

Introduction to the *Theorema* system

Isabela Drămnesc

Computer Science Department
West University of Timișoara
Romania

FROM 2024
17 September
Timișoara, Romania

- 1 Introduction to the system
- 2 Algorithm synthesis
- 3 Algorithm verification
- 4 Conclusions

The *Theorema* system

Web page:

www.risc.jku.at/theorema

- Conceived and initiated around 1995 by *Bruno Buchberger* and reflects his view of “doing mathematics”.

Theorema 2.0 is a major re-launch

- Mainly developed by *Wolfgang Windsteiger*.

Implementation: Mathematica

- Proving uses only the rewrite mechanism of Mathematica.

Supports:

- Development of mathematical theories in natural style.
- Proving in natural style.
- Definition and execution of algorithms.
- Construction of provers for various domains.

Installation

Home page:

`www.risc.jku.at/research/theorema/software`

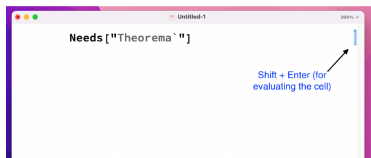
User mode

- Mathematica software needed
- Download the Theorema package and copy it in the Mathematica folder

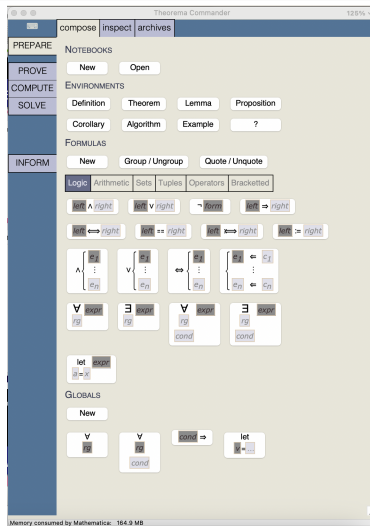
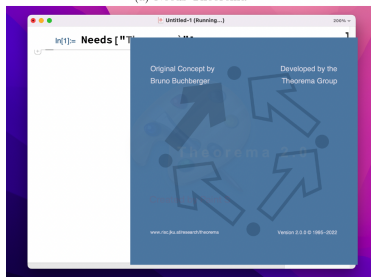
Developer mode

- Needed: Mathematica, Eclipse, JDK, Workbench
- Download the Theorema package and include it in Eclipse
- Installation guide **here**.

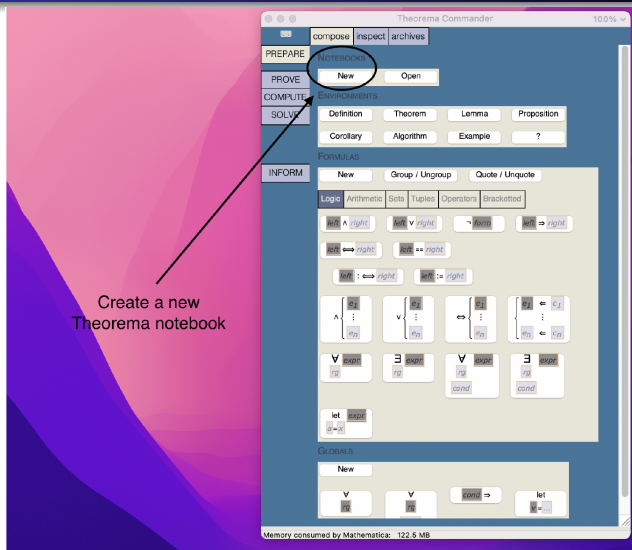
Loading *Theorema* in user mode



(a) Needs Theorema



Create a new notebook



Create a new
Theorema notebook

Proving

The image displays two windows from the Theorema 2.0 software. The left window, titled 'demo.nb', shows a 'Simple proving' section with a 'PROPOSITION (...)' box and a '???' placeholder. The right window, titled 'Theorema Commander', shows a toolbar with buttons for 'compose', 'inspect', 'archives', 'PREPARE', 'PROVE', 'COMPUTE', 'SOLVE', and 'INFORM'. A red circle highlights the 'Proposition' button in the 'SOLVE' section, with an arrow pointing to it. The 'Theorema Commander' window also shows various mathematical symbols and operators in the 'FORMULAS' and 'GLOBALS' sections.

Proving

Simple proving

Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:

1.

$$(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$$

We write the corresponding Proposition in Theorema using the templates in the Commander.

PROPOSITION (...)

???

■

[...]

Click inside the box and choose the template from Theorema Commander

Theorema Commander

compose inspect archives

PREPARE NOTEBOOKS
New Open

PROVE ENVIRONMENTS
Definition Theorem Lemma Proposition

SOLVE
Corollary Algorithm Example ?

FORMULAS
New Group / Ungroup Quote / Unquote

Logic Arithmetic Sets Tuples Operators Bracketed

\forall A right

\forall V right

\neg form

\Rightarrow right

\Leftrightarrow right

\wedge right

\wedge == right

\wedge : cond right

\wedge < right

$\left\{ \begin{array}{l} A \\ : \\ B \end{array} \right\}$

$\left\{ \begin{array}{l} V \\ : \\ B \end{array} \right\}$

$\left\{ \begin{array}{l} A \\ : \\ B \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} C \\ : \\ D \end{array} \right\}$

$\left\{ \begin{array}{l} A \\ : \\ B \end{array} \right\} \Rightarrow C$

$\left\{ \begin{array}{l} A \\ : \\ B \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} C \\ : \\ D \end{array} \right\}$

\forall exp

\exists exp

\forall exp

\exists exp

\wedge cond

\wedge cond

let exp

let exp

GLOBALS
New

\forall \wedge cond \Rightarrow let

Memory consumed by Mathematica: 122.7 MB

Proving

The screenshot displays the Mathematica software interface. On the left, the 'Basic Math Assistant' palette is open, showing various mathematical function categories: Mathematical Constants, Numeric Functions, Elementary Functions, Trigonometric Functions, Integer Functions, and Random Functions. Below these is a 'Typesetting' section with a grid of symbols and operators. The main notebook window, titled 'demo.nb', shows a document with the text 'Theorem proving' and 'Theorema system, using'. Below this, there is a logical expression $\Rightarrow (\neg P)$ and a paragraph starting with 'Proposition in Theorema using the templates in the'. The right sidebar contains a 'compose' panel with buttons for 'PREPARE', 'PROVE', 'COMPUTE', 'SOLVE', and 'INFORM', along with a 'FORMULA' panel showing logical symbols and a 'Memory consumed by M' indicator at the bottom.

Proving

The screenshot shows the Theorema 2.0 web interface. The main window displays a document titled "Simple proving" with the following text:

Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:

1.

$$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

We write the corresponding Proposition in Theorema using the templates in the Commander.

Below this text is a "PROPOSITION (...)" input field. The formula $((P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$ has been entered into the field. To the right of the field is a button labeled "...".

On the right side of the interface is the "Theorema Commander" panel. It contains several sections: "PREPARE" (compose, inspect, archives), "PROVE" (New, Open), "COMPUTE" (Definition, Theorem, Lemma, Proposition, Corollary, Algorithm, Example, ?), "SOLVE", and "INFORM" (New, Group / Ungroup, Quote / Unquote). Below these are "FORMULAS" and "Logic" sections with various logical symbols and operators.

