CICM 2015 – Work in Progress Preface

The Conference on Intelligent Computer Mathematics has been held annually since 2008 at various locations in Europe and North America. The conference is made up of tracks on aspects of handling mathematical knowledge embodied in documents, proofs, algorithms and software systems. The 2015 conference was held 13–17 July in Washington D.C., USA. For several years now, CICM has had a track devoted to works in progress. This informal venue provides a forum to present original work that is not yet in a suitable form for presentation as a full paper or system description. Authors have found this useful to communicate that they are working on particular topics, and allows early feedback from the research community. For CICM 2015 we have continued this tradition. In response to a call for participation, several submissions were received, covering a broad span of CICM topics. These were reviewed by a committee consisting of the General PC Chair and all the track PC Chairs. Upon deliberation, it was decided that, bearing in mind the informal intent of the track, the quality of the submissions supported acceptance of six submissions. These are presented here in these proceedings alphabetically according to the surname of the first author.

July 2015

Manfred Kerber: General CICM PC Chair
Jacques Carette: Calculemus Track Chair
Cezary Kaliszyk: Mathematical Knowledge Management Track Chair
Florian Rabe: Systems & Data Track Chair
Volker Sorge: Digital Mathematical Libraries Track Chair

Content

Arithmetic in Metamath, Case Study: Bertrand's Postulate 3
Mario Carneiro

Auto-hyperlinking the Stacks Project 19
Johan Commelin, Josef Urban

The **SMGloM** Project and System 25
Deyan Ginev, Mihnea Iancu, Constantin Jucovshi, Andrea Kohlhase, Michael Kohlhase, Jürgen Schefter, Wolfram Sperber

Parsing Texts and Checking Proofs in LaTeX 29
Bob Neveln, Bob Alps

A Web Environment for Geometry 44
Pedro Quaresma, Vanda Santos, Milena Marić

Automatic and Transparent Transfer of Theorems along Isomorphisms in the Coq Proof Assistant 50
Theo Zimmermann, Hugo Herbelin

© of the articles is with the authors.

Arithmetic in Metamath, Case Study: Bertrand's Postulate

Mario Carneiro

The Ohio State University, Columbus OH, USA

Abstract. Unlike some other formal systems, the proof system Metamath has no built-in concept of "decimal number" in the sense that arbitrary digit strings are not recognized by the system without prior definition. We present a system of theorems and definitions and an algorithm to apply these as basic operations to perform arithmetic calculations with a number of steps proportional to an arbitrary-precision arithmetic calculation. We consider as case study the formal proof of Bertrand's postulate, which required the calculation of many small primes. Using a Mathematica implementation, we were able to complete the first formal proof in Metamath using numbers larger than 10. Applications to the mechanization of Metamath proofs are discussed, and a heuristic argument for the feasability of large proofs such as Tom Hales' proof of the Kepler conjecture is presented.

Keywords: Arithmetic · Metamath · Bertrand's postulate · decimal number · natural numbers · large proofs · ATP · Mathematica · formal proof

1 Introduction

The Metamath system, consisting of a formal proof language and computer verification software, was developed for the purpose of formalizing mathematics in a foundational theory which is as minimal as possible while still being able to express proofs "efficiently" (in a sense we will make more precise later) [1]. Although Metamath supports arbitrary axiom systems, we are mainly concerned in this paper with the `set.mm` database, which formalizes much of the traditional mathematics curriculum into a ZFC-based axiomatization [2].

In this context, one can define a model of the real numbers and show that it satisfies the usual properties, and within this set we have the natural numbers $\mathbb{N}$ and can give a name to the numbers $1, 2, 3, \ldots \in \mathbb{N}$. One important point of comparison to other proof languages such as Mizar [3] is that not every sequence of decimal digits is interpreted as a natural number. In particular, the goal is to show rigorous derivations directly from ZFC axioms with complete transparency and not to depend on indirect proofs of correctness of, say, a computer arithmetic algorithm. A sequence of digits is treated as any other identifier and only represents a natural number if it has been defined to do so. The list of sequences

so defined is quite short – we have the definitions

$$2 := 1+1, \quad 3 := 2+1, \quad 4 := 3+1, \quad 5 := 4+1, \quad 6 := 5+1,$$

$$7 := 6+1, \quad 8 := 7+1, \quad 9 := 8+1, \quad 10 := 9+1,$$

and this constitutes a complete listing of the defined integers in `set.mm` (not including 0 and 1, which are defined as part of the field operations).

The primary application of the `set.mm` has been for abstract math, and such small numbers were sufficient for prior theorems, but initial attempts at Bertrand's postulate showed that a new method was needed in order to systematize arithmetic on large numbers.

