The Challenge of Computer Mathematics

Henk Barendregt
Nijmegen University
The Netherlands

1. Computer Mathematics: what?

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

1. Computer Mathematics: why?

Reasons for Computer Mathematics

- Highest degree of reliability
- Integration of proving and computing
- Certified library of theorems and algorithms
- Dependencies easy to track
- Beauty

Eventually to assist humans to learn and develop mathematics

At present an interesting foundational problem

Spin-off to computer science (actually IT): reliable hardware & software

1. Computer Mathematics: how?

- Representing "computable" objects

$\sqrt{2}$ becomes a symbol α

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

- Representing "non-computable" objects

Hilbert space H, again just a symbol

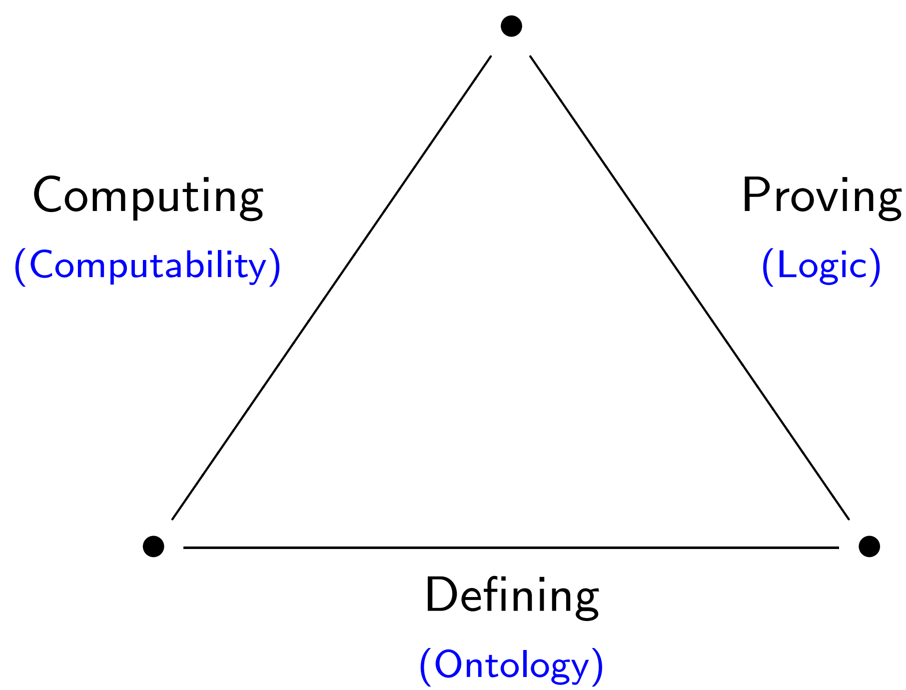
$P(H) :=$ "H *is locally compact*" is not decidable

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

But then we need formalized proofs

2. History: The mathematical enterprise

Mathematical activity: defining, computing, proving



2. History: logic, the axiomatic-deductive method

Aristotle (384-322 BC)

- The axiomatic method

objects	properties
primitive	axioms
defined	derived

defining proving
computing

- The quest for logic: try to chart reasoning

  (finished by Frege 1879; proved complete by Gödel in 1930)

- Poof-checking vs theorem proving

2. History: Computing vs. proving

Abstract history of mathematics 4000 BC – 2100 AD

Computing		Proving
Egyptians, Chinese		Thales
Babylonians		Eudoxos, Euclid
	Archimede	
	al-Khowârizmî	
Leibniz		Newton
Euler		
	Cauchy	
...	...	...
Computer Algebra		Proof-checkers
	Mathematical Assistants	
...	...	...

