Human-Readable Machine-Verifiable Proofs for Teaching Constructive Logic

Andreas Abel, Bor-Yuh Evan Chang, and Frank Pfenning

Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs
International Joint Conference on Automated Reasoning
Siena, Italy
June 19, 2001

Disclaimer: Work in Progress!

Acknowledgments: Steve Awodey, University Education Council

A Course in Constructive Logic

- Website: `http://www.cs.cmu.edu/~fp/courses/logic/`
- Outline:
  - Intuitionistic propositional logic
  - Proofs as programs
  - Recursion
  - First-order logic
  - Arithmetic
  - Structural induction
  - Decidable fragments
- One goal: teach how to prove formally
- Audience: mostly 3rd/4th year undergraduate Computer Science students
- Computer support desirable for assignments

Tutch - A *Tut*orial Proof *Ch*ecker

- Compiler-like tool
  - input: a text file with proofs written following a strict grammar
  - output: indication of acceptance or of gaps remaining in the proofs
- Linear syntax of single-step natural deduction (ND) proofs
- Also supports proofs given by proof terms
- Contrast with interactive proof tutor systems
- Well received in its initial use in an undergraduate course.

Overview

- Tutch syntax for single-step natural deduction proofs
  - examples
  - experiences from usage in an undergraduate logic course
- Toward human-readable machine-verifiable proofs
  - motivation for extending Tutch
- Extending Tutch
  - contrasting examples
  - focused proofs
- Conclusion

Tutch Syntax

- Linearization of natural deduction trees
- Sequence of assertions
- Step must follow using a single inference rule from already proven propositions
- Final step is the assertion proven
- Brackets scope use of assumptions – *frames*
- No explicit justification necessary

Example: Modus Ponens

$$\dfrac{\dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A}\,\wedge\mathcal{E}_1 \qquad \dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A \supset B}\,\wedge\mathcal{E}_2}{\dfrac{B}{A \wedge (A \supset B) \supset B}\,\supset\mathcal{I}^u}\,\supset\mathcal{E}$$

```
proof mp: A & (A=>B) => B =
begin
  [ A & (A=>B);
    A;
    A=>B;
    B ];
  A & (A=>B) => B
end;
```

Tutch Syntax

Proof	S^+	$::=$	A	*Final step*
		$\mid$	$S\,;S^+$	*Step sequence*
Step	S	$::=$	A	*Assertion*
		$\mid$	$\texttt{[}H\,;S^+\texttt{]}$	*Frame*
Hypothesis	H	$::=$	A	*Assertion* $(\supset\mathcal{I}, \vee\mathcal{E})$
		$\mid$	$x{:}\tau$	*Parameter* $(\forall\mathcal{I})$
		$\mid$	$x{:}\tau, A(x)$	*Constraint* $(\exists\mathcal{E})$

Tutch Syntax

- Notational definitions

$$\begin{array}{rcl} \neg A & = & A \supset \bot \\ A \equiv B & = & (A \supset B) \wedge (B \supset A) \end{array}$$

- Concrete syntax

$\top, \bot$	`T, F`	truth, absurdity
$A \equiv B$	`A <=> B`	A if and only if B
$A \supset B$	`A => B`	A implies B
$A \vee B$	`A | B`	A or B
$A \wedge B$	`A & B`	A and B
$\neg A$	`~A`	not A
$\exists x{:}\tau. A(x)$	`?x:t.A(x)`	there exists $x{:}t$ s.t. $A(x)$
$\forall x{:}\tau. A(x)$	`!x:t.A(x)`	for all $x{:}t$, $A(x)$

Example: Exist-Not Implies Not-All

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\exists x{:}\tau.\neg A(x)}\,u \qquad \dfrac{\dfrac{}{\neg A(c)}\,w \qquad \dfrac{\dfrac{}{\forall x{:}\tau.A(x)}\,v \quad \dfrac{}{c{:}\tau}\,c}{A(c)}\,\forall\mathcal{E}}{\bot}\,{\supset}\mathcal{E}}{\bot}\,\exists\mathcal{E}^{c,w}}{\neg\forall x{:}\tau.A(x)}\,{\supset}\mathcal{I}^{v}}{\exists x{:}\tau.\neg A(x) \supset \neg\forall x{:}\tau.A(x)}\,{\supset}\mathcal{I}^{u}$$

```
proof EnnA : (?x:t.~A(x)) => (~!x:t.A(x)) =
begin
  [ ?x:t.~A(x);
    [ !x:t.A(x);
      [ c:t, ~A(c);
        A(c);
        F ];
      F ];
    ~!x:t.A(x) ];
  (?x:t.~A(x)) => (~!x:t.A(x))
end;
```