1.1 Bertrand's Postulate

Theorem 1 (Chebyshev, Erdős). *For every $n \in \mathbb{N}$ there is a prime p satisfying $n < p \leq 2n$.*

This statement was conjectured by Joseph Bertrand in 1845, from which the problem gets its name, and it was proven in 1852 by Chebyshev. Our version of the theorem looks like this:

$$\frac{}{\vdash n \in \mathbb{N} \rightarrow \exists p \in \mathbb{P}(n < p \wedge p \leq 2n)} \; \mathsf{bpos}^{1}$$

Although the complete proof of this theorem is not the purpose of this paper, this was the motivating problem that led to the developments described here. For our formalization we targeted not Chebyshev's proof but rather a simpler proof due to Paul Erdős. The proof is based on a detailed asymptotic analysis of the central binomial coefficients $\binom{2n}{n}$, and by keeping track of the bounds involved this shows Theorem 1 for $n > 4000$. A usual exposition of the proof will observe that the other 4000 cases can be shown by means of the sequence

$$2, 3, 5, 7, 13, 23, 43, 83, 139, 163, 317, 631, 1259, 2503, 4001, \tag{1}$$

which can be seen to consist of primes p_k such that $p_{k+1} < 2p_k$.

It is this last statement which is the most problematic for a formal system which cannot handle large numbers, because verifying that large numbers are prime requires even larger calculations, and doing such calculations by hand on numbers of the form $1+1+\cdots+1$ (as in our definition of the numbers up to 10) is quite impractical, requiring $O(n^2)$ operations to multiply numbers of order n.

[1] The sans-serif labels mentioned in this paper refer to theorem statements in `set.mm`; they can be viewed at e.g. `http://us.metamath.org/mpegif/bpos.html` for bpos.

1.2 Decimal Arithmetic

To solve the problem of the space requirements of unary numbers, the standard approach is to use base-10 arithmetic, or more generally base-b arithmetic for $b \geq 2$, in which a nonnegative integer $n \in \mathbb{N}_0$ is represented by an expression of the form

$$n = \sum_{i=0}^{k} a_i b^i = b \cdot (b \cdot (b \cdot a_k) + \cdots + a_1) + a_0. \tag{2}$$

Here $0 \leq a_i < b$, and arithmetic is performed relative to this representation, with "addition with carry" and long multiplication, using Horner's method in order to efficiently recurse through the structure of the expression. (We postpone the precise description of these (grade school) algorithms to Section 2.)

All Calculation is Addition, Multiplication and Ordering of $\mathbb{N}_0$. One important observation which is helpful to identify is that

Observation 1. All calculations of a numerical nature can be reduced to the three operations $x + y, x \cdot y, x < y$ applied to nonnegative integers.

For instance:

$$\sqrt{2} > 1.414 \quad \text{because} \quad 1414 \cdot 1414 < 2 \cdot 1000 \cdot 1000, \tag{3}$$

$$11 \text{ is prime} \quad \text{because} \quad 11 = 2 \cdot 5 + 1 = 3 \cdot 3 + 2, \tag{4}$$

$$\frac{4^4}{4} < \binom{2 \cdot 4}{4} \quad \text{because} \quad 4 \cdot 4 \cdot 4 \cdot 2 \cdot 3 \cdot 4 < 5 \cdot 6 \cdot 7 \cdot 8. \tag{5}$$

Note that as this is a formal proof the goal is not in *calculating* the result itself, which we already know or can verify externally, but rather in *efficiently verifying the numerical claim*, which comes with its own challenges. For primality testing, this process is known as a primality certificate, and for larger primes we use so-called "Pratt certificates" for prime verification [4]. In general, there may be many non-numerical steps involved to set up a problem into an appropriate form, such as the squaring and multiplication by 1000 in equation 3, or we may want to add such steps as an additional reduction on the equation so that we avoid an unnecessarily complicated calculation, such as canceling the common factor $6 \cdot 4$ in equation 5. However, these "framing" steps are comparatively few so that it is worthwhile to coerce these calculations into the framework described here for even moderately large numbers, say $n > 30$.

It is not really possible to prove Observation 1 as it is not a strictly mathematical statement without additional explanation of what exactly is meant by "calculation" or "numerical nature". Its primary purpose is in justifying the scope of the arithmetic system to be built, and it is justified by the observation that computers can compute the various constants and special functions using only a small fixed instruction set and an arithmetic logic unit (ALU) that supports only these basic operations on bit strings. This observation can also be

viewed as a combination of the Church-Turing thesis and the observation that arithmetic on $\mathbb{Q}$ can be expressed in the language of Peano Arithmetic.

