The Matita Interactive Theorem Prover

Andrea Asperti[1], Wilmer Ricciotti[1], Claudio Sacerdoti Coen[1], and Enrico Tassi[2]

[1] Department of Computer Science, University of Bologna
Mura Anteo Zamboni, 7 — 40127 Bologna, ITALY
{asperti,ricciott,sacerdot}@cs.unibo.it
[2] Microsoft Research - INRIA Joint Centre
enrico.tassi@inria.fr

Abstract. Matita is an interactive theorem prover being developed by the Helm team at the University of Bologna. Its stable version 0.5.x may be downloaded at http://matita.cs.unibo.it. The tool originated in the European project MoWGLI as a set of XML-based tools aimed to provide a mathematician-friendly web-interface to repositories of formal mathematical knoweldge, supporting advanced content-based functionalities for querying, searching and browsing the library. It has since then evolved into a light but fully fledged ITP, particularly suited for the assessment of innovative ideas, both at foundational and logical level. In this paper, we give an account of the whole system, its peculiarities and its main applications.

1 The system

Matita is an interactive proof assistant, adopting a dependent type theory - the Calculus of (Co)Inductive Constructions (CIC) - as its foundational language for describing proofs. It is thus compatible, at the proof term level, with Coq [27], and the two systems are able to check each other's proof objects. Since the two systems do not share a single line of code, but are akin to each other, it is natural to take Coq as the main term of comparison, referring to other systems (most notably, Isabelle and HOL) when some ideas or philosophies characteristic of these latter tools have been imported into our system.

Similarly to Coq, Matita follows the so called *De Bruijn principle*, stating that proofs generated by the system should be verifiable by a small and trusted component, traditionally called *kernel*. Unsurprisingly, the kernel has roughly the same size in the two tools, in spite of a few differences in the encoding of terms: in particular, Matita's kernel handles *explicit substitutions* to mimic Coq's Section mechanism, and can cope with *existential metavariables*, i.e. non-linear placeholders that are Curry-Howard isomorphic to holes in the proofs. Metavariables cannot be instantiated by the kernel: they are considered as opaque constants, with a declared type, only equal to themselves.

While this extension does not make the kernel sensibly more complex or fragile, it has a beneficial effect on the size of the type inference subsystem, here

called *refiner*. In particular, the refiner can directly call the kernel to check the complex and delicate conditions (guardedness, positivity) needed to ensure the termination of recursive functions and detect erroneous definition of inductive types leading to logical paradoxes. The Matita refiner implements several advanced features like coercive subtyping [16] and subset coercions [24], that allow for some automatic modifications of the user input to fix a type error or annotate simple programs with proof obligations.

The kernel compares types up to conversion, that is a decidable relation involving β-reduction, constant unfolding and recursive function computation. On the contrary, the refiner deals with incomplete terms, and compares types with a higher order unification algorithm in charge of finding an instantiation for metavariables that makes the two types convertible. Higher order unification is in general semi-decidable, and is thus usually implemented as an extension of the usual first order algorithm equipped with some extra heuristics. To avoid the inherent complexity of combining together many heuristics, Matita enables the user to extend unification by means of *unification hints* [4], that give explicit solutions for the cases not handled by the basic algorithm.

Remarkably, many ad-hoc mechanisms studied in the last years for dealing with the formalization of algebraic structures, including Canonical Structures, Type Classes [25,28], and Coercion Pullbacks [21], can be implemented on top of unification hints.

Library. Besides the aforementioned components, that make up the core of all theorem provers, the most important aspect of Matita is its document-centric philosophy. Matita is meant, first of all, as an interface between the user and the mathematical library, storing definitions, theorems and notation. An important consequence of this is that once a concept has been defined and added to the library, it will stay visible unless it is removed or invalidated, with no need for the user to explicitly reference or include a part of the library.

Objects are stored in the library together with metadata, which are used for indexing and searching. The searching facility provided by Matita, that is a key component of the system, has been described in [2].

Disambiguation. A well known, complex problem for interactive provers is that, at a linguistic level, ordinary mathematical discourse systematically overloads symbols and abuses notations in ways that make mechanical interpretation difficult. This originates from various sources, including conflicting parsing rules, implicit information that a human reader can recover from the context, overloading of operators, and so on. The complexity of the problem is due to the fact that the comprehension of the text sometimes requires not just knowledge of the notation and conventions at play but some understanding of the relevant mathematical aspects of the discipline (e.g. the fact that a given set *can be equipped* by a suitable algebraic structure), requiring the system *to dig* into its base of knowledge, in order *to correctly parse* the statement!