A context menu is open over the "PROPOSITION (...)" field, showing options such as "Cut", "Copy", "Paste", "Evaluate Cell", "Add/Remove Cell Tags...", "Style", "Background Color", "Size", "Clear Formatting", and "Save Selection As...".

Proving

The screenshot shows the Theorema 2.0 interface. The main window displays a notebook titled "demo.nb" with the following content:

Simple proving

Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:

1.

$$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

We write the corresponding Proposition in Theorema using the templates in the Commander.

PROPOSITION (P1)

$$((P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$$

At the bottom of the notebook, there is a button labeled "Evaluate the cell" with an arrow pointing to it.

The right-hand side of the image shows the Theorema Commander palette, which contains various templates for defining propositions and formulas. The palette is organized into sections: PREPARE, PROVE, COMPUTE, SOLVE, FORMULAS, and GLOBALS. The FORMULAS section is currently selected, showing various logical operators and quantifiers.

Proving

The screenshot displays the Theorema 2.0 user interface. The main window, titled 'demo.nb', shows a document with the heading 'Simple proving'. Below the heading, it states: 'Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:'. A numbered list starts with '1.' followed by a highlighted formula: $(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$. The text continues: 'We write the corresponding Proposition in Theorema using the templates in the Commander.' Below this, a 'PROPOSITION (P1)' box contains the same formula. In the input area, 'In[2]:=' is followed by the formula $((P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$ and a small '(p1)' label. To the right, the 'Theorema Commander' window is open, showing a menu with 'goal', 'knowledge', 'built-in', 'prover', 'submit', and 'inspect'. Below the menu are buttons for 'PREPARE', 'PROVE', 'COMPUTE', 'SOLVE', and 'INFORM'. The 'COMPUTE' button is highlighted, and the text '[P1] $P \Rightarrow Q \Rightarrow ((\neg Q) \Rightarrow (\neg P))$ ' is visible in the input field. At the bottom right of the Commander window, it says 'Memory consumed by Mathematica: 121.9 MB'.

Options to choose from

The screenshot displays the Theorema 2.0 interface. The main window, titled 'demo.nb', shows a 'Simple proving' task. It asks to generate automatic proofs for the formulae: $(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$. The user has entered this formula in the 'PROPOSITION (P1)' field. The 'Theorema Commander' panel on the right is open, showing the 'PROOF RULES' section. An arrow points to the 'Basic Theorema Language Rules' button with the text 'Choose a prover'. Below this, the 'PROOF RULES SETUP' section shows a list of rules with checkboxes: 'Basic Theorema Language Rules' (unchecked), 'Rules for Proof Termination' (checked), 'Quantifier Rules' (unchecked), 'Rules for Logical Connectives' (unchecked), 'Rules for Equality' (unchecked), 'Rules based on Rewriting' (checked), 'Special Arithmetic' (unchecked), and 'Prove by contradiction' (unchecked). The 'PROOF STRATEGY' section shows 'Apply once + Level saturation' selected. The 'PROOF SEARCH LIMITS' section shows 'Search Depth' set to 30 and 'Search Time' set to 360. The bottom status bar indicates 'Memory consumed by Mathematics: 122. MB'.

Prove submit

The screenshot shows the Theorema 2.0 interface. The main window displays a proof goal and the Theorema Commander panel on the right.

Main Window:

Simple proving

Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:

- $$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

We write the corresponding Proposition in Theorema using the templates in the Commander.

PROPOSITION (P1)

In[2]:=
$$((P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$$
 (p1) X

Theorema Commander:

- goal knowledge built-in prover submit inspect
- PREPARE: (arrow points to this field)
- PROVE: SELECTED PROOF GOAL
- COMPUTE: (p1)
- SOLVE:
$$(P \Rightarrow Q \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$$
- SELECTED KNOWLEDGE BASE: No knowledge selected
- SELECTED BUILT-INS:
- INFORM:
 - Sets
 - Tuples
 - Arithmetic
 - Equal
 - Plus, Minus, Times, MultiInverse, Power, Radical, Factorial, Less, LessEqual, Greater, GreaterEqual, AbsValue, SumOf, ProductOf
 - Logic
 - Not, And, Or, Implies, If, Nand, Nor, Xor, Xnor, Equal
 - Forall, Exists, Let, Componentwise, Such, SuchUnique
 - Domains
 - Programming
 - SELECTED RULES:
 - (1) Knowledge base contains contradicting formulae
 - (1) The proof goal is True
 - (1) Knowledge base contains a formula False
 - (1) Knowledge base contains the proof goal
 - (2) Maximum rules for tuples
 - (2) Inequality rules
 - (2) Knowledge base contains two contradicting universally
 - (2) Knowledge base contains a universally quantified form

Memory consumed by Mathematica: 122.1 MB

The proof and the proof tree

The image shows three overlapping windows from the Theorema 2.0 software interface.

Left Window: Displays a logical proposition $(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$. It asks the user to "Write the corresponding Propositional Commander." Below this, it shows the proposition **PROPOSITION (P1)** and a command `In[2]:= ((P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P)))`. A section titled "Proof of (p1) #1: Show proof" is also visible.

Middle Window: Titled "Theorema 2.0", it shows the "Proof Simplification" interface. It lists alternatives for proving the proposition:

- Alternative 1: In order to prove (p1) we assume $P \Rightarrow Q$ and then prove $(\neg Q) \Rightarrow (\neg P)$.
- Alternative 2: In order to prove (Gp1) we assume $\neg Q$ and then prove $\neg P$.

Right Window: Displays a complex proof tree diagram. The tree starts with a root node (green circle) and branches out into various nodes represented by different symbols (triangles, circles, squares) in various colors (green, blue, yellow). The tree structure represents the logical derivation of the proposition. At the bottom, it says "Abort proof" and "Memory consumed by Mathematica: 126.4 MB".

Simplify the proof

Theorema Proof 125% ▾

Proof Simplification

- ☒ Eliminate failing/pending branches
- ☐ Eliminate superfluous steps
- ☐ Eliminate unused formulae

Display with new settings

We prove:

$$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

(p1)

with no assumptions.

We have several alternatives to continue the proof.

☒ Alternative 1:

In order to prove (p1) we assume

$$P \Rightarrow Q$$

(A#0)

and then prove

$$(\neg Q) \Rightarrow (\neg P).$$

(G#1)

We have several alternatives to continue the proof.

☒ Alternative 1:

In order to prove (G#1) we assume

Theorema Commander 100% ▾

goal knowledge built-in prover submit inspect

PREPARE

PROVE
COMPUTE
SOLVE

INFORM

Abort proof

Memory consumed by Mathematics: 126.4 MB

Both simplified and full proof

demo.nb 150%

Theorema 2.0

Simple proving

Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:

- $$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

We write the corresponding Proposition in Theorema using the templates in the Commander.

PROPOSITION (P1)

In[2]:= $((P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P)))$ (p1)

✓ ☒ Proof of (p1) #1: [Show simplified proof](#) [Show full proof](#)

knowledge built-in prover Restore settings

Theorema Commander 100%

compose inspect archives

PREPARE
New Open

PROVE

COMPUTE

SOLVE

ENVIRONMENTS

Definition Theorem Lemma Proposition
Corollary Algorithm Example ?

