Exploring an Algorithm for Polynomial Interpolation in the *Theorema* System

*Wolfgang Windsteiger**

RISC Institute, University of Linz
A-4232 Hagenberg, Austria

Abstract

We present a case study using the *Theorema* system to explore an algorithm for polynomial interpolation. The emphasis of the case study lies on formulating mathematical knowledge in *one* language that appears in its syntax close to common mathematical language but is precise enough to formulate all details necessary for proving. Moreover, the language allows the computation of concrete examples without any further translation into an executable language.

---

* This work has been supported by the "Spezialforschungsbereich for Numerical and Symbolic Scientific Computing" (SFB F013) at the University of Linz and the european union "CALCULEMUS Project" (HPRN-CT-2000-00102). The author would like to thank B.Buchberger and M.Rosenkranz for many valuable discussions during the case study that led to this paper.

---

1 Introduction

Existing mathematical software systems (e.g. Mathematica, MAPLE, Gap etc.) have made big progress over the past decades by providing the users with comprehensive libraries of sophisticated algorithms in various areas of mathematics. In parallel, basically independently, the past decades have also produced enormous progress in the automation of the proving activity of mathematicians. These approaches, however, have put their emphasis mainly on proving isolated theorems, where systems like Otter, Spass, or Vampire are rather successful.

The challenge for the future is the theoretical foundation, the design, and implementation of *integral software systems* that guide, support, and at least partially automate the entire process of inventing, proving, and applying mathematical knowledge using mathematical knowledge and method libraries and, as a result, expanding these libraries by the result of this process. Recently, this integral view and research program for the next generation of mathematical software systems has been named “Mathematical Knowledge Management” (MKM) in the first international workshop on MKM, Sept. 14-16, 2001, organized at RISC-Linz by B. Buchberger, see [MKM 03].

From the very beginning the *Theorema* project was meant to give a logical and software technological frame for the entire mathematical knowledge management as an integral coherent process with the following key design objectives and ideas:

- The three main activities of mathematics—proving, computing, and solving—should all be available in one logical and software technological frame. Moreover, the natural interplay between proving, computing, and solving should be supported by the system.

- Domains, functors and categories as a natural and powerful structuring mechanism for generic build-up of systematic knowledge and methods.
- Preference to special proving, simplifying, and solving algorithms for special mathematical theories as opposed to a one-method-approach to proving all of mathematics (e.g. resolution method) with the possibility to link powerful and provenly correct algebraic algorithms (e.g. Cylindrical Algebraic Decomposition for quantifier elimination, Gröbner bases method for systems of algebraic equations, Risch's algorithm for symbolic integration, or Zeilberger's algorithm for sum identities).
- High usability and attractiveness for the working mathematician by using various software technological advances, e.g. flexible syntax imitating the usual textbook-style, proof presentation with high readability and postprocessing capabilities.
- Knowledge management tools for the construction, maintenance, and modification of large mathematical knowledge bases together with system support for integral theory exploration: failure analysis for proofs and conjecture generation based on failure analysis; build-up of libraries of problem types, knowledge types and algorithm types and their systematic use in the theory exploration process.

In this paper, we would like to demonstrate part of the currently available features in *Theorema* in a case study, namely polynomial interpolation. In this case study, we show: 1) How the domain of univariate polynomials can be built up in generic form using the functor construct available in *Theorema*. 2) We demonstrate how the problem of interpolation can be specified in this setting. 3) How a special solution of the interpolation problem, namely by the Neville algorithm, can be formulated. 4) How its correctness proof can be given. 5) How the algorithm can finally be applied to concrete problems.
With this case study we want to demonstrate the following aspects, which play a crucial role in an integral view of the mathematical knowledge management process: 1) Problem specification, algorithmic formulation, correctness proof, and computation (application to concrete examples) can be done within the one and uniform logic and system frame of *Theorema*. 2) The possibility for formulating mathematical knowledge and methods in a generic way that guarantees applicability and re-usability in a wide range of (hierarchically built-up) domains. 3) Attractive choice of syntax, which is close to the common usage of notation in mathematical textbooks and at the same time is formally rigorous in the sense that all formulae (including the algorithms) are just formulae in the underlying predicate logic. We want to particularly emphasize the didactic challenge in this context: on the one hand, every detail must be spelled out unambiguously whereas, on the other hand, we want to stay close to common use (and often ab-use) of mathematical notation used in mathematical texts.
In this paper, we do not yet talk about possibilities in *Theorema* for guiding and supporting the invention process. However, note for example, that proofs for elementary properties of e.g. polynomial evaluation, which are needed for polynomial interpolation, are naturally suggested by the structure of the polynomial domain as defined in the polynomial functor. However, we would also like to emphasize that systematic methods for mathematical exploration, in our view, are not only a desirable goal for completely automating the invention process (which will never be possible by the inherent incompleteness of mathematics) but are a very reasonable and worthwhile research goal for improving the didactics and heuristics of mathematics. In concrete terms this means that at certain stages in the invention process instead of getting support from the system the user may also interact with the system by allowing the user to guide the prover or suggesting the prover the general structure of an algorithm.
This case study is taken from lecture notes used in courses, whose goal is to present the entire content of the first year of mathematics study in an algorithmic fashion. The *Theorema* language turns out to provide a suitable frame for these courses, because the entire mathematical knowledge including all algorithms can be formulated in a style, which later allows *proving* all the properties in subsequent courses. Similar case studies have been initiated for other topics such as Gaussian elimination or Gröbner bases theory.

