Introduction to the *Theorema* system

Isabela Drămnesc

Computer Science Department
West University of Timișoara
Romania

FROM 2024
17 September
Timișoara, Romania

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

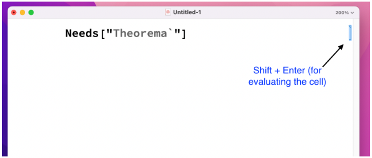
User mode

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

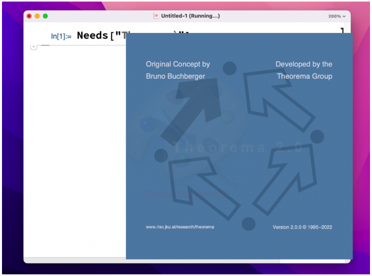
Developer mode

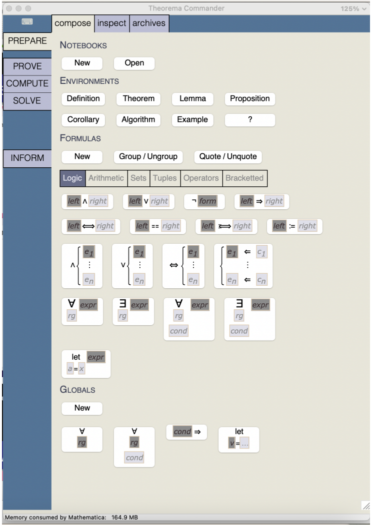
- Needed: Mathematica, Eclipse, JDK, Workbench
- Download the Theorema package and include it in Eclipse
- Installation guide **here**.

