# formal proof between mathematics and computer science

Freek Wiedijk

Radboud University Nijmegen
The Netherlands

June 13, 2015

## the questions

- do formal proof in mathematics and formal proof in computer science require different approaches?

## the questions

- do formal proof in mathematics and formal proof in computer science require different approaches?

- what are the ingredients of a formal proof system?
  (in case you want to build one yourself)

## the questions

- do formal proof in mathematics and formal proof in computer science require different approaches?

- what are the ingredients of a formal proof system?
  (in case you want to build one yourself)

- what are the unsolved problems in formal proof?
  (in case you think you can do better than the state of the art)

## the questions

- do formal proof in mathematics and formal proof in computer science require different approaches?
- what are the ingredients of a formal proof system? (in case you want to build one yourself)
- what are the unsolved problems in formal proof? (in case you think you can do better than the state of the art)
- can one be certain a formal proof is fully correct?

## the questions

- do formal proof in mathematics and formal proof in computer science require different approaches?
- what are the ingredients of a formal proof system? (in case you want to build one yourself)
- what are the unsolved problems in formal proof? (in case you think you can do better than the state of the art)
- can one be certain a formal proof is fully correct?
- how does computation fit into formal proof?

# my research interests

## interactive formal proof

- **formal proof for mathematics**

  formal proof languages
  formal proof interfaces

  FEAR project
  formal proof and computer algebra

  logical frameworks
  partiality in formal proof

- **formal proof for computer science**

  $CH_2O$ project
  formal version of the C standard
  Robbert Krebbers

  static analysis + interactive formal proof

# formal proof for computer science: zero days exploits

## $CH_2O$ operational semantics

$$\frac{\Gamma, m \vdash (v_1 \circledcirc v_2) \text{ defined}}{([v_1]_{\Omega_1} \circledcirc [v_2]_{\Omega_2}, m) \rightharpoonup_{\mathsf{h}} ([v_1 \circledcirc v_2]_{\Omega_1 \cup \Omega_2}, m)}$$

$$(\mathsf{load}\ [a]_\Omega, m) \rightharpoonup_{\mathsf{h}} ([m\langle a\rangle_\Gamma]_\Omega, \mathsf{force}_\Gamma\ a\ m)$$

$$\frac{\mathsf{writable}_\Gamma\ a\ m \qquad \tau = \mathsf{typeof}\ a \qquad \Gamma, m \vdash (\tau)v \text{ defined} \qquad v' = (\tau)v}{([a]_{\Omega_1} := [v]_{\Omega_2}, m) \rightharpoonup_{\mathsf{h}} ([v']_{\{a\} \cup \Omega_1 \cup \Omega_2}, \mathsf{lock}_\Gamma\ a\ (m[a := v']_\Gamma))}$$

---

$$\frac{(e_1, m_1) \rightharpoonup_{\mathsf{h}} (e_2, m_2)}{\mathbf{S}(\mathcal{P}[\square], \mathcal{E}[e_1], m_1) \rightharpoonup \mathbf{S}(\mathcal{P}[\square], \mathcal{E}[e_2], m_2)}$$

context $\mathcal{E}[\square]$ chosen non-deterministically

$$\mathbf{S}(\mathcal{P}[\square], (\searrow, e), m) \rightharpoonup \mathbf{S}(\mathcal{P}[\square_e], e, m)$$

$$\mathbf{S}(\mathcal{P}[\square_e], [v]_\Omega, m) \rightharpoonup \mathbf{S}(\mathcal{P}[\square], (\nearrow, e), \mathsf{unlock}\ \Omega\ m)$$

$$\mathbf{S}(\mathcal{P}[\square], (\searrow, \mathsf{if}\ (e)\ s_1\ \mathsf{else}\ s_2), m) \rightharpoonup \mathbf{S}(\mathcal{P}[\mathsf{if}\ (\square_e)\ s_1\ \mathsf{else}\ s_2], e, m)$$

$$\frac{m \vdash (\mathsf{zero}\ v_b) \text{ defined} \qquad \neg\, \mathsf{zero}\ v_b}{\mathbf{S}(\mathcal{P}[\mathsf{if}\ (\square_e)\ s_1\ \mathsf{else}\ s_2], [v_b]_\Omega, m) \rightharpoonup \mathbf{S}(\mathcal{P}[\mathsf{if}\ (e)\ \square\ \mathsf{else}\ s_2], (\searrow, s_1), \mathsf{unlock}\ \Omega\ m)}$$