FORMULAS

INFORM

New Group / Ungroup Quote / Unquote

Logic Arithmetic Sets Tuples Operators Bracketted

\wedge right \vee right \neg term \Rightarrow right
 \Leftrightarrow right \Leftarrow right
 \Leftarrow == right \Leftarrow right

\wedge \vee \Leftrightarrow \Leftarrow

\forall \exists \forall \exists

let

GLOBAL

New

\wedge \vee \Leftarrow \Rightarrow \Leftarrow

Memory consumed by Mathematica: 126.3 MB

The simplified proof

Theorema Proof

125%

Proof Simplification ☆ Time spent for simplifying the proof: 0.000987s

We prove:

$$(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$$

(p1)

with no assumptions.

In order to prove (p1) we assume

$$P \Rightarrow Q$$

(A#0)

and then prove

$$(\neg Q) \Rightarrow (\neg P)$$

(G#1)

In order to prove (G#1) we assume

$$\neg Q$$

(A#3)

and then prove

$$\neg P$$

(G#4)

We augment the knowledge base:

From (A#3), using (A#0), we can deduce

$$\neg P$$

(A#10)

The goal (G#4) is identical to formula (A#10) in the knowledge base. Thus, this part of the proof is finished.

Theorema Commander

100%

goal knowledge built-in prover submit inspect

PREPARE

PROVE

COMPUTE

SOLVE

INFORM

Abort proof

Memory consumed by Mathematica: 126.3 MB

Exercises (proving)

Exercises. Consider the following formulae:

1. $(P \implies Q) \implies (Q \implies P)$
2. $P \vee (P \implies Q)$
3. $((P \implies Q) \wedge (Q \implies R)) \implies ((P \wedge Q) \implies R)$
4. $((Q \implies P) \wedge (Q \implies R)) \implies ((P \vee Q) \implies R)$

For each of these formulae:

- (a) following the examples shown, generate both the full proof and the simplified proofs in the Theorema system;
- (b) generate in Theorema different proofs by choosing different provers, different inference rules to be applied, change the search depth, change the search time;

Theory exploration

Exploration of LISTS theory in Theorema.nb

Lists Theory Exploration

$\langle \rangle$ is the empty list, $a \sim U$ (a is the first element in the list, U is the tail)

FIRST ELEMENT

DEFINITION (FIRST ELEMENT)

In[181]:=
$$\left(\forall_{a,U} \text{FirstEl}[a \sim U] := a \right)$$
 (first_elem) X

The TAIL of a list

DEFINITION (TAIL OF A LIST)

In[182]:=
$$\left(\forall_{a,U} \text{TailOf}[a \sim U] := U \right)$$
 (tail_of) X

Theorema Commander

compose inspect archives

PREPARE

PROVE

COMPUTE

SOLVE

INFORM

NOTEBOOKS

New Open

ENVIRONMENTS

Definition Theorem Lemma Proposition

Corollary Algorithm Example ?

FORMULAS

New Group / Ungroup Quote / Unquote

Logic Arithmetic Sets Tuples Operations Bracketted

left \wedge right left \vee right \neg form left \Rightarrow right

left \Leftarrow right left $==$ right left \neq right left \neq right

$\bigwedge \begin{Bmatrix} E_1 \\ \vdots \\ E_n \end{Bmatrix}$ $\bigvee \begin{Bmatrix} E_1 \\ \vdots \\ E_n \end{Bmatrix}$ $E_1 \Leftrightarrow E_2$ $E_1 \Leftrightarrow E_2$

$\forall_{ra} \text{expr}$ $\exists_{ra} \text{expr}$ $\forall_{ra} \text{cond}$ $\exists_{ra} \text{cond}$

let expr $a = \text{expr}$

GLOBALS

New

\forall_{ra} \forall_{ra} $\text{cond} \Rightarrow$ let expr

Compute with definitions

The screenshot shows the Theorema 2.0 interface. The main window displays the definition of the **INSERT** function. The title bar indicates the file is "Exploration of LISTS theory in Theorema.nb" and the zoom level is 125%. The interface includes a menu bar with options like "new", "knowledge", "built-in", and "setup". A sidebar on the right contains buttons for "PREPARE", "PROVE", "COMPUTE", "SOLVE", and "INFORM", along with a "New" button. The main content area shows the following:

INSERT

This function inserts an element in a sorted list such that the result remains sorted.

DEFINITION (INSERT)

In[219]:= $\forall_a \text{Insert}[a, \langle \rangle] := a \sim \langle \rangle$ (insert -1) ✕

In[220]:= $\forall_{a,b,U} \text{Insert}[a, b \sim U] := a \sim (b \sim U)$ (insert -2) ✕

In[221]:= $\forall_{a,b,U} \text{Insert}[a, b \sim U] := b \sim \text{Insert}[a, U]$ (insert -3) ✕

The bottom of the window shows a command input area with a blue cursor.

Select KB

The screenshot displays the Theorema 2.0 environment. The main window shows the definition of the **INSERT** function, which inserts an element into a sorted list while maintaining its order. The definition is given as a series of rewrite rules:

- DEFINITION (INSERT)**
- in[262] = $\forall a \text{ Insert}[a, \langle \rangle] := a - \langle \rangle$ (insert-1)
- in[263] = $\forall a, b, u \text{ Insert}[a, b - u] := a - (b - u)$ (insert-2)
- in[264] = $\forall a, b, u \text{ Insert}[a, b - u] := b - \text{Insert}[a, u]$ (insert-3)

An example application is shown at the bottom:

```
in[ ] = Insert[7, 2 - (3 - (6 - (8 - (10 - ( ) ) ) ) ) ]
```

The right sidebar, titled "Theorema Commander", contains a list of knowledge items:

- PREPARE**
 - ☐ FOUNDATION (F13)
- PROVE**
- COMPUTE**
- SOLVE**
 - ☐ **Property P12**
 - ☐ PROPOSITION (P12)
 - ☐ **Length of a list**
 - ☐ DEFINITION (LENGTH OF A LIST)
 - ☐ **Searching an element in a list**
 - ☐ DEFINITION (SEARCH ELEM IN LIST)
- INFORM**
 - ☐ $L^?$
 - ☐ ALGORITHM (MAXA)
 - ☐ ALGORITHM (MAXIMUM OF A LIST)
 - ☐ $L^?$
 - ☐ ALGORITHM (TRIMMA)
 - ☐ ALGORITHM (TRIMM)
 - ☐ $L^?$
 - ☐ ALGORITHM (MAXSORT)
 - ☐ 19
 - ☐ 20
 - ☐ **INSERT SORT**
 - ☒ **INSERT**
 - ☒ DEFINITION (INSERT)
 - ☒ insert-1
 - ☒ insert-2
 - ☒ insert-3
 - ☐ **INSERT-SORT**
 - ☐ ALGORITHM (INSERT-SORT)

At the bottom of the sidebar, there is a button labeled "OK, next ...".

Compute Insert

Exploration of LISTS theory in Theorema.nb 150% Theorema 2.0

new knowledge built-in setup

PREPARE TRACE COMPUTATION
PROVE ☒ Trace user-definitions
COMPUTE DEMO MODE
SOLVE ☐ Restore settings before each computation
INFORM

INSERT

This function inserts an element in a sorted list such that the result remains sorted.

