Theorema 2.0: An Open-Source Mathematical Assistant System for Automated and Interactive Reasoning

Wolfgang Windsteiger

RISC, JKU Linz
4232 Hagenberg, Austria

Wolfgang.Windsteiger@risc.jku.at

www.risc.jku.at/home/wwindste/

October 24, 2013

History

≈ 1995: Project initiated by Bruno Buchberger.

Goal 1: create a system that does proofs "like us".

Goal 2: create an integrated system (algorithms + proofs in one system).

First Implementation

- Implementation based on Mathematica 3 (and higher):
  - \+ Notebook interface and two-dimensional syntax.
  - \+ Many proving components implemented by various people (Buchberger, Jebelean, Windsteiger, etc.).
  - \+ Proofs generated and printed in natural style.
  - – System architecture reached its limits (combination of the components).
  - – Although user interface was always counted as a plus, the system was difficult to use in practice.
  - – Execution of algorithms not efficient enough (translation Mathematica $\leftrightarrow$ Theorema).

Re-Design and Re-Implementation

- Implementation based on Mathematica 7 (and higher):
  - Preserve the “+” and improve on the “−”.
  - GUI elements as part of the Mathematica programming language (dynamic expressions).
  - Create an appealing/nice/easy to use interface.
  - Consider interactive proving in the system design from the beginning.
  - Integration of computation and proving.
  - Speed: Implementation in Mathematica programming language $\rightsquigarrow$ try to use efficient built-in operations as much as possible.

In This Talk: Mostly Demo

Program Verification: Theorema could be used as a frame for it.

Demo

1. Computation on top-level
2. Computation in proof
3. Execution of algorithm in proof (does not work yet)
4. Algorithm is an object in the language

In This Talk: Mostly Demo

Program Verification: Theorema could be used as a frame for it.

Proving: General mechanism / On the use of rewriting.

Computation: On the use of rewriting.

GUI: on the fly!

Demo Warm-up: An Easy Proving Example

1. Select proof goal
2. Compose KB
3. activate/deactivate rules
4. activate/deactivate output
5. restore settings
6. automatic status of cells
7. proof display in notebook: tree $\rightsquigarrow$ nested cells
8. proofs are human-readable
9. proofs are human-comprehensible
10. how to input formulas $\rightsquigarrow$ global quantifiers, parentheses

On the Role of Rewriting

Modus Ponens rule: If we know $A \Rightarrow B$ and A then we can infer B.

How would we implement it: If we have $A \Rightarrow B$ in the KB then search for A in the KB and add B to the KB in case of success.

In mathematics very frequently: We know $\underset{x,y,\ldots}{\forall} A_{x,y,\ldots} \Rightarrow B_{x,y,\ldots}$ and have $A_{t,s,\ldots}$. We then want to infer $B_{t,s,\ldots}$. (Combination of instantiation and Modus Ponens.)

How would we implement it: If we have $\underset{x,y,\ldots}{\forall} A_{x,y,\ldots} \Rightarrow B_{x,y,\ldots}$ in the KB then search for a substitution σ such that $A_{x,y,\ldots}\sigma$ is in the KB (see it as a hint for instantiation!) and add $B_{x,y,\ldots}\sigma$ to the KB in case of success.

On the Role of Rewriting: Implementation Aspects

- Main Goal: Try to implement efficiently in Mathematica.
- E.g. use pattern matching and rewrite rules from Mathematica instead of implementing matching and replacement for Theorema expressions in the Mathematica programming language.
- Modus Ponens: turn $A \Rightarrow B$ into `A:>B` and apply it to A, technically `Replace[A,A:>B]` gives `B`.
- Quantified Modus Ponens: turn $\underset{x,y,\ldots}{\forall} A_{x,y,\ldots} \Rightarrow B_{x,y,\ldots}$ into

  $$\mathtt{A}_{\mathtt{x\_,y\_,\ldots}} \mathtt{:>} \mathtt{B}_{\mathtt{x,y,\ldots}}$$

  When this is applied to $A_{t,s,\ldots}$, Mathematica will match the patterns `x_`, `y_`, ... against $t, s, \ldots$, thus find the substitution σ, and give $B_{t,s,\ldots}$, i.e. $B_{x,y,\ldots}\sigma$.