Matita was designed keeping in mind that ambiguous notation is not an unfortunate accident, but a powerful tool in the hands of mathematicians. For

this reason, the user is allowed the maximum degree of flexibility in defining notation. To manage ambiguous notation, Matita provides an ambiguous parser (described in [23]) associating to the user input the set of all its possible interpretations (according to the defined notation). While ambiguous parsing is a potentially expensive operation, Matita is able to preserve efficiency by means of a sophisticated algorithm, capable of detecting semantic errors as early as possible, in order to prevent semantic analysis of a multitude of incorrect interpretations. At the end of the process, it is possible that we are left with more than one interpretation: in this case, Matita asks the user to select the correct interpretation, then it stores it into the script to avoid interrogating the user again the next time the script is executed.

2 Proof authoring

The user interface of Matita was inspired by that of CtCoq [11] and Proof General [9] and is, in our experience, quite easy to learn for newcomers. The main language used to write proof scripts is procedural, in the LCF tradition, and essentially similar to the one used by Coq. In addition to that, Matita features a declarative language ([13]), in the style usually ascribed to Trybulec (the so called Mizar-style [19]), and made popular in the ITP community mostly by the work of Wenzel [29].

Despite many similarities to Coq, Matita departs from it in more than one respect (see [5] for details). The sequent-window is based on a MathML-compliant GTK-widget providing a sophisticated bidimensional rendering and supporting hyperlinks. During proof authoring, *direct manipulation* of terms is available on the generated MathML markup: the user can follow hyperlinks to library objects, visually select semantically meaningful subterms using the mouse, and perform contextual actions on them like copy&paste or tactic application. To textually represent graphical selections Matita adopts *patterns*, that are generated on-the-fly from visual selections and consistently used by all tactics.

Step by step tacticals. LCF-tacticals (operations combining a number of tactics in complex proof strategies), which are used for a better syntactical structuring of proof scripts, are also provided by Matita. Tacticals provide syntax for expressing concepts like branching, mimicking the tree structure of proofs at the script level. Since branches of proof trees usually corresponds to conceptual parts of pen & paper proofs, the branching tactical helps improving script readability.

In other systems, the practical use of these constructs is limited by the need of executing each tactical in a single step, even though it is composed of multiple tactics. Instead, Matita offers the possibility of interrupting the execution of a script at intermediate evaluation steps of a tactical, allowing the user to inspect changes in the status and, if needed, edit the script. This is a notable improvement in the overall user experience. Step by step tacticals – also called *tinycals* – are described in [22].

Automation. Automation is a well known weak point of Coq, only partially compensated by powerful reflexive tactics. Matita was intended to fill this gap, with a particular attention to support the automation of those small logical transformations (*small step* automation) needed to *match* [7] the current goal versus the knowledge bases of already proved results, and which constitute the underlying *glue* [8] of the actual mathematical discourse. A large part of this glue can be expressed in form of rewritings, allowing mathematicians to freely move between different incarnations of the same entity without even mentioning the transformation (sometimes referred to as Poincaré's principle). For this reason, the main component of Matita automation is a powerful *paramodulation tool*[1] able to exploit the *whole library* of known equational *facts* (unit equalities) in order to solve equational goals. Paramodulation is also used to supply matching *up to equational rewriting* between a goal and a given statement, and this in turn is used to support a simple, smooth but effective integration between equational reasoning and a backward-based, Prolog-like resolution procedure. Again, this automation tactic exploits the *whole library* of visible results, adopting a philosophy already advocated and implemented by several successful systems (see e.g. [18]), and contrasting with the approach of Coq, requiring the user to thoroughly select a collection of theorems to be used. On the other side, we are not eager to extend our SLD-approach to a more general resolution technique, since the prolog style, being closer to the LCF backward based procedural approach, allows a better interaction between the user and the application, permitting the user to *drive* the automatic search, e.g. by pruning or reordering the search space [6].

3 Formalizations

In the last years, Matita was successfully used in formalizations of considerable complexity, spanning on the following areas:

Number theory These formalizations include results about Möbius's μ, Euler's ϕ and Chebyshev's ψ and θ functions, up to a fully arithmetical proof of the property of prime numbers known as Bertrand's postulate ([1,3]).

Constructive analysis The main result is Lebesgue's dominated convergence theorem in the new, abstract setting of convex uniform spaces [20]. The formalization stresses some features peculiar of Matita, like coercions pullback.

Programming languages metatheory Comprises the formalization of several different solutions of part 1A of the POPLmark challenge [10], characterized by a different treatment of binding structures.

Hardware formalization Matita has been used to provide two realistic and executable models of microprocessors for the Freescale 8-bit family and the 8051/8052 microprocessors respectively. The formalization also captures the intensional behaviour, comprising exact execution times.

[1] Matita paramodulation tool took part in the UEQ category of the 2009 CASC competition ([26]), scoring better than Metis [14], Otter [17], and iProver [15].

Software verification Matita is employed in the FET Open EU Project CerCo (Certified Complexity)[2] [12] for the verification of the first formally certified complexity preserving compiler. The compiler, which targets the 8051 microprocessor, annotates the input program (in C) with the exact computational cost of every O(1) program slice. The costs, that are dependent on the compilation strategy, are directly computed from the generated object code. Hence it will be possible to reason on a hard real time program at the C level, knowing that the compiled code will have the same behaviour. The formalization in Matita will include executable formal models of every intermediate languages, a dependently typed implementation of the compiler, and the proof of preservation of extensional and intensional properties.