## mathematics versus computer science

### different cultures?

# mathematics versus computer science

## different cultures?

Alexander Grothendieck
(Fields medal 1966)


Ken Thompson & Dennis Ritchie
(Turing award 1983)

## trying out formal proof: four formal proof systems

= proof assistants
= interactive theorem provers

| *primarily developed for mathematics* | *primarily developed for computer science* |
|---|---|
| | **Coq** |
| Mizar | **Isabelle/HOL** |
| | HOL Light |

## trying out formal proof: five formal proof systems

= proof assistants
= interactive theorem provers

| *primarily developed for mathematics* | *primarily developed for computer science* |
|---|---|
| | **Coq** |
| Mizar | **Isabelle/HOL** |
| | HOL Light |
| | HOL4 |

## biggest successes per system

- **Coq**
  four color theorem (mathematics)
  odd order theorem (mathematics)
  CompCert C compiler (computer science)
- **Isabelle/HOL**
  seL4 operating system (computer science)
- **HOL Light**
  prime number theorem (mathematics)
  Flyspeck project = Kepler conjecture (mathematics)
- **HOL4**
  ARM processor (computer science)
  CakeML compiler (computer science)
- **Mizar**
  textbook on continuous lattices (mathematics)

## state of the art

- **mathematics**
  formal proof: not yet routine
  serious formal proof convincingly demonstrated

  just one practical example: Flyspeck

- **theoretical computer science**
  formal proof: routine

  conferences: ITP, CPP, POPL

- **practical computer science**
  formal proof: not yet routine
  scalable formal proof not yet convincingly demonstrated

  spin-off technologies: model checking, SAT/SMT solvers

## the ingredients

### teaching an interactive theorem prover

- **statements**
- **proof steps**
- **definitions**
- imports
- proof automation

# the ingredients

## teaching an interactive theorem prover

- **statements** 13%
- **proof steps** 59%
- **definitions** 6%
- imports 1%
- proof automation 4%

# the ingredients

## teaching an interactive theorem prover

- **statements** 13%
- **proof steps** 59%
- **definitions** 6%
- imports 1%
- proof automation 4%
- comments 7%
- blank lines 10%

## statement example: Sylow theorem

- informal statement

  Let $p$ be a prime factor with multiplicity $n$ of the order of a finite group $G$, so that the order of $G$ can be written as $p^n m$, where $n > 0$ and $p$ does not divide $m$. Let $n_p$ be the number of Sylow $p$-subgroups of $G$. Then the following hold:

  - $n_p$ divides $m$, which is the index of the Sylow $p$-subgroup in $G$.
  - $n_p \equiv 1 \bmod p$.
  - $n_p = |G : N_G(P)|$, where $P$ is any Sylow $p$-subgroup of $G$ and $N_G$ denotes the normalizer.

- informal statement

  Let $p$ be a prime factor with multiplicity $n$ of the order of a finite group $G$, so that the order of $G$ can be written as $p^n m$, where $n > 0$ and $p$ does not divide $m$. Let $n_p$ be the number of Sylow $p$-subgroups of $G$. Then the following hold:

  - $n_p$ divides $m$, which is the index of the Sylow $p$-subgroup in $G$.
  - $n_p \equiv 1 \bmod p$.
  - $n_p = |G : N_G(P)|$, where $P$ is any Sylow $p$-subgroup of $G$ and $N_G$ denotes the normalizer.

- Coq statement

```
[/\ ∀ P, [max P | p.-subgroup(G) P] = p.-Sylow(G) P,
    [transitive G, on 'Syl_p(G) | 'JG],
    ∀ P, p.-Sylow(G) P → #|'Syl_p(G)| = #|G : 'N_G(P)|
 & prime p → #|'Syl_p(G)| %% p = 1%N].
```

## statement example: Sylow theorem

Google: 100 theorems

- informal statement

  Let $p$ be a prime factor with multiplicity $n$ of the order of a finite group $G$, so that the order of $G$ can be written as $p^n m$, where $n > 0$ and $p$ does not divide $m$. Let $n_p$ be the number of Sylow $p$-subgroups of $G$. Then the following hold:

  - $n_p$ divides $m$, which is the index of the Sylow $p$-subgroup in $G$.
  - $n_p \equiv 1 \mod p$.
  - $n_p = |G : N_G(P)|$, where $P$ is any Sylow $p$-subgroup of $G$ and $N_G$ denotes the normalizer.