3. Foundations (ontology): Set theory

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

Postulated are

$$
\begin{array}{rcl}
\mathbb{N} & & \\
A, B & \mapsto & A \times B \\
A & \mapsto & \{X \mid X \subseteq A\} = \mathcal{P}(A) \\
A & \mapsto & \{a{\in}A \mid P(a)\} \\
A & \mapsto & \{F(a) \mid a{\in}A\}
\end{array}
$$

Giving monsters like $\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{P}^n(\mathbb{N}))$

3. Foundations (ontology): Type Theory

Type Theory is an interesting alternative to set theory

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

Function and Product Types

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

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

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

Recursively defined functions

given n_0:`nat`, h:`nat,nat⟶nat` we postulate an f:`nat⟶nat` such that

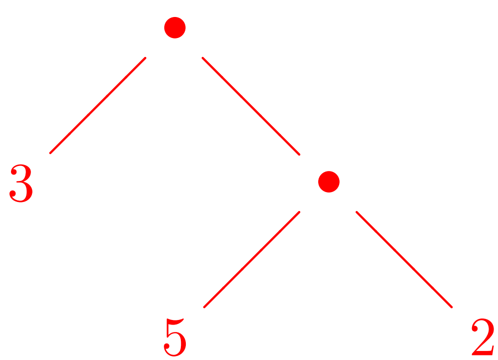
$$\begin{aligned} f(0) &= n_0 \\ f(S(n)) &= h(f(n), n) \end{aligned}$$

For example

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

Other data type: (binary) trees.

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




```
p(leaf(3),p(leaf(5),leaf(2)))
```

Primitive recursion over trees: we postulate functions like

$$\begin{aligned} \text{mirror}(\text{leaf}(n)) &= \text{leaf}(n) \\ \text{mirror}(p(t_1, t_2)) &= p(\text{mirror}(t_2), \text{mirror}(t_1)) \end{aligned}$$

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

3. Foundations (logic): first-, second- and higher-order logic

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 $\forall X{\subseteq}U$, $\exists X{\subseteq}U$

A sentence is a theorem if it belongs to all sets containing the axioms that are closed under deductions.

This definition is not allowed in pure first order logic.

In second-order logic:

$$t{\in}S \;\Leftrightarrow\; \forall X{\subseteq}S[axioms \subseteq X \;\&\; (\forall x, y, z.x, y{\in}X \;\&\; x, y \vdash z) \;\Rightarrow\; z{\in}X] \;\Rightarrow\; t{\in}X$$

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

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

3. Foundations (logic): Intuitionism

Brouwer: Aristotelian logic is unreliable

It may promise existence without being able to give a witness

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

Heyting: charted Brouwer's logic

Gentzen: gave it a nice form

Example of such an A

$$A(n) \Leftrightarrow (n = 0 \;\&\; \mathsf{P}{\neq}\mathsf{NP}) \vee (n = 1 \;\&\; \mathsf{P}{=}\mathsf{NP})$$

3. Foundations (logic): "Intuitionism has become technology" (Constable)

THEOREM-CLASSICAL [No effectiveness]
For every non-deterministic finite automaton (NDFA) $\mathcal{M}$ *there is a DFA* $\mathcal{M}'$ *such that* $L(\mathcal{M}) = L(\mathcal{M}')$.

THEOREM-CLASSICAL [This does not give the theorem]
There is a Turing machine TM such that for every NDFA $\mathcal{M}$ *the result TM*$(\mathcal{M})$ *is a DFA with the same language.*

