merging the procedural and declarative proof styles

Freek Wiedijk

Radboud University Nijmegen

Mathematics, Algorithms and Proofs

ICTP, Trieste, Italy

2008 08 25, 14 : 00

Mathematical Logic and Computers

satellite workshop of ALC 10

Kobe, Japan

2008 08 31, 14 : 50

which system is best for formal mathematics?

80 out of 100 theorems

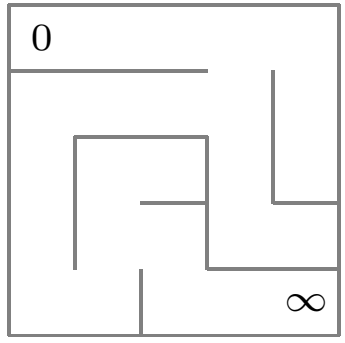
1.	The Irrationality of the Square Root of 2	≥ 17
2.	Fundamental Theorem of Algebra	4
3.	The Denumerability of the Rational Numbers	6
4.	Pythagorean Theorem	6
5.	Prime Number Theorem	2
6.	Gödel's Incompleteness Theorem	3
7.	Law of Quadratic Reciprocity	4
8.	The Impossibility of Trisecting the Angle and Doubling the Cube	1
9.	The Area of a Circle	1
10.	Euler's Generalization of Fermat's Little Theorem	4
11.	The Infinitude of Primes	6
12.	The Independence of the Parallel Postulate	0
13.	Polyhedron Formula	1

...

five systems

HOL	HOL Light	69	procedural
	ProofPower	42	procedural
	Isabelle	40	declarative + procedural
	Coq	39	procedural
	Mizar	45	declarative

procedural versus declarative



- **procedural**

  E E S E N E S S S W W W S E E E

  HOL, Coq, Isabelle

- **declarative**

  (0,0) (1,0) (2,0) (3,0) (3,1) (2,1) (1,1) (0,1) (0,2) (0,3) (0,4) (1,4) (1,3) (2,3) (2,4) (3,4) (4,4)

  Mizar, Isabelle

the state of the art in formal mathematics

- Georges Gonthier

  Coq

  Four Color Theorem, 2004

  Neil Robertson, Daniel Sanders, Paul Seymour, Robin Thomas

  The four colour theorem

  43 pp.

- John Harrison

  HOL

  Prime Number Theorem, 2008

  Donald Newman

  Analytic Number Theory, chapter VII

  9 pp.

the systems

HOL ← most theorems

Isabelle

Coq ← four color theorem + intuitionistic weirdness

Mizar ← most mathematical

HOL Light

overview

1972 – today

LCF → HOL → { HOL4 | **HOL Light** | ProofPower }

Robin Milner → Mike Gordon → **John Harrison**

Stanford, US → Cambridge, UK → Portland, US

higher order logic = weak, typed set theory

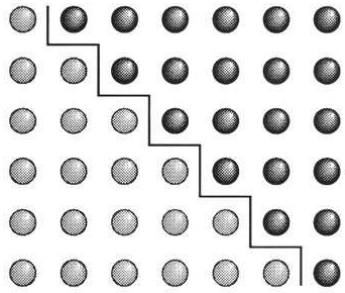
very nice open architecture

only 394 lines of ocaml code need to be trusted

relatively strong automation

example: session

```
        HOL Light 2.20++, built 11 March 2008 on OCaml 3.09.2 with ckpt

val it : unit = ()
# g `!n. nsum(1..n) (\i. i) = (n*(n + 1)) DIV 2`;;
val it : goalstack = 1 subgoal (1 total)

`!n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2`

# e INDUCT_TAC;;
val it : goalstack = 2 subgoals (2 total)

  0 [`nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2`]

`nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2`

`nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2`

# e (ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG]);;
val it : goalstack = 1 subgoal (2 total)

`(if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2`

#
```



example: session (continued)

```
‘(if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2‘

# e ARITH_TAC;;
val it : goalstack = 1 subgoal (1 total)

  0 [‘nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2‘]

‘nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2‘

# e (ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG]);;
val it : goalstack = 1 subgoal (1 total)

  0 [‘nsum (1..n) (ι. i) = (n * (n + 1)) DIV 2‘]

‘(if 1 <= SUC n then (n * (n + 1)) DIV 2 + SUC n else (n * (n + 1)) DIV 2) =
 (SUC n * (SUC n + 1)) DIV 2‘

# e ARITH_TAC;;
val it : goalstack = No subgoals

#
```

