Using algorithms in proofs

Péter Diviánszky

translated by
Zoltán A. Kocsis

Debrecen, 29 November 2012

Motivation

The four color theorem

- *conjecture:* Francis Guthrie, 1852
- *computer-assisted proof:* Appel & Haken, 1976
  - proof by exhaustion with billions of cases
  - correctness of the program used wasn't demonstrated satisfactorily
- provenly correct use of computers: Georges Gonthier, 2005

How can we utilize the abilities of computers in a reliable way?

Proof Assistants

We need a minimalist language in which verifying proofs is simple. The proof checking algorithm shall be reliable.

We also need a high-level language in which giving proofs is simple, and a transformation algorithm translating this to the minimal language.

$+$ the ability to use algorithms during the proof process

Outline

1. Denoting proofs

Rules of inference

Implication introduction and elimination:

$$\dfrac{\begin{matrix} A \\ \vdots \\ B \end{matrix}}{A \to B} \qquad\qquad \dfrac{A \to B \qquad A}{B}$$

Deduction tree

$$A \to B \to C$$

$$\vdots$$

$$B$$

$$\vdots$$

$$A$$

$$\vdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A\to B\to C \qquad A}{B\to C} \qquad B}{C}}{A\to C}}{B\to A\to C}}{(A \to B \to C) \to B \to A \to C}$$

Denoting proofs

$x : A \quad \leftrightharpoons \quad x$ describes a deduction of A

Implication introduction and elimination:

$$\dfrac{\begin{array}{c} x : A \\ \vdots \\ y : B \end{array}}{\lambda\, x \to y : A \to B} \qquad\qquad \dfrac{x : A \to B \qquad y : A}{x\, y : B}$$

Along with the predicate, we obtain a description of its proof, sufficient to reconstruct the deduction tree.

Obtaining a proof

$$f : A \to B \to C$$

$$\vdots$$

$$y : B$$

$$\vdots$$

$$x : A$$

$$\vdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f : A \to B \to C \qquad x : A}{f\ x : B \to C} \qquad y : B}{f\ x\ y : C}}{\lambda\, x \to f\ x\ y : A \to C}}{\lambda\, y \to \lambda\, x \to f\ x\ y : B \to A \to C}}{\lambda\, f \to \lambda\, y \to \lambda\, x \to f\ x\ y : (A \to B \to C) \to B \to A \to C}$$

Universal quantification

Universal quantifier introduction & elimination:

$$\frac{\begin{array}{c} A\ \texttt{prop} \\ \vdots \\ B \end{array}}{\forall A : B} \qquad \frac{\forall A : B \qquad C\ \texttt{prop}}{B[A \rightsquigarrow C]}$$

the same thing in a different notation:

$$\frac{\begin{array}{c} A : \mathbb{P} \\ \vdots \\ B \end{array}}{(A : \mathbb{P}) \to B} \qquad \frac{(A : \mathbb{P}) \to B \qquad C : \mathbb{P}}{B[A \rightsquigarrow C]}$$

Example

$$\dfrac{\dfrac{\dfrac{\begin{matrix}A : \mathbb{P}\\ \vdots \\ A \\ \vdots \\ A\end{matrix}}{A \to A}}{(A : \mathbb{P}) \to A \to A} \qquad (B \to A) : \mathbb{P}}{(B \to A) \to B \to A}$$

Universal quantification (2)

Explicitly denoted proofs for universal quantifier introduction & elimination

$$\frac{\begin{array}{c} A : \mathbb{P} \\ \vdots \\ x : B \end{array}}{\lambda A \to x : (A : \mathbb{P}) \to B} \qquad \frac{f : (A : \mathbb{P}) \to B \qquad C : \mathbb{P}}{f\ C : B[A \rightsquigarrow C]}$$

Example

$$\dfrac{\dfrac{\dfrac{\begin{array}{c} A : \mathbb{P} \\ \vdots \\ x : A \\ \vdots \\ x : A \end{array}}{\lambda\, x \to x : A \to A}}{\lambda\, A \to \lambda\, x \to x : (A : \mathbb{P}) \to A \to A} \qquad (B \to A) : \mathbb{P}}{(\lambda\, A \to \lambda\, x \to x)\,(B \to A) : (B \to A) \to B \to A}$$

Remark

The rules governing implication and universal quantification can be unified, reducing mathematical logic to just two rules instead of the usual four.