- HOL Light statement

```
!e op i G p.
    group (G,op,i,e)
    ==> FINITE G
    ==> prime p
    ==> (!n m. CARD G = p EXP n * m
               ==> coprime (p,m)
               ==> CARD {K | subgroup op i K G /\ CARD K = p EXP n}
                                                         MOD p = 1)
```

## statement example: Sylow theorem

Google: 100 theorems

- informal statement

  Let $p$ be a prime factor with multiplicity $n$ of the order of a finite group $G$, so that the order of $G$ can be written as $p^n m$, where $n > 0$ and $p$ does not divide $m$. Let $n_p$ be the number of Sylow $p$-subgroups of $G$. Then the following hold:

  - $n_p$ divides $m$, which is the index of the Sylow $p$-subgroup in $G$.
  - $n_p \equiv 1 \bmod p$.
  - $n_p = |G : N_G(P)|$, where $P$ is any Sylow $p$-subgroup of $G$ and $N_G$ denotes the normalizer.

- Mizar statement

```
for G being finite Group, p being prime (natural number) holds
  card the_sylow_p-subgroups_of_prime(p,G) mod p = 1 &
  card the_sylow_p-subgroups_of_prime(p,G) divides ord G;
```

# proof example with procedural tactics: Coq/Ssreflect

## proof example with procedural tactics: Coq/Ssreflect

```
...
pose maxp A P := [max P | p.-subgroup(A) P]; pose S := [set P | maxp G P].
pose oG := orbit 'JG%act G.
have actS: [acts G, on S | 'JG].
  apply/subsetP⇒ x Gx; rewrite 3!inE; apply/subsetP⇒ P; rewrite 3!inE.
  exact: max_pgroupJ.
have S_pG P: P \in S → P \subset G ∧ p.-group P.
  by rewrite inE ⇒ /maxgroupp/andP[].
have SmaxN P Q: Q \in S → Q \subset 'N(P) → maxp 'N_G(P) Q.
  rewrite inE ⇒ /maxgroupP[/andP[sQG pQ] maxQ] nPQ.
  apply/maxgroupP; rewrite /psubgroup subsetI sQG nPQ.
  by split⇒ // R; rewrite subsetI -andbA andbCA ⇒ /andP[_]; exact: maxQ.
have nrmG P: P \subset G → P <| 'N_G(P).
  by move⇒ sPG; rewrite /normal subsetIr subsetI sPG normG.
have sylS P: P \in S → p.-Sylow('N_G(P)) P.
  move⇒ S_P; have [sPG pP] := S_pG P S_P.
  by rewrite normal_max_pgroup_Hall ?nrmG //; apply: SmaxN; rewrite ?normG.
have{SmaxN} defCS P: P \in S → 'Fix_(S |'JG)(P) = [set P].
  move⇒ S_P; apply/setP⇒ Q; rewrite {1}in_setI {1}afixJG.
  apply/andP/set1P⇒ [[S_Q nQP]|->{Q}]; last by rewrite normG.
  apply/esym/val_inj; case: (S_pG Q) ⇒ //= sQG _.
  by apply: uniq_normal_Hall (SmaxN Q _ _ _) ⇒ //=; rewrite ?sylS ?nrmG.
...
```

