Mathematician-friendly Proof Assistants

Henk Barendregt
Nijmegen University
The Netherlands

HAPPY BIRTHDAY BRUNO!

1. Computer Mathematics
2. Status quo
3. Case studies
4. Challenge

??

1. Computer Mathematics: goals

Mathematical activity: defining, computing, proving

Mathematical assistant helps human user:

Representing arbitrary mathematical notions	(defining)
Manipulating these	(computing)
Proving results about them	(proving)

in an impeccable way

Eventually to assist humans to learn and develop mathematics

At present an interesting foundational problem

1. Computer Mathematics: method

- Representing "computable" objects

  $\sqrt{2}$ becomes a symbol α

  $\alpha^2 - 2$ becomes 0 $\qquad\qquad$ $\alpha + 1$ cannot be simplified

- Representing "non-computable" objects

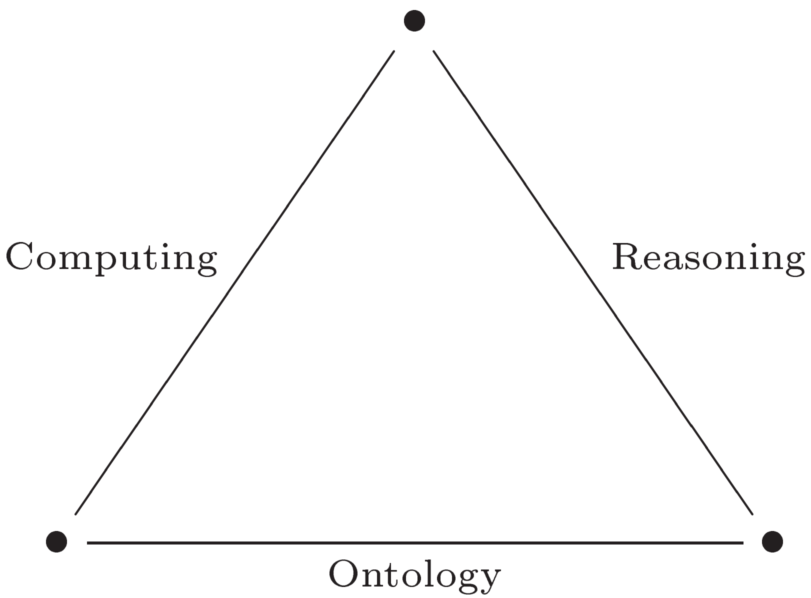
  Hilbert space H, again just a symbol

  $P(H) :=$ "*H is locally compact*" is not decidable

  But $\quad \vdash p :^1 P(H) \quad$ is decidable $\quad$ ($^1 p$ is a proof of $P(H)$)

But then we need formalized proofs

1. Computer Mathematics: choices



Ontology (what objects do there exist?)

Classical mathematics (before the 19-th century)
only needed a few fixed spaces

Modern mathematics needs a wealth of new spaces
and ample energy is devoted to the construction of these

Set Theory has the virtue that it unifies all needed concepts in one framework

Type Theory based on

- inductively defined data types with their
- recursively defined functions and closed under
- function spaces and dependent products

is an interesting alternative

Function space types

If A, B are types, then $A \to B$ is the type of functions from objects of type A into objects of type B.

$$\frac{a : A \quad f : (A \to B)}{(f\ a) : B} \qquad \frac{f : (A \to B) \quad g : (B \to C)}{(g \circ f) : (A \to C)}$$

Dependent products

$$\frac{\Gamma, n{:}A \vdash B(n) : \mathit{type}}{\vdash \Pi_{n:A}.B(n) : \mathit{type}}$$

Functional abstraction

$$\lambda x.f(x)$$

stands for the function $x \longmapsto f(x)$. For example, $g \circ f = \lambda x.g(f(x))$

Inductive types (freely generated data types)

Natural numbers `nat`

```
nat := 0 | S(nat)
nat := 0:nat | S:nat→nat
```

Primitive recursion over `nat`: we postulate an $f : \text{nat},\text{nat}^k \to \text{nat}$ such that

$$\begin{aligned} f(0,\vec{x}) &= g(\vec{x}) \\ f(S(n),\vec{x}) &= h(f(n,\vec{x}),n,\vec{x}) \end{aligned}$$

for $g : \text{nat}^k \to \text{nat}$, $h : \text{nat},\text{nat},\text{nat}^k \to \text{nat}$.

