A Tactic Language for Declarative Proofs

Serge Autexier **Dominik Dietrich**

German Research Center for Artificial Intelligence (DFKI), Bremen, Germany
autexier@dfki.de dominik.dietrich@dfki.de

ITP 2010 - International Conference on Interactive Theorem Proving
Edinburgh, UK, July 11-14, 2010

Procedural vs. Declarative Proof

- recent trend towards declarative proof languages, inspired by MIZAR

procedural style

```
theorem natcomp:
        "(a::nat) + b = b+a"
  apply (induct a)
  apply (subst add_0)
  apply (subst add_0_right)
  apply (rule refl)
  apply (subst add_Suc_right)
  apply (subst add_Suc)
  apply (simp)
done
```

- + more efficient processing
- + faster proof development
- + usually shorter

declarative style

```
theorem natcomplus:
        "(a::nat) + b = b+a"
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (simp)
    also have "...=b+0" by (simp)
    finally show ?thesis .
  qed
  next ...
```

- + easier to read (**explicit context**)
- + easier to maintain, error recovery
- **portable** (at least to some degree)

Procedural vs. Declarative Proof

- recent trend towards declarative proof languages, inspired by MIZAR

procedural style

```
theorem natcomp:
        "(a::nat) + b = b+a"
  apply (induct a)
  apply (subst add_0)
  apply (subst add_0_right)
  apply (rule refl)
  apply (subst add_Suc_right)
  apply (subst add_Suc)
  apply (simp)
done
```

declarative style

```
theorem natcomplus:
        "(a::nat) + b = b+a"
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (simp)
    also have "...=b+0" by (simp)
    finally show ?thesis .
  qed
  next ...
```

+ more efficient processing
+ faster proof development
+ usually shorter

+ easier to read (**explicit context**)
+ easier to maintain, error recovery
- **portable** (at least to some degree)

- **common practice:**
  - **do not use** declarative style
  - **explore** and find proof using **procedural style**, **rewrite** it in **declarative style**

```
theorem natcomp:  "a + b =
b+a"
 apply (induct a)
 apply (subst add_0)
 apply (subst add_0_right)
 apply (rule refl)

goal (1 subgoal):
 1.  a + b = b + a
```

```
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (subst add_0)
    also have "...=b+0" by (subst
    finally show ?thesis .
   qed
  next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
```

- **common practice:**
  - **do not use** declarative style
  - **explore** and find proof using **procedural style**, **rewrite** it in **declarative style**

```
theorem natcomp:  "a + b =
b+a"
 apply (induct a)
 apply (subst add_0)
 apply (subst add_0_right)
 apply (rule refl)

goal (2 subgoals):
 1.  0 + b = b + 0
 2.  !!a.  a + b = b + a
==> Suc a + b = b + Suc a
```

```
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (subst add_0)
    also have "...=b+0" by (subst
    finally show ?thesis .
   qed
  next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
```

Constructing Declarative Proofs

- **common practice:**
  - **do not use** declarative style
  - **explore** and find proof using **procedural style**, **rewrite** it in **declarative style**

```
theorem natcomp:  "a + b =
b+a"
 apply (induct a)
 apply (subst add_0)
 apply (subst add_0_right)
 apply (rule refl)

goal (2 subgoals):
 1.  b = b + 0
 2.  !!a.  a + b = b + a
==> Suc a + b = b + Suc a
```

```
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (subst add_0)
    also have "...=b+0" by (subst
    finally show ?thesis .
   qed
  next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
```

Constructing Declarative Proofs

- **common practice:**
  - **do not use** declarative style
  - **explore** and find proof using **procedural style**, **rewrite** it in **declarative style**

```
theorem natcomp:  "a + b =
b+a"
 apply (induct a)
 apply (subst add_0)
 apply (subst add_0_right)
 apply (rule refl)

goal (2 subgoals):
1.  b = b
2.  !!a.  a + b = b + a ==>
Suc a + b = b + Suc a
```

```
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (subst add_0)
    also have "...=b+0" by (subst
    finally show ?thesis .
   qed
  next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
```