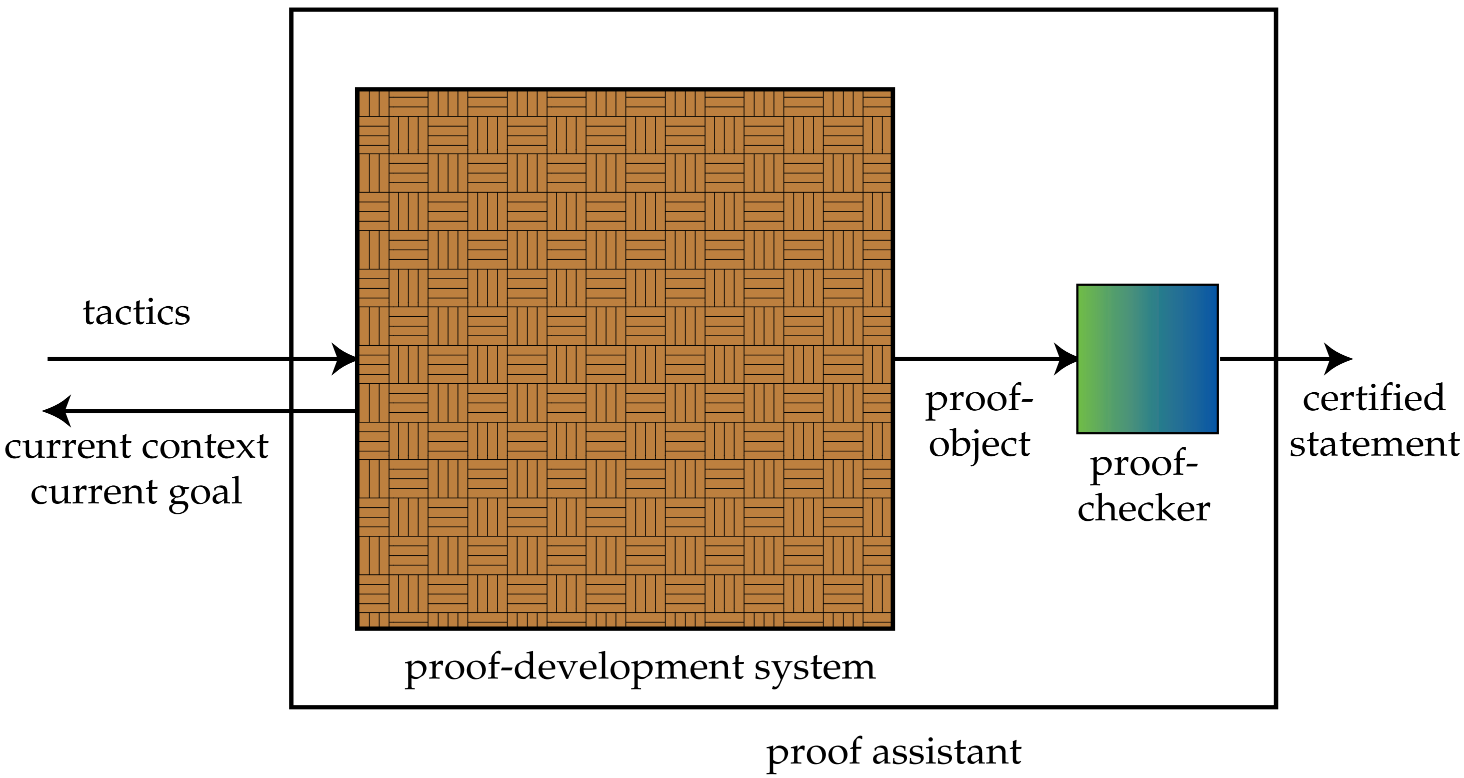
THEOREM-CLASSICAL [We do not always want to be explicit]
Let $\mathrm{TM} = \langle\langle q_0, ..\rangle, \ldots\rangle$. *Then TM is a Turing machine and for every NDFA* $\mathcal{M}$ *the result TM(*$\mathcal{M}$*) is a DFA with the same language.*

THEOREM-CONSTRUCTIVELY. [Effectiveness]
For every NFDA $\mathcal{M}$ *there exists a FDA* $\mathcal{M}'$ *having the same language*

Building an intuitionistic library provides certified tools

4. Status quo: Proof-assistants

Assistance



Reliability?

The de Bruijn criterion: have a small checker.

4. Status quo: some systems

- Mizar based on classical ZFC set theory
- Isabelle-HOL based on classical higher order logic with λ-terms
- Coq based on impredicative intuitionistic Type Theory

5. Proofs by computation

Goal to prove

$$A(t)$$

Full generalization

First try to prove

$$\forall x.A(x)$$

obtaining $A(t)$ *a fortiori*

Example

$$10^9 + 9^{10} = 9^{10} + 10^9$$

is proved best by first proving

$$\forall x, y{\in}\mathbb{N}.x + y = y + x$$

5. Proofs by computation

Goal to prove

$$A(t)$$

Pattern generalization

Strategy: write $t = f(s)$ with $s{\in}L$

and try to prove

$$\forall x{\in}L.A(f(x))$$

giving $A(f(s))$, hence $A(t)$.

This method is particularly powerful if combined with reflection.

But we need to prove $f(s) = t$.

5. Proofs by computation

How does one give formal proofs of

- Computations

$$(xy - x^2 + y^2)(x^3 - y^3 + z^3) \quad = \quad x^4y - xy^4 + xyz^3 - x^5 + x^2y^3 - x^2z^3 + y^2x^3 - y^5 + y^2z^3.$$

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

- Intuition

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

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

Then f is continuous.