For example

$$\begin{aligned} 0+x &= x & 0*x &= 0 & 0! &= 1 \\ S(n)+x &= S(n+x) & S(n)*x &= (n*x)+x & (S(n))! &= n!*(n+1) \end{aligned}$$

Other data type: (binary) trees.

```
tree := leaf:nat→tree | p:tree,tree→tree
```

Primitive recursion over trees: given g, h we postulate F such that

$$\begin{array}{rcl} F(\mathrm{leaf}(n), \vec{x}) & = & g(n, \vec{x}) \\ F(p(t_1, t_2), \vec{x}) & = & h(F(t_1, \vec{x}), F(t_2, \vec{x}), t_1, t_2) \end{array}$$

For example

$$\begin{array}{rcl} \mathrm{mirror}(\mathrm{leaf}(n)) & = & \mathrm{leaf}(n) \\ \mathrm{mirror}(p(t_1, t_2)) & = & p(\mathrm{mirror}(t_2), \mathrm{mirror}(t_1)) \end{array}$$

No need to code such structures into numbers via the Chinese remainder theorem (Gödel)

First order rules for $\rightarrow$, & , $\vee$, $\neg$, $\Leftrightarrow$, $\forall x{\in}U$, $\exists x{\in}U$

Continuity: $\forall\epsilon{>}0\forall x\exists\delta\forall y\ldots$, uniform continuity: $\forall\epsilon{>}0\exists\delta\forall x,y\ldots$.

Second-order rules for $\forall X{\subseteq}U$, $\exists X{\subseteq}U$

An element x in a group G has torsion iff $\exists n{\in}\mathbb{N}.x^n = e$

This definition is not allowed in pure first order logic.

In second-order logic:

$$\forall X{\subseteq}G[x{\in}X \;\&\; (\forall y{\in}X.xy{\in}X) \;\Rightarrow\; e{\in}X]$$

Higher-order A topology $\mathcal{O}$ on U is an element of $\mathcal{P}(\mathcal{P}(U))$

Third order statement:

There exists a topology on U such that F is continuous

Brouwer: Aristotelian logic is unreliable

It may promise existence without being able to give a witness

$$\vdash \exists n{\in}\mathbb{N}.P(n), \text{ but } \not\vdash P(0), \not\vdash P(1), \ldots$$

Heyting: charted Brouwer's logic

Gentzen: gave it a nice form

Example of such a P

$$P(n) \;\Leftrightarrow\; (n = 0 \;\&\; \mathrm{GRH}) \vee (n = 1 \;\&\; \neg\mathrm{GRH})$$

1. Logic: "Intuitionism has become technology" (Constable)

THEOREM-CLASSICAL [No effectiveness]
For every ideal $I = (p_1, \ldots, p_n)$ *in* $\mathbb{Q}[\vec{x}]$ *there is a Gröbner basis* P *of* I*.*