Constructing Declarative Proofs

- **common practice:**
  - **do not use** declarative style
  - **explore** and find proof using **procedural style**, **rewrite** it in **declarative style**

```
theorem natcomp:  "a + b =
b+a"
 apply (induct a)
 apply (subst add_0)
 apply (subst add_0_right)
 apply (rule refl)

goal (1 subgoal):
 1.  !!a.  a + b = b + a
==> Suc a + b = b + Suc a
```

```
proof (induct a)
  show "0 + b = b + 0"
  proof (-)
    have "0+b=b" by (subst add_0)
    also have "...=b+0" by (subst
    finally show ?thesis .
   qed
  next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
```

```
theorem natcomplus: "a + b = b+a"
```

Tactic execution

```
theorem natcomplus: "a + b = b+a"
proof (induct a)
  show "0 + b = b + 0"
  proof
  ...
  qed next
    fix a
    assume IH: "a+b=b+a"
    show "Suc a + b = b + Suc a"
    proof
```

Goals/Contributions

- **first class support** of **declarative proofs** at the tactic level
  - declarative proof scripts **without** loosing advantages of apply style
- capture **high level structure** of the proof in the tradition of **proof planning** [Bun88] or **proof sketches** [Wie04]
- specification of tactics **within** proof document

Generation of Declarative Proofs

1. generate declarative proof script from **proof term** [Coe10]
   - procedural proof script → proof term → declarative proof script
2. generate declarative proof script from **assertion level proof** [DSW08]
   - procedural proof script → proof tree → declarative proof script

Proof Script Generation

- **Stylistic choices** in expressing proofs, leading to **granularity problem**
  - include **intermediate results** or express them as **separate lemmas**
  - skip **trivial steps** completely

3. **Observation:** **proof plans** can be expressed as declarative proof scripts (ISAPLANNER [Dix05])
   - similarities and differences are discussed at the end of the talk

Overview

1. Development of Declarative Tactics

2. Dynamic Patterns and Iteration

Basic Declarative Tactics (1)

- procedural tactics, **simplest case**: sequence of inference applications
  - involve **parameters** (such as induction variable)
- declarative tactic: sequence of (declarative) proof commands
- **abstract** over common structure of proof scripts to obtain **schematic proof script**

```
theorem natcomplus: a+b = b+a
proof
 subgoals by (induct a)
  subgoal 0+b = b+0
  subgoal Suc a+b = b+Suc a
    using IH: a+b=b+a
 end
qed
```

```
theorem natcomplus: a+b = b+a
proof
 subgoals by (induct b)
  subgoal a+0 = 0+a
  subgoal a+Suc b = Suc b+a
    using IH: a+b=b+a
 end
qed
```

Basic Declarative Tactics (2)

```
strategy natinduct
cases * ⊢ P x
  with x in (analyzeinductvars "P")
->
proof
  subgoals by (induct x)
    subgoal P 0
    subgoal P (suc x) using IH: P x
  end
```

precondition

action

- make **context** available via precondition
- allow for **internal computations**
- **schematic proof script** as body

Realization

- define tactic language on top of **proof language**
  - declarative specification of the tactic **within proof document**
- justification is a **declarative proof script**
  - **natural integration** into existing frameworks

Basic Declarative Tactics (2)


```
strategy natinduct
 cases * ⊢ P x
   with x in (analyzeinductvars "P")
 ->
proof
  subgoals by (induct x)
   subgoal P 0
   subgoal P (suc x) using IH: P x
  end
```

precondition

action

- make **context** available via precondition
- allow for **internal computations**
- **schematic proof script** as body

Realization

- define tactic language on top of **proof language**
  - declarative specification of the tactic **within proof document**
- justification is a **declarative proof script**
  - **natural integration** into existing frameworks

Basic Declarative Tactics (2)

```
strategy natinduct
  cases * ⊢ P x
    with x in (analyzeinductvars "P")
  ->
proof
  L1:  P 0
  L2:  assume P x thus P (suc x)
  P x from L1,L2 by(induct x)
qed
```

precondition

action

- make **context** available via precondition
- allow for **internal computations**
- **schematic proof script** as body