## proof example with procedural tactics: Coq/Ssreflect

```
...
pose maxp A P := [max P | p.-subgroup(A) P]; pose S := [set P | maxp G P].
pose oG := orbit 'JG%act G.
have actS: [acts G, on S | 'JG].
  apply/subsetP⇒ x Gx; rewrite 3!inE; apply/subsetP⇒ P; rewrite 3!inE.
  exact: max_pgroupJ.
have S_pG P: P \in S → P \subset G ∧ p.-group P.
  by rewrite inE ⇒ /maxgroupp/andP[].
have SmaxN P Q: Q \in S → Q \subset 'N(P) → maxp 'N_G(P) Q.
  rewrite inE ⇒ /maxgroupP[/andP[sQG pQ] maxQ] nPQ.
  apply/maxgroupP; rewrite /psubgroup subsetI sQG nPQ.
  by split⇒ // R; rewrite subsetI -andbA andbCA ⇒ /andP[_]; exact: maxQ.
have nrmG P: P \subset G → P <| 'N_G(P).
  by move⇒ sPG; rewrite /normal subsetIr subsetI sPG normG.
have sylS P: P \in S → p.-Sylow('N_G(P)) P.
  move⇒ S_P; have [sPG pP] := S_pG P S_P.
  by rewrite normal_max_pgroup_Hall ?nrmG //; apply: SmaxN; rewrite ?normG.
have{SmaxN} defCS P: P \in S → 'Fix_(S |'JG)(P) = [set P].
  move⇒ S_P; apply/setP⇒ Q; rewrite {1}in_setI {1}afixJG.
  apply/andP/set1P⇒ [[S_Q nQP]|->{Q}]; last by rewrite normG.
  apply/esym/val_inj; case: (S_pG Q) ⇒ //= sQG _.
  by apply: uniq_normal_Hall (SmaxN Q _ _ _) ⇒ //=; rewrite ?sylS ?nrmG.
...
```

# proof example with declarative steps only: Mizar

```
...
   ex h being Element of G st y = h & Q9 |^ h = Q9
   proof
     set h = y;
     the carrier of Q9 c= the carrier of G by GROUP_2:def 5;
     then reconsider h as Element of G by A33;
     take h;
     thus y = h;
     for g being Element of G holds g in Q9 iff g in Q9 |^ h
     proof
       let g be Element of G;
       hereby
         assume
A34:     g in Q9;
         ex g9 being Element of G st g = g9 |^ h & g9 in Q9
         proof
           set g9 = h * g * h";
           take g9;
           thus g9 |^ h = h" * g9 * h by GROUP_3:def 2
             .= h" * (h * (g * h")) * h by GROUP_1:def 3
             .= (h" * h) * (g * h") * h by GROUP_1:def 3
             .= 1_G * (g * h") * h by GROUP_1:def 5
...
```

## flavors of definitions

- abbreviations (mathematics, computer science)

  $f(x_1, \ldots, x_n) := \ldots$

- characterisations (mathematics)

  $f(x_1, \ldots, x_n) :=$ 'the unique $y$ such that $P(x_1, \ldots, x_n, y)$'

- recursive definitions (computer science)

  $f(x_1, \ldots, x_n) := \ldots \quad \longleftarrow$ may contain $f$

- algebraic datatypes (computer science)

  lists, trees, syntax

- inductively defined predicates (computer science)

  smallest relation closed under certain inference rules

## differences

- **formal proof in mathematics**

  specific choice of definition not important

  few small definitions, relatively easy to get correct

  proofs need insight
  proofs interesting

  mostly first order reasoning

- **formal proof in computer science**

  specific choice of definition matters

  many large definitions, difficult to get correct

  proofs largely trivial
  proofs mostly not interesting

  many inductions with many cases

## the problems

### unimportant issues

- looking for the ‘right’ logical foundations

  set theory, HOL, type theory, HoTT
  all work just as well

## the problems

### unimportant issues

- looking for the 'right' logical foundations

  set theory, HOL, type theory, HoTT
  all work just as well

- making it look like natural language

  the COBOL fallacy
  formal proof language $\approx$ programming language

## important issues

- **automation**

  integration of computer algebra
  MetiTarski

## important issues

- **automation**

  integration of computer algebra
  MetiTarski

  hammers
  Sledgehammer, HOL(y)Hammer

## important issues

- **automation**

  integration of computer algebra
  MetiTarski

  hammers
  Sledgehammer, HOL(y)Hammer

- libraries

  $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \ldots$
  semigroup $\supset$ groups $\supset$ rings $\supset$ fields $\supset$ ordered fields $\supset \ldots$
  scalars $\subset$ vectors $\subset$ matrixes $\subset$ tensors $\subset \ldots$