2 The Polynomial Functor

We present a case study in the domain of univariate polynomials over a field K. Polynomials can be defined to be infinite K-sequences with only finitely many non-zero elements, i.e. the (infinite) direct sum of (infinitely many copies of) the coefficient field. Thus, for each such sequence there must be an index, such that the sequence consists of only zeroes after this index. This is an appropriate setting for a computer-representation of polynomials, since it allows to naturally represent a polynomial by a K-tuple of its coefficients up to the last non-zero entry in the sequence.

In the *Theorema* language, the domain of univariate polynomials over a coefficient field K can be introduced nicely by a *Functor*. Functors are a well-known concept (e.g. in category theory) and the hierachical construction of mathematical domains by functors has already been used in Computer Algebra systems (the use of domains and categories is one of the distinctive design features of the well-known AXIOM system, see [AXIOM]). The algorithmic nature of functors as introduced in the *Theorema* system (see [Buchberger 96a], [Buchberger 96b], and [Windsteiger 99]) relates to how functors are available in the programming language ML. In general, a functor allows to construct a new domain from an already existing domain. In the concrete case, we assume a domain K and construct the domain of polynomials over K, named Poly[K], by defining the characteristic property for the elements in Poly[K] and by defining operations in Poly[K] based on available operations in the underlying domain K. Note that we will present here only part of the functor, namely just those definitions that are relevant for further discussion on the polynomial interpolation algorithm presented in Section 4. The functor definition shown in Figure 1 must be read as follows: The domain Poly[K] is such a domain P, where, for any p, q, n, a, the following operations are defined:

- $\underset{P}{\in}[p]$ (p is an element in P) iff p is a tuple of positive length with elements from K.
- $\underset{P}{x}$ (a new constant x in P) is the tuple $\langle \underset{K}{0}, \underset{K}{1} \rangle$.
- $\deg_{P}[p]$ (the degree of p in P) is either 0 or it is such an i between 1 and $|p|$ such that … . (The "such a-quantifier" $\underset{i=1,\ldots,|p|}{\ni} \ldots$ is a special language construct available in the *Theorema* language, which stands for "such an i between 1 and $|p|$ satisfying the property …". It provides a formal frame for giving *implicit function definitions*.)
- etc.

The constant x in the polynomial domain plays exactly the role of the "polynomial indeterminate" x when thinking of polynomials as "arithmetic terms" of the form $\sum_{k=0}^{n} p_i\, x^i$. In the functor notation all function, predicate, and object constants carry the domain, for which they are defined, as an underscript, e.g. $\underset{\text{Poly}[K]}{-}$ for subtraction in the domain of polynomials as opposed to $\underset{K}{-}$ for subtraction in the domain K. In the remainder of this paper, all text in gray boxes is *Theorema* input or output as it appears in a *Theorema* session. The syntax used—including all special symbols and typesetting facilities—is machine-readable and the *Theorema* parser translates it unambiguously into *Theorema*'s internal representation.