DEFINITION (INSERT)

In[262]:= $\forall a \text{ Insert}[a, \langle \rangle] := a \sim \langle \rangle$ (insert - 1) X

In[263]:= $\forall a, b, U \text{ as } b \text{ Insert}[a, b \sim U] := a \sim (b \sim U)$ (insert - 2) X

In[264]:= $\forall a, b, U \text{ as } b < a \text{ Insert}[a, b \sim U] := b \sim \text{Insert}[a, U]$ (insert - 3) X

In[269]:= $\text{Insert}[7, 2 \sim (3 \sim (6 \sim (8 \sim (10 \sim \langle \rangle))))]$
Out[269]:= $2 \sim (3 \sim (6 \sim (7 \sim (8 \sim (10 \sim \langle \rangle)))))$

Show computation X

knowledge built-in Computation Restore settings

Show computation steps

The screenshot displays the **Theorema Computation** window, which is used for algorithm synthesis and verification. The interface is divided into several panels:

- Left Panel (Definition):** Contains the definition of the **INSERT** function. It shows the function signature `Insert[a, < >]` and its implementation using nested `if` statements and `insert` operations. The definition is shown in a box labeled **DEFINITION (INSERT)**.
- Top Panel (Computation):** Shows the step-by-step execution of the `Insert[7, 2 ~ (3 ~ (6 ~ (8 ~ (10 ~ {}))))]` function. The steps are:
 - Initial call: `Insert[7, 2 ~ (3 ~ (6 ~ (8 ~ (10 ~ {}))))]`
 - Step 1: `2 ≤ 3` is `False`. The function calls `Insert[7, 2 ~ (3 ~ (6 ~ (8 ~ (10 ~ {}))))]`.
 - Step 2: `2 < 7` is `True`. The function calls `insert-3` to insert 2 into the list `3 ~ (6 ~ (8 ~ (10 ~ {})))`.
 - Step 3: `3 ≤ 3` is `False`. The function calls `Insert[7, 3 ~ (6 ~ (8 ~ (10 ~ {})))]`.
 - Step 4: `3 < 7` is `True`. The function calls `insert-3` to insert 3 into the list `6 ~ (8 ~ (10 ~ {}))`.
 - Step 5: `6 ≤ 6` is `False`. The function calls `Insert[7, 6 ~ (8 ~ (10 ~ {}))]`.
 - Step 6: `6 < 7` is `True`. The function calls `insert-3` to insert 6 into the list `8 ~ (10 ~ {})`.
 - Step 7: `7 ≤ 8` is `True`. The function calls `insert-2` to insert 7 into the list `10 ~ {}`.
 - Final result: `2 ~ (3 ~ (6 ~ (7 ~ (8 ~ (10 ~ {})))))`
- Right Panel (Controls):** Contains buttons for **PREPARE**, **PROVE**, **COMPUTE**, **SOLVE**, and **INFORM**. It also has a **TRACE COMPUTATION** section with a checkbox for **Trace user-definitions** (checked) and a **DEMO MODE** section with a checkbox for **Restore settings before each com** (unchecked).

Compute with predicate definitions

Exploration of LISTS theory in Theorema.nb 125%

Theorema 2.0

Here, because we define a predicate, one has two definitions : one for Compute and one for Prove.

Definition for Compute

DEFINITION (OCCURS IN- FOR COMPUTE)

In[230]:= $\forall a \quad a < \langle \rangle := \text{False}$ (occurs_in_1) X

In[231]:= $\forall a, X \quad a < (a \sim X) := \text{True}$ (occurs_in_2) X

In[232]:= $\forall a, b, X \quad a < (b \sim X) \Leftrightarrow a < X$ (occurs_in_3) X

In[270]:= $a < (a \sim \langle \rangle)$
Out[270]= True

In[271]:= $a < \langle \rangle$
Out[271]= False

In[272]:= $1 < (1 \sim (2 \sim \langle \rangle))$
Out[272]= True

In[273]:= $1 < (2 \sim (2 \sim (2 \sim (2 \sim (2 \sim (2 \sim (2 \sim \langle \rangle)))))))$
Out[273]= False

In[274]:= $1 < (2 \sim (2 \sim (2 \sim (2 \sim (2 \sim (1 \sim \langle \rangle)))))))$
Out[274]= True

Theorema Commander

new knowledge built-in setup

PREPARE TRACE COMPUTATION

PROVE ☐ Trace user-definitions

COMPUTE DEMO MODE

SOLVE ☐ Restore settings before each computation

INFORM

Compute with predicate definitions

Exploration of LISTS theory in Theorema.nb 125%

Theorema 2.0

Here, because we define a predicate, one has two definitions : one for Compute and one for Prove.

Definition for Compute

DEFINITION (OCCURS IN- FOR COMPUTE)

In[230]:= $\forall_a \ a \triangleleft \langle \rangle := \text{False}$ (occurs_in_1) X

In[231]:= $\forall_{a,X} \ a \triangleleft (a - X) := \text{True}$ (occurs_in_2) X

In[232]:= $\forall_{a,b,X} \ a \triangleleft (b - X) \Leftrightarrow a \triangleleft X$ (occurs_in_3) X

Definition used in proving

DEFINITION (OCCURS IN)

In[233]:= $\forall_a \ \neg a \triangleleft \langle \rangle$ (occ1) X

In[234]:= $\forall_{a,X} \ a \triangleleft (a - X)$ (occ2) X

In[235]:= $\forall_{a,b,X} \ a \triangleleft (b - X) \Leftrightarrow a \triangleleft X$ (occ3) X

Theorema Commander

new knowledge built-in setup

PREPARE TRACE COMPUTATION

PROVE ☐ Trace user-definitions

COMPUTE DEMO MODE

SOLVE ☐ Restore settings before each computation

INFORM

References

- [1] Isabela Drămnesc, Erika Ábrahám, Tudor Jebelean, Gábor Kusper, and Sorin Stratulat. **Experiments with Automated Reasoning in the Class**. In Proc. of the 15th International Conference on Intelligent Computer Mathematics (CICM'22), volume 13467 of LNCS, pages 287–304. Springer, 2022, Tbilisi, Georgia.
- [2] Isabela Drămnesc, Erika Ábrahám, Tudor Jebelean, Gábor Kusper, Sorin Stratulat, **ARC: An Educational Project on Automated Reasoning in the Class** In Proceedings of EdMedia+ Innovate Learning 2022, pages 934–943, 2022, New York City, USA, AACE.
- [3] Isabela Drămnesc, Erika Ábrahám, Tudor Jebelean, Gábor Kusper, Sorin Stratulat, **Automated Reasoning in the Class**, Journal of Computer Algebra Rundbrief, Nr. 71, pages 21–26, 2022.
- [4] Isabela Drămnesc, Tudor Jebelean, Erika Ábrahám, Sorin Stratulat, Gábor Kusper, Mircea Marin, Adrian Crăciun, Csaba Bíró, Gergely Kovásznai, Nikolaj Popov, **Computational Logic: A Practical Approach**. Editura Universitatii de Vest, 2022.

*Acknowledgements: This work was co-funded by the European Union, Erasmus+ project **ARC: Automated Reasoning in the Class**, 2019-1-RO01-KA203-063943, 2019 - 2022,*

www.arc.info.uvt.ro

AlCons: A Prover for Deductive Synthesis of Sorting Algorithms in *Theorema*

Outline

- Problem and approach
- AlCons implementation overview
- Demo in *Theorema*
 - Algorithm generation on binary trees
 - Compute with the extracted algorithm

Algorithm synthesis

Starting from a **specification** of a problem

- find an **algorithm** which satisfies the specification including **auxiliary algorithms** (subroutines)

Synthesis by proving:

- specification \longrightarrow conjecture
- **prove**
- proof \longrightarrow algorithm

Main goal:

- Design **methods**, **inference rules** and **strategies** for **constructing proofs** (efficient, natural style) and for **synthesizing auxiliary functions**.