So we've obtained our minimalist language and (implicity) a proof-checking algorithm.

Now we'll present our high-level language and the translation to the minimalist one.

Naming proofs

We name our proofs to facilitate their reuse:

$$\begin{aligned} \mathrm{id} &: (A : \mathbb{P}) \to A \to A \\ \mathrm{id} &= \lambda A \to \lambda x \to x \end{aligned}$$

Then $\mathrm{id}\,(B \to A) : (B \to A) \to B \to A$.

Simplifying the notation:

$$\begin{aligned} &\mathrm{id} : (A : \mathbb{P}) \to A \to A \\ &\mathrm{id}\,A\,x = x \end{aligned}$$

Hidden arguments

Some arguments can be inferred.

$$\begin{aligned}&\mathrm{id} \;:\; (A : \mathbb{P}) \to A \to A\\ &\mathrm{id}\, A\, x \;=\; x\end{aligned}$$

can be replaced with

$$\begin{aligned}&\mathrm{id} \;:\; \{A : \mathbb{P}\} \to A \to A\\ &\mathrm{id}\, x \;=\; x\end{aligned}$$

Then using id requires fewer arguments:

$\mathrm{id} \;:\; (B \to A) \to B \to A$.

Agda demonstration

$$\mathsf{flip} : \{A\,B\,C : \mathbb{P}\} \to (A \to B \to C) \to B \to A \to C$$
$$\mathsf{flip}\; f\; y\; x \;=\; f\; x\; y$$

$$\mathsf{K} : \{A\,B : \mathbb{P}\} \to A \to B \to A$$
$$\mathsf{K}\; x\; y \;=\; x$$

$$\mathsf{S} : \{A\,B\,C : \mathbb{P}\} \to (A \to B \to C) \to (A \to B) \to A \to C$$
$$\mathsf{S}\; f\; x\; g \;=\; f\; x\; (g\; x)$$

2. Logical connectives

Logical AND

Logical AND can be introduced using three postulates:

$$
\begin{aligned}
\_\wedge\_ &: \mathbb{P} \to \mathbb{P} \to \mathbb{P} \\
\_,\_ &: \{A\ B\ :\ \mathbb{P}\} \to A \to B \to A \wedge B \\
\wedge\text{-elim} &: \{A\ B\ C\ :\ \mathbb{P}\} \to (A \to B \to C) \to A \wedge B \to C
\end{aligned}
$$

In our shorthand this becomes

$$
\begin{aligned}
&\texttt{data}\ \_\wedge\_\ (A\ B\ :\ \mathbb{P})\ :\ \mathbb{P}\quad \texttt{where} \\
&\quad \_,\_\ :\ A \to B \to A \wedge B
\end{aligned}
$$

The `data` keyword generates guaranteedly consistent rules, so this is regarded as a definition, not a postulate.

Pattern matching

Pattern matching is a convenient way of hiding elimination rules, e.g.

$$
\begin{aligned}
&\wedge\text{-comm} \;:\; \{A\,B \;:\; \mathbb{P}\} \to A \wedge B \to B \wedge A\\
&\wedge\text{-comm}\,(x\;,\;y) \;=\; y\;,\;x
\end{aligned}
$$

abbreviates

$$
\begin{aligned}
&\wedge\text{-comm} \;:\; \{A\,B \;:\; \mathbb{P}\} \to A \wedge B \to B \wedge A\\
&\wedge\text{-comm} \;=\; \wedge\text{-elim}\,(\lambda\,x \to \lambda\,y \to (y\;,\;x))
\end{aligned}
$$

Logical constants and negation

```
data ⊤  : ℙ   where
  tt : ⊤

data ⊥  : ℙ   where
```

Elimination rule generated for $\bot$:

$\bot\text{-elim} : \{A : \mathbb{P}\} \to \bot \to A$

Logical negation:

$\neg A = A \to \bot$

Logical OR

Constructive OR:

```
data _⊎_ (A B : ℙ) : ℙ  where
    inj₁ : A → A ⊎ B
    inj₂ : B → A ⊎ B
```

Classical OR:

$$A \vee B \;=\; \neg\neg(A \uplus B)$$

Classical implication:

$$A \Rightarrow B \;=\; A \to \neg\neg B$$

Agda demonstration

$\{A : \mathbb{P}\} \to A \to \neg\neg A$

$\{A : \mathbb{P}\} \to \neg\neg A \Rightarrow A$, reductio ad absurdum

$\{A : \mathbb{P}\} \to A \vee \neg A$, tertium non datur

$\{A\ B : \mathbb{P}\} \to A \uplus B \to A \vee B$, i.e. $\_\uplus\_$ is stronger than $\_\vee\_$

$\{A\ B : \mathbb{P}\} \to (A \to B) \to A \Rightarrow B$, i.e. $\_\to\_$ is stronger than $\_\Rightarrow\_$

$+$ the usual theorems about logical connectives

3. Sets and functions

Dependently typed functions

$f\ :\ (x : A)\ \rightarrow\ B$ is a dependently typed function, i.e. x can occur in B.

When B does not depend on x we write $f\ :\ A \rightarrow B$.

Example:
If $\mathsf{Fin}\ n$ is the set of all natural numbers less than n, then
$a\ :\ (n : \mathbb{N})\ \rightarrow\ \mathsf{Fin}\ (n+1)$ is a sequence such that $a_n < n+1$.

Remark:
$|(x : A)\ \rightarrow\ B| = \prod_{x \in A} |B|$,

e.g. $$\begin{aligned} &|(n : \mathsf{Fin}\ m)\ \rightarrow\ \mathsf{Fin}\ (n+1)| \\ &\quad = \prod_{n \in \mathsf{Fin}\ m} |\mathsf{Fin}\ (n+1)| = \prod_{n \in \mathsf{Fin}\ m} n = m! \end{aligned}$$

Embedding $\mathbb{P}$ in Set

$\mathbb{P} = \mathsf{Set}$, non-empty sets correspond to true propositions.

Implication and universal quantification map to dependently typed function spaces:

$$\begin{aligned} &\mathsf{flip} : \{A\ B\ C : \mathsf{Set}\} \to (A \to B \to C) \to B \to A \to C \\ &\mathsf{flip}\ f\ y\ x \ = \ f\ x\ y \end{aligned}$$

Logical AND corresponds to the Cartesian product:

$$\begin{aligned} &\mathtt{data}\ \_\wedge\_\ (A\ B : \mathsf{Set}) : \mathsf{Set} \quad \mathtt{where} \\ &\quad \_,\_ : A \to B \to A \wedge B \end{aligned}$$

The intuitive definition of Set

Set is the set of all sets.

The contradiction is resolved by the stratification
$\mathsf{Set}_0 \ : \ \mathsf{Set}_1 \ : \ \mathsf{Set}_2 \ : \ \ldots.$

$A : B \quad \leftrightharpoons \quad A$ is an element of the set B

The coloring $x : A : \mathsf{Set}$ is used to improve readability.

Natural numbers

The definition of the naturals:

```
data ℕ : Set where
    zero : ℕ
    suc  : ℕ → ℕ
```

From this, mathematical induction follows as the elimination rule:

```
ℕ-elim :
    (P : ℕ → Set) →
    P zero →
    ((i : ℕ) → P i → P (suc i)) →
    (n : ℕ) → P n
```

Agda demonstration

Definitions of addition and multiplication:

$$
\begin{aligned}
&\_+\_ \; : \; \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&\mathsf{zero} + n \; = \; n \\
&(\mathsf{suc}\; m) + n \; = \; \mathsf{suc}\,(m + n)
\end{aligned}
$$

As before, pattern matching conceals a use of the elimination rule:

$$
\begin{aligned}
&\_+\_ \; : \; \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&k + n \; = \; (\mathbb{N}\text{-}\mathsf{elim}\,(\lambda\, i \;\to\; \mathbb{N})\; n\,(\lambda\, i\; p \;\to\; \mathsf{suc}\; p))\; k
\end{aligned}
$$

The elimination rule guarantees that our functions are well-defined.