Student Experience

- Midterm evaluation:
  - *Utility* (avg. score: 4.28)
    * 15 out of 26 students rated Tutch *very helpful* (5 out of 5 points)
    * only 1 student found it *unhelpful* (1 point)
  - *Usability* (avg. score: 3.96)
    * attribute to the similarity to programming
- Personal experience:
  - Forced understanding of each step
  - Motivated appreciation of logical system
  - Appreciated familiar programming-like interface

Issues

- Becomes tedious to explicitly state one-step inferences in the natural deduction calculus after the logic has been mastered
- Granularity of single step in the natural deduction calculus is too small
- Proving mathematical theorems or properties of programs is infeasible in this manner
- Explicitness interrupts rather than support flow of reasoning
- Rigorous mathematical proofs rely on humans applying rules "in the background"

Toward Human-Readable Machine-Verifiable Proofs

- Two extremes:
  - supply each ND proof step (Tutch linear syntax)
  - give only proposition (fully automated theorem prover)
- Compromise: Language for proofs that are
  - readable for humans (in the way JAVA source code is readable)
  - efficiently verifiable by machine
- Size of proof steps should be logically justified
  - *Focused Proofs* (Andreoli)
  - *Assertion Level Proofs* (Huang)

Focused Proofs

- Classification of Sequent Calculus rules

	Left Rules (Hypotheses)	Right Rules (Conclusion)
Invertible	$\vee L$, $\exists L$, $\wedge L$, $\bot L$	$\supset R$, $\forall R$, $\wedge R$, $\top R$
Non-Invertible	$\supset L$, $\forall L$, $\wedge L_1$, $\wedge L_2$	$\vee R_1$, $\vee R_2$, $\exists R$

- Strategy of *focusing* is complete [Andreoli '92][Pfenning '99]

  1. Apply invertible rules
  2. Focus on a hypothesis or the conclusion and apply sequence of non-invertible rules

Proofs on the Assertion Level

- Proof presentation for classical logic (PROVERB project)
- Three levels of justifications [Huang '94]

  Logical level Tutch as described above operates at this level where each step is explictly expressed.

  Assertion level Humans in mathematical proofs give justification at this level by citing axioms, definitions, and theorems.

  Proof level Justifications such as "by analogy" are at the proof level.

- Proof step at the assertion level is equivalent to a chain of non-invertible rules.
- *Goal*: Extend Tutch to allow steps at the assertion level. Plus: Chain invertible rules.

Extending Tutch - Guiding Principle

- What is considered a single proof step in mathematical practice?

  1. Introduction of new hypotheses ("assume", "let") and parameters ("fix").
  2. Application of an axiom, a definition, a lemma or a theorem.
  3. Application of a local lemma.
  4. Distinguishing cases.
  5. ~~Initiating mathematical induction.~~
  6. ~~Reference to the induction hypothesis.~~
  7. ~~Use of a special inference rule for a special area of mathematics.~~

Old and New Syntax

```
P = (A&B | C) & (A=>B=>D) => (C | D)
```

```
proof ex1 : P =
begin
  [ (A&B | C) & (A=>B=>D);
    A => B => D;
    A&B | C;
    [ A&B;
      A;
      B => D;
      B;
      D;
      C | D];
    [ C;
      C | D];
    C | D ];
  P
end;
```

```
assertion proof ex1 : P =

assume (A&B | C) & (A=>B=>D) in

  case A&B | C of
       A&B -->




               D

    || C   --> C

  proves C | D

end;
```

