From Formal Natural Deduction Proofs to LaTeX Proof Trees

International Conference on Mathematical and Computational Linguistics for Proofs

Andrija Urošević & Sana Stojanović Đurđević

University of Belgrade
Faculty of Mathematics
{andrija.urosevic,sana.stojanovic.djurdjevic}@matf.bg.ac.rs

Université Paris–Saclay, September 17, 2025.

Overview

1. Introduction
2. Natural Deducton & Proof Trees
3. Motivation
4. Proof Assistant Theodore
5. Demo
6. Conclusion, Future Work & Related Work

Introduction

Proof Assistant: Theodore

- Proof assistant based on first-order logic (FOL) with natural deduction (ND) rules.
- Developed in the functional programming language Haskell.
- Executed in the interactive environment GHCi.
- A way to generate LaTeX proof trees in a bussproofs-style.
- Online at: `https://github.com/andrija-urosevic/theodore`

Introduction

Proof Assistant: Theodore

- Proof assistant based on first-order logic (FOL) with natural deduction (ND) rules.
- Developed in the functional programming language Haskell.
- Executed in the interactive environment GHCi.
- A way to generate LaTeX proof trees in a bussproofs-style.
- Online at: `https://github.com/andrija-urosevic/theodore`

Example: Distributivity of Universal Quantifier over Conjuction

$$\forall x.(A(x) \land B(x)) \Rightarrow (\forall x.A(x) \land \forall x.B(x))$$

Natural Deduction & Proof Trees

System of reasoning based on infrence rules of the form:

$$\frac{\mathcal{H}_1 \qquad \mathcal{H}_2 \qquad \dots \qquad \mathcal{H}_n}{\mathcal{C}}$$

- We call $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_n$ hypothesis or premises, and $\mathcal{C}$ conclusion.
- Hypothesis and conclusion are of form $\Gamma \vdash A$, meaning that Γ is a context (list of formulas) from which the formula A can be derived or proven.

There are two distinct types of inference rules:

- introduction rules;
- elimination rules.

Natural Deduction & Proof Trees

The proof or derivation is a sequence of steps (inference rules). Because of its branching structure we call it proof tree, that is consisted of:

Leaves Empty nodes followed by assumption rule.

Inter nodes Formulas obtained be applying inference rules.

Root The conclusion derived from the assumptions.

Simple Example: Derivation of conclusion R from context $\Gamma := P, P \Rightarrow Q, Q \Rightarrow R$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P} \quad \dfrac{}{\Gamma \vdash P \Rightarrow Q}}{\Gamma \vdash Q} \qquad \dfrac{}{\Gamma \vdash Q \Rightarrow R}}{\Gamma \vdash R}$$

Natural Deduction & Proof Trees

The proof or derivation is a sequence of steps (inference rules). Because of its branching structure we call it proof tree, that is consisted of:

- **Leaves** At the top of the trees are assumptions.
- **Inter nodes** Formulas obtained be applying inference rules.
- **Root** The conclusion derived from the assumptions.

Simple Example (Alternative representation)

$$\dfrac{\dfrac{[P] \qquad [P \Rightarrow Q]}{Q} \qquad [Q \Rightarrow R]}{R}$$

ND Infrence Rules for Intuicionistic FOL

Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B} \Rightarrow_E$$

Equivalence

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash B \Rightarrow A}{\Gamma \vdash A \Leftrightarrow B} \Leftrightarrow_I \qquad \frac{\Gamma \vdash A \Leftrightarrow B}{\Gamma \vdash A \Rightarrow B} \Leftrightarrow_{E_1} \qquad \frac{\Gamma \vdash A \Leftrightarrow B}{\Gamma \vdash B \Rightarrow A} \Leftrightarrow_{E_2}$$

Exact

$$\frac{}{\Gamma, A \vdash A} \text{asm}$$

ND Infrence Rules for Intuicionistic FOL

Negation

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_I \qquad\qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg_E$$

Conjuction

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_I \qquad\qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_{E_1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_{E_2}$$

Disjunction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee_{I_1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee_{I_2} \qquad\qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ \wedge_E$$

ND Infrence Rules for Intuicionistic FOL

Logical Constants

$$\frac{}{\Gamma \vdash \top} \top_I \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot_E$$