References

1. Andrea Asperti and Cristian Armentano. A page in number theory. *Journal of Formalized Reasoning*, 1:1–23, 2008.
2. Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A content based mathematical search engine: Whelp. In *Proc. of TYPES'04*, volume 3839 of *LNCS*, pages 17–32. Springer-Verlag, 2004.
3. Andrea Asperti and Wilmer Ricciotti. About the formalization of some results by Chebyshev in number theory. In *Proc. of TYPES'08*, volume 5497 of *LNCS*, pages 19–31. Springer-Verlag, 2009.
4. Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. Hints in unification. In *TPHOLs 2009*, volume 5674/2009 of *LNCS*, pages 84–98. Springer-Verlag, 2009.
5. Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. User interaction with the Matita proof assistant. *J. Autom. Reasoning*, 39(2):109–139, 2007.
6. Andrea Asperti and Enrico Tassi. An interactive driver for goal directed proof strategies. In *Proc. of UITP'08.*, volume 226 of *ENTCS*, pages 89–105, 2009.
7. Andrea Asperti and Enrico Tassi. Smart matching. In *Proc. of MKM 2010*, volume 6167 of *LNCS*, pages 263–277, 2010.
8. Andrea Asperti and Enrico Tassi. Superposition as a logical glue. In *Proc. of TYPES'09*, EPTCS (to appear), 2010.
9. David Aspinall. Proof General: A generic tool for proof development. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2000*, volume 1785 of *LNCS*. Springer-Verlag, January 2000.
10. Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *Proc. of TPHOLs 2005)*, volume 3603 of *LNCS*, pages 50–65, 2005.
11. Yves Bertot. The CtCoq system: Design and architecture. *Formal Aspects of Computing*, 11:225–243, 1999.

[2] The project CerCo acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 243881.

12. The CerCo website.
`http://cerco.cs.unibo.it`.
13. Claudio Sacerdoti Coen. Declarative representation of proof terms. *J. Autom. Reasoning*, 44(1-2):25–52, 2010.
14. Joe Hurd. First-order proof tactics in higher-order logic theorem provers. Technical Report NASA/CP-2003-212448, Nasa technical reports, 2003.
15. Konstantin Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In *Proc. of IJCAR 2008*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.
16. Zhaohui Luo. Coercive subtyping. *J. Logic and Computation*, 9(1):105–130, 1999.
17. William McCune and Larry Wos. Otter - the CADE-13 competition incarnations. *J. of Autom. Reasoning*, 18(2):211–220, 1997.
18. Jia Meng, Claire Quigley, and Lawrence C. Paulson. Automation for interactive proof: First prototype. *Inf. Comput.*, 204(10):1575–1596, 2006.
19. The Mizar proof-assistant.
`http://mizar.uwb.edu.pl/`.
20. Claudio Sacerdoti Coen and Enrico Tassi. A constructive and formal proof of Lebesgue's dominated convergence theorem in the interactive theorem prover Matita. *Journal of Formalized Reasoning*, 1:51–89, 2008.
21. Claudio Sacerdoti Coen and Enrico Tassi. Working with mathematical structures in type theory. In *Proc. of TYPES'07*, volume 4941/2008 of *LNCS*, pages 157–172. Springer-Verlag, 2008.
22. Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. Tinycals: step by step tacticals. In *Proc. of UITP 2006*, volume 174 of *ENTCS*, pages 125–142. Elsevier Science, 2006.
23. Claudio Sacerdoti Coen and Stefano Zacchiroli. Efficient ambiguous parsing of mathematical formulae. In *Proc. of MKM 2004*, volume 3119 of *LNCS*, pages 347–362. Springer-Verlag, 2004.
24. Matthieu Sozeau. Subset coercions in Coq. In *Proc. of TYPES'06*, volume 4502/2007 of *LNCS*, pages 237–252. Springer-Verlag, 2006.
25. Matthieu Sozeau and Nicolas Oury. First-class type classes. In *Proc. of TPHOLs 2008*, pages 278–293, 2008.
26. Geoff Sutcliffe. The CADE-22 automated theorem proving system competition - CASC-22. *AI Commun.*, 23:47–59, January 2010.
27. The Coq Development Team. The Coq proof assistant reference manual.
`http://coq.inria.fr/doc/main.html`.
28. Markus Wenzel. Type classes and overloading in higher-order logic. In *Proc. of TPHOLs 1997*, volume 1275 of *LNCS*, pages 307–322, 1997.
29. Markus Wenzel. Isar – a generic interpretative approach to readable formal proof documents. In *Proc. of TPHOLS 1999*, volume 1690 of *LNCS*, pages 167–184. Springer, 1999.