lemmas and tactics

```
# NSUM_CLAUSES_NUMSEG;;
val it : thm =
  |- (!m. nsum (m..0) f = (if m = 0 then f 0 else 0)) /\
     (!m n.
          nsum (m..SUC n) f =
          (if m <= SUC n then nsum (m..n) f + f (SUC n) else nsum (m..n) f))
# INDUCT_TAC;;
val it : tactic = <fun>
# ASM_REWRITE_TAC;;
val it : thm list -> tactic = <fun>
# ARITH_TAC;;
val it : tactic = <fun>
#
```

example: in the file

```
let ARITHMETIC_PROGRESSION_SIMPLE = prove
 (‘!n. nsum(1..n) (\i. i) = (n*(n + 1)) DIV 2‘,
  INDUCT_TAC THEN ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG] THEN
  ARITH_TAC);;
```

Mizar

overview

1973 – today

Andrzej Trybulec

Białystok, Poland

first order logic +

Tarski-Grothendieck set theory

= ZFC + arbitrarily large inaccessible cardinals

very nice natural language-like proof language

very nice type system

MML = Mizar Mathematical Library

2.2 million lines of code, 55 thousand lemmas

example

```
theorem Th1:
  (for i holds s.i = i) implies for n holds Partial_Sums(s).n = n*(n + 1)/2
proof
  assume
A1: for i holds s.i = i;
  defpred X[Element of NAT] means Partial_Sums(s).$1 = $1*($1 + 1)/2;
  Partial_Sums(s).0 = s.0 by SERIES_1:def 1
    .= 0*(0 + 1)/2 by A1;
  then
A2: X[0];
A3: now let n;
    assume X[n];
    then Partial_Sums(s).(n + 1) = n*(n + 1)/2 + s.(n + 1) by SERIES_1:def 1
      .= n*(n + 1)/2 + (n + 1) by A1;
    hence X[n + 1];
  end;
  thus for n holds X[n] from NAT_1:sch 1(A2,A3);
end;
```

lemmas

```
definition
  let s be Real_Sequence;
  func Partial_Sums(s) -> Real_Sequence means
:: SERIES_1:def 1
    it.0 = s.0 & for n holds it.(n + 1) = it.n + s.(n + 1);
end;

scheme :: NAT_1:sch 1
  Ind { P[Nat] } : for k being Element of NAT holds P[k]
provided
  P[0]
and
  for k being Element of NAT st P[k] holds P[k + 1];
```

the best of both worlds

a proof assistant that is not **too** frustrating

HOL, Isabelle, Coq, Mizar

I know three intimately

they are all frustrating in different ways

- **just learn Isabelle!**

  $$\text{Isabelle} = \text{HOL-like system} + \text{Mizar-like proofs}$$

  does not **integrate** the procedural and declarative proof styles

- **build an $(n+1)$st system!**

- **improve an existing system!**

current attempt: improve HOL!

yet another Mizar-style proof language on top of HOL Light

Mizar-style proofs as **first class objects**

Mizar-style **user interface**

- **'A Mizar Mode for HOL'** by John Harrison — TPHOLs 1996
- **Mizar Light** — TPHOLs 2001
- **Mizar Light II** — unpublished
- **'Changing proof style'** by John Harrison — HOL Light tutorial
- **Mizar Light III**

evolution of Mizar Light syntax

- **Mizar Light**

```
prove(‘a ==> a‘,
  ASSUME_A(0,‘a:bool‘) THEN
  THUS_A(1,‘a:bool‘) (BY [0][]));;
```

- **Mizar Light II**

```
theorem "a ==> a"
 [assume "a";
  hence "a"];;
```

- **Mizar Light III**

```
a ==> a
proof
  assume a;
  thus a;
end;
```

the example in Mizar Light

```
!n. nsum(0..n) (\i. i) = (n*(n + 1)) DIV 2
proof
  nsum(0..0) (\i. i) = 0 by NSUM_CLAUSES_NUMSEG;
    .= (0*(0 + 1)) DIV 2 [A1] by ARITH_TAC;
  now let n be num;
    assume nsum(0..n) (\i. i) = (n*(n + 1)) DIV 2;
    nsum(0..SUC n) (\i. i) = (n*(n + 1)) DIV 2 + SUC n
      by NSUM_CLAUSES_NUMSEG,ARITH_RULE `0 <= SUC n`;
    thus .= ((SUC n)*(SUC n + 1)) DIV 2 by ARITH_TAC;
  end;
qed by INDUCT_TAC,A1;
```

- Mizar proof language
- HOL Light statements without quotes!
- HOL Light tactics in the justifications!

tactics in justifications?