  just a single 0

- **automation**

  integration of computer algebra
  MetiTarski

  hammers
  Sledgehammer, HOL(y)Hammer

- libraries

  $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \dots$
  semigroup $\supset$ groups $\supset$ rings $\supset$ fields $\supset$ ordered fields $\supset \dots$
  scalars $\subset$ vectors $\subset$ matrixes $\subset$ tensors $\subset \dots$

  just a single 0

  MathML fragment of mathematics

# certainty

## what could possibly go wrong?

- **bugs in the implementation of the proof system**

## certainty

### what could possibly go wrong?

- **bugs in the implementation of the proof system**
  not a problem

## certainty

### what could possibly go wrong?

- **bugs in the implementation of the proof system**
  not a problem

- **logical foundations are inconsistent**
  not a serious problem

## certainty

### what could possibly go wrong?

- **bugs in the implementation of the proof system**
  not a problem

- **logical foundations are inconsistent**
  not a serious problem

- **definitions do not match informal understanding**
  **the real problem**

  Andrzej Trybulec:

  *definitions are a debt*
  *proved lemmas pay back that debt*

## de Bruijn criterion: what to trust?

formalization

## de Bruijn criterion: what to trust?

formalization

proof system
sources

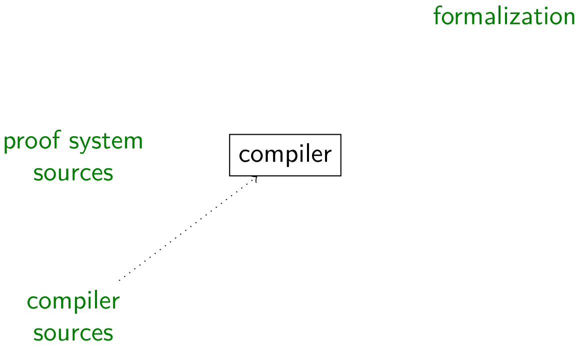
## de Bruijn criterion: what to trust?

formalization

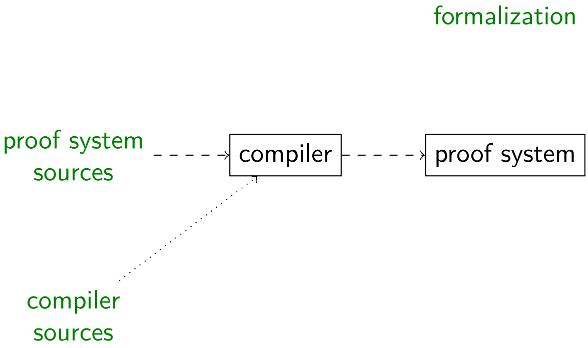
proof system
sources

compiler
sources

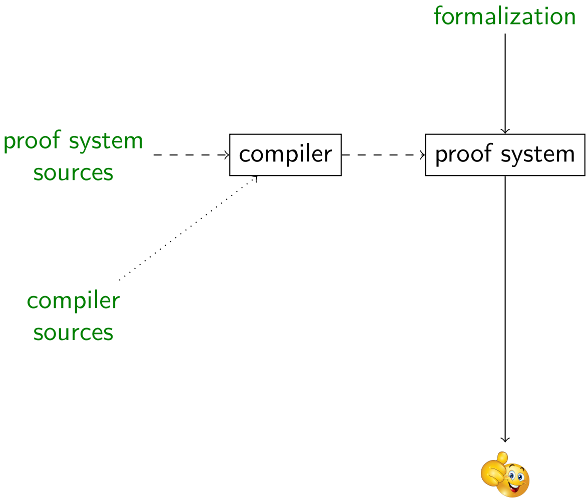
# de Bruijn criterion: what to trust?



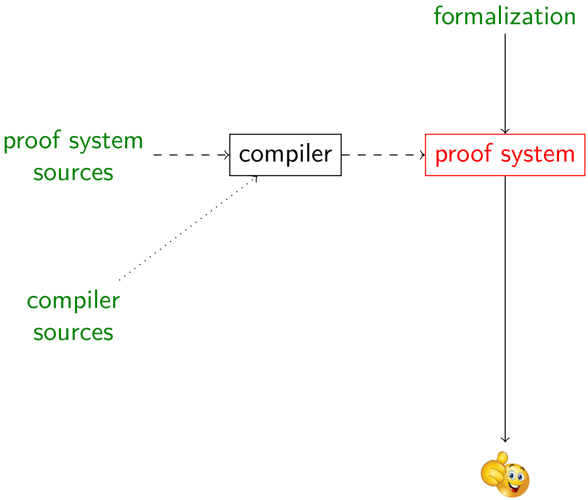
## de Bruijn criterion: what to trust?