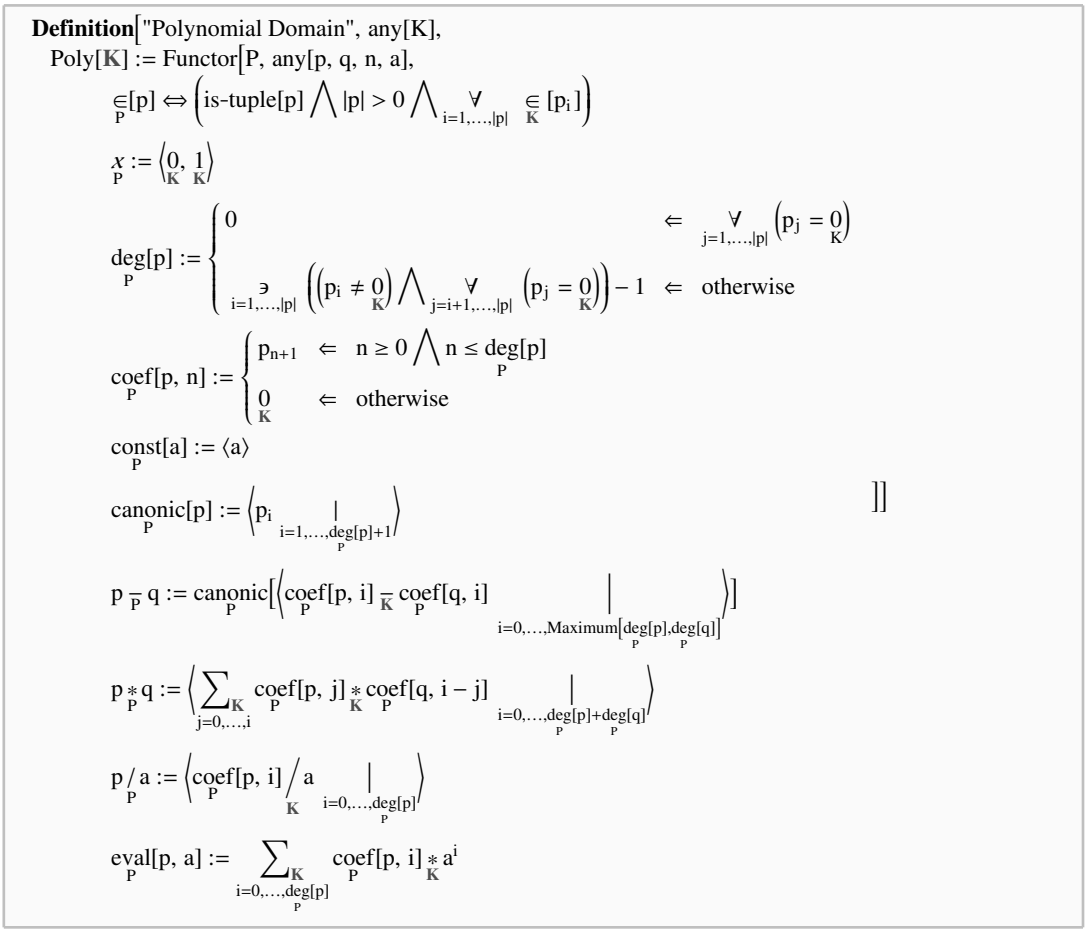
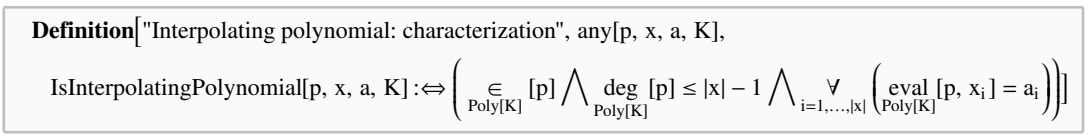


Figure 1: The functor defining the domain of univariate polynomials.

3 Problem Specification: Polynomial Interpolation

Given a polynomial p over K and two tuples x and a, one might ask, whether p evaluates (in Poly[K]) to a_i at x_i (for all $i = 1, \ldots, |x|$), i.e. whether p is an interpolating polynomial for x and a in Poly[K]. This consideration is natural because then the "polynomial function associated with p" would run through all the given points $\langle x_i, a_i \rangle$, which is a crucial property for many applications in mathematics (e.g. several methods for solving equations are based on iteratively solving equations for interpolating polynomials). In *Theorema*, this property can be expressed as follows:



Under certain restrictions—the tuples x and a must be non-empty and have equal length and the elements of x must be mutually distinct—it can be shown that for given x, a, and K there exists a unique polynomial p over K of degree less equal $|x| - 1$ such that IsInterpolatingPolynomial[p, x, a, K]. Of course, it is then desirable to come up with an *algorithm* that computes the interpolating polynomial for given tuples x, a and coefficient field K.

