
  SUBGOAL_THEN `~(u cross w = vec 0) /\ ~(v cross w = vec 0)` ASSUME_TAC THENL
   [REPEAT(POP_ASSUM MP_TAC) THEN REWRITE_TAC[GSYM DOT_EQ_0] THEN VEC3_TAC;
    ALL_TAC] THEN
  ASM_SIMP_TAC[VECTOR_ANGLE_EQ] THEN
  SIMP_TAC[vector_norm; GSYM SQRT_MUL; DOT_POS_LE] THEN
  ASM_REWRITE_TAC[DOT_CROSS; REAL_MUL_LZERO; REAL_SUB_RZERO] THEN
  REWRITE_TAC[REAL_ARITH `(x * y) * (z * y):real = (y * y) * x * z`] THEN
  SIMP_TAC[SQRT_MUL; DOT_POS_LE; REAL_LE_SQUARE; REAL_LE_MUL] THEN
  SIMP_TAC[SQRT_POW_2; DOT_POS_LE; GSYM REAL_POW_2] THEN REAL_ARITH_TAC);;
```

reliability of proof assistants

why would we believe a proof assistant?

... a proof assistant is just another program ...

to attain the utmost level of reliability:

- precise description of the rules and the logic of the system.
- have a small "kernel": all proofs can be reduced to a small number of basic proof steps
  high level steps are defined in terms of the small basic ones.

LCF approach [Milner]:

- have an abstract data type of theorems `thm`
- only constants of type `thm` are the axioms
- only functions to `thm` are logical inference rules

Robin Milner

why would we believe a proof assistant?

...a proof assistant is just another program ...

other possibility to increase the reliability of the proof assistant:

De Bruijn criterion

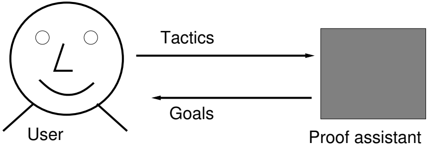
a proof assistant satisfies the De Bruijn criterion if

- it generates proof objects
- that can be checked independently of the system that created them
- using a simple program that a skeptical user can write him/herself

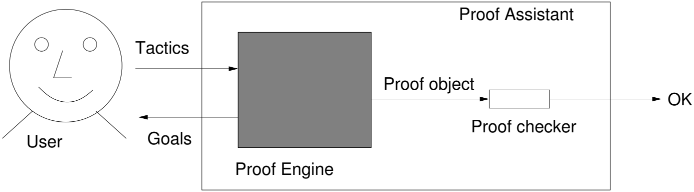
why would we believe a proof assistant?

De Bruijn criterion: separate the proof checker ("simple") from the proof engine ("powerful")

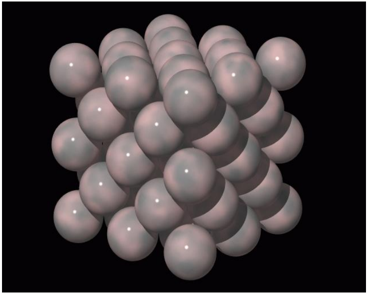
proof assistant (interactive theorem prover)



proof assistant with a small kernel that satisfies the De Bruijn criterion



Kepler conjecture



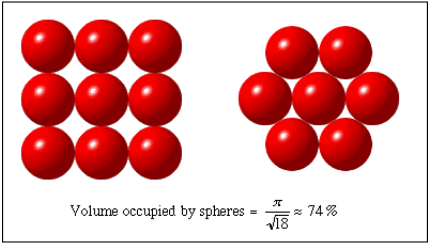
face-centric cubic ball packing

Johannes Kepler
1611

strena seu de nive sexangula
on the six-angled snowflake

Kepler conjecture

the most compact way of stacking balls of the same size is a pyramid.

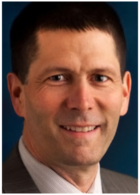


the Flyspeck project

- 1996: proof of the Kepler conjecture

  book of 334 pages

  giga bytes of data and
  days of computer calculations



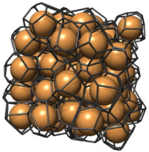
Tom Hales

- reviewers of the *Mathematische Annalen*:

  we cannot find mistakes, but
  too complicated to verify in full detail
  (reviewers did not study the programs)

- 2003: start the Flyspeck project =
  create a completely formal version of the proof

  HOL Light proof assistant
  (+ Isabelle proof assistant)



Voronoi cells

- 2014: formal proof of the Kepler conjecture completed
  impossible that there is still a mistake somewhere

Essential Computer Assistance in the Flyspeck formal proof

the proof of Hales rests on a number of computer calculations:

a. *a program that lists all 19.715 "tame graphs", that potentially may produce a counterexample to the Kepler conjecture.*

   this program was originally written in Java
   now, it is written and verified in Isabelle and exported to ML

b. *a computer calculation that verifies that a list of 43.078 linear programs are unsolvable.*

   each linear program in this list has about 100 variables and a similar list of equations.

c. *a computer verification that 23.242 non-linear equations with at most 6 variables hold.*

   this is the verification where originally interval-arithmetic was used.