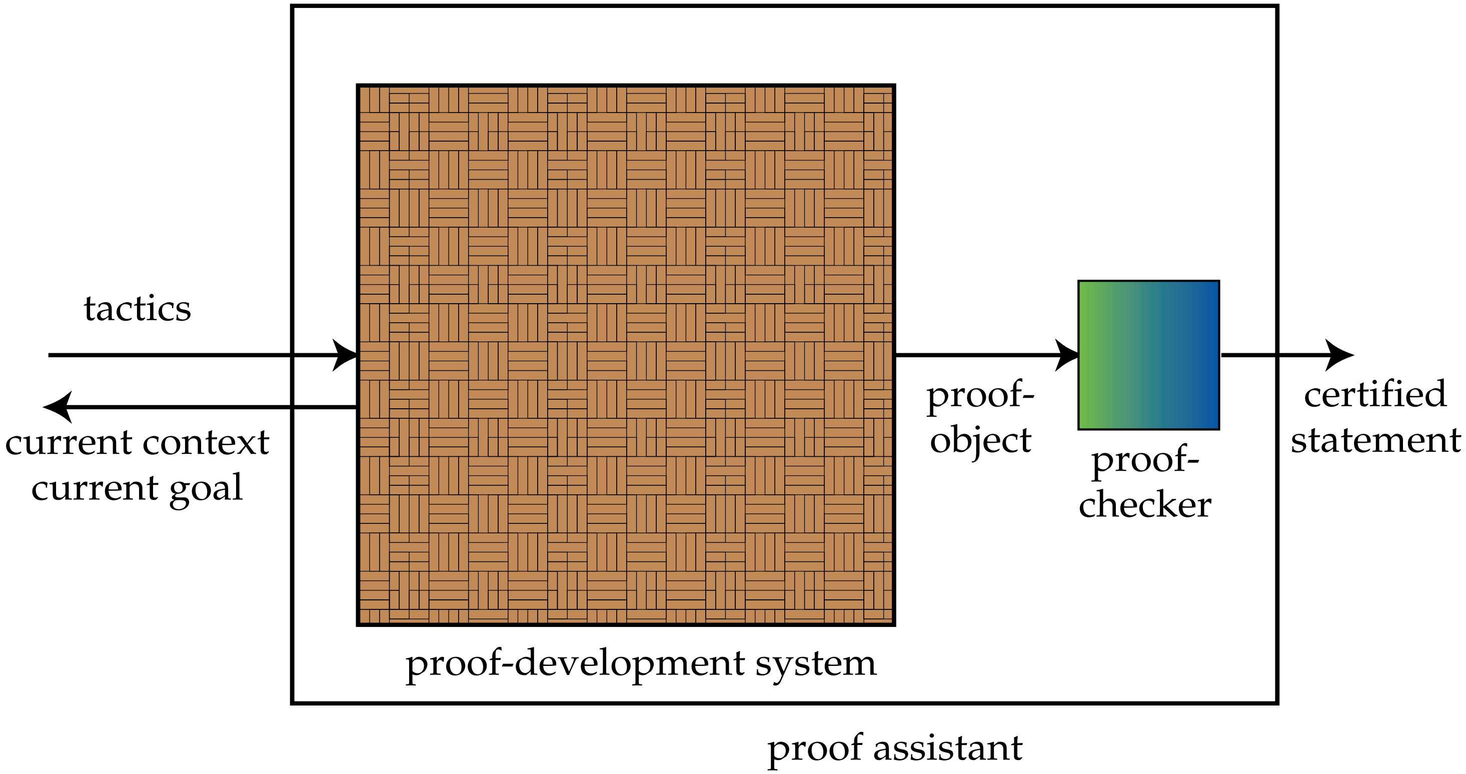
THEOREM-CLASSICAL [This does not give the theorem]
There is a Turing machine TM such that for every ideal $I = (p_1, \ldots, p_n)$ *in* $\mathbb{Q}[\vec{x}]$ *the result* $TM(p_1, \ldots, p_n) = P$ *is a Gröbner basis of* I*.*

THEOREM-CLASSICAL [We do not always want to be explicit]
Let $\mathrm{TM} = \langle\langle q_0, ..\rangle, \ldots\rangle$*. Then TM is a Turing machine and for every ideal* $I = (p_1, \ldots, p_n)$ *in* $\mathbb{Q}[\vec{x}]$ *the result* $TM(p_1, \ldots, p_n) = P$ *is a Gröbner basis of* I*.*

THEOREM-CONSTRUCTIVELY. [Effectiveness]
For every ideal $I = (p_1, \ldots, p_n)$ *in* $\mathbb{Q}[\vec{x}]$ *there is a Gröbner basis* P *of* I*.*

Building an intuitionistic library provides certified tools

Assistance



Reliability?

The de Bruijn criterion: have a small checker.

2. Status Quo: some systems

- Mizar based on classical ZFC

- Isabelle-HOL based on classical higher order logic with λ-terms

- Coq based on impredicative intuitionistic Type Theory

2. Status Quo: Computing vs. proving

The Chinese-Babylonian vs Greek tradition

Chinese-Babylonian: computing

Greek: proving

It took until the 19-th century until the two traditions fully came together

20-th century: a temporary separation

- Computer algebra: Chinese-Babylonian
- Formalization of proofs: Greek

21-st century: synthesis is emerging

How does one give formal proofs of

- Computations

$$\begin{aligned}(xy - x^2 + y^2)(x^3 - y^3 + z^3) \quad = \quad & x^4y - xy^4 + xyz^3 - x^5 + \\ & x^2y^3 - x^2z^3 + y^2x^3 - y^5 + y^2z^3.\end{aligned}$$

  It is important to formally prove computations, not just for computational statements, but also for statements involving *intuition*

- Intuition

  Let $f : \mathbb{R} \to \mathbb{R}$ be defined by

$$f(x) = \frac{e^x + e^{-x}}{2} + e^{\sin^2 x} + e^{\cos^2 x}.$$

  Then f is continuous.