Realization

- define tactic language on top of **proof language**
  - declarative specification of the tactic **within proof document**
- justification is a **declarative proof script**
  - **natural integration** into existing frameworks

Syntax and Semantics

- a declarative tactic **expands** to a **lazy list** of **declarative proof scripts** (cf. justification function of LCF tactics)
- choice points are due to **matching**, **internal computations**
- access to **underlying programming** language only at specific points
- general form

```
strategy name
cases
 matcher
 where cond
 with assignments
-> proofscript
with assignments
   ⋮
```

Expansion

1. match context
2. evaluate **metalevel condition**
   - compute value of **schematic variables** in where part
3. insert proof script
   - compute value of remaining **schematic variables**

Integration of External Systems

- natural integration of external systems
- procedural decision procedure to close gap

```
strategy maximafactorabs
cases
 * |- ((abs(GOALLHS)) < GOALRHS) ->
 proof
  L1:   abs(Y) < GOALRHS
  proof
   L2:(Y = GOALLHS) by abeliandecide
   L3:abs(Y) = abs(GOALLHS) by (f=abs in arg_cong) from L2
  qed
 qed
 with Y = (maxima-factor "GOALLHS")
```

A first example

The following problem is taken from [JKK+05]. Given the goal

$$3 + f(x) + g(x, y) = x + g(x, y) + h(y, x)$$

write a tactic that cancels common summands to obtain

$$3 + f(x) = x + h(y, x)$$

```
strategy cancelsum
cases * |- A_1 + .. + A_N = B_1 + .. + B_M
 proof
  L1: C_1 + .. + C_N = D_1 + .. + D_M
  A_1 + .. + A_N = B_1 + .. + B_M from L1
 qed
 with
  foreach I = 1..N where (not (member ``A_I'' ``B'')) C_I = A_I
  foreach I = 1..M where (not (member ``B_I'' ``A'')) D_I = B_I
```

A first example

The following problem is taken from [JKK+05]. Given the goal

$$3 + f(x) + g(x, y) = x + g(x, y) + h(y, x)$$

write a tactic that cancels common summands to obtain

$$3 + f(x) = x + h(y, x)$$

```
strategy cancelsum
cases * |- A_1 + ..  + A_N = B_1 + ..  + B_M
 proof
  L1:  C_1 + ..  + C_N = D_1 + ..  + D_M
  A_1 + ..  + A_N = B_1 + ..  + B_M from L1
 qed
 with
  foreach I = 1..N where (not (member ‘‘A_I’’ ‘‘B’’)) C_I = A_I
  foreach I = 1..M where (not (member ‘‘B_I’’ ‘‘A’’)) D_I = B_I
```

- Heuristic: Factor bounding

  "The following rule is stated for simplicity using only two factors, but the rule is implemented for a product of any number of factors."

  (cf. [Bee98])

$$\frac{\Gamma, |\alpha| < \delta \vdash |\gamma| < M \qquad \Gamma, |\alpha| < \delta \vdash |\beta| < \epsilon/(M+1)}{\Gamma, |\alpha| < \delta \vdash |\beta\gamma| < \epsilon}$$

- combines
  - **integration of external systems**
  - **dynamic continuation**


```
strategy factorbound
 cases
  abs(LHS)<RHS,* |- abs(GOALLHS) < GOALRHS
  where (and (variable-eigenvar.is "GOALRHS")
             (metavar-is "RHS")
             (some #'(lambda (x) (term= "LHS" "x")) "Y_1 ..  Y_N"))
  with  Y_1 * ..  * Y_N = (maxima-factor "GOALLHS")
        j = (termposition "LHS" "Y_1 ..  Y_N")
  ->
  proof
    L1:  GOALLHS=Y_1 * ..  * Y_N by abeliandecide
    foreach i in 1..N where (not (= "j" "i"))
      Y_j <= MV_j by linearbound
    end
    L2:  abs(GOALLHS)=abs(Y_1 * ..  * Y_N) from L1
    .<= abs(Y_1) * ..  * abs(Y_N)
    .< MV_1 * ..  * MV_N
    .<= GOALRHS
  qed
  with foreach i in 1..N
         M_i = (if (= "i" "j") "RHS" (make-metavar (term-type "RHS")))
```