Loading *Theorema* in user mode

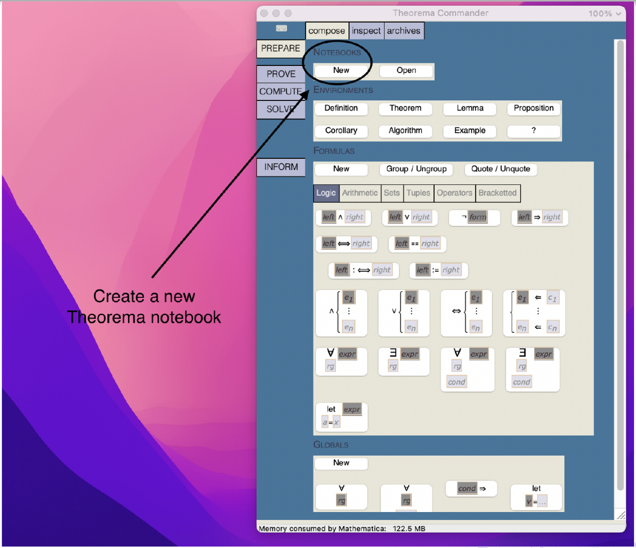


(a) Needs Theorema

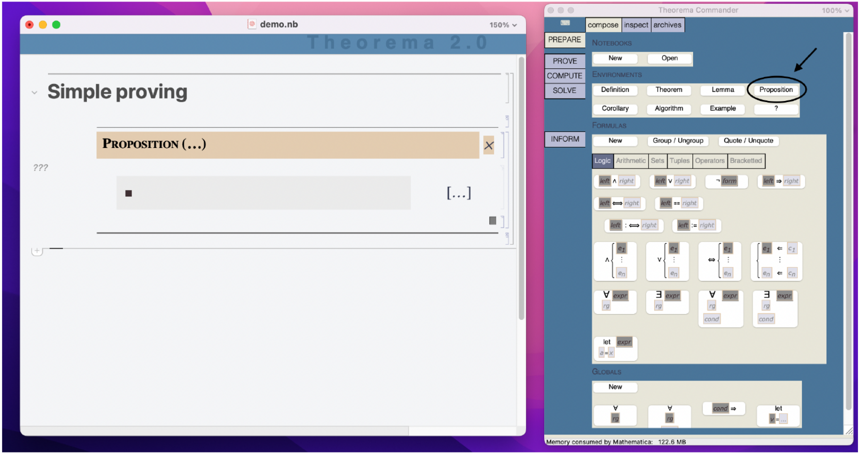




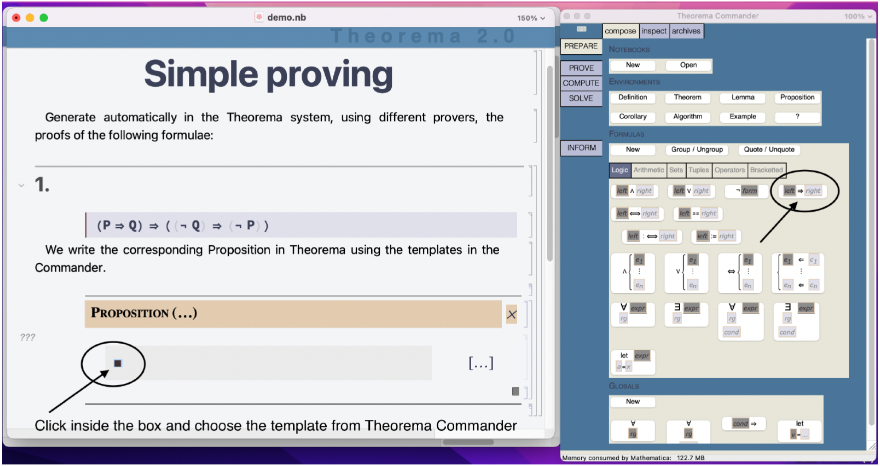
Create a new notebook



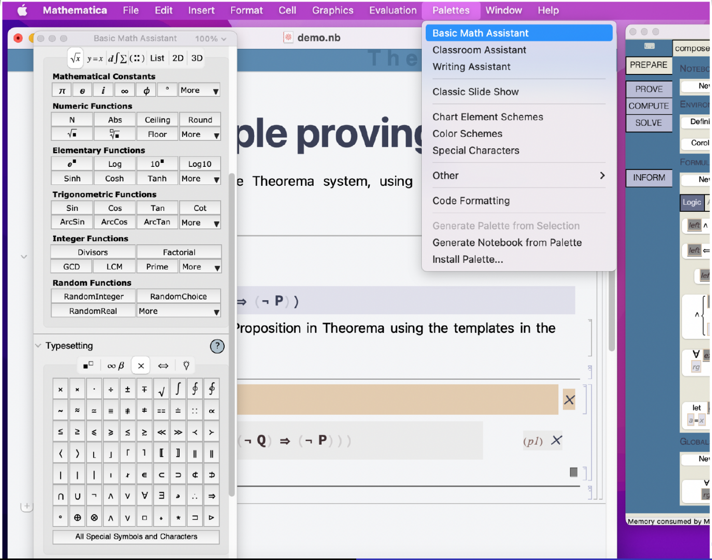
Proving


demo.nb
150%
Theorema 2.0
Simple proving
Proposition (…)
???
[…]
Theorema Commander
100%
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
Operators
Bracketted
Globals
New
Memory consumed by Mathematica: 122.6 MB

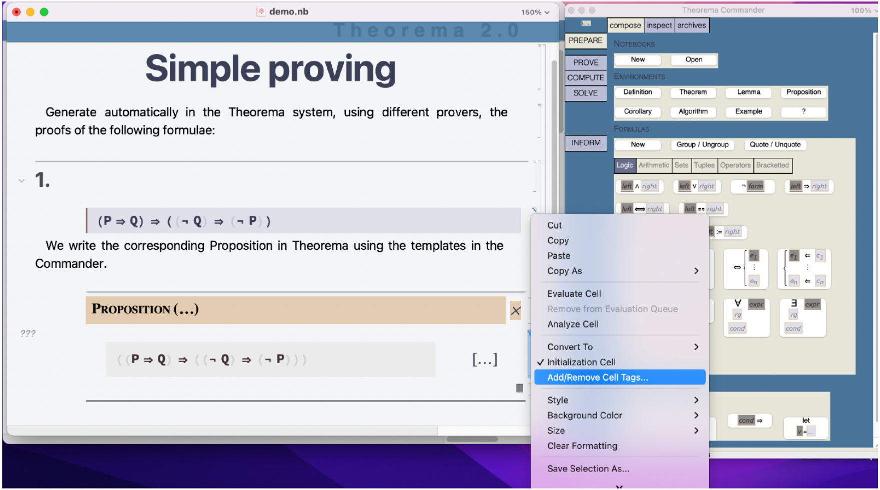
Proving


demo.nb
150%
Theorema 2.0
Simple proving
Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:
1.
(P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ 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
100%
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
Operators
Bracketted
GLOBALS
New
Memory consumed by Mathematica: 122.7 MB

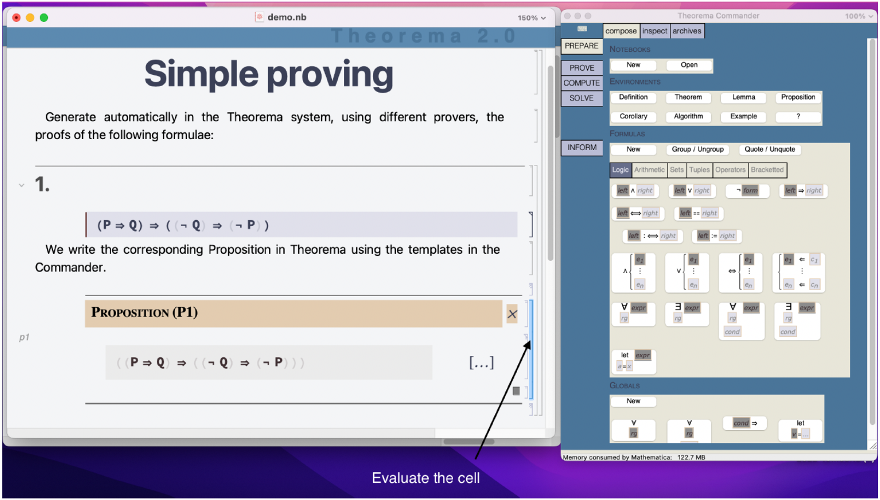
Proving


Mathematica File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
Basic Math Assistant
demo.nb
Basic Math Assistant
Classroom Assistant
Writing Assistant
Classic Slide Show
Chart Element Schemes
Color Schemes
Special Characters
Other
Code Formatting
Generate Palette from Selection
Generate Notebook from Palette
Install Palette...
Mathematical Constants
Numeric Functions
N Abs Ceiling Round
Floor More
Elementary Functions
Log Log10
Sinh Cosh Tanh More
Trigonometric Functions
Sin Cos Tan Cot
ArcSin ArcCos ArcTan More
Integer Functions
Divisors Factorial
GCD LCM Prime More
Random Functions
RandomInteger RandomChoice
RandomReal More
Typesetting
All Special Symbols and Characters
ple provin
e Theorema system, using
roposition in Theorema using the templates in the
PREPARE
PROVE
COMPUTE
SOLVE
INFORM
Logic