the 23.242 non-linear equations with at most 6 variables typically look like this, with the variables ranging over specific intervals

$$\frac{\begin{array}{c}-x_1x_3 - x_2x_4 + x_1x_5 + x_3x_6 - x_5x_6 + \\ x_2(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6)\end{array}}{\sqrt{4x_2\left(\begin{array}{c}x_2x_4(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6) + \\ x_1x_5(x_2 - x_1 + x_3 + x_4 - x_5 + x_6) + \\ x_3x_6(x_2 + x_1 - x_3 + x_4 + x_5 - x_6) \\ -x_1x_3x_4 - x_2x_3x_5 - x_2x_1x_6 - x_4x_5x_6\end{array}\right)}} < \tan(\frac{\pi}{2} - 0.74)$$

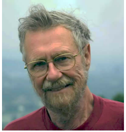
use computer programs to verify these inequalities.

Proving programs correct

John McCarthy (1927 – 2011)
1961, Computer Programs for Checking Mathematical Proofs

> *Proof-checking by computer may be as important as proof generation. It is part of the definition of formal system that proofs be machine checkable.*
>
> *. . .*
>
> *For example, instead of trying out computer programs on test cases until they are debugged, one should prove that they have the desired properties.*

- we have techniques to prove high level programs correct (Dijkstra, Hoare)




E. Dijkstra T. Hoare

$$\{\text{pre}\} \quad \texttt{program} \quad \{\text{post}\}$$

- that a `program` satisfies a `specification` is a formal mathematical statement that we can prove using a proof assistant
- first formalize the programming and specification language and their semantics
- similar techniques can be applied to hardware design

Holy Grail

> *'Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.'*

Bill Gates, 18 april 2002

- verifying an optimizing compiler from C to x86/ARM/PowerPC code
- implemented using Coq's functional language
- verified using using Coq's proof language

Xavier Leroy

why?

- your high level program may be correct, maybe you've proved it correct ...
- ... but what if it is compiled to wrong code?
- compilers do a lot of optimizations: switch instructions, remove dead code, re-arrange loops, ...
- for critical software the possibility of miscompilation is an issue

C-compilers are generally not correct
Csmith project *Finding and Understanding Bugs in C Compilers*,
X. Yang, Y. Chen, E. Eide, J. Regehr, University of Utah.

> *... we have found and reported more than 325 bugs in mainstream C compilers including GCC, LLVM, and commercial tools.*
> *Every compiler that we have tested, including several that are routinely used to compile safety-critical embedded systems, has been crashed and also shown to silently miscompile valid inputs.*

> *As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task.*

some other large formalization projects in Computer Science

- formalization of the C standard in Coq
  Krebbers and Wiedijk Nijmegen 2015.
- the ARM microprocessor
  proved correct in HOL4
  Anthony Fox University of Cambridge, 2002

Robbert Krebbers

- the L4 operating system,
  proved correct in Isabelle
  Gerwin Klein NICTA, Australia, 2009
  200,000 lines of Isabelle
  20 person-years for the correctness proof
  160 bugs before verification
  0 bugs after verification

Gerwin Klein

- Conference Interactive Theorem Proving, every paper is supported by a formalization

create large comprehensive reusable formal libraries

- formalize all of the bachelor undergraduate mathematics

automation

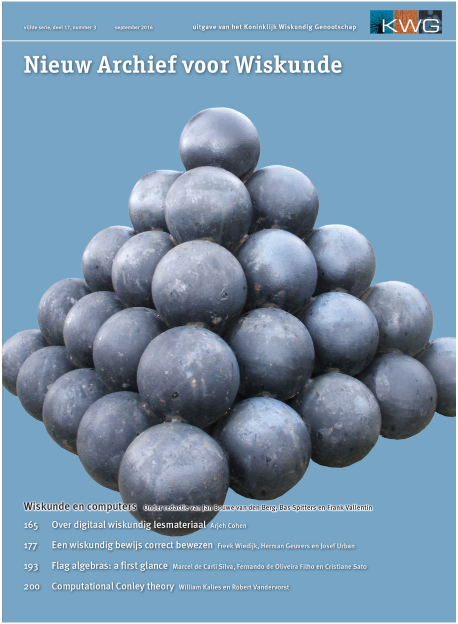
- combination of automated theorem proving and machine learning
  use machine learning to produce a hint database that can be fed to an automated theorem prover: the Hammer approach
- domain specific tactics and domain specific automation

Summarizing

- proof assistants for formal verification is becoming standard technology in computer science
- in mathematics there are more and more fields where the computer is indispensable for checking large proofs
  NB. one can prove that

  there will always be short formulas with large proofs

further reading:
Freek Wiedijk, Herman Geuvers, Josef Urban,
Een wiskundig bewijs correct bewezen: De meest efficiënte manier om bollen op te stapelen (in Dutch),
Nieuw Archief voor Wiskunde (NAW) 5/17 nr 3, september 2016, pp. 177-183.


vijfde serie, deel 17, nummer 3 september 2016 uitgave van het Koninklijk Wiskundig Genootschap

Nieuw Archief voor Wiskunde

Wiskunde en computers Onder redactie van Jan Bouwe van den Berg, Bas Spitters en Frank Vallentin

165 Over digitaal wiskundig lesmateriaal Arjeh Cohen

177 Een wiskundig bewijs correct bewezen Freek Wiedijk, Herman Geuvers en Josef Urban

193 Flag algebras: a first glance Marcel de Carli Silva, Fernando de Oliveira Filho en Cristiane Sato

200 Computational Conley theory William Kalies en Robert Vandervorst