Challenge (3)

theorem *th1:* $\lim_{x \to 3} \frac{x^2-5}{x-2} = 4$
proof
  subgoals
    subgoal $|\frac{x^2-5}{x-2} - 4| < \epsilon$ **using** *A1:*$\epsilon > 0$ **and** *A2:*$|x-3| <?\delta$ **by** *factorbound*
    subgoal $?\delta > 0$ **using** $\epsilon > 0$
  end by *limdefbw*
qed

Challenge (3)

theorem *th1:* $\lim_{x\to 3} \frac{x^2-5}{x-2} = 4$
proof
  subgoals
    subgoal $|\frac{x^2-5}{x-2} - 4| < \epsilon$ **using** *A1:* $\epsilon > 0$ **and** *A2:* $|x-3| <?\delta$
    proof
      L1: $\frac{x^2-5}{x-2} - 4 = (x-3) * (\frac{1}{x-2}) * (x-1)$ **by** *abeliandecide*
      $|x-1| \leq ?MV1$ **by** *linearbound*
      $|\frac{1}{x-2}| \leq ?MV2$ **by** *linearbound*
      L2: $|\frac{x^2-5}{x-2} - 4| \leq |(x-3) * (\frac{1}{x-2}) * (x-1)|$ **from** *L1*
      $. \leq |x-3| * |\frac{1}{x-2}| * |x-1|$
      $. <?\delta * ?MV1 * ?MV2$
      $. \leq \epsilon$
    qed
    subgoal $?\delta > 0$ **using** $\epsilon > 0$
  end by *limdefbw*
qed

Related Work

- MATITA: translation of proof terms to declarative proofs
  - **granularity** problem
  - does not address **specification of tactics**
- ISAPLANNER
  - generates **declarative proof scripts**
  - reasoning techniques **specified in ML**, not within proof document
  - provides **gap** command
- LTAC (intermediate tactic language in procedural style, COQ)
  - also provides **pattern matching** constructs
- ACL2:
  - **syntaxp** metalevel statement to check for a particular structure, no proof obligation
  - **meta-function**, need to be proved by **meta-rule**
  - **bind-free** binds free variables of a rule

Summary

- **declarative tactics** in analogy to procedural tactics to **automate declarative proof**
- defined **on top** of declarative proof language
  - **declarative specification** within proof document
  - **declarative justification** by expansion
- in the spirit of **proof planning** and **proof sketches**
  - in particular: **integration of external systems** (oracle mechanism)
- **expressive** pattern language
  - to provide **context** in form of **preconditions**
  - to **analyze** the result of tactics/external systems and express **continuation** accordingly
- restricted access to underlying programming language at specific points

References I

Michael Beeson.
Automatic generation of epsilon-delta proofs of continuity.
In Jacques Calmet and Jan A. Plaza, editors, *Artificial Intelligence and Symbolic Computation, International Conference AISC'98, Plattsburgh, New York, USA, September 16-18, 1998, Proceedings*, volume 1476 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 1998.

Alan Bundy.
The use of explicit plans to guide inductive proofs.
In R. Lusk and R. Overbeek, editors, *Proceedings CADE-9*, LNAI, pages 111–120. Springer, 1988.

Claudio Sacerdoti Coen.
Declarative representation of proof terms.
J. Autom. Reasoning, 44(1-2):25–52, 2010.

L. Dixon.
A Proof Planning Framework for Isabelle.
PhD thesis, University of Edinburgh, 2005.

Dominik Dietrich, Ewaryst Schulz, and Marc Wagner.
Authoring verified documents by interactive proof construction and verification in text-editors.
In *Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics. July 31 - August 1, Birmingham,*

United Kingdom, volume 5144 of *Lecture Notes in Artificial Intelligence, LNAI*, pages 398–414. Springer, Berlin, Heidelberg, 2008.

Warren A. Hunt Jr., Matt Kaufmann, Robert Bellarmine Krug, J. Strother Moore, and Eric Whitman Smith.
Meta reasoning in acl2.
In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2005.

Freek Wiedijk.
Formal proof sketches.
In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers*, volume

3085 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2004.