Section 2 describes the metatheory of "decimal numbers" as they appear in `set.mm`, and the theorems which form the basic operations upon which the algorithm is built. Section 3 describes the Mathematica implementation of a limited-domain automated theorem prover (ATP) for arithmetic using the decimal theorems as a base, and Section 4 surveys the outcome of the project.

2 Setting Up "Decimal" Arithmetic Theorems in `set.mm`

The State of the Database Before this Project.

Definition 1. *When describing terms, we will use the notation* $\langle x\rangle$ *to refer to the defined number term with value* x*, assuming* $0 \le x \le 10$*, e.g.* $3\cdot 3$ *is a literal expression "*$3\cdot 3$*" while* $\langle 3\cdot 3\rangle$ *is the term* 9*.*

We take for granted the following facts, already derived in the database:

- We have general theorems for the field operations, so that $a + b = b + a$, $a \cdot b = b \cdot a$, $a + 0 = a$, and $a \cdot 1 = a$. We also have theorems of the form $x = x$; $x = y \vdash y = x$; $x = y, y = z \vdash x = z$ available.
- As we have already mentioned in Section 1, the numbers 0-10 have been defined, yielding definitional theorems of the form $\langle n + 1\rangle = n + 1$ for each $n \in \{1, \ldots, 9\}$, and for each such number n we have the theorems $n \in \mathbb{N}_0$ and $n \in \mathbb{N}$ (except $n = 0$ which only has $0 \in \mathbb{N}_0$).
- We have addition and multiplication facts for these numbers. That is, if $1 \le n \le m \le 10$ and $m+n \le 10$ we have the theorem $m+n = \langle m+n\rangle$, and similarly if $1 < n \le m \le 10$ and $mn \le 10$ then $m\cdot n = \langle mn\rangle$ is a theorem. By combining these facts with the general theorems on field operations we can construct the complete subsection of "addition table" and "multiplication table" expressible using numbers less than 10. (Recall that $6 + 5 = 11$ is not a theorem because even though $6 + 5$ is well-defined, "11" does not name a number and so the statement itself makes no sense unless 11 is first given a definition.)
- We have general inequality theorems like transitivity, and theorems of the form $m < n$ for each $0 \le m < n \le 10$.

This listing motivates the following definition:

Definition 2. *A* basic fact *is a theorem of the form* $x \in \mathbb{N}$*,* $x \in \mathbb{N}_0$*,* $x = y + z$*,* $x = y \cdot z$*, or* $x < y$ *where* x, y, z *are each number terms selected from* $0, \ldots, 10$*, which asserts a true statement about integers* x, y, z*.*

Theorem 2. *There is an integer* N *such that every basic fact is provable in less than* N *steps.*

Proof. Immediate since there are only finitely many basic facts. □

It is not relevant for the analysis how large N is, but by explicitly enumerating such facts one can show that $N \leq 10$, or $N \leq 3$ if closure steps $x \in \mathbb{N}_0, \mathbb{R}, \mathbb{C}$ are ignored.

We begin by defining the concept of "numeral".

Definition 3. *A* numeral *is a term of the language of* `set.mm` *defined recursively by the following rules:*

- *The terms* $0, 1, 2, 3$ *are numerals.*
- *If* n *is a nonzero numeral and* $a \in \{0, 1, 2, 3\}$*, then the term* $(4 \cdot n) + a$ *is a numeral.*

Why Quaternary? It is easily seen that this definition enumerates not base-10 integer representations but rather base-4 representation. It is of course necessary to commit to a base to work in for practical purposes, and the most logical decision for a system such as `set.mm` which prioritizes abstract reasoning over numerical calculation is base 10. The reason this choice was rejected is because the "multiplication table" for base 10 would require many more basic facts like $7 \cdot 8 = 10 \cdot 5 + 6$, while the multiplication table of base 4 requires only numbers as large as 9, which fits inside our available collection of basic facts. Furthermore, within this constraint a large base allows for shorter representations, which directly translates to shorter proofs of closure properties and other algorithms (in addition to increasing readability). The specific choice of $4 = 2^2$ also works well with algorithms like exponentiation by squaring, used in the calculation of $x^k \bmod n$ in primality tests (see Section 4).

Remark 1. It is important to recognize that "n is a numeral" is not a statement of the object language, but rather of the metalanguage describing the actual structure of terms. Furthermore, these are terms for "concrete" integers, not variables over them, and there are even valid terms for nonnegative integers that are not equal to any numeral, such as $\text{if}(\text{CH}, 1, 0)$ (where CH is the Continuum Hypothesis or any other independent statement), which is provably a member of $\mathbb{N}_0$ even though neither $\vdash \text{if}(\text{CH}, 1, 0) = 0$ nor $\vdash \text{if}(\text{CH}, 1, 0) = 1$ are theorems. However, we will show that these pathologies do not occur in the evaluation of multiplication, addition, and ordering on other numerals.