The Poincaré Principle:

If $f(a) = b$, via a computation, then $\vdash f(a) = b$ axiomatically

The class $\mathcal{P}$ of f's for which this is postulated may vary

$\mathcal{P} = \emptyset$	(Isabelle-HOL: ephemeral proofs)
$\mathcal{P} = \{f \mid \text{prim. rec. over a data type}\}$	(Coq)
$\mathcal{P} = \{f \mid f \text{ is representable in a CAS}\}$	(PVS)

The Poincaré Principle is in tension with the de Bruijn criterion

2. Status Quo: sliding styles

The Poincaré Principle:

If $f(a) = b$, via a computation, then $\vdash f(a) = b$ axiomatically

The class $\mathcal{P}$ of f's for which this is postulated may vary

$\mathcal{P} = \emptyset$	(Isabelle-HOL: ephemeral proofs)
$\mathcal{P} = \{f \mid \text{prim. rec. over a data type}\}$	(Coq)
$\mathcal{P} = \{f \mid f \text{ is representable in a CAS}\}$	(PVS)

The Poincaré Principle is in tension with the de Bruijn criterion

2. Status Quo: using the Poincaré Principle

$$\begin{tikzcd}
\log a \arrow[r, "2\cdot_"] \arrow[d, "e^{-}"'] & \log b \arrow[d, "e^{-}"] \\
a \arrow[r, "\text{square}"] & b
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
a^{+} \arrow[r, "f"] \arrow[d] & b^{+} \arrow[d] \\
a \arrow[r, "F"] & b
\end{tikzcd}$$

Logarithms $\qquad\qquad$ $f \in \mathcal{P}$

2. Status Quo: using reflection

In our case the map $+$ is not the logarithm but usually of a syntactic coding nature (reflection): $((x+y)^2)^+ = \mathtt{sq}(\mathtt{plus\ var_x\ var_y})$ and f is something like a `simplify` function on these syntactic expressions. The role of the exp function is played by the semantic function $[\![\ ]\!]$.

$$\begin{tikzcd}
\mathtt{times(minus\ x\ y)(plus\ x\ y)} \arrow[r, "\mathtt{smpl.}"] \arrow[d, "{[\![\ ]\!]}"'] & \mathtt{minus(sq\ x)(sq\ y)} \arrow[d, "{[\![\ ]\!]}"] & \in L \\
(x-y)(x+y) \arrow[r, "=_{\mathtt{provably}}"] & (x^2-y^2) & \in R
\end{tikzcd}$$

In order to apply this freely one has to show

$$\forall e{:}L.[\![e]\!] = [\![\mathtt{smpl\ e}]\!]$$

once and for all

Formalized in Coq (intuitionistic proofs)

THEOREM 1.
[Formalization: Geuvers, Wiedijk, Zwanenburg, Pollack, Niqui]
Every non-constant polynomial $p(x)$ *over* $\mathbb{C}$ *has a* root x.

THEOREM 2. Collaboration between Coq and GAP
[Formalization: Oostdijk, Caprotti, Elbers]
The number 90262580833849968604493660721423078019 63 *is a prime.*
(Based on Fermat's little theorem and Pocklington.)

THEOREM 3.
[Formalization: Capretta]
Correctness of the Fast Fourier Transform.

THEOREM 4. [Formalization: Person, Théry]
Correctness of an efficient Gröbner base algorithm.

THEOREM 5. [Formalization: Cruz-Félipe]
Fundamental theorem of calculus

THEOREM 6. [Formalization: Danos, Gonthier, Werner]
Main lemma for the four colour theorem.

For this, Coq needed an overdrive: compilation rather than interpretation

This compiler was proved correct in the simpler version of Coq

	romantic	Cool	Super cool
Maths	Human mind	Coq	Compiled Coq
Biology	Human eye	Light microscope	Electronic microscope

4. Challenge: to construct a mathematician-friendly assistant

And to formalize substantial parts of mathematics Start with undergraduate mathematics

Needed

Libraries: theories, algorithms.