Motivation

Investigate the effect of different proof techniques
→ synthesize different algorithms.

Study the structure of natural style proving.

Explore the appropriate theories (by adding properties and functions to the knowledge base during proof attempts).

Find efficient proof strategies and inference rules appropriate to the domains of **lists**, **binary trees**, and **multisets**.

Use multisets to express naturally the fact that two lists/trees have the same elements and to guide the synthesis proof.

Problem and Method

Problem: Given a specification, find a correct algorithm.

Specification: Input condition (I), output condition (O).

Synthesis conjectures:

- Unary functions: $\forall_X (I[X] \implies \exists_Y O[X, Y])$.
- Binary functions: $\forall_X \forall_Y (I[X, Y] \implies \exists_Z O[X, Y, Z])$.

Proofs:

- Automatically generated by **AICons**:
 - success: **proof** \longrightarrow **algorithm**
 - failure: “**cascading**” (generates further synthesis conjectures for synthesis of auxiliary functions)

Algorithm extraction:

- One or more algorithms from one proof.

Case studies: Sorting of lists and **binary trees**.

Types

Elements: a, b, c

total order: $a < b, a \leq b$

Lists: U, V, W, X, Y

inductive domain: $\langle \rangle, a \smile U$

extended order: $a < U, a \leq U, U < a, U \leq a, U < V, U \leq V$

Binary trees: L, R, S, T, X, Y, Z

inductive domain: $\varepsilon, \langle L, a, R \rangle$

extended order: $a < L, a \leq L, L < a, L \leq a, L < R, L \leq R$

Multisets:

$$\begin{array}{l|l} \mathcal{M}[\langle \rangle] = \emptyset & \mathcal{M}[\varepsilon] = \emptyset \\ \mathcal{M}[a \smile V] = \{\{a\}\} \uplus \mathcal{M}[V] & \mathcal{M}[\langle L, a, R \rangle] = \mathcal{M}[L] \uplus \{\{a\}\} \uplus \mathcal{M}[R] \end{array}$$

AICons: Implementation Overview

The prover: a collection of rewrite rules corresponding to inferences (called *proof steps*).

- a proof situation (assm, goal) $\xrightarrow{\text{rule}}$ new proof situation
- Alternatives: an *AND-OR proof tree* is created,
- In contrast to the usual behavior of *Theorema* provers, **AICons** follows *all alternatives* \longrightarrow may lead to different algorithms.
- Some of the alternatives may also fail – the *termination* is ensured by the *Theorema* mechanism for controlling the depth of the proof tree.

Implementation Overview

AICons is a *first order* prover based on **the methods**:

- *Classical* propositional and first order inferences similar to the ones from sequent calculus.
- *Cover set induction* applied both to universal and to existential goals.
- *Domain specific* methods for the types handled by the prover (multisets, lists, and binary trees).

These methods are implemented as:

- *Inference rules*: describe how to change the proof status in one proof step.
- *Strategies*: describe how to combine several inference rules.

Cover Set: Set of terms covering the domain.

Each element of the domain instantiates exactly one cover term.

For binary trees: $\{\varepsilon, \langle L, a, R \rangle\}$ (L, a, R : variables).

Target goal: $\forall_{XY} \exists P[X, Y] \xrightarrow{\text{Skolem}} \exists_Y P[X_0, Y] \xrightarrow{\text{metavar}} P[X_0, Y^*]$

X_0 : target constant, Y^* : target metavariable

On Skolem constants: Prove $P[\varepsilon, Y^*]$ and $P[\langle L_0, a_0, R_0 \rangle, Y^*]$

- may use $P[L_0, F[L_0]]$ and $P[L_0, F[R_0]]$ (F : synth. funct.)
- determines the decomposition of the input

On metavariables:

Prove $P[X_0, \varepsilon]$ and $P[X_0, \langle L^*, a^*, R^* \rangle]$

- may replace L^* by $F[L_1^*]$ and R^* by $F[R_1^*]$
- determines the structure of the output

In a nested way on the new Skolem constants and metavariables

→ *Algorithms with nested recursion, and with recursion on*

several arguments

Dynamic induction

Dynamically generates new induction hypotheses during the proof.

Noetherian induction based on the well-founded ordering:

$$L \prec R \text{ is } \mathcal{M}[L] \subset \mathcal{M}[R]$$

Checked *syntactically* at meta-level: meta-relation between terms induced by the strict inclusion of the multisets of symbols

- Example: $\langle L_0, a_0, R^* \rangle \prec \langle L_0, a_0, \langle R^*, b_0, S^* \rangle \rangle$

Usage:

- ground term $t \prec X_0$ (target constant):
 add $P[t, \text{Sort}[t]]$ to assumptions.
- metavariable $L^* \prec Y^*$ (target metavariable):
 replace in goal L^* by $F[L^*]$ (target function applied to a new metavariable)

Strategy: Cascading (conjecture generation)

Synthesizing auxiliary functions:

- generate conjecture
- prove conjecture \longrightarrow algorithm
- use auxiliary function in proof

Conjecture generation:

- Skolem constants from goal \longrightarrow universal x, x', \dots
- metavariables from goal \longrightarrow existential y, y', \dots
- conjecture:

$$\forall_{xx'} \dots (P[x, x', \dots] \implies \exists_{yy'} \dots Q[x, x', \dots, y, y', \dots])$$

- $P[x, x', \dots]$: from the assumptions containing only the Skolem constants present in goal
- $Q[x, x', \dots, y, y', \dots]$: from the goal

Strategy: Cascading (function usage)

Using the generated functions $F[x, x', \dots]$, $F'[x, x', \dots]$:

- new assumption:

$$\forall_{x, x'} \dots (P[x, x', \dots] \implies Q[x, x', \dots, F[x, x', \dots], F'[x, x', \dots], \dots])$$

- use the new function symbols in the goal
- contributes to *automatic* theory exploration

Group multisets

The goal contains the equality: $\mathcal{M}[Y^*] = \mathcal{M}[t_1] \uplus \mathcal{M}[t_2] \uplus \dots$

Proof flow: transform the union into $\mathcal{M}[t]$ (then $Y^* = t$)

Stepwise: groups pairs $\mathcal{M}[t_1] \uplus \mathcal{M}[t_2]$

(different groupings \longrightarrow proof alternatives)

For each pair, find the function F such that:

$$\mathcal{M}[F[t_1, t_2]] = \mathcal{M}[t_1] \uplus \mathcal{M}[t_2]$$

Possibilities:

- F is already known, the proof works by predicate logic;
- induction can be applied (if F is the target function)
- cascading is necessary for the synthesis of F

Example: Group multisets and cascading

The goal is: $IsSorted[Y^*] \wedge \mathcal{M}[Y^*] = \mathcal{M}[a_0] \uplus \mathcal{M}[R_0] \uplus \dots$

The assumptions contain: $IsSorted[R_0]$

Group pair: $\mathcal{M}[a_0] \uplus \mathcal{M}[R_0]$

Cascading: synthesize *Insert* from the conjecture:

$$\forall \forall \exists (IsSorted[R] \implies (IsSorted[Y] \wedge \mathcal{M}[Y] = \mathcal{M}[a] \uplus \mathcal{M}[R]))$$