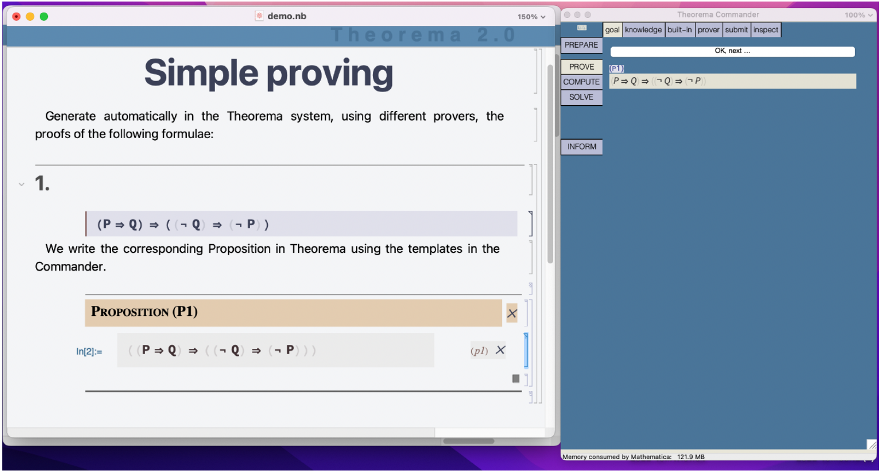
Proving


demo.nb
Theorema 2.0
Simple proving
Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:
1.
(P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P))
We write the corresponding Proposition in Theorema using the templates in the Commander.
PROPOSITION (…)
???
((P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P)))
[…]
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
Operators
Bracketted
Cut
Copy
Paste
Copy As
Evaluate Cell
Remove from Evaluation Queue
Analyze Cell
Convert To
Initialization Cell
Add/Remove Cell Tags...
Style
Background Color
Size
Clear Formatting
Save Selection As...

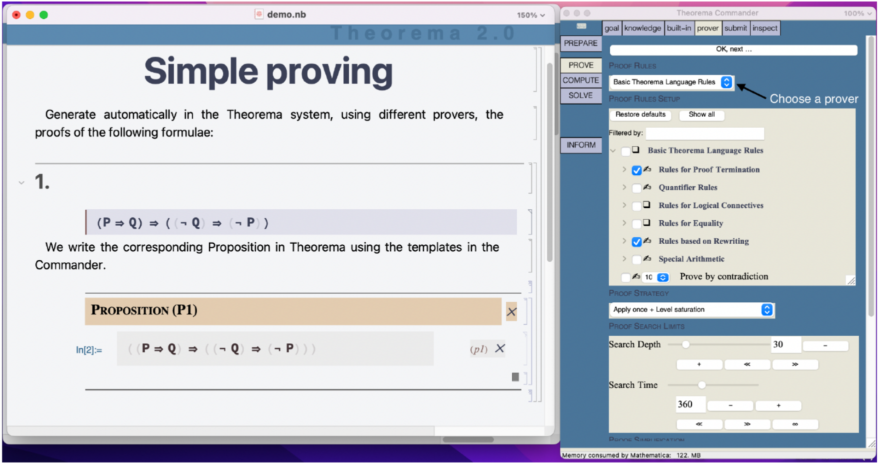
Proving


demo.nb
150%
Theorema 2.0
Simple proving
Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:
1.
(P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P))
We write the corresponding Proposition in Theorema using the templates in the Commander.
PROPOSITION (P1)
p1
((P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P)))
[...]
Theorema Commander
100%
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
Operators
Bracketted
GLOBALS
New
Memory consumed by Mathematica: 122.7 MB
Evaluate the cell

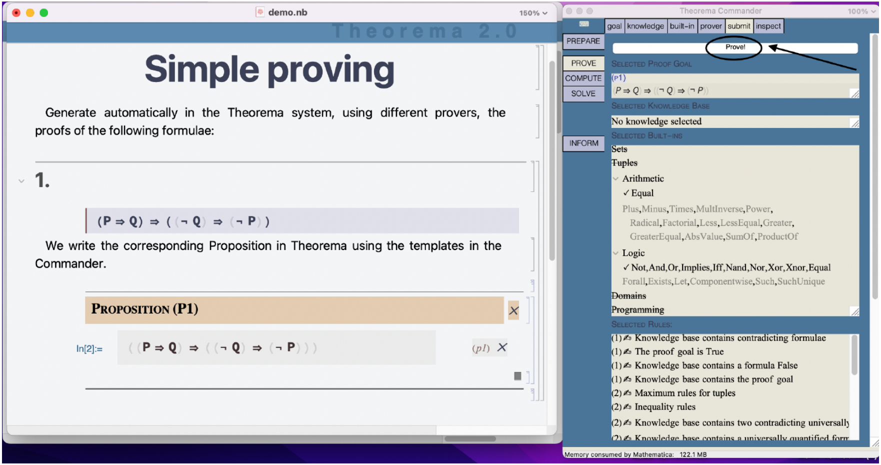
Proving


demo.nb
Theorema 2.0
Simple proving
Generate automatically in the Theorema system, using different provers, the proofs of the following formulae:
1.
(P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P))
We write the corresponding Proposition in Theorema using the templates in the Commander.
PROPOSITION (P1)
In[2]:=
((P ⇒ Q) ⇒ ((¬ Q) ⇒ (¬ P)))
(p1)
Theorema Commander
goal
knowledge
built-in
prover
submit
inspect
PREPARE
PROVE
COMPUTE
SOLVE
INFORM
OK, next ...
(P1)
P ⇒ Q ⇒ ((¬ Q) ⇒ (¬ P))
Memory consumed by Mathematica: 121.9 MB