On the Role of Rewriting: Implementation Aspects

Use the same idea for the situation where we know
$\underset{x,y,\ldots}{\forall} A_{x,y,\ldots} \Rightarrow B_{x,y,\ldots}$ and have to prove $B_{t,s,\ldots}$.
It suffices to prove $A_{t,s,\ldots}$.

Turn $\underset{x,y,\ldots}{\forall} A_{x,y,\ldots} \Rightarrow B_{x,y,\ldots}$ into

$$\mathtt{B}_{\mathtt{x\_,y\_,\ldots}} \mathtt{:>} \mathtt{A}_{\mathtt{x,y,\ldots}}$$

When this is applied to $B_{t,s,\ldots}$, we get $A_{t,s,\ldots}$.

On the Role of Rewriting: How it is Applied

Whenever a new formula comes to the KB, all universal quantifiers at the beginning are stripped and all conditions on the quantified variables are collected (from special ranges and explicit conditions).

$$\underset{x_1}{\forall} \ldots \underset{x_l}{\forall} P_{x_1,\ldots,x_l} \rightsquigarrow \{P_{x_1,\ldots,x_l}, \{C_1, \ldots, C_l\}, \{x_1, \ldots, x_l\}\}$$

where the C_i are the conditions on the x_i.

For forward application:

$$\mathtt{C1}_{\mathtt{x}_{1}\mathtt{\_},\ldots,\mathtt{x}_{l}\mathtt{\_}} \mathtt{:>} \mathtt{P}_{\mathtt{x}_1,\ldots,\mathtt{x}_l} \quad \text{and check } \mathtt{Cj}_{\mathtt{x}_{1}\mathtt{\_},\ldots,\mathtt{x}_{l}\mathtt{\_}} \in \text{KB for } j \neq 1$$

For backward application:

$$\mathtt{P}_{\mathtt{x}_{1}\mathtt{\_},\ldots,\mathtt{x}_{l}\mathtt{\_}} \mathtt{:>} \mathtt{C1}_{\mathtt{x}_1,\ldots,\mathtt{x}_l} \wedge \ldots \wedge \mathtt{Cl}_{\mathtt{x}_1,\ldots,\mathtt{x}_l}$$

On the Role of Rewriting: How it is Applied

Special cases:

- P has the form $C \Rightarrow Q$: C is an additional condition.
- $P \Rightarrow Q$ is equivalent to $\neg P \Rightarrow \neg Q \rightsquigarrow$ various rewrite rules for negated occurrences.
- P has the form $P_1 \wedge \ldots \wedge P_n$: For backward application:

$$\mathtt{P}_{\mathtt{j}_{\mathtt{x}_{1\_},\ldots,\mathtt{x}_{l\_}}} \mathtt{:>} \mathtt{C}_{1_{\mathtt{x}_1,\ldots,\mathtt{x}_l}} \wedge \ldots \wedge \mathtt{C}_{l_{\mathtt{x}_1,\ldots,\mathtt{x}_l}} \quad \textbf{for each } j = 1, \ldots, n$$

Similar Situations

1. Expanding definitions
2. Rewriting using equalities
3. Rewriting using equivalences

Equalities/equivalences are used bi-directional: the one that can be applied first deletes the other one in order to avoid cycles. (No experience yet, but think of $x = y$.)

Demo: Rewriting

1. Use Goal-Rewriting AND Knowledge Rewriting (interestingly an alternative is proved twice)
2. Use Knowledge Rewriting ONLY
3. Use Goal-Rewriting ONLY
4. Show proof simplification

Demo: Interactive Proving

1. Select next step
2. Instantiate existential goal

Availability

- Via GitHub.
- Hopefully still 2013.
- `www.risc.jku.at/research/theorema`

Future work:

Availability

- Via GitHub.
- Hopefully still 2013.
- `www.risc.jku.at/research/theorema`

Future work:

$$\infty$$