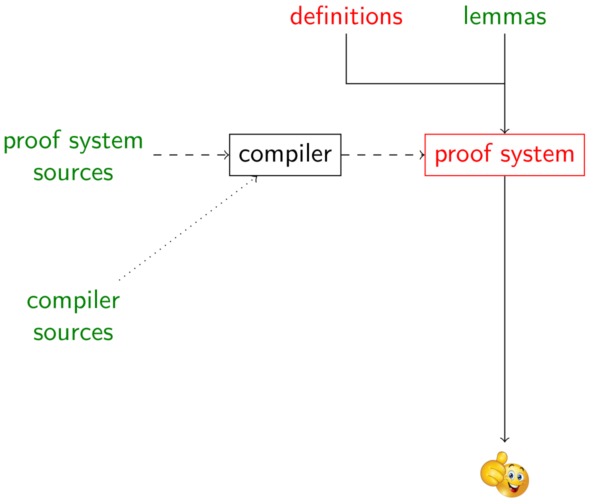
## de Bruijn criterion: what to trust?



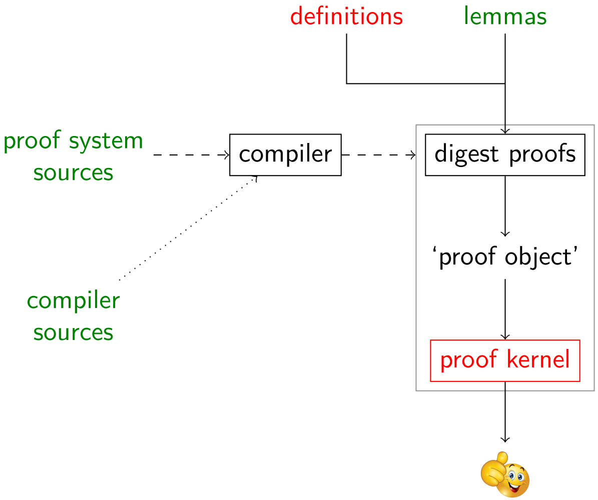
# de Bruijn criterion: what to trust?



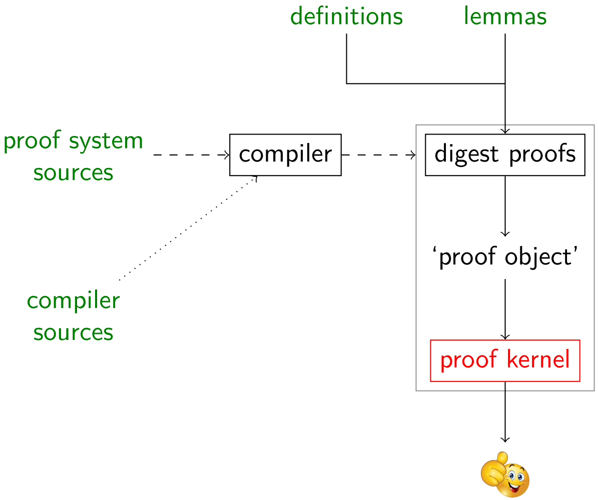
## de Bruijn criterion: what to trust?



## de Bruijn criterion: what to trust?



## de Bruijn criterion: what to trust?



## CakeML and the verified HOL Light kernel

Magnus Myreen, Ramana Kumar, Scott Owens, e.a.
'Cake' = Cambridge & Kent

| | *implemented in* | *verified in* |
|---|---|---|
| Poly/ML | Poly/ML | — |
| HOL4 kernel | Poly/ML | — |
| HOL4 system | Poly/ML | — |
| OCaml | OCaml | — |
| HOL Light kernel | OCaml | — |
| HOL Light system | OCaml | — |
| CakeML | HOL4* | HOL4 |
| vHOL kernel | HOL4* | HOL4 |
| | *extracted to CakeML | HOL4 |