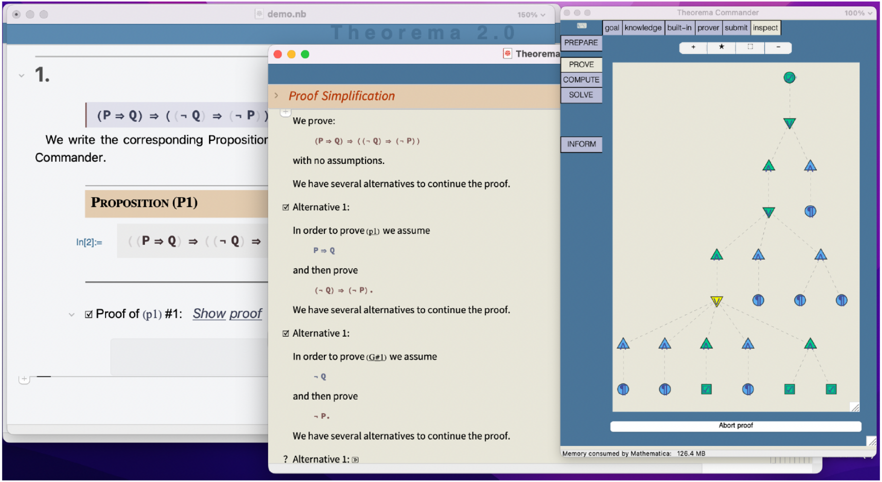
Options to choose from



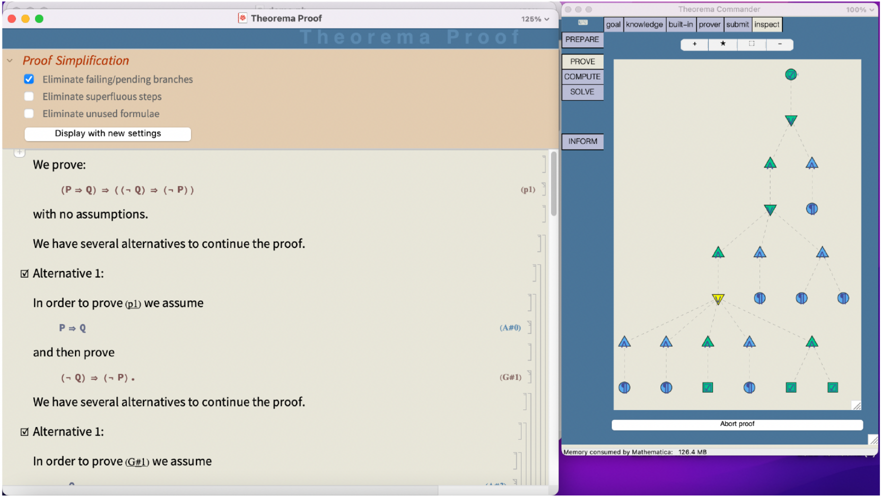
Prove submit



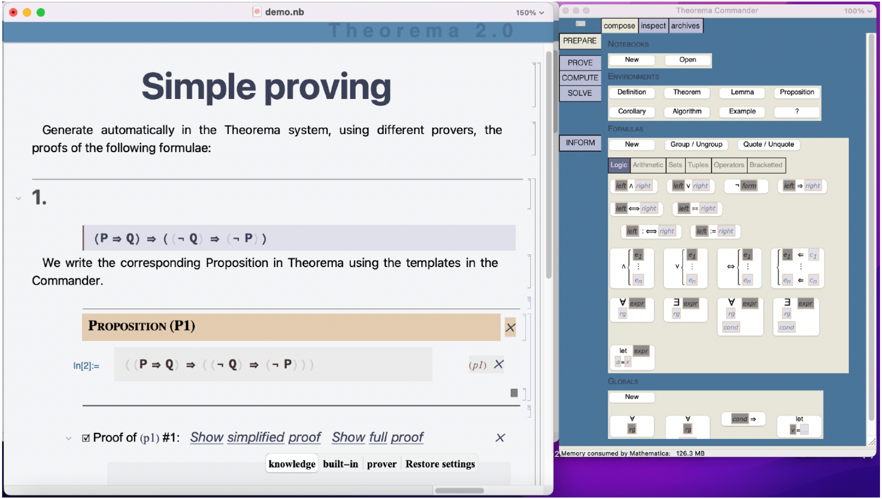
The proof and the proof tree



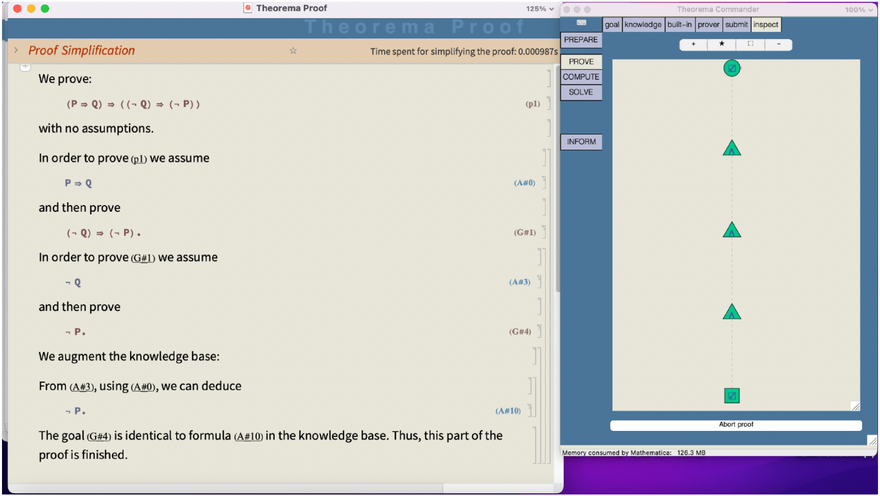
Simplify the proof



Both simplified and full proof



The simplified proof



Exercises (proving)

Exercises. Consider the following formulae:

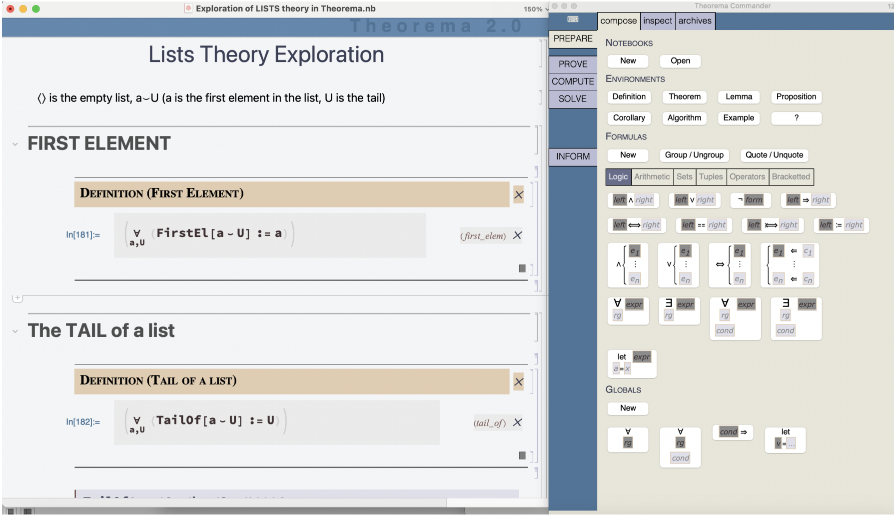
1. $(P \Longrightarrow Q) \Longrightarrow (Q \Longrightarrow P)$
2. $P \vee (P \Longrightarrow Q)$
3. $((P \Longrightarrow Q) \wedge (Q \Longrightarrow R)) \Longrightarrow ((P \wedge Q) \Longrightarrow R)$
4. $((Q \Longrightarrow P) \wedge (Q \Longrightarrow R)) \Longrightarrow ((P \vee Q) \Longrightarrow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

For each of these formulae:

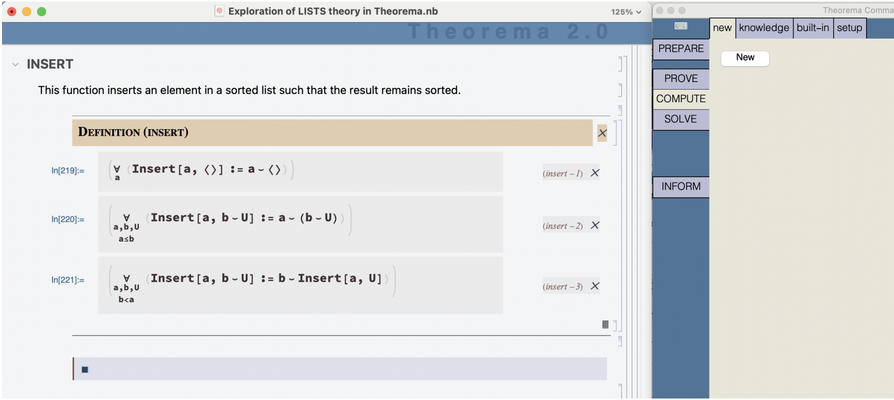
(a) following the examples shown, generate both the full proof and the simplified proofs in the Theorema system;

(b) generate in Theorema different proofs by choosing different provers, different inference rules to be applied, change the search depth, change the search time;