Universal Quantifier

$$\frac{\Gamma \vdash A[y/x]}{\Gamma \vdash \forall x.A} \forall_I \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[y/x]} \forall_E$$

Existential Quantifier

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A} \exists_I \qquad \frac{\Gamma \vdash \exists x.A \qquad \Gamma, A[t/x] \vdash B}{\Gamma \vdash B} \exists_E$$

Example $\LaTeX$ Proof Trees

Distributivity of Universal Quantifier over Conjuction

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\forall x.(A(x) \land B(x)) \vdash \forall x.(A(x) \land B(x))}\,\text{ams}
}{\forall x.(A(x) \land B(x)) \vdash A(a) \land B(a)}\,\forall_E\ (a)
}{\forall x.(A(x) \land B(x)) \vdash A(a)}\,\land_{E_1}
}{\forall x.(A(x) \land B(x)) \vdash \forall x.A(x)}\,\forall_I\ (a)
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\forall x.(A(x) \land B(x)) \vdash \forall x.(A(x) \land B(x))}\,\text{ams}
}{\forall x.(A(x) \land B(x)) \vdash A(a) \land B(a)}\,\forall_E\ (a)
}{\forall x.(A(x) \land B(x)) \vdash B(a)}\,\land_{E_2}
}{\forall x.(A(x) \land B(x)) \vdash \forall x\, B(x)}\,\forall_I\ (a)
}{
\dfrac{\forall x.(A(x) \land B(x)) \vdash \forall x.A(x) \land \forall x.B(x)}{\vdash \forall x.(A(x) \land B(x)) \Rightarrow (\forall x.A(x) \land \forall x.B(x))}\,\Rightarrow_I
}\,\land_I
$$

Motivation

Goal

- Enhance process of constructing and verifying formal ND proofs.
- Enhance process of reading formal ND proofs.

Current proof assistants

- Do enhance process of constructing and verifying formal ND proofs.
- But fail to enhance process of reading formal ND proofs.

Let's examine some of the proof assistants through the lens of ND proofs.

Motivation

```
lemma all_conj_distrib:
  "(∀x. P x ∧ Q x) ⟶ (∀x. P x) ∧ (∀x. Q x)"
  apply (rule impI)
  apply (rule conjI)
   apply (rule allI)
   apply (erule_tac x=x in allE)
   apply (erule conjE)
   apply assumption
  apply (rule allI)
  apply (erule_tac x=x in allE)
  apply (erule conjE)
  apply assumption
  done
```

Isabelle's apply-scripts

- gives lists of rules
- lacks structure

Motivation

```
lemma all_conj_distrib:
  "(∀x. P x ∧ Q x) ⟶ (∀x. P x) ∧ (∀x. Q x)"
proof (rule impI)
  assume h: "∀x. P x ∧ Q x"
  show "(∀x. P x) ∧ (∀x. Q x)"
  proof (rule conjI)
    show " ∀x. P x"
    proof (rule allI)
      fix x
      from h show "P x"
      proof (erule_tac x="x" in allE)
        assume "P x ∧ Q x"
        then show "P x"
        proof (rule conjE)
        ...
```

Isar Proofs

- gives structure
- a lot of boilerplate code

Motivation

```
lemma all_conj_distrib:
  "(∀x. P x ∧ Q x) ⟶ (∀x. P x) ∧ (∀x. Q x)"
  by auto
```

Automated Proofs

- easily writen
- super unreadable

Motivation

```
universe u
variable {α : Type u} {P Q : α → Prop}

theorem all_conj_distrib :
  (∀ x, P x ∧ Q x) → ((∀ x, P x) ∧ (∀ x, Q x)) :=
by
  intro h
  constructor
  · intro x
    exact (h x).left
  · intro x
    exact (h x).right
```

Lean4 Proofs

- gives structure
- types and tactics

Motivation

```
universe u
variable {α : Type u} {P Q : α → Prop}

theorem all_conj_distrib :
  (∀ x, P x ∧ Q x) → ((∀ x, P x) ∧ (∀ x, Q x)) :=
fun h => ⟨
    (fun x => (h x).left),
    (fun x => (h x).right)
⟩
```

Lean4 term-style

- gives structure
- types

Motivation

```
_×_ : (A : Set) (B : Set) → Set
A × B = Σ A λ _ → B

variable
    A : Set
    P Q : A → Set

all-conj-distrib : ((∀ x → P x) × (∀ x → Q x))
                 → (∀ x → P x × Q x)
all-conj-distrib (hP , hQ) x = hP x , hQ x
```

Agda's pattern-matching

- split between elimination and introduction
- Σ-types and Π-types

Proof Assistant Theodore

Designed for backward chaining or top-down approach (goal oriented).
Two modules:

- `FOL` — used for stating and manipulating FOL formulas;
- `Theodore` — used for stating, proving, and exporting proofs as LaTeX proof trees.