Estimated development time: 140 manyear (7M$)

4. Challenge: Style "λ-term"

THEOREM. Euclid : $\forall d{>}0, n\, \exists q, r[r{<}d \;\&\; n = qd + r]$.

Proof.

```
[d:nat; p:(d>0)]
 [P:=[n:nat]
  (EX q:nat | (EX r:nat | (lthan r d)/\n=(plus (times d q) r)))]
   (cv_ind P
    [n:nat; ih:(before n P)]
     [H:=(ltgeq n d)]
      (or_ind (lthan n d) (geq n d)
       (EX q:nat | (EX r:nat | (lthan r d)/\n=(plus (times d q) r)))
        [H0:(lthan n d)]
         (ex_intro nat
          [q:nat](EX r:nat | (lthan r d)/\n=(plus (times d q) r))
           0 (ex_intro nat
            [r:nat](lthan r d)/\n=(plus (times d 0) r) n
             (conj (lthan n d) n=(plus (times d 0) n) H0
              (eq_ind_r nat (times 0 d) [n0:nat]n=(plus n0 n)
               (req nat n) (times d 0) (times_com d 0)))))
                [H0:(geq n d)] [n':=(monus n d)]
                 [H1:=(ltm n d (leseq_trans one d n p H0) p)]
                  [H2:=(ih n' H1)](ex_ind nat [q:nat]
                  (EX r:nat | (lthan r d)/\nn=(plus (times d q) r))
                   (EX q:nat |
                    (EX r:nat | (lthan r d)/\n=(plus (times d q) r)))
                     [q':nat;
                      H3:(EX r:nat|(lthan r d)/\nn=(plus(times d q')r))]
                       (ex_ind nat
                        [r:nat](lthan r d)/\n'=(plus (times d q') r)
(EX q:nat |(EX r:nat|(lthan r d)/\n=(plus(times d q)r)))
 [r':nat; H4:((lthan r' d)/\n'=(plus (times d q') r'))]
  (and_ind (lthan r' d) n'=(plus (times d q') r')
   (EX q:nat|(EX r:nat|(lthan r d)/\n=(plus (times d q) r)))
    [H5:(lthan r' d); H6:(n'=(plus (times d q') r'))]
     (ex_intro nat [q:nat]
      (EX r:nat | (lthan r d)/\n=(plus (times d q) r))
       (suc q') (ex_intro nat
        [r:nat]
         (lthan r d)/\n=(plus (times d (suc q')) r) r'
          (conj (lthan r' d)
           n=(plus (times d (suc q')) r') H5
            [H7:=(f_equal nat nat (plus d) n'
            (plus (times d q') r') H6)]
             [H8:=(eq_ind_r nat (plus (monus n d) d)
              [n0:nat]n0=n (pdmon n d H0)
               (plus d (monus n d))
                (plus_com d (monus n d)))]
                 (eq_ind nat (plus d n')
                  [n0:nat]n0=(plus (times d (suc q')) r')
                   (eq_ind_r nat
                    (plus d (plus (times d q') r'))
                     [n0:nat]
                      n0=(plus (times d (suc q')) r')
                      (compute q' r' d)
                       (plus d n')H7) n H8))))H4)H3)H2)H)). QED
```

2. Challenge: Style "Script"

```
Theorem. Euclid : (d:nat)(0<d)->(n:nat)(EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r)))).
Proof.
Intros d p.
LetTac P:=[n:nat]
(EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r))))
Apply '(cv_ind P).
Intro n.
Intro ih.
Unfold before in ih.
Assert ((lthan n d)\/(geq n d)).
Apply ltgeq.
Unfold P.
Intuition.
Exists 0.
Exists n.
Split.
Try Assumption.
Rewrite -> times_com.
Try Assumption.
Simpl.
Apply req.
LetTac nn := (n[-]d).
Assert (lthan nn n).
Unfold nn.
Apply ltm.
Intuition.
Apply (leseq_trans one d n).
Intuition.
Intuition.
Intuition.
Assert (P nn).
Apply ih.
Try Assumption.
Unfold P in H1.
Pick H1 qq.
Pick H2 rr.
Intuition.
Exists (suc qq).
Exists rr.
Intuition.
Assert ((d[+]nn)=(d[+](d[*]qq[+]rr))).
Apply (f_equal ? ? (plus d)).
Try Assumption.
Assert ((d[+]nn)=n).
Unfold nn.
R plus_com.
Apply  pdmon.
Try Assumption.
Rewrite <- H4.
Rewrite -> H1.
Apply compute.
Qed.
```