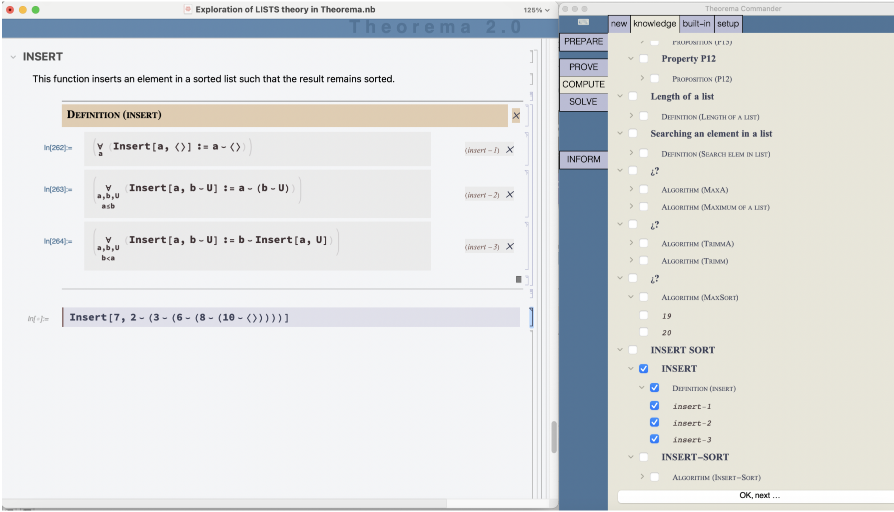
Theory exploration



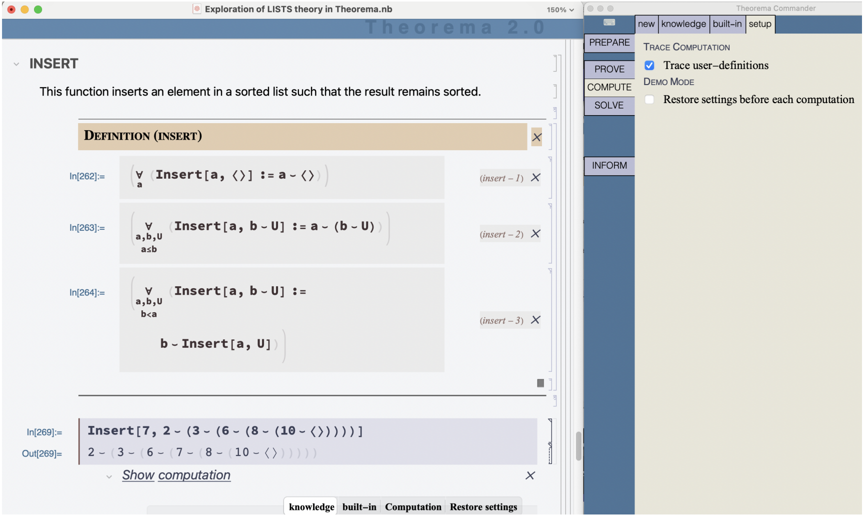
Compute with definitions



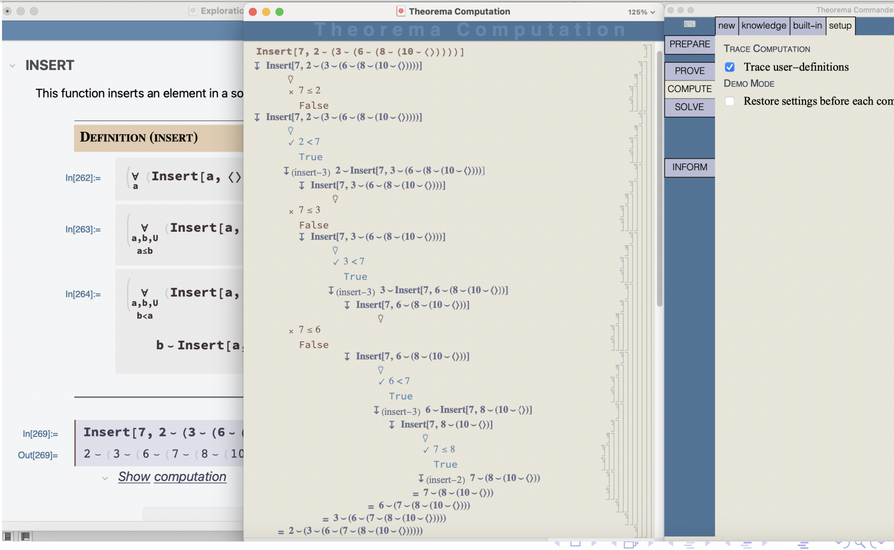
Select KB


Exploration of LISTS theory in Theorema.nb
125%
Theorema 2.0
INSERT
This function inserts an element in a sorted list such that the result remains sorted.
Definition (Insert)
In[262]:=
∀ a Insert[a, ⟨⟩] := a ⌣ ⟨⟩
(insert – 1)
In[263]:=
∀ a,b,U a≤b Insert[a, b ⌣ U] := a ⌣ (b ⌣ U)
(insert – 2)
In[264]:=
∀ a,b,U b<a Insert[a, b ⌣ U] := b ⌣ Insert[a, U]
(insert – 3)
In[ ]:=
Insert[7, 2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩))))]
Theorema Commander
new
knowledge
built-in
setup
PREPARE
PROVE
COMPUTE
SOLVE
INFORM
Property P12
Proposition (P12)
Length of a list
Definition (Length of a list)
Searching an element in a list
Definition (Search elem in list)
¿?
Algorithm (MaxA)
Algorithm (Maximum of a list)
¿?
Algorithm (TrimmA)
Algorithm (Trimm)
¿?
Algorithm (MaxSort)
19
20
INSERT SORT
INSERT
Definition (Insert)
insert-1
insert-2
insert-3
INSERT–SORT
Algorithm (Insert–Sort)
OK, next ...

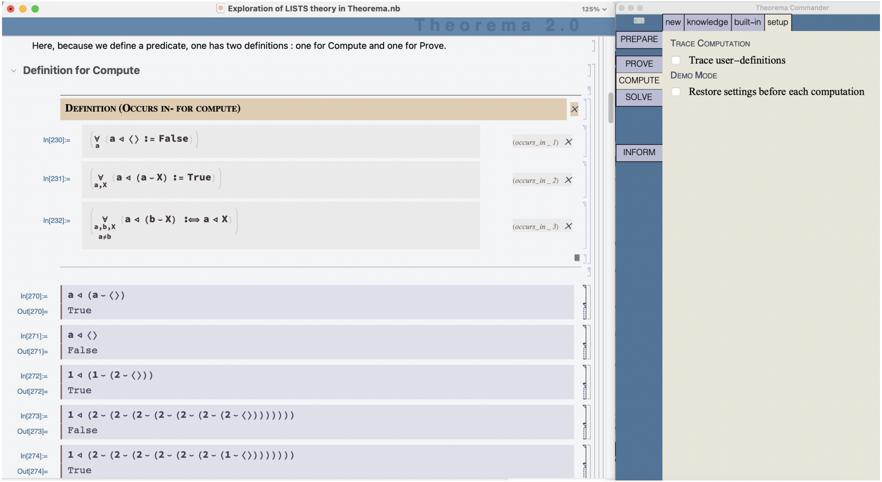
Compute Insert



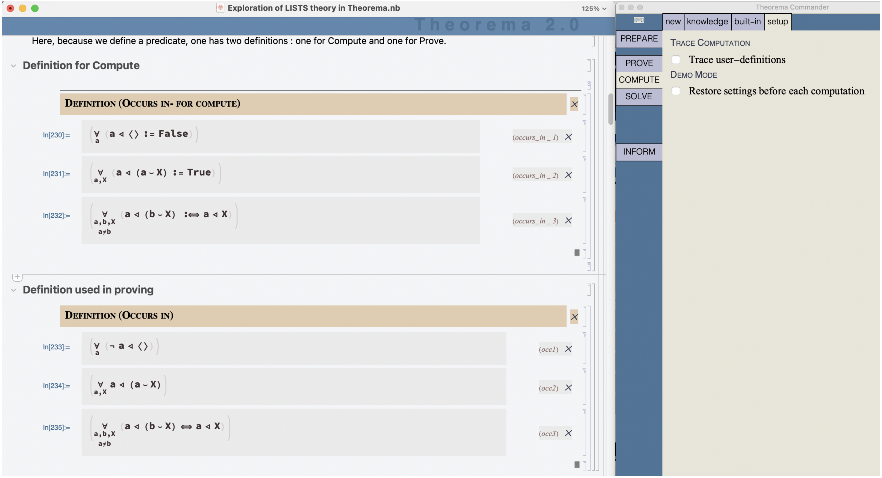
Show computation steps



Compute with predicate definitions



Compute with predicate definitions



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
$\longrightarrow$ 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: $\underset{X}{\forall}(I[X] \implies \underset{Y}{\exists} O[X, Y])$.
- Binary functions: $\underset{X\,Y}{\forall\forall}(I[X, Y] \implies \underset{Z}{\exists} O[X, Y, Z])$.