There are three additional methods that are employed to shorten decimal representations, by adding clauses to Definition 3. We will call these "extended numerals".

- We can include the numbers $4, 5, 6, 7, 8, 9, 10$ as numerals. Although it is disruptive to some of the algorithms to allow these in the lower digits (e.g. considering $4 \cdot 3 + 7$ to be a numeral), it is easy to convert such "non-standard" digits in the most significant place, via the (object language) theorems

$$\frac{}{\vdash (4 \cdot 1) + 0 = 4}\ \mathsf{dec4}, \quad \ldots \quad \frac{}{\vdash (4 \cdot 2) + 2 = 10}\ \mathsf{dec10}.$$

- We can drop 0 when it occurs in a numeral; this amounts to adding the rule "if n is a numeral then $4 \cdot n$ is a numeral" to the definition of a numeral. The conversion from this form to the usual form is provided by the theorem

$$\frac{\vdash n \in \mathbb{N}_0}{\vdash 4n + 0 = 4n} \text{ dec0u.}$$

- We can define a function $(x : y) = 4x + y$ and add the rule "if x is a numeral and $y \in \{0, 1, 2, 3\}$ then $(x : y)$ is a numeral". Since multiplication and addition have explicit grouping in `set.mm`, this halves the number of parentheses and improves readability, from $((4 \cdot ((4 \cdot 3) + 2)) + 1)$ to $((3 : 2) : 1)$. The conversion for this form is provided by the theorem

$$\frac{\vdash x \in \mathbb{N}_0 \quad \vdash y \in \mathbb{N}_0}{\vdash 4x + y = (x : y)} \text{ decfv.}$$

In remark 1 we observed that not all nonnegative integers are numerals, but we can show that the converse is true, so that in the above theorems, we can use the antecedent $n \in \mathbb{N}_0$ instead of "n is a numeral" which is not possible since this is not a statement of the object language. This weaker notion is sufficient to assure that n has all the general properties we can expect of numerals, like $n + 0 = n$ or $n \geq 0$, which is what we need for most of the theorems on numerals.

Theorem 3. *If n is a numeral, then $\vdash n \in \mathbb{N}_0$.*

Proof. By induction. If $n \in \{0, 1, 2, 3\}$, then $n \in \mathbb{N}_0$ is a basic fact. Otherwise, $n = 4m + a$ for some numeral m and $a \in \{0, 1, 2, 3\}$, and by induction we can prove $m \in \mathbb{N}_0$ and $a \in \mathbb{N}_0$ is a basic fact. Then the result follows from application of the theorem

$$\frac{\vdash m \in \mathbb{N}_0 \quad \vdash a \in \mathbb{N}_0}{\vdash 4m + a \in \mathbb{N}_0} \text{ decclc.}$$

□

Theorem 4. *If n is a nonzero numeral, then $\vdash n \in \mathbb{N}$.*

Proof. By induction; the theorems involved are

$$\frac{\vdash m \in \mathbb{N}}{\vdash 4m + 0 \in \mathbb{N}} \text{ decnncl2} \qquad \frac{\vdash m \in \mathbb{N}_0 \quad \vdash a \in \mathbb{N}}{\vdash 4m + a \in \mathbb{N}} \text{ decnnclc.}$$

If $n \in \{1, 2, 3\}$, then $n \in \mathbb{N}$ is a basic fact. Otherwise, $n = 4m + a$ for some nonzero numeral m and $a \in \{0, 1, 2, 3\}$, and by induction $m \in \mathbb{N}$. If $a = 0$, then by decnncl2 $n \in \mathbb{N}$; otherwise $a \in \mathbb{N}$ is a basic fact and $n \in \mathbb{N}$ follows from decnnclc. □

Theorem 5. *The numeral representation is unique, in the sense that if $m = n$ as integers, then m and n are identical terms.*

Proof. Follows from the uniqueness of base-4 representation of integers. □

Remark 2. Note that each of the extended representations of numerals invalidate this theorem, e.g. $(4\cdot 2)+0 = 8 = 4\cdot 2 = (2:0)$ would all be distinct representations of the same integer value whose standard form is $(4\cdot 2)+0$. Nevertheless, there is still a weaker form of uniqueness which is preserved. If m, n are extended numerals and $m = n$ as integers, then $\vdash m = n$ is provable; this follows from the respective "conversion" theorems for each extension together with the equality theorems for addition and multiplication ($a = b, c = d \vdash a+b = c+d,\ a\cdot b = c\cdot d$).

The converse of this, $\vdash m = n \implies m = n$, follows from soundness of ZFC from our interpretation of terms in the object logic as integers