## CakeML and the verified HOL Light kernel

Magnus Myreen, Ramana Kumar, Scott Owens, e.a.
'Cake' = Cambridge & Kent

| | *implemented in* | *verified in* |
|---|---|---|
| Poly/ML | Poly/ML | — |
| HOL4 kernel | Poly/ML | — |
| HOL4 system | Poly/ML | — |
| OCaml | OCaml | — |
| HOL Light kernel | OCaml | — |
| HOL Light system | OCaml | — |
| CakeML | HOL4* | HOL4 |
| vHOL kernel | HOL4* | HOL4 |
| vHOL system | CakeML | HOL4 |
| | *extracted to CakeML | HOL4 |

## CakeML and the verified HOL Light kernel

Magnus Myreen, Ramana Kumar, Scott Owens, e.a.
'Cake' = Cambridge & Kent

| | *implemented in* | *verified in* |
|---|---|---|
| Poly/ML | Poly/ML | — |
| HOL4 kernel | Poly/ML | — |
| HOL4 system | Poly/ML | — |
| OCaml | OCaml | — |
| HOL Light kernel | OCaml | — |
| HOL Light system | OCaml | — |
| CakeML | vHOL* | vHOL |
| vHOL kernel | vHOL* | vHOL |
| vHOL system | CakeML | vHOL |
| | *extracted to CakeML | vHOL |

## CakeML and the verified HOL Light kernel

Magnus Myreen, Ramana Kumar, Scott Owens, e.a.
'Cake' = Cambridge & Kent

| | *implemented in* | *verified in* |
|---|---|---|
| CakeML | vHOL* | vHOL |
| vHOL kernel | vHOL* | vHOL |
| vHOL system | CakeML | vHOL |
| | *extracted to CakeML | vHOL |

# proofs that depend on programs

## examples

- **four color theorem**
  - check unavoidability of configurations
  - check reducibility of configurations
- **Mertens conjecture**
  - compute zeroes of Riemann zeta function to many decimals
- **Kepler conjecture**
  - solve many linear programs
  - check many non-linear inequalities
  - enumerate a collection of tame graphs

## proofs that depend on programs

### examples

- **four color theorem**
  - check unavoidability of configurations
  - check reducibility of configurations
- **Mertens conjecture**
  - compute zeroes of Riemann zeta function to many decimals
- **Kepler conjecture**
  - solve many linear programs
  - check many non-linear inequalities
  - enumerate a collection of tame graphs

how to know these programs are correct?

## proofs that depend on programs

### examples

- **four color theorem**
  - check unavoidability of configurations
  - check reducibility of configurations
- **Mertens conjecture**
  - compute zeroes of Riemann zeta function to many decimals
- **Kepler conjecture**
  - solve many linear programs
  - check many non-linear inequalities
  - enumerate a collection of tame graphs

how to know these programs are correct?

how do these programs fit into a formal proof?

## a spectrum of programming languages for mathematics

- computation by deduction
  HOL
- high-level functional programming languages
  ML
- low-level imperative programming languages
  C
- machine code
  x86

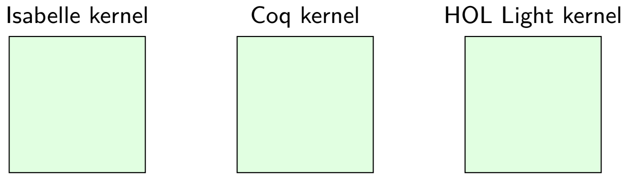
## the Poincaré principle: fitting computation to the logic

formally proving

$$\vdash F(M) \twoheadrightarrow N$$

formally proving

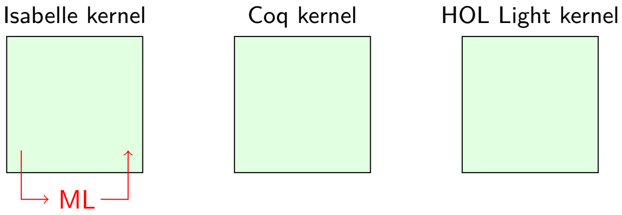
$$\vdash F(M) \longrightarrow\!\!\!\!\!\rightarrow N$$