Extending Tutch - Syntax

Proof	S^+	$::=$	$S \mid S; S^+$
Step	S	$::=$	$\texttt{assume}\ H_1, \ldots, H_n\ \texttt{in}\ S^+\ \texttt{end}$
		$\mid$	$\texttt{case}\ \vec{A}\ \texttt{of}\ \vec{K}^1 \longrightarrow S^{+1} \mid\mid \ldots \mid\mid \vec{K}^n \longrightarrow S^{+n}$
			$\texttt{proves}\ C$
		$\mid$	$A\ \texttt{by lemma}\ l$
		$\mid$	$\texttt{triv}\ A$
Hypothesis	H	$::=$	$A \mid x{:}\tau$
Constraint	K	$::=$	$\langle x_1{:}\tau_1, \ldots, x_m{:}\tau_m \rangle A$

Extending Tutch - Syntax Classification

	Left Rules (Hypotheses)	Right Rules (Conclusion)
Inv.	$\vee L$, $\exists L$, $\bot L$	$\supset R$, $\forall R$
Structure	Case distinction and witness extraction.	Hypothesis and parameter introduction.
	`case`	**`assume`**
Non-Inv.	$\supset L$, $\forall L$, $\wedge L_1$, $\wedge L_2$	$\vee R_1$, $\vee R_2$, $\exists R$, $\wedge R$, $\top R$, $\supset R^-$, $\forall R^-$, $\bot L$
Strategy	*Focusing*	*Finishing*
	`lemma`, **`triv`** (focus on hyp.)	**`triv`** (focus on conclusion)

- $\wedge L$ is always available
- $\supset R^-$ and $\forall R^-$ are the non-invertible forms of $\supset R$ and $\forall R$

Extending Tutch - How to Verify Assertion Proofs

Before Verify a step by checking that it follows directly using a single inference rule.

Now Verify a step by focused proof search.

- still decidable
- polynomial complexity
- prototype implementation in Twelf
- soundness formally proven
- completeness wrt. one-step inferences formally proven
- logically justified $\longrightarrow$ intuitive(?)

Example: Split Natural Numbers

axiom *indNat* : $P(0) \supset (\forall x{:}nat.\, P(x) \supset P(s(x))) \supset \forall n{:}nat.P(n)$;
axiom *eq0* : $0 = 0$;
axiom *eqS* : $\forall x{:}nat.\forall y{:}nat.\, x = y \supset s(x) = s(y)$;

assertion proof *splitNat* **:** $\forall x{:}nat.\, 0 = x \vee \exists y{:}nat.s(y) = x \equiv$
assume $x{:}nat$ **in**

  % Induction on $x{:}nat$

  % Base case: $x = 0$
  $0 = 0$ **by axiom** *eq0*;

  % Step case: $x = s(x')$
  assume $x'{:}nat,\ 0 = x' \vee \exists y{:}nat.s(y) = x'$ **in**
    case $0 = x' \vee \exists y{:}nat.s(y) = x'$ **of**
      $0 = x' \longrightarrow s(0) = s(x')$ **by axiom** *eqS*
     || $y{:}nat$ **where** $s(y) = x' \longrightarrow s(s(y)) = s(x')$ **by axiom** *eqS*
    proves $0 = s(x') \vee \exists y{:}nat.s(y) = s(x')$
  end;

  $0 = x \vee \exists y{:}nat.s(y) = x$ **by axiom** *indNat*
end;

Related Work

- *Mizar* [Rudnicki '92]
  - Mathematics formalized in syntax close to natural language
- *Isar* [Wenzel '99]
  - High-level proof language for theorem prover Isabelle
  - Derived inference rules instead of focusing proofs
  - No chaining of left-invertible rules
  - Interface to tactics
- Proof verbalization - PROVERB [Huang & Fiedler '97]

Future Work

- Implement big-step checking in Tutch
- Syntax for induction
- Add support for equational reasoning

Summary

- Compiler-like proof checker Tutch
  - linearization of intuitionistic natural deduction proofs
  - noted positive experience in the classroom due to programming like interface
- Human-readable machine-verifiable proofs
  - Four basic constructs (**`assume`**, **`case`**, **`lemma`**, **`triv`**)
  - Derived from focused proof search
  - Applicable in other logics (classical, linear, temporal, modal, ...)