Proof Automation and Proof Style

John Harrison
University of Cambridge

(Åbo Akademi University)

- What is a proof?
- Assessment of proof styles
- Classification of proof styles
- Some existing systems
- Full programmability
- Procedural vs. declarative
- The best of both worlds?

What is a proof?

We can distinguish between:

- Informal proof sketches
- Machine-checkable proof sketches
- Proof objects in a formal system
- Canonical proofs

We are concerned with machine-checkable proof sketches; with the general problem of *how to communicate a proof to a machine.*

Assessment of proof styles

Some qualities we might be interested in include:

- Ease of writing
- Ease of reading
- Maintainability
- Efficiency of processing
- Tool support
- Explicit control

Classification of proof styles

Some variables include:

- The level of automation provided, and conversely, the level of explicit control provided.
- The emphasis on a declarative style (stating what is to be proved) or a procedural one (stating how to prove it).
- Proof direction; whether one proceeds forward, backward or can mix the two or do something more flexible still.
- Whether one works interactively or in batch mode.

Some existing systems

We can see some sharp contrasts by considering existing systems:

- AUTOMATH: Very low automation, completely procedural.
- Mizar: Low but effective automation, almost completely declarative.
- PVS: Highly automated but with good interactive features, largely procedural.
- NQTHM: Very highly automated, no interactive features, totally declarative.

Declarative proof example

```
let f be A->A;
assume L:antecedent;
antisymmetry: (!x y. x <= y /\ y <= x ==> (x = y)) by L;
transitivity: (!x y z. x <= y /\ y <= z ==> x <= z) by L;
monotonicity: (!x y. x <= y ==> f x <= f y) by L;
least_upper_bound:
  (!X. ?s:A. (!x. x IN X ==> s <= x) /\
             (!s'. (!x. x IN X ==> s' <= x) ==> s' <= s))
  by L;
set Y_def: Y = {b | f b <= b};
Y_thm: !b. b IN Y = f b <= b
  by Y_def,IN_ELIM_THM,BETA_THM;
consider a such that
    lub: (!x. x IN Y ==> a <= x) /\
         (!a'. (!x. x IN Y ==> a' <= x) ==> a' <= a)
  by least_upper_bound;
take a;
now let b be A;
    assume b_in_Y: b IN Y;
    then L0: f b <= b by Y_thm;
    a <= b by b_in_Y, lub;
    so f a <= f b by monotonicity;
    hence f a <= b by L0, transitivity;
    end;
so Part1: f(a) <= a by lub;
so f(f(a)) <= f(a) by monotonicity;
so f(a) IN Y by Y_thm;
so a <= f(a) by lub;
hence thesis by Part1, antisymmetry;
```

Procedural proof example

```
REPEAT GEN_TAC THEN
REWRITE_TAC[contl; LIM; REAL_SUB_RZERO] THEN
BETA_TAC THEN DISCH_TAC THEN X_GEN_TAC "e:real" THEN
DISCH_TAC THEN
FIRST_ASSUM(UNDISCH_TAC o assert is_conj o concl) THEN
DISCH_THEN(CONJUNCTS_THEN MP_TAC) THEN
DISCH_THEN(\th. FIRST_ASSUM(MP_TAC o MATCH_MP th)) THEN
DISCH_THEN(X_CHOOSE_THEN "d:real" STRIP_ASSUME_TAC) THEN
DISCH_THEN(MP_TAC o SPEC "d:real") THEN
ASM_REWRITE_TAC[] THEN
DISCH_THEN(X_CHOOSE_THEN "c:real" STRIP_ASSUME_TAC) THEN
EXISTS_TAC "c:real" THEN ASM_REWRITE_TAC[] THEN
X_GEN_TAC "h:real" THEN
DISCH_THEN(ANTE_RES_THEN MP_TAC) THEN
ASM_CASES_TAC "&0 < abs(f(x + h) - f(x))" THENL
 [UNDISCH_TAC "&0 < abs(f(x + h) - f(x))" THEN
  DISCH_THEN(\th. DISCH_THEN(MP_TAC o CONJ th)) THEN
  DISCH_THEN(ANTE_RES_THEN MP_TAC) THEN
  REWRITE_TAC[REAL_SUB_ADD2];
  UNDISCH_TAC "~(&0 < abs(f(x + h) - f(x)))" THEN
  REWRITE_TAC[GSYM ABS_NZ; REAL_SUB_0] THEN
  DISCH_THEN SUBST1_TAC THEN
  ASM_REWRITE_TAC[REAL_SUB_REFL; ABS_0]]);;
```

The procedural-declarative distinction

Full programmability

It is very useful to have a full programming language as in HOL. Users of PVS feel the need for it. We have the 'Java problem' but LCF systems solve it.

However it brings disadvatages! One needs to support full programming when designing interfaces, debuggers and other tools.

Programmability is mainly used for:

- Substantial enhancements to the proof system
- Small one-off nonce programs

Perhaps with a suitable choice of primitives, the second is not needed? We can have something like the Coq approach.

Procedural vs. declarative (1)

- Ease of writing: Declarative style is certainly easier for the beginner because it requires a smaller 'vocabulary'. For experienced users, which is better may well depend on the proof.
- Readability: : Declarative proofs are normally much easier to read. They make the context manifest, whereas in the procedural style it need to be reconstructed by *executing* the previous steps.

Procedural vs. declarative (2)

- Maintainability: Declarative proofs are certainly easier to fix when broken. In general, as they are less explicit, they break less easily if the proof system or the problem changes. Terms are made manifest, and so can be changed by global editing.
- Efficiency: : Procedural proofs clearly win as they involve much less search.
- Tool support: : Declarative proofs are better, again because it is easier to establish context.
- Style of working: Declarative proofs support batch working because error recovery is better.

Procedural vs. declarative (3)

We can tentatively conclude that the declarative style is normally better for pure mathematics, but that the procedural style may be better in a number of verification applications.

However there may be some instances where the reverse is true, e.g. mathematical proofs established with computer assistance, or the Gonthier-Doligez proof of garbage collection algorithms.

How nice if we could have a free choice of both!

The best of both worlds?

We have implemented a system supporting Mizar-style declarative proofs within a slight extension of the HOL tactic framework.

The only extension to the tactic system is a method for labelling assumptions.

Mizar constructs have a procedural interpretation as HOL tactics, and we use automated theorem proving to turn the declarative proof outlines into concrete proofs.

We still retain a Coq-style ability to write (safe) new proof procedures in ML and link them into the Mizar mode. Moreover Mizar and HOL proofs can be arbitrarily intermixed.

Conclusions

The question of proof style is a fundamental one, and deserves more general consideration.

The success of Mizar shows the strength of the declarative style, at least within pure mathematics.

The right style probably depends on the proof. Perhaps, then, the ability to mix these and other styles is the ideal for a general theorem prover.