## the Poincaré principle: fitting computation to the logic

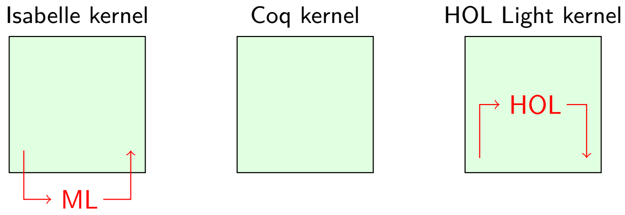
formally proving

$$\vdash F(M) \longrightarrow\!\!\!\!\!\!\rightarrow N$$



formally proving

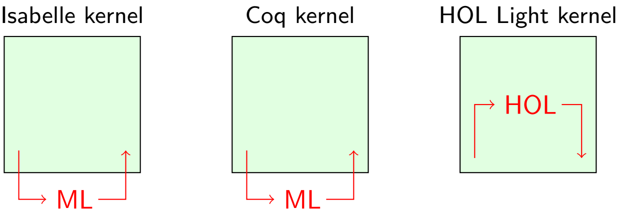
$\vdash F(M) \twoheadrightarrow N$



## the Poincaré principle: fitting computation to the logic

formally proving

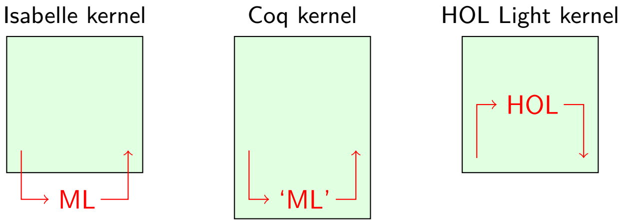
$\vdash F(M) \twoheadrightarrow N$



# the Poincaré principle: fitting computation to the logic

formally proving

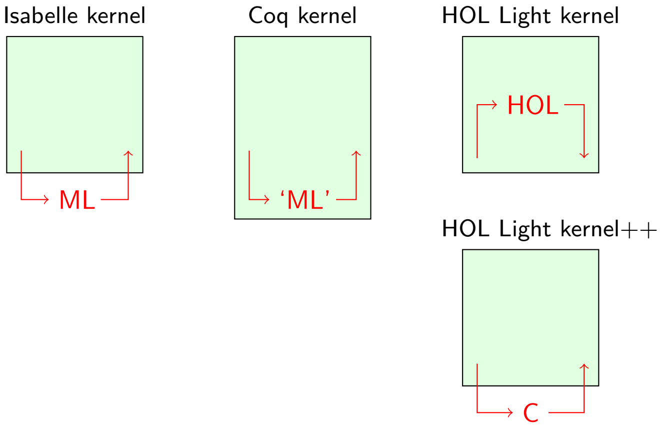
$$\vdash F(M) \twoheadrightarrow N$$



## the Poincaré principle: fitting computation to the logic

formally proving

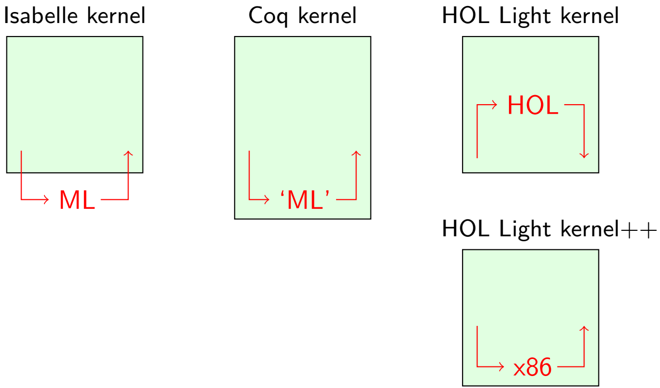
$\vdash F(M) \twoheadrightarrow N$



# the Poincaré principle: fitting computation to the logic

formally proving

$$\vdash F(M) \twoheadrightarrow N$$



# conclusions

## developing a formal proof system

- without good **automation** it will not be very usable

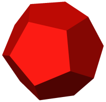
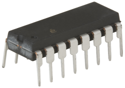
- without a good lemma **library** it will not be very appealing

## conclusions

### developing a formal proof system

- without good **automation** it will not be very usable
- without a good lemma **library** it will not be very appealing
- without a small **proof kernel** it will not be utterly reliable
- without **computation** it will not handle proofs that depend on large programs

## any questions?

# table of contents

## contents