4 Solution to the Interpolation Problem: Neville Algorithm

An ad-hoc solution for an interpolation algorithm can immediately be extracted from the proof of unique existence of the interpolating polynomial. The proof of this fact can be reduced to prove solvability of a system of linear equations, which is always guaranteed under the given restrictions on x and a. The interpolating polynomial can then be computed by solving the linear system. However, there are better algorithms for finding the interpolating polynomial, for instance the *Neville algorithm*, which proceeds by *recursion* over the tuples x and a. Written in *Theorema* the algorithm is given as follows:

Algorithm["Neville", any[x, a, x0, $\overline{x}$, xn, a0, $\overline{a}$, an, K],

NevillePolynomial[⟨x⟩, ⟨a⟩, K] = $\underset{\text{Poly[K]}}{\text{const}}$[a]

NevillePolynomial[⟨x0, $\overline{x}$, xn⟩, ⟨a0, $\overline{a}$, an⟩, K] =

$$\left(\left(x \underset{\text{Poly[K]}}{-} \underset{\text{Poly[K]}}{\text{const}}[x0]\right) \underset{\text{Poly[K]}}{*} \text{NevillePolynomial}[\langle \overline{x}, xn\rangle, \langle \overline{a}, an\rangle, K] \underset{\text{Poly[K]}}{-}\right.$$

$$\left.\left(x \underset{\text{Poly[K]}}{-} \underset{\text{Poly[K]}}{\text{const}}[xn]\right) \underset{\text{Poly[K]}}{*} \text{NevillePolynomial}[\langle x0, \overline{x}\rangle, \langle a0, \overline{a}\rangle, K]\right) \underset{\text{Poly[K]}}{/} \left(xn \underset{K}{-} x0\right)$$

5 Correctness of the Algorithm

The correctness theorem for the Neville algorithm written in *Theorema* syntax is as follows:

Theorem["Neville polynomial is interpolating polynomial", any[is-tuple[x], a, K], with[|x| > 0 ∧ |a| = |x|]
IsInterpolatingPolynomial[NevillePolynomial[x, a, K], x, a, K]]

For automatically proving a formula *for all tuples* x, we can use the *tuple induction prover* available in the *Theorema* system. This prover implements a special prove technique available for tuples, namely Noetherian induction. In order to call this prover, we issue the *Theorema* command

Prove[Theorem["Neville polynomial is interpolating polynomial"], using → KB, by → TupleInduction],

where KB contains the knowledge base of auxiliary assumptions needed for the proof. We will refer to required knowledge from KB in the proof later. The tuple induction prover comes up with a successful and complete proof. Due to space limitations, we show only the key steps of this proof (text in boxes contains explanation of the prove techniques applied, *all the rest*—including formula labels, references, and intermediate text—is generated completely automatically by the prover).

Since x is a tuple an induction over x is set up. Since nothing is known about a and K the prover chooses a, K arbitrary but fixed.

Induction base: x = ⟨x1⟩ for arbitrary but fixed x1. We have to show:

(1) |a| = 1 ⇒ IsInterpolatingPolynomial[NevillePolynomial[⟨x1⟩, a, K], ⟨x1⟩, a, K],

We assume

(2) |a| = 1,

and show

(3) IsInterpolatingPolynomial[NevillePolynomial[⟨x1⟩, a, K], ⟨x1⟩, a, K].

From (2), we can infer:

(4) a = ⟨a1⟩,

for some new constant a1.

Formula (3), using (4) and (Algorithm (Neville)), is implied by:

(5) IsInterpolatingPolynomial[$\underset{\text{Poly[K]}}{\text{const}}$[a1], ⟨x1⟩, ⟨a1⟩, K],

which, using (Definition (Polynomial Domain)), is implied by:

(6) $\text{IsInterpolatingPolynomial}[\langle a1\rangle, \langle x1\rangle, \langle a1\rangle, K]$,

which, using (Definition (Interpolating polynomial: characterization)), is implied by:

(7) $\underset{\text{Poly}[K]}{\in}[\langle a1\rangle] \bigwedge \underset{\text{Poly}[K]}{\deg}[\langle a1\rangle] \le |\langle x1\rangle| - 1 \bigwedge \underset{i=1,\ldots,|\langle x1\rangle|}{\forall} \left(\underset{\text{Poly}[K]}{\text{eval}}[\langle a1\rangle, \langle x1\rangle_i] = \langle a1\rangle_i\right)$,

Formula (7) can now be easily verified by expanding the polynomial operations defined in the functor.