Proof Assistant Theodore

FOL Term

- A term can be:
  - variable (Ex. $x, y, z, \ldots$);
  - constant (Ex. $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$);
  - functional symbols applied to terms (Ex. $f(t_1, t_2, \ldots, t_n), g(l_1, l_2, \ldots, l_m), \ldots$).

```
data Term = Var String
          | Fun String [Term]
```

Functional symbols without arguments is a constant.

```
mkConstTerm :: String -> Term
mkConstTerm c = Fun c []
```

Proof Assistant Theodore

FOL Formula

- A FOL fromula can be:
  - logical constant $(\top, \bot)$;
  - logical variable (Ex. $p, q, r, \ldots$);
  - relational symbols (Ex. $P(f(a), b)$);
  - negation of formula (Ex. $\neg A$);
  - conjuction of formulas (Ex. $A \wedge B$);
  - disjuction of formulas (Ex. $A \vee B$);
  - implication of formulas (Ex. $A \Rightarrow B$);
  - equivalence of formulas (Ex. $A \Leftrightarrow B$);
  - universaly quantified formula (Ex. $\forall x.A(x)$);
  - existentialy quantified formula (Ex. $\exists x.A(x)$);

Proof Assistant Theodore

FOL Formula

```
data Formula = Top
             | Bot
             | Rel String [Term]
             | Neg Formula
             | Conj Formula Formula
             | Disj Formula Formula
             | Impl Formula Formula
             | Eqiv Formula Formula
             | Alls String Formula
             | Exis String Formula
```

Relational symbol without arguments is a logical variable.

- ```
  mkVar :: String -> Formula
  mkVar p = Rel p []
  ```
  ```

Proof Assistant Theodore

Distributivity of Universal Quantifier over Conjuction

```
let f = Impl
            (Alls "x"
                (Conj
                    (Rel "A" [Var "x"])
                    (Rel "B" [Var "x"])
                )
            )
            (Conj
                (Alls "x" (Rel "A" [Var "x"]))
                (Alls "x" (Rel "B" [Var "x"]))
            )

> f
∀x.(A(x) ∧ B(x)) → (∀x.A(x) ∧ ∀x.B(x))
```

Proof Assistant Theodore

Theodore's Meta Variables & Context

- We define $\mathcal{U}$ meta variables as a list of strings.

```
type MetaVars = [String]
```

- Also, we define assumption or premise as a named FOL formula.

```
data Assumption = Assumption { name     :: String
                             , formula  :: Formula }
```

- That let us define context or hypothesis Γ as a list of assumptions.

```
type Assumptions = [Assumption]
```

Proof Assistant Theodore

Theodore's Conjecture/Subgoal & Goal

- We can define conjecture or subgoal as $\mathcal{U}, \Gamma \vdash C$, where C is FOL formula.

```
data Subgoal = Subgoal { mvars :: MetaVars
                       , assms :: Assumptions
                       , cncls :: Formula }
```

- Finally, it is natural to define the goal as a list of subgoals.

```
type Goal = [Subgoal]
```

Goal with one subgoal and without meta variables

- ```
  mkGoal :: Assumptions -> Formula -> Goal
  mkGoal assms cncls = [ Subgoal [] assms cncls ]
  ```
  ```

Proof Assistant Theodore

Distributivity of Universal Quantifier over Conjuction

```
let lemmaAllConjDistrib = mkGoal [] f


> lemmaAllConjDistrib
Goal (1 subgoals):

1. subgoal
⊢ ∀x.(A(x) ∧ B(x)) → (∀x.A(x) ∧ ∀x.B(x))
```