$R \ a \ Y$

Add new assumption:

$$IsSorted[R] \implies (IsSorted[Insert[a, R]] \wedge \mathcal{M}[Insert[a, R]] = \mathcal{M}[a] \uplus \mathcal{M}[R])$$

Change goal to: $IsSorted[Y^*] \wedge \mathcal{M}[Y^*] = Insert[a_0, R_0] \uplus \dots$

The extracted algorithm: set of conditional equalities

- $\{Q_k \Rightarrow (F[\mathcal{X}] = T_k)\}_{k=1}^n$ (\mathcal{X} is a pattern – cover set term)
- Q_k are formulae; T_k are terms (dependent on variables of the pattern)

Insert-Sort: (cover set on Skolem constant)

$$\forall_{a,L,R} \left(\begin{array}{l} \text{Sort}[\varepsilon] = \varepsilon \\ \text{Sort}[\langle L, a, R \rangle] = \text{Insert}[a, \text{Sort}[\text{Concat}[L, R]]] \end{array} \right)$$

Quick-Sort: (cover set on metavariable)

$$\forall_{a,L,R} \left(\begin{array}{l} \text{Sort}[\varepsilon] = \varepsilon \\ \text{Sort}[\langle L, a, R \rangle] = \\ \langle \text{SmallerEq}[a, \text{Sort}[\text{Concat}[L, R]], \\ a, \\ \text{Bigger}[a, \text{Sort}[\text{Concat}[L, R]]] \rangle \end{array} \right)$$

Proving in *Theorema*

The screenshot displays the Theorema 2.0 interface. The main workspace contains three propositions, each with a goal and a proof strategy configuration:

- PROPOSITION (INSERT-FUNCTION-SORTED)**
Goal: $\forall x \text{ IsSorted}[X] \Rightarrow \forall a (M[\text{Insert}[a, X]] = \{(a)\} \uplus M[X] \wedge \text{IsSorted}[\text{Insert}[a, X]])$
Strategy: (insert - function - sorted) x
- PROPOSITION (INSERT-FUNCTION)**
Goal: $\forall x \text{ IsSorted}[X] \Rightarrow \forall a (M[\text{Insert}[a, X]] = \{(a)\} \uplus M[X])$
Strategy: (insert - function) x
- PROPOSITION (INSERT)**

The right-hand side features the **Theorema Commander** panel, which includes tabs for goal, knowledge, built-in, prover, submit, and inspect. The **PROVE** tab is active, showing the **PROOF RULES SETUP** section. The **Filtered by:** field is empty. The **PROOF RULES** section lists various rules with checkboxes and dropdown menus for configuration:

- Initialization**: ☒ 1
- Propositional**: ☒ 1
- Quantifiers**: ☒ 5 (Skolemize the universal goal), ☒ 5 (Introduce meta-variables in the existential)
- Simplify**: ☒ 1 (Simplify goal by one rule), ☒ 1 (Simplify goal by several rules), ☒ 1 (Solve metavariable from single multiset eq), ☒ 1 (Solve metavariable from multiset equality i), ☒ 1 (Solve target function from single multiset e), ☒ 1 (Solve target function from multiset equality)
- Induction**: ☒ 1 (Use a cover set from the alternatives), ☒ 5 (Apply alternatively several cover sets to Sk), ☒ 10 (Generate alternative pairings of multiset te)
- Fail: end proof**: ☐ 100
- Options**: ☐

Generated Proof and Proof Tree in *Theorema*

The screenshot displays the **Theorema Proof** window, which is part of the Wolfram Mathematica 11.3 environment. The interface is divided into several sections:

- Top Bar:** Includes standard Mathematica menus (File, Edit, Insert, Format, Cell, Graphics, Evaluation, Palettes, Window, Help) and a **Theorema Commander** panel with buttons for **goal**, **knowledge**, **built-in**, **prover**, **submit**, and **inspect**.
- Left Panel (Proof Simplification):** Shows the step-by-step simplification of a goal.
 - Initial goal: $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge \text{IsSorted}[R0] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0) \wedge (a0 \leq R0)$. (Ge591)
 - Using "A#227.4", the goal (Ge591) is simplified to: $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge \text{IsSorted}[R0] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0)$. (Ge592)
 - Using "A#227.2", the goal (Ge592) is simplified to: $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0)$. (Ge593)
 - Using "insert-less-elem-mg", the goal (Ge593) is simplified to: $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge ((\overline{a}7 \leq a0) \wedge (L0 \leq a0))$. (Ge594)
 - Using "A#227.3", the goal (Ge594) is simplified to: $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge (\overline{a}7 \leq a0)$. (Ge595)
 - The conjunction (A#448) is split into:
 - $M[\text{Insert}[\overline{a}7, L0]] = \{ \{ (\overline{a}7) \} \} \setminus M[L0]$. (A#448.1)
 - $\text{IsSorted}[\text{Insert}[\overline{a}7, L0]]$. (A#448.2)
 - Using "A#448.2", the goal (Ge595) is simplified to: $\overline{a}7 \leq a0$. (Ge645)
 - The goal is used as a condition in the following clause of the algorithm:

$$(\overline{a}7 \leq a0) \Rightarrow (\text{Insert}[\overline{a}7, \langle L0, a0, R0 \rangle] = \text{Insert}[\overline{a}7, L0], a0, R0)$$
. (A#646)
 - The process concludes with **Success.**
- Right Panel (Proof Tree):** Displays a hierarchical proof tree. The root node is a triangle with a red flag. It branches into several nodes, each represented by a triangle with a red flag and a small icon. The tree structure shows the logical derivation from the initial goal to the final simplified goal.
- Bottom Bar:** Contains a row of navigation icons (back, forward, search, etc.) and an **Abort proof** button.

AICons Demo

- The automatically generated proof of the above conjecture, by AICons.
- Basic properties used in the proof.
- Highlighting the solutions obtained.
- The extracted algorithm from the proof.

Proof animation

Computation with the Extracted Algorithm

The extracted algorithm is:

Algorithm (Insert an element in a tree)

$$h[2] = \left(\forall_a \text{ Insert}[a, \epsilon] := \langle \epsilon, a, \epsilon \rangle \right) \quad (D10) \quad \times$$

$$h[3] = \left(\forall_{\substack{a,b,L,R \\ a \leq b}} \text{ Insert}[a, \langle L, b, R \rangle] := \langle \text{Insert}[a, L], b, R \rangle \right) \quad (D11) \quad \times$$

$$h[4] = \left(\forall_{\substack{a,b,L,R \\ a > b}} \text{ Insert}[a, \langle L, b, R \rangle] := \langle L, b, \text{Insert}[a, R] \rangle \right) \quad (D13) \quad \times$$

$$h[5] = \text{Insert}[3, \epsilon]$$

$$\text{Out}[5] = \langle \epsilon, 3, \epsilon \rangle$$

$$h[6] = \text{Insert}[3, \langle \epsilon, 5, \epsilon \rangle]$$

$$\text{Out}[6] = \langle \langle \epsilon, 3, \epsilon \rangle, 5, \epsilon \rangle$$

$$h[7] = \text{Insert}[3, \langle \langle \epsilon, 3, \epsilon \rangle, 5, \epsilon \rangle]$$

$$\text{Out}[7] = \langle \langle \langle \epsilon, 3, \epsilon \rangle, 3, \epsilon \rangle, 5, \epsilon \rangle$$

$$h[8] = \text{Insert}[10, \langle \langle \epsilon, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \epsilon, 10, \epsilon \rangle \rangle \rangle]$$

$$\text{Out}[8] = \langle \langle \epsilon, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \langle \epsilon, 10, \epsilon \rangle, 10, \epsilon \rangle \rangle \rangle$$

$$h[9] = \text{Insert}[2, \langle \langle \epsilon, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \epsilon, 10, \epsilon \rangle \rangle \rangle]$$

$$\text{Out}[9] = \langle \langle \langle \epsilon, 2, \epsilon \rangle, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \epsilon, 10, \epsilon \rangle \rangle \rangle$$

$$h[10] = \text{Insert}[22, \langle \langle \epsilon, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \epsilon, 10, \epsilon \rangle \rangle \rangle]$$

$$\text{Out}[10] = \langle \langle \epsilon, 3, \epsilon \rangle, 5, \langle \epsilon, 7, \langle \epsilon, 10, \langle \epsilon, 22, \epsilon \rangle \rangle \rangle \rangle$$