Induction hypothesis: We assume for arbitrary but fixed $n \ge 1$

(8) $\underset{x}{\forall}\,(|x| = n \wedge |a| = |x|) \Rightarrow \text{IsInterpolatingPolynomial}[\text{NevillePolynomial}[x, a, K], x, a, K]$,

and show

(9) $(|x| = n + 1 \wedge |a| = |x|) \Rightarrow \text{IsInterpolatingPolynomial}[\text{NevillePolynomial}[x, a, K], x, a, K]$.

We assume

(10) $|x| = n + 1$,

(11) $|a| = |x|$,

From (10) and (11), we can infer:

(12) $x = \langle x0, \overline{x}, xn\rangle$,

(13) $a = \langle a0, \overline{a}, an\rangle$,

for new constants x0, xn, a0, an and new constant sequences $\overline{x}$ and $\overline{a}$ of length $n - 1$.

The prover guesses this particular structure for representing x and a from the definition of NevillePolynomial.

It remains to show

(14) $\text{IsInterpolatingPolynomial}[\text{NevillePolynomial}[\langle x0, \overline{x}, xn\rangle, \langle a0, \overline{a}, an\rangle, K], \langle x0, \overline{x}, xn\rangle, \langle a0, \overline{a}, an\rangle, K]$.

Formula (14), using (Algorithm (Neville)), is implied by:

(15) $\text{IsInterpolatingPolynomial}\Big[\Big(\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[x0]\Big) \underset{\text{Poly}[K]}{*} \text{NevillePolynomial}[\langle \overline{x}, xn\rangle, \langle \overline{a}, an\rangle, K]\Big) \underset{\overline{\text{Poly}[K]}}{}$,

$\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[xn]\Big) \underset{\text{Poly}[K]}{*} \text{NevillePolynomial}[\langle x0, \overline{x}\rangle, \langle a0, \overline{a}\rangle, K]\Big) \Big/_{\text{Poly}[K]} \big(xn \underset{K}{\overline{\ }} x0\big)$,

$\langle x0, \overline{x}, xn\rangle, \langle a0, \overline{a}, an\rangle, K\Big]$

Membership in the polynomial domain and the degree bound for the interpolating polynomial are not too difficult to prove. For proving the evaluation property we need some auxiliary knowledge about polynomial evaluation, such as e.g. $\text{eval}[p + q, a] = \text{eval}[p, a] + \text{eval}[q, a]$, which can be proven by *another special prover*, which can handle formulae containing the $\sum$-quantifier. In our approach of theory exploration, we suppose that this knowledge has already been proven in a previous exploration phase and is for this proof available in the knowledge base KB. After several simplifications we arrive at the following formula to be proved:

$$\underset{i=1,\ldots,n+1}{\forall}\Bigg(\Big(\underset{\text{Poly}[K]}{\text{eval}}\Big[\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[x0]\Big), \langle x0, \overline{x}, xn\rangle_i\Big] \underset{K}{*} \underset{\text{Poly}[K]}{\text{eval}}\Big[$$

$$\text{NevillePolynomial}[\langle \overline{x}, xn\rangle, \langle \overline{a}, an\rangle, K], \langle x0, \overline{x}, xn\rangle_i\Big] \underset{K}{\overline{\ }} \underset{\text{Poly}[K]}{\text{eval}}\Big[\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[xn]\Big), \langle x0, \overline{x}, xn\rangle_i\Big] \underset{K}{*}$$

$$\underset{\text{Poly}[K]}{\text{eval}}\,[\text{NevillePolynomial}[\langle x0, \overline{x}\rangle, \langle a0, \overline{a}\rangle, K], \langle x0, \overline{x}, xn\rangle_i]\Big) \Big/_{K} \big(xn \underset{K}{\overline{\ }} x0\big) = \langle a0, \overline{a}, an\rangle_i\Bigg)$$

Since $\langle x0, \overline{x}, xn\rangle_i$ (and $\langle a0, \overline{a}, an\rangle_i$) can be simplified in case $i = 1$ or $i = n + 1$ a case distinction is now made:

Case $i = 1$: We have to show

$$\Big(\underset{\text{Poly}[K]}{\text{eval}}\Big[\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[x0]\Big), x0\Big] \underset{K}{*} \underset{\text{Poly}[K]}{\text{eval}}\,[\text{NevillePolynomial}[\langle \overline{x}, xn\rangle, \langle \overline{a}, an\rangle, K], x0] \underset{K}{\overline{\ }}$$

$$\underset{\text{Poly}[K]}{\text{eval}}\Big[\Big(x \underset{\text{Poly}[K]}{\overline{\text{Poly}[K]}} \underset{\text{Poly}[K]}{\text{const}}[xn]\Big), x0\Big] \underset{K}{*} \underset{\text{Poly}[K]}{\text{eval}}\,[\text{NevillePolynomial}[\langle x0, \overline{x}\rangle, \langle a0, \overline{a}\rangle, K], x0]\Big) \Big/_{K} \big(xn \underset{K}{\overline{\ }} x0\big) = a0$$

which, using (Definition (Polynomial Domain)), is implied by:

(22) $0 \underset{K}{\overline{\ }} \big((x0 - xn) \underset{K}{*} a0\big) \Big/_{K} \big(xn \underset{K}{\overline{\ }} x0\big) = a0$.

Formula (22) can be verified by auxiliary knowledge on arithmetic in K. The remaining cases proceed analogously.

6 Application of the Algorithm to Concrete Examples

The recursive Algorithm["Neville"] can be used immediately in computations *without any translation* to some machine-executable language. The *Theorema* command "Compute" can perform rewriting using the recursive definition as given in Section 4. Rewriting is done by the interpreter of the underlying Mathematica system. In addition, it can access semantics for the algorithmic language constructs provided by the *Theorema* language (e.g. finite tuples, quantifiers with finite ranges, arithmetic on numbers, etc.), which is needed for performing polynomial arithmetic as defined in the polynomial functor in Section 2.

```
Compute[NevillePolynomial[⟨1, 2, 3, 4, 5⟩, ⟨3, 1, 5, 2, 6⟩, ℚ],
  using → ⟨Definition["Polynomial Domain"], Algorithm["Neville"]⟩]
```

$$\left\langle 51, -\frac{1093}{12}, \frac{443}{8}, -\frac{161}{12}, \frac{9}{8} \right\rangle$$

This computation tells that the Neville-polynomial over $\mathbb{Q}$ for the tuples $\langle 1, 2, 3, 4, 5\rangle$ and $\langle 3, 1, 5, 2, 6\rangle$ is the polynomial $\langle 51, -\frac{1093}{12}, \frac{443}{8}, -\frac{161}{12}, \frac{9}{8}\rangle$, which would commonly be written as the arithmetic term $51 - \frac{1093}{12}\, x + \frac{443}{8}\, x^2 - \frac{161}{12}\, x^3 + \frac{9}{8}\, x^4$.

7 Conclusion

The *Theorema* system has been used in formal development of part of a mathematical theory. An entire exploration cycle—from defining mathematical concepts, over stating mathematical properties, computer-supported proving, until finally applying mathematical knowledge to concrete objects—has been carried through inside the system. The main objective of this case study is to show the integration of *proving* and *computing* in combination with an attractive mathematics-oriented syntax inside *one system*.

References

[AXIOM] Axiom. *Developed by IBM Research, directed by R. Jenks*. http://www.nag.com/symbolic_software.asp.

[Buchberger 96a] B. Buchberger: *Symbolic Computation: Computer Algebra and Logic*. In: Frontiers of Combining Systems (F. Baader, K.U. Schulz eds.), pp. 193-220. Applied Logic Series. Kluwer Academic Publishers, 1996.

[Buchberger 96b] B. Buchberger: *Mathematica as a Rewrite Language*. Invited paper in: Proceedings of the Fuji Conference on Functional Logic Programming, Shonan Village, Nov 1-4, 1996, (T. Ida ed.), pp. 1–13,Telos Publishing.

[MKM 03] Bruno Buchberger, Gaston Gonnet, Michiel Hazewinkel (eds.). *Mathematical Knowledge-ment Management*. Special issue of the journal Annals of Mathematics and Artificial Intelligence, Kluwer Publishing Company, 2003.

[Windsteiger 99] W. Windsteiger: *Building up Hierarchical Mathematical Domains Using Functors in Theorema*. In: A. Armando and T. Jebelean, editors, Electronic Notes in Theoretical Computer Science, vol 23-3, p. 83-102, Elsevier, 1999.