Proof Assistant Theodore

Proof Structure

```
data Proof = ToDo
           | Exact  { assmName  :: String }
           | ImplI  { assmName  :: String
                    , proof     :: Proof }
           | ConjI  { proofA    :: Proof
                    , proofB    :: Proof }
           | DisjlI { proof     :: Proof }
           | DisjrI { proof     :: Proof }
           | EqivI  { assmName  :: String
                    , proofA    :: Proof
                    , proofB    :: Proof }
           | NegI   { assmName  :: String
                    , proof     :: Proof }
           | AllsI  { mvar      :: String
                    , proof     :: Proof }
           | ExisI  { mvar      :: String
                    , proof     :: Proof }
           ...
           | ImplE  { assmName  :: String
                    , proofA    :: Proof
                    , proofB    :: Proof }
           | ConjE  { assmName  :: String
                    , proof     :: Proof }
           | DisjE  { assmName  :: String
                    , proofA    :: Proof
                    , proofB    :: Proof }
           | EqivE  { assmName  :: String
                    , proof     :: Proof }
           | NegE   { assmName  :: String
                    , proof     :: Proof }
           | AllsE  { mvar      :: String
                    , assmName  :: String
                    , proof     :: Proof }
           | ExisE  { mvar      :: String
                    , assmName  :: String
                    , proof     :: Proof }
```

Proof Assistant Theodore

Applying The Proof

- To check proof we use the following function:

  ```
  apply :: Proof -> Goal -> Goal
  ```

- Function `apply` traverses the proof tree, and at each node it applies inference rule that transforms current goal.
- That allows us to incrementally update our proof tree, based on the current goal.
- Applying invalid proof to the goal will result in an error.
- Valid proof will result in an empty goal.

Demo

Demo

Step 0

```
let proofAllConjDistrib = ToDo

> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (1 subgoals):

1. subgoal
⊢ ∀x.(P(x) ∧ Q(x)) → (∀x.P(x) ∧ ∀x.Q(x))
```

Demo

Step 1

```
let proofAllConjDistrib = ImplI "h" ToDo
```

```
> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (1 subgoals):

1. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.P(x) ∧ ∀x.Q(x)
```

Demo

Step 2

```
let proofAllConjDistrib = ImplI "h" (ConjI ToDo ToDo)

> apply proofAllConjDistrib lemmaAllConjDistrib

Goal (2 subgoals):

1. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.P(x)

2. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.Q(x)
```

Demo

Step 3

```
let proofAllConjDistrib =
    ImplI "h"
        (ConjI
            (AllsI "a" ToDo)
            ToDo
        )
```

Demo

Step 3

```
> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (2 subgoals):

1. subgoal
- a
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ P(a)

2. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.Q(x)
```

Demo

Step 4

```
let proofAllConjDistrib =
    ImplI "h"
        (ConjI
            (AllsI "a"
                (AllsE "a" "h" ToDo)
            )
            ToDo
        )
```

Demo

Step 4

```
> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (2 subgoals):

1. subgoal
- a
• P(a) ∧ Q(a) (h)
⊢ P(a)

2. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.Q(x)
```

Demo

Step 5

```
let proofAllConjDistrib =
    ImplI "h"
        (ConjI
            (AllsI "a"
                (AllsE "a" "h"
                    (ConjE "a" ToDo)
                )
            )
            ToDo
        )
```

Demo

Step 5

```
> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (2 subgoals):

1. subgoal
- x
• P(x) (h1)
• Q(x) (h2)
⊢ P(x)

2. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.Q(x)
```

Demo

Step 6

```
let proofAllConjDistrib =
    ImplI "h"
        (ConjI
            (AllsI "a"
                (AllsE "a" "h"
                    (ConjE "h"
                       Exact "h1"
                    )
                )
            )
            ToDo
        )
```

Demo

Step 6

```
> apply proofAllConjDistrib lemmaAllConjDistrib
Goal (1 subgoals):

1. subgoal
• ∀x.(P(x) ∧ Q(x)) (h)
⊢ ∀x.Q(x)
```

Demo

Steps 7-10

```
let proofAllConjDistrib =
    ImplI "h"
        (ConjI
            (AllsI "a"
                (AllsE "a" "h"
                    (ConjE "h"
                        Exact "h1"
                    )
                )
            )
            (AllsI "a"
                (AllsE "a" "h"
                    (ConjE "h"
                        Exact "h2"
                    )
                )
            )
        )
```

Demo

Steps 7-10

```
> apply proofAllConjDistrib lemmaAllConjDistrib

Nothing to prove!
```