⟨statement⟩ `by` ⟨tactic⟩, ⟨theorem⟩, ..., ⟨label⟩, ...;

- ⟨tactic⟩ should have ML type `thm list -> tactic`
- the step naturally corresponds to a certain goal
  the ⟨label⟩s label the assumptions in that goal
- just keep the assumptions in the goal that occur in the list
- run the tactic with both theorems and assumptions as argument
- if the tactic succeeds: turn resulting subgoals into statements
  these should all be in the list

growing a proof procedurally

```
now
  thus | [A1] by #;
end;
```

growing a proof procedurally

```
now
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A1] by #INDUCT_TAC;
end;
```

growing a proof procedurally

```
now
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A1] by #;
  now [A2]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A3];
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A4] by #;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A5] by INDUCT_TAC,A1,A2;
end;
```

growing a proof procedurally

```
now
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A1]
    by #REWRITE_TAC,NSUM_CLAUSES_NUMSEG;
  now [A2]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A3];
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A4] by #;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A5] by INDUCT_TAC,A1,A2;
end;
```

growing a proof procedurally

```
now
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 [A1] by #ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A2]
    by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A1;
  now [A3]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A4];
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A5] by #;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A6] by INDUCT_TAC,A2,A3;
end;
```

growing a proof procedurally

```
now
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 [A1] by ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A2]
    by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A1;
  now [A3]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A4];
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A5] by #;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A6] by INDUCT_TAC,A2,A3;
end;
```

growing a proof procedurally

```
now
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 [A1] by ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A2]
    by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A1;
  now [A3]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A4];
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A5]
      by #REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A4;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A6] by INDUCT_TAC,A2,A3;
end;
```

growing a proof procedurally

```
now
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 [A1] by ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A2]
    by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A1;
  now [A3]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A4];
    (if 1 <= SUC n then (n * (n + 1)) DIV 2 + SUC n else (n * (n + 1)) DIV 2 =
      (SUC n * (SUC n + 1)) DIV 2 [A5] by #ARITH_TAC;
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A6]
      by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A4,A5;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A7] by INDUCT_TAC,A2,A3;
end;
```

growing a proof procedurally

```
now
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 [A1] by ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A2]
    by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A1;
  now [A3]
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A4];
    (if 1 <= SUC n then (n * (n + 1)) DIV 2 + SUC n else (n * (n + 1)) DIV 2 =
      (SUC n * (SUC n + 1)) DIV 2 [A5] by ARITH_TAC;
    thus nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2 [A6]
      by REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A4,A5;
  end;
  thus !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A7] by INDUCT_TAC,A2,A3;
end;
```

text typed while 'growing' the proof

```
!n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2
INDUCT_TAC
REWRITE_TAC,NSUM_CLAUSES_NUMSEG
ARITH_TAC
REWRITE_TAC,NSUM_CLAUSES_NUMSEG,A4
ARITH_TAC
```

this corresponds to the traditional session:

```
g `!n. nsum(1..n) (ι. i) = (n*(n + 1)) DIV 2`;;
e INDUCT_TAC;;
e (ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG]);;
e ARITH_TAC;;
e (ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG]);;
e ARITH_TAC;;
```

merging the declarative and procedural proof styles

two ways of working on a Mizar Light proof:

- **the declarative way**: free form text editing
  just type and edit the proof yourself

- **the procedural way**:
  generate new steps by executing a tactic at an unjustified line

both freely mixed

not two 'modes' in the proof script

Mizar-style error messages

```
!n. nsum(0..n) (\i. i) = (n*(n + 1)) DIV 2
proof
  nsum(0..0) (\i. i) = (0*(0 + 1)) DIV 2;
::                                      #1
  now let n be nat;
::                  #6
    assume nsum(0..n) (\i. i) = (n*(n + 1)) DIV 2;
    thus nsum(0..SUC n) (\i. i) = ((SUC n)*(SUC n + 1)) DIV 2 by #;
  end;
qed;
:: #1
```

porting formal mathematics

formalization for the future

currently, when a proof assistant dies its mathematical library dies
formal proof languages should be **generic**

three approaches to porting formal mathematics:

- *convert the low-level 'proof objects'*
  works, but no converted proofs on the user-level
- *port procedural scripts using similar tactics in the other system*
  does not work
- *port declarative scripts by just translating the statements*
  all declarative proof languages are basically the same!
  works approximately, but *with converted proofs on the user-level!*

the HOL Light example (repeat)

```
let ARITHMETIC_PROGRESSION_SIMPLE = prove
 (‘!n. nsum(1..n) (\i. i) = (n*(n + 1)) DIV 2‘,
  INDUCT_TAC THEN ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG] THEN
  ARITH_TAC);;
```

the Mizar Light conversion of the HOL Light example

```
!n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2
proof
  (if 1 = 0 then 0 else 0) = (0 * (0 + 1)) DIV 2 by ARITH_TAC;
  nsum (1..0) (\i. i) = (0 * (0 + 1)) DIV 2 [A1]
    by ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG];
  !n. nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2
      ==> nsum (1..SUC n) (\i. i) = (SUC n * (SUC n + 1)) DIV 2
  proof
    let n be num;
    assume nsum (1..n) (\i. i) = (n * (n + 1)) DIV 2 [A2];
    (if 1 <= SUC n then (n * (n + 1)) DIV 2 + SUC n else (n * (n + 1)) DIV 2) =
      (SUC n * (SUC n + 1)) DIV 2 by ARITH_TAC;
  qed by ASM_REWRITE_TAC[NSUM_CLAUSES_NUMSEG],A2;
qed by INDUCT_TAC,A1;
```

the Mizar example (repeat)

```
theorem
  (for i holds s.i = i) implies for n holds Partial_Sums(s).n = n*(n + 1)/2
proof
  assume
A1: for i holds s.i = i;
  defpred X[Element of NAT] means Partial_Sums(s).$1 = $1*($1 + 1)/2;
  Partial_Sums(s).0 = s.0 by SERIES_1:def 1
    .= 0*(0 + 1)/2 by A1;
  then
A2: X[0];
A3: now let n;
    assume X[n];
    then Partial_Sums(s).(n + 1) = n*(n + 1)/2 + s.(n + 1) by SERIES_1:def 1
      .= n*(n + 1)/2 + (n + 1) by A1
    hence X[n + 1];
  end;
  thus for n holds X[n] from NAT_1:sch 1(A2,A3);
end;
```

the Mizar Light conversion of the Mizar example

```
!s. (!i. s i = i) ==> !n. nsum(0..n) s = (n*(n + 1)) DIV 2
proof
  let s be num->num;
  assume !i. s i = i [A1];
  set X = \n. (nsum(0..n) s = (n*(n + 1)) DIV 2);
  nsum(0..0) s = s 0 by NSUM_CLAUSES_NUMSEG';
    .= 0 by A1;
    .= 0*(0 + 1) DIV 2 by ARITH_TAC;
  X 0 [A2];
  now [A3] let n be num;
    assume X n;
    nsum(0..n + 1) s = (n*(n + 1)) DIV 2 + s (n + 1) by NSUM_CLAUSES_NUMSEG';
      .= (n*(n + 1)) DIV 2 + (n + 1) by A1;
    thus X (n + 1) by ARITH_TAC;
  end;
  !n. X n by MATCH_MP_TAC,num_INDUCTION',A2,A3;
qed;
```

outlook

Mizar Light III is not **just** vaporware . . .

implementation currently just starting

what is there:

- parser and checker (including tactics in the justifications!)
- all examples in this talk are processed correctly

what is not there (yet):

- proper error messages
- a Mizar-style user interface
- 'growing' a proof by executing tactics
- conversion from procedural HOL Light or Mizar (or other systems)

the history and future of mathematics

Euclid

proof

Cauchy

rigor

de Bruijn

formality

needed for this third revolution of formality to happen:

- formalization should be much closer to traditional mathematics
- full automation of high school mathematics