5. Proofs by computation

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

5. Proofs by computation

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

5. Proofs by computation: using the Poincaré Principle

$$\begin{array}{ccc} \log a & \xrightarrow{2 \cdot \_} & \log b \\ {\scriptstyle e^{-}}\downarrow & & \downarrow{\scriptstyle e^{-}} \\ a & \xrightarrow{\text{square}} & b \end{array} \qquad\qquad \begin{array}{ccc} a^{+} & \xrightarrow{f} & b^{+} \\ \downarrow & & \downarrow \\ a & \xrightarrow{F} & b \end{array}$$

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

5. Proofs by Computation: using reflection

Here the map $+$ is not the logarithm but usually of a syntactic nature (reflection): $((x+y)(x-y))^{+} = \texttt{times}(\texttt{minus x y})(\texttt{plus x y})$ and f is something like a `simplify` function on these syntactic expressions. The role of the exp function is played by the semantic function $[\![\ ]\!]$.

$$\begin{array}{ccccc}
\texttt{times}(\texttt{minus x y})(\texttt{plus x y}) & \xrightarrow{\texttt{smpl.}} & \texttt{minus}(\texttt{sq x})(\texttt{sq y}) & & \in L \\
{\scriptstyle [\![\ ]\!]}\big\downarrow & & \big\downarrow{\scriptstyle [\![\ ]\!]} & & \\
(x-y)(x+y) & \xrightarrow{=_{\texttt{provably}}} & (x^2-y^2) & & \in R
\end{array}$$

In order to apply this freely one has to show

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

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 $x \in \mathbb{C}$.

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-Filipe]
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

7. 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
```

7. 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.
```

7. 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.
....
....
```

7. Challenge: "Best Mathematical Style"

THEOREM. Let $d \in \mathtt{nat}$ with $0 < d$. Then

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

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

$$\forall n \in \mathtt{nat}. P(n)$$

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 $q_1, r_1 \in \mathtt{nat}$ one has $r_1 < d \;\&\; n_1 = q_1 d + r_1$.
Take $q = q_1 + 1, r = r_1$. Then $r < d$ and

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

7. Challenge: Style “Mathmode” (M. Giero)

```
Lemma Euclid  : (d:nat)(O<d)->(n:nat)(EX q:nat|((EX r:nat|(r<d)/\n=((d[x]q)[+]r)))).
Proof. MLet d be nat. Assume     (O < d) (A4).
LetTac P:=[n:nat]((EX q:nat|(EX r:nat|((r<d)/\n=((d[x]q)[+]r))))).
Claim ((n:nat)(before n P)->(P n)) (A1).
  MLet n be nat. Assume (before n P) (A6).
  Case 1 (n<d) (A2).
    Take zero and prove (EX r:nat|r<d/\n=d[x]zero[+]r).
    Take n and prove (n<d/\n=d[x]zero[+]n).
    Done (n<d) [by A2].
    Done (n=d[x]zero[+]n) [by times_com].
  Case 2 (n>=d) (A5).
    Claim ( (monus n d) <n).
      Have (O<n) [by A4, A5, lt_le_imp_lt].
    Hence claim done [by A4, pos_imp_mon_lt].
    Then (P (n-,d)) [by A6].
    Then consider q such that
       ([q:nat](EX r:nat|r<d/\n-,d=d[x]q[+]r)).
    Then consider r such that
      ([r:nat](r<d/\n-,d=d[x]q[+]r)) (A8).
    Take (S q) and prove (EX r:nat|r<d/\n=d[x](S q)[+]r).
    Take r and prove (r<d/\n=d[x](S q)[+]r).
    Firstly (r<d) [by A8].
    Secondly we have
         n  =  ((n-,d)[+]d)              [by ge_imp_mon_plus_eq, A5].
            eqr (d[x]q[+]r[+]d)          [by A8]. Hence
            eqr (d[x](S q)[+]r)          [by compute].
  End_cases [by dichotomy].
So we have proved (A1).
Finally we need to prove ((n:nat)(P n)).
Done [by cv_ind, A1]. Qed.
```

7. Challenge

Mathematician-friendly systems for Computer Mathematics can be built

Needed

mathematical interface (M-mode)
libraries
certified tools

It will take 150 manyear ($\sim$ 5 M UKpound) to build them
for the topics of a master's in mathematics