Proofs:

- Automatically generated by **AlCons**:
  - 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: $a < b, \quad a \leq b$

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: ε, $\langle L, a, R \rangle$

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

Multisets:

$$\mathcal{M}[\langle\rangle] = \emptyset \qquad \mathcal{M}[a \smile V] = \{\{a\}\} \uplus \mathcal{M}[V]$$

$$\mathcal{M}[\varepsilon] = \emptyset \qquad \mathcal{M}[\langle L, a, R \rangle] = \mathcal{M}[L] \uplus \{\{a\}\} \uplus \mathcal{M}[R]$$

AlCons: 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, **AlCons** 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

AlCons 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: $\underset{X\,Y}{\forall\exists} P[X, Y] \overset{\text{Skolem}}{\longrightarrow} \underset{Y}{\exists} P[X_0, Y] \overset{\text{metavar}}{\longrightarrow} P[X_0, Y^*]$
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
$\longrightarrow$ *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 $P[t, Sort[t]]$ 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 $x, x', \ldots$
- metavariables from goal $\longrightarrow$ existential $y, y', \ldots$
- conjecture:

$$\underset{x\,x'}{\forall\forall} \ldots (P[x, x', \ldots] \Longrightarrow \underset{y\,y'}{\exists\exists} \ldots Q[x, x', \ldots, y, y', \ldots])$$

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

Strategy: Cascading (function usage)

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

- new assumption:

$$\underset{x}{\forall}\underset{x'}{\forall} \ldots (P[x, x', \ldots] \Longrightarrow Q[x, x', \ldots, F[x, x', \ldots], F'[x, x', \ldots], \ldots])$$

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

Group multisets

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

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: $IsSorted[Y^*] \;\wedge\; \mathcal{M}[Y^*] = \mathcal{M}[a_0] \uplus \mathcal{M}[R_0] \uplus \ldots$
The assumptions contain: $IsSorted[R_0]$

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

Cascading: synthesize *Insert* from the conjecture:

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

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: $IsSorted[Y^*] \;\wedge\; \mathcal{M}[Y^*] = Insert[a_0, R_0] \uplus \ldots$

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)

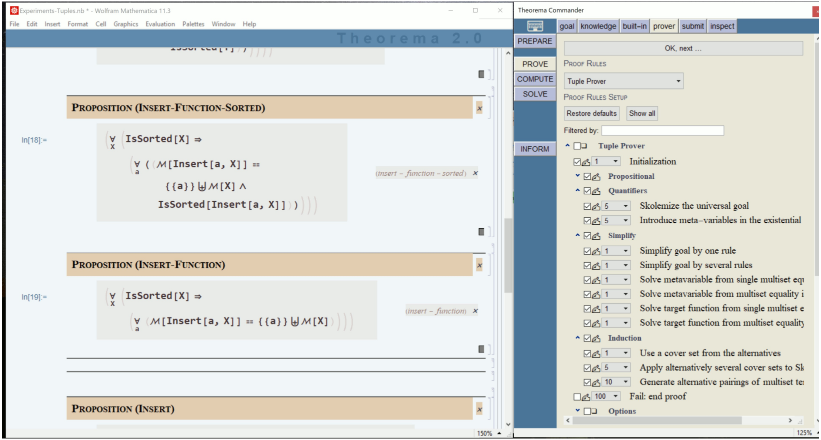
Insert-Sort: (cover set on Skolem constant)

$$\underset{a,L,R}{\forall}\begin{pmatrix} Sort[\varepsilon] = \varepsilon \\ Sort[\langle L, a, R\rangle] = Insert[a, Sort[Concat[L, R]]] \end{pmatrix}$$

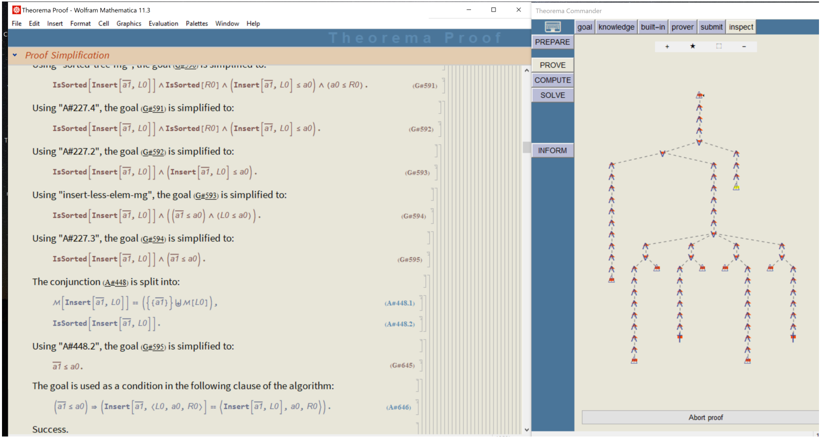
Quick-Sort: (cover set on metavariable)

$$\underset{a,L,R}{\forall}\begin{pmatrix} Sort[\varepsilon] = \varepsilon \\ Sort[\langle L, a, R\rangle] = \\ \langle\ SmallerEq[a, Sort[Concat[L, R]], \\ a, \\ Bigger[a, Sort[Concat[L, R]]]]\ \rangle \end{pmatrix}$$

Proving in *Theorema*



Generated Proof and Proof Tree in *Theorema*

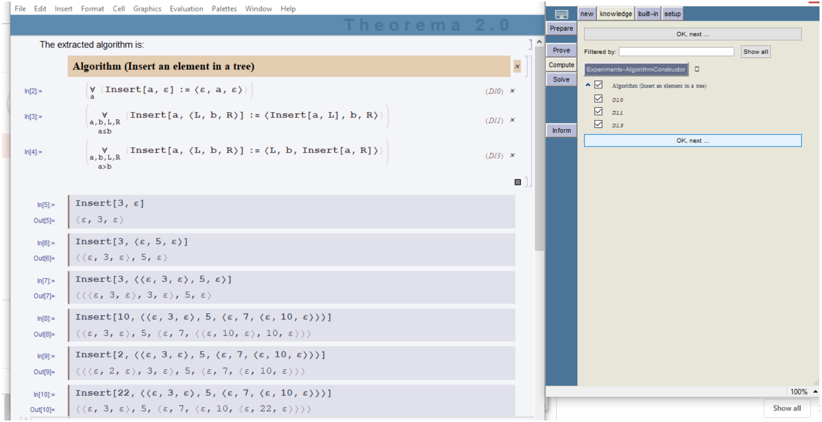


AlCons Demo

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

Proof animation

Computation with the Extracted Algorithm



Results

- **AlCons**: 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. **AlCons: 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.