The right panel shows the **Theorema 2.0** interface with the **new** tab selected. The **Solve** button is active, and the **Algorithm (Insert an element in a tree)** is selected. The **OK, next ...** button is visible at the bottom right.

Results

- **AICons**: a powerful system for proof-based algorithm synthesis on lists and binary trees using multisets.
- The proofs are generated in a few seconds and are easy to understand.
- The most important proof strategies are: use cover sets together with multiset based Noetherian induction, pairing of multisets, and cascading.
- By using cover sets, no algorithm scheme and no concrete induction principles are needed in advance, as they are dynamically produced during the proof, and even nested induction algorithms can be generated automatically.

References

- [5] Isabela Drămnesc and Tudor Jebelean. **Synthesis of list algorithms by mechanical proving**. Journal of Symbolic Computation, 68:61–92, 2015.
- [6] Isabela Drămnesc, Tudor Jebelean, Sorin Stratulat. **Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques**. Journal of Symbolic Computation 90: 3–41, 2019.
- [7] Isabela Drămnesc and Tudor Jebelean. **Synthesis of sorting algorithms using multisets in Theorema**. Journal of Logical and Algebraic Methods in Programming, 119(100635), 2021.
- [8] Isabela Drămnesc and Tudor Jebelean. **AICons: Deductive Synthesis of Sorting Algorithms in Theorema**. In Proc. of the 18th International Colloquium on Theoretical Aspects of Computing (ICTAC'21), volume 12819 of LNCS, pages 314–333. Springer, 2021.

Verification of algorithms in *Theorema*

Outline

- Problem and approach
- Implementation overview: some special rules
- Results

Problem and Method

Problem: Given an algorithm, prove that the algorithm is correct (including the verification of **auxiliary algorithms**).

Specification: the output is sorted and preserves the multiset.
 → **two logical conjectures**

Proofs:

- Automatically generated by the *Theorema* prover:
 - success: **proof** → **the correctness of the algorithm**
 - failure: “**cascading**” (generates further conjectures for verification of auxiliary functions)
- Script in Coq

Case study

- Verification of: Bubble-Sort, Insert-Sort, Merge-Sort, Quick-Sort, Patience-Sort, Min-Sort, Max-Sort, Min-Max-Sort on lists, by using **the Theorema** and Coq systems.

Motivation

Algorithm certification or program verification have *an increasing importance* in the current technological landscape, due to the sharp increase in the complexity of software (adverse effects in case of failure)

- for instance robots constitute a particular class of systems that can present high risks of software failures.

Sorting has *a growing area of applications*,

- in particular the ones where organizing huge data collections is critical, as for instance in environmental applications.

Compare the characteristics and the performance of **Theorema** and **Coq**.

The logical conjecture in Theorema

Conjecture

$$\forall X \left(\text{IsSorted}[\text{Sort}[X]] \wedge \mathcal{M}[X] = \mathcal{M}[\text{Sort}[X]] \right)$$

- The conjecture is split in two;
- The proofs in *Theorema* are automatically generated by our prover which uses:
 - some special inference rules;
 - definitions and additional lemmas in the knowledge base.

Implementation Overview

The prover: a collection of rewrite rules corresponding to inferences.

- A proof situation (assumptions, goal) $\xrightarrow{\text{rule}}$ new proof situation
- Alternatives: an *AND-OR proof tree* is created,
- Some of the alternatives may fail – the *termination* is ensured by the *Theorema* mechanism for controlling the depth of the proof tree.

Implementation Overview

The **Theorema**-based prover uses the **the methods**:

- *Classical* propositional and first order inferences similar to the ones from sequent calculus.
- *Generalized Noetherian induction* that finds inductive hypotheses automatically.
- *Domain specific* methods for the types handled by the prover (multisets, lists).

These methods are implemented as:

- *Inference rules*: describe how to change the proof status in one proof step.
- *Strategies*: describe how to combine several inference rules.

R1. Generalized induction

- Uses the **Noetherian induction** based on the well-founded ordering between lists;
- Checked *syntactically* by a meta-relation between terms induced by the strict inclusion of the multisets of the terms ($U < V$ is determined by $\mathcal{M}[U] \subset \mathcal{M}[V]$).

When needed, this rule applies to introduce a novel induction hypothesis with a smaller list.

- Example: the main goal is $IsSorted[Insert[a_0, b_0 \smile U_0]]$, then $IsSorted[Insert[a_0, U_0]]$ is assumed, because U_0 is smaller than $b_0 \smile U_0$.

R2. Cascading

- When a goal cannot be proved, a new conjecture is generated from the current goal and by using only the assumptions that are needed (the ones which contain the Skolem constants occurring in the goal);
- The conjecture is of the form $\forall_{X,Y} A[X, Y] \implies G[X, Y]$ where
 - X, Y are the Skolem constants from the current goal,
 - A is composed from the needed assumptions, and
 - G is the current goal;
- A new proof attempt starts, and typically, the novel generated conjecture corresponds to the verification of auxiliary functions used in the sorting algorithms.

R3. Preserving multisets

This rule is based on the following principle: the formula

$$E_1[x_1, x_2, \dots, X_1, X_2, \dots] \leq E_2[y_1, y_2, \dots, Y_1, Y_2, \dots],$$

where only \cup , *Insert*, and *Merge* occur in the expressions E_1 and E_2 , can be transformed into:

$$x_1 \leq y_1, y_2, \dots, Y_1, Y_2, \dots \wedge x_2 \leq y_1, y_2, \dots, Y_1, Y_2, \dots \wedge \dots$$

$$\wedge X_1 \leq y_1, y_2, \dots, Y_1, Y_2, \dots \wedge X_2 \leq y_1, y_2, \dots, Y_1, Y_2, \dots \wedge \dots$$

(Each argument of E_1 is smaller than each argument of E_2 .)

Example: A goal (or an assumption) of the form

$$\text{Insert}[a, T] \leq \text{Merge}[U, b \cup V]$$

is transformed into

$$a \leq U \wedge a \leq b \wedge a \leq V \wedge T \leq U \wedge T \leq b \wedge T \leq V.$$

The algorithms

- *Insert-Sort*
- *Merge-Sort*
- *Bubble-Sort*
- *Quick-Sort*
- *Patience-Sort*
- *Min-Sort*
- *Max-Sort*
- *Min-Max-Sort*

Example: *Min-Max-Sort* returns the sorted version of the input list. It places the minimum of the input list at the beginning of the output and the maximum at the end of it, and then applies recursively to the list of the remaining elements, selected by the function *TrimMM*.