4. Automatic simplification

Rearranging proofs

Implication eliminations occuring right before introductions can be simplified:

$$\dfrac{\dfrac{\begin{array}{c} x \;:\; S \\ \vdots \\ y \;:\; T \end{array}}{\lambda\, x \to y : S \to T} \qquad z \;:\; S}{(\lambda\, x \;\to\; y)\; z \;:\; T} \qquad \rightsquigarrow \qquad y[x \rightsquigarrow z] \;:\; T$$

i.e.

$$(\lambda\, x \;\to\; y)\; z \qquad \rightsquigarrow \qquad y[x \rightsquigarrow z]$$

Simplification rules

Along with the rule on the previous slide, `data` can introduce other simplification rules, e.g.

$\texttt{data}\ \_\wedge\_\ (A\ B\ :\ \mathsf{Set})\ :\ \mathsf{Set}\quad \texttt{where}$
$\quad \_,\_\ :\ A \to B \to A \wedge B$

introduces the following:

$\_,\_\ :\ \{A\ B\ :\ \mathsf{Set}\} \to A \to B \to A \wedge B,$
$\wedge\text{-elim}\ :\ \{A\ B\ C\ :\ \mathsf{Set}\} \to (A \to B \to C) \to A \wedge B \to C,$
$\wedge\text{-elim}\ f\ (x, y) \quad \rightsquigarrow \quad f\ x\ y$

Induced rules

$$\begin{aligned}
\_+\_ &: \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\mathsf{zero} + n &= n \\
(\mathsf{suc}\ m) + n &= \mathsf{suc}\ (m + n)
\end{aligned}$$

From the implicit simplification rules for $\mathbb{N}$-elim we obtain

$$\begin{aligned}
\mathsf{zero} + n &\rightsquigarrow n \\
(\mathsf{suc}\ m) + n &\rightsquigarrow \mathsf{suc}\ (m + n)
\end{aligned}$$

which follow intuitively from the definition anyway.

Automatic simplification

Automatic application of simplification rules allows us to use algorithms in the proof process.

- The equality of $1 + 2 * (3 + 4 * 5)$ and 47 can be demonstrated in a single step (because we need to prove the equality of 47 and 47).
- The equality of $3 + (4 + n)$ and $7 + n$ is one step as well.
- The equality proof for $n + 7$ and $3 + (4 + n)$ has to be reduced to the previous case with one additional step.

Remark

The more classical connectives ($\_\vee\_$, $\_\Rightarrow\_$) we replace with constructive ones ($\_\uplus\_$, $\_\rightarrow\_$), the more efficient automated simplification becomes, so it's worthwhile to find the most constructive variant of a given theorem or proof.

For the most part, mathematics is constructive. We could do our proofs in a completely classical environment, but then we'd have to give up automatic simplification almost completely.

Gonthier's proof of the four-color theorem wouldn't be possible without automatic simplification.

Agda demonstration

- definition of equality
- $(\mathbb{N}, \_+\_, \mathsf{zero})$ is a unital Abelian group.
- definition of existential quantification
- definitions of basic concepts of set theory
- definitions and proofs of certain choice axioms

Summary

Intensional type theory is excellent for proof-checking.

- is simple (dependent function spaces & some rules)
- has the strength of infinite-order logic
- has automatic simplification
- the concepts of classical and constructive mathematics can be easily defined

Summary (2)

The Agda language is excellent for proof-giving.

- proofs can be named
- hidden arguments can be inferred automatically
- `data` declarations, pattern matching
- interactive proofs, unicode characters, etc.

References

1. Agda home page
2. **Classical Mathematics for a Constructive World**
   Russell O'Connor
   Mathematical Structures in Computer Science, Volume 21, Special Issue 04, August 2011, pp 861-882, DOI
3. **A computer-checked proof of the four colour theorem**
   Georges Gonthier, 2005