$\longrightarrow$ **two logical conjectures**

Proofs:

- Automatically generated by the *Theorema* prover:
    - success: proof $\longrightarrow$ 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

$$\underset{X}{\forall}\Big(IsSorted[Sort[X]] \wedge \mathcal{M}[X] = \mathcal{M}[Sort[X]]\Big)$$

- 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 $\underset{X,Y}{\forall} A[X, Y] \implies G[X, Y]$ where
  - X, Y are the Skolem contants 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

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

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

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

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

(Each argument of E_1 is smaller than each argument of E_2.)
Example: A goal (or an assumption) of the form

$$\mathit{Insert}[a, T] \leq \mathit{Merge}[U, b \smile V]$$

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 the it, and then applies recursively to the list of the remaining elements, selected by the function *TrimMM*.

Algorithm

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

Algorithm

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

Computation with the Algorithms

Theorema 2.0

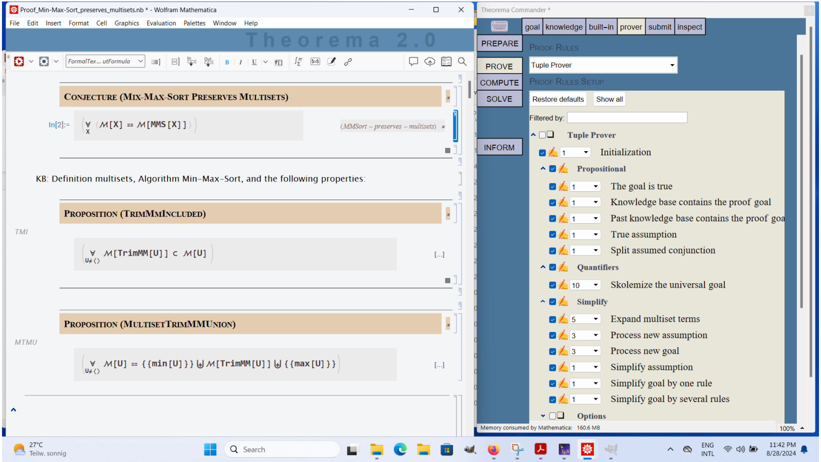
ALGORITHM (MINMAXSORT)

```
In[83]:=  MMS[⟨⟩] == ⟨⟩                                                        (Min – Max – Sort – 1)
In[84]:=  ∀_a (MMS[a ⌣ ⟨⟩] == a ⌣ ⟨⟩)                                          (Min – Max – Sort – 2)
In[85]:=  ∀_{a,b,U} (MMS[a ⌣ (b ⌣ U)] == min[a ⌣ (b ⌣ U)] ⌣ (MMS[TrimMM[a ⌣ (b ⌣ U)]] ⌢ max[a ⌣ (b ⌣ U)]))   (Min – Max – Sort – 3)
```

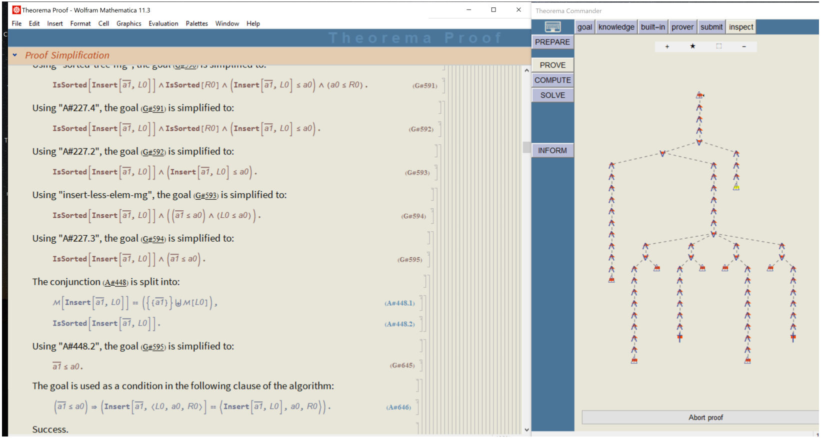
Compute

```
In[90]:=  MMS[3 ⌣ ⟨⟩]
Out[90]=  3 ⌣ ⟨⟩

In[91]:=  MMS[3 ⌣ (2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩))))))]
Out[91]=  2 ⌣ (3 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩)))))

In[92]:=  MMS[10 ⌣ (2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩))))))]
Out[92]=  2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ (10 ⌣ ⟨⟩)))))

In[93]:=  MMS[1 ⌣ (2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩))))))]
Out[93]=  1 ⌣ (2 ⌣ (3 ⌣ (6 ⌣ (8 ⌣ (10 ⌣ ⟨⟩)))))

In[94]:=  MMS[10 ⌣ (4 ⌣ (3 ⌣ (2 ⌣ (8 ⌣ (1 ⌣ ⟨⟩))))))]
Out[94]=  1 ⌣ (2 ⌣ (3 ⌣ (4 ⌣ (8 ⌣ (10 ⌣ ⟨⟩)))))
```

Proving in *Theorema*



Generated Proof and Proof Tree in *Theorema*



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 work[1]

- 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!