4. Challenge: Style "Script"

```
Theorem. Euclid :
(d:nat)(d>0)->(n:nat)(EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r)))).

Proof.
Intros d p.
LetTac P:=[n:nat](EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r))))
Apply '(cv_ind P).
Intro n.
Intro ih.
....
....
```

4. Challenge: Style "Mathmode"

```
Theorem. Euclid : (d:nat)(d>0)->(n:nat)(EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r)))).
Proof.
Given d.
assume 'd>0.
towards '(n:nat)(EX q:nat|(EX r:nat|(r<d/\(n=(d[*]q[+]r))))).
LetTac P:=[n:nat]
((EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r))))).
We must prove (n:nat)(P n).
Apply (cv_ind P).
Given n.
assume '(before n P).
towards '(P n).
Remember_in before H0.
We_have '((n<d) \/ (n>=d)).
By ltgeq.
Intuition.
remember P.
Take 0.
Take n.
Split.
Trivial.
RR times_com.
Simpl.
By req.
LetTac nn := (n[-]d).
Then '( nn <n).
remember nn.
By ltm.
Apply (leseq_trans one d n).
Assumption.
Assumption.
We_have '(P nn).
Unfold P in H3.
pick qq H3.
pick rr H4.
Exists (suc qq).
Exists rr.
Intuition.
R times_com.
Simpl.
R times_com.
Assert ((d[+]nn)=n).
Unfold nn.
R plus_com.
Apply  pdmon.
Trivial.
RR H3.
R H5.
R plus_com.
RR plus_ass.
We_have '(d[*]qq[+]d[+]rr)=(d[*]qq[+](d[+]rr)).
R plus_ass.
By req.
RR plus_ass.
Apply (f_equal ? ? (plus (d[*]qq))).
R plus_com.
By req.
Qed.
```

4. Challenge: Style "Math mode"

```
Theorem. Euclid :
(d:nat)(d>0)->(n:nat)(EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r)))).

Proof.
Given d.
assume 'd>0.
towards '(n:nat)(EX q:nat|(EX r:nat|(r<d/\(n=(d[*]q[+]r))))).
LetTac P:=[n:nat]((EX q:nat|(EX r:nat|(r<d/\n=(d[*]q[+]r))))).
We must prove (n:nat)(P n).
Apply (cv_ind P).
Given n.
assume '(before n P).
towards '(P n).
....
....
```

4. Challenge: "Best Mathematical Style"

THEOREM. Let $d \in \mathbb{N}$ with $0 < d$. Then

$$\forall n \in \mathbb{N}\, \exists q, r \in \mathbb{N} [r < d \,\&\, n = qd + r].$$

PROOF. Let $d \in \mathbb{N}$ with $0 < d$ be given.
Write $P(n) := \exists q, r \in \mathbb{N}.[r < d \,\&\, n = qd + r]$.
We will show

$$\forall n \in \mathbb{N}. P(n)$$

by course of value induction. So assume

$$\forall k < n. P(k), \qquad \text{(ih)}$$

in order to prove $P(n)$.
If $n < d$, then we can take $q = 0, r = n$.
If on the other hand $n \geq d$, define $n_1 = n \dot{-} d$.
Then $n_1 < n$ by ltm. Therefore $P(n_1)$ by (ih).
Hence for some $q_1, r_1 \in \mathbb{N}$ one has $r_1 < d \,\&\, n_1 = q_1 d + r_1$.
Take $q = q_1 + 1, r = r_1$. Then $r < d$ and

$$\begin{aligned} n &= d + (n \dot{-} d), && \text{by lemma pdmon,} \\ &= d + n_1 \\ &= d + q_1 d + r_1 \\ &= (q_1 + 1)d + r_1, && \text{by computation,} \\ &= qd + r. && \text{QED} \end{aligned}$$

4. Challenge

Mathematician-friendly systems for Computer Mathematics exist

It will take 10 to 50 years to build them

Logo