Demo

Generating $\LaTeX$ Proof Tree

```
> genLatexTree proofAllConjDistrib lemmaAllConjDistrib
```

```latex
\begin{prooftree}
\AxiomC{}
\RightLabel{$\mathsf{asm}$}
\UnaryInfC{$a; A(a), B(a) \vdash B(a)$}
\RightLabel{$\land_E$}
\UnaryInfC{$a; A(a) \land B(a) \vdash B(a)$}
\RightLabel{$\forall_E(a)$}
\UnaryInfC{$a; \forall x. (A(x) \land B(x)) \vdash A(a)$}
\RightLabel{$\forall_I(a)$}
\UnaryInfC{$\forall x. (A(x) \land B(x)) \vdash \forall x. B(x)$}
\AxiomC{}
\RightLabel{$\mathsf{asm}$}
\UnaryInfC{$a; A(a), B(a) \vdash A(a)$}
\RightLabel{$\land_E$}
\UnaryInfC{$a; A(a) \land B(a) \vdash A(a)$}
\RightLabel{$\forall_E(a)$}
\UnaryInfC{$a; \forall x. (A(x) \land B(x)) \vdash A(a)$}
\RightLabel{$\forall_I(a)$}
\UnaryInfC{$\forall x. (A(x) \land B(x)) \vdash \forall x. A(x)$}
\RightLabel{$\land_I$}
\BinaryInfC{$\forall x. (A(x) \land B(x)) \vdash \forall x. A(x) \land \forall x. B(x)$}
\RightLabel{$\implies_I$}
\UnaryInfC{$ \vdash \forall x. (A(x) \land B(x)) \implies (\forall x. A(x) \land \forall x. B(x))$}
\end{prooftree}
```

Demo

Generating LaTeX Proof Tree

```
> genLatexTree proofAllConjDistrib lemmaAllConjDistrib
```

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{asm}
\UnaryInfC{$a; A(a), B(a) \vdash B(a)$}
\RightLabel{$\wedge_E$}
\UnaryInfC{$a; A(a) \wedge B(a) \vdash B(a)$}
\RightLabel{$\forall_E(a)$}
\UnaryInfC{$a; \forall x.(A(x) \wedge B(x)) \vdash A(a)$}
\RightLabel{$\forall_I(a)$}
\UnaryInfC{$\forall x.(A(x) \wedge B(x)) \vdash \forall x.B(x)$}
\AxiomC{}
\RightLabel{asm}
\UnaryInfC{$a; A(a), B(a) \vdash A(a)$}
\RightLabel{$\wedge_E$}
\UnaryInfC{$a; A(a) \wedge B(a) \vdash A(a)$}
\RightLabel{$\forall_E(a)$}
\UnaryInfC{$a; \forall x.(A(x) \wedge B(x)) \vdash A(a)$}
\RightLabel{$\forall_I(a)$}
\UnaryInfC{$\forall x.(A(x) \wedge B(x)) \vdash \forall x.A(x)$}
\RightLabel{$\wedge_I$}
\BinaryInfC{$\forall x.(A(x) \wedge B(x)) \vdash \forall x.A(x) \wedge \forall x.B(x)$}
\RightLabel{$\implies_I$}
\UnaryInfC{$\vdash \forall x.(A(x) \wedge B(x)) \implies (\forall x.A(x) \wedge \forall x.B(x))$}
\end{prooftree}
$$

Conclusion

Theodore provides:

- writing formal proofs in FOL using ND rules;
- exporting checked formal proofs in LaTeX proof tree format;
- minimal software that can be formaly verified.

Theodore challenges:

- only one style of LaTeX proof tree (for now);
- not integrated with text editors (possible pluggins).

Future Work

A lot can be done:

- Editor pluggins.
- Interface for instanciating previously proved theorems as rules.
- Extending with optinal classical rules.
- Connecting to the other proof assistants, and exporting their proofs to Theodore.
- ...

Related Work

- **Jape** (Just Another Proof Editor) (Richard Bornat & Bernard Sufrin, 1992)
- **Pandora** (Krysia Broda at al., 2007)
- **NaDeA** (Jørgen Villadsen at al., 2019)
- **PeaCoq** (Valentin Robert, 2015)
- **Traf** (Hideyuki Kawabata, 2018)
- **PaperProof** (Evgenia Karunus & Anton Kovsharov, 2024)

Thank you!