Algorithm

$$\left(\begin{array}{l} \forall_{a,b,U} (TrimMM[a \smile (b \smile U)] = TrimMmA[a, b, U]) \\ a \leq b \\ \forall_{a,b,U} (TrimMM[a \smile (b \smile U)] = TrimMmA[b, a, U]) \\ b < a \end{array} \right)$$

Algorithm

$$\left(\begin{array}{l} \forall_{a,b} (TrimMmA[a, b, \langle \rangle] = \langle \rangle) \\ \forall_{a,b,c,U} (TrimMmA[a, b, c \smile U] = a \smile TrimMmA[c, b, U]) \\ c < a \\ \forall_{a,b,c,U} (TrimMmA[a, b, c \smile U] = c \smile TrimMmA[a, b, U]) \\ (a \leq c \wedge c \leq b) \\ \forall_{a,b,c,U} (TrimMmA[a, b, c \smile U] = b \smile TrimMmA[a, c, U]) \\ b < c \end{array} \right)$$

Computation with the Algorithms

Theorema 2.0

ALGORITHM (MINMAXSORT)

In[83]:= $\text{MMS}[\langle \rangle] == \langle \rangle$ (Min - Max - Sort - 1) ✕

In[84]:= $\forall_a (\text{MMS}[a - \langle \rangle] == a - \langle \rangle)$ (Min - Max - Sort - 2) ✕

In[85]:= $\forall_{a,b,U} \text{MMS}[a - (b - U)] == \min[a - (b - U)] \sim (\text{MMS}[\text{TrimMM}[a - (b - U)]] \sim \max[a - (b - U)])$ (Min - Max - Sort - 3) ✕

Compute

In[90]:= $\text{MMS}[3 - \langle \rangle]$
Out[90]:= $3 - \langle \rangle$

In[91]:= $\text{MMS}[3 - (2 - (3 - (6 - (8 - (10 - \langle \rangle)))))]$
Out[91]:= $2 - (3 - (3 - (6 - (8 - (10 - \langle \rangle)))))$

In[92]:= $\text{MMS}[10 - (2 - (3 - (6 - (8 - (10 - \langle \rangle)))))]$
Out[92]:= $2 - (3 - (6 - (8 - (10 - (10 - \langle \rangle)))))$

In[93]:= $\text{MMS}[1 - (2 - (3 - (6 - (8 - (10 - \langle \rangle)))))]$
Out[93]:= $1 - 2 - (3 - (6 - (8 - (10 - \langle \rangle))))$

In[94]:= $\text{MMS}[10 - (4 - (3 - (2 - (8 - (1 - \langle \rangle)))))]$
Out[94]:= $1 - 2 - (3 - (4 - (8 - (10 - \langle \rangle))))$

Proving in *Theorema*

The screenshot shows the Wolfram Mathematica interface with the Theorema 2.0 application. The main window displays a conjecture about multisets and a proof goal. The right sidebar shows the 'Theorema Commands' panel with options like 'goal', 'knowledge', 'built-in', 'prover', 'submit', and 'inspect'. The 'PROVE' section is active, showing 'Tuple Prover' and 'PROOF RULES SETUP'. The 'SOLVE' section shows 'Restore defaults' and 'Show all'. The 'INFORM' section shows 'Initialization', 'Propositional', 'Quantifiers', and 'Simplify'.

Generated Proof and Proof Tree in *Theorema*

Thorema Proof - Wolfram Mathematica 11.3

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

Thorema Proof

Proof Simplification

Using "A#227.4", the goal (Ge591) is simplified to:

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge \text{IsSorted}[R0] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0) \wedge (a0 \leq R0).$$

(Ge591)

Using "A#227.4", the goal (Ge591) is simplified to:

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge \text{IsSorted}[R0] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0).$$

(Ge592)

Using "A#227.2", the goal (Ge592) is simplified to:

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge (\text{Insert}[\overline{a}7, L0] \leq a0).$$

(Ge593)

Using "insert-less-elem-mg", the goal (Ge593) is simplified to:

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge ((\overline{a}7 \leq a0) \wedge (L0 \leq a0)).$$

(Ge594)

Using "A#227.3", the goal (Ge594) is simplified to:

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]] \wedge (\overline{a}7 \leq a0).$$

(Ge595)

The conjunction (A#448) is split into:

$$M[\text{Insert}[\overline{a}7, L0]] = \{ \{ (\overline{a}7) \} \} \cup M[L0],$$

(A#448.1)

$$\text{IsSorted}[\text{Insert}[\overline{a}7, L0]].$$

(A#448.2)

Using "A#448.2", the goal (Ge595) is simplified to:

$$\overline{a}7 \leq a0.$$

(Ge645)

The goal is used as a condition in the following clause of the algorithm:

$$(\overline{a}7 \leq a0) \Rightarrow (\text{Insert}[\overline{a}7, \langle L0, a0, R0 \rangle] = \text{Insert}[\overline{a}7, L0], a0, R0).$$

(A#646)

Success.

Thorema Commander

goal knowledge built-in prover submit inspect

PREPARE

PROVE

COMPUTE

SOLVE

INFORM

Abort proof

Results

- A Theorema prover for proof-based algorithm synthesis and verification on lists using multisets.

This case study

- shows that, even though the algorithms are very well-known, **proving the correctness is not trivial**;
- the use of multisets is very important as it allows to express more easily the fact that two lists have the same elements;
- the techniques used lead to more efficient proofs, and simplify the proving process → in *Theorema* the proofs are generated in a few seconds and are easy to understand.

References

- [9] Isabela Drămnesc, Tudor Jebelean, and Sorin Stratulat. **Certification of Tail Recursive Bubble-Sort in Theorema and Coq**. In LPAR 2024 Complementary Volume, volume 18 of Kalpa Publications in Computing, pages 53–68. EasyChair, 2024, Port Louis, Mauritius.
- [10] Isabela Drămnesc, Tudor Jebelean, and Sorin Stratulat. **Certification of Sorting Algorithms using Theorema and Coq**. In Proc. of the 10th International Symposium on Symbolic Computation in Software Science (SCSS'24), volume 14991 of LNCS, pages 38–56. Springer, 2024, Tokyo, Japan.
- [11] Isabela Drămnesc, Tudor Jebelean, and Sorin Stratulat. **Formal Certification of Synthesized Sorting Algorithms**. In Proc. of the 13th International Conference on Logic and Applications (LAP'24), 2024, Dubrovnik, Croatia (to appear).
- [12] Isabela Drămnesc, Tudor Jebelean, and Sorin Stratulat. **Certification of Insert–Sort and Merge–Sort in Coq**. In Proc. of the 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME'24), 2024, Male, Maldives (to appear).

Theorema is a very nice automated theorem prover that can be used for both

- Teaching:
 - Automated Theorem Proving,
 - Algorithm Synthesis and Mathematical Theory Exploration.
- Research: algorithm synthesis and verification.

Ongoing and future work¹

- Verify robotic algorithms in *Theorema* (e.g. Dijkstra, A*, ...)
- Increase the automation of proving and of finding necessary lemmata in *Theorema*

¹*Acknowledgements: This work is co-funded by the European Union through the Erasmus+ project **AiRobo: Artificial Intelligence-based Robotics**, 2023-1-RO01-KA220-HED-000152418, <https://airobo.info.uvt.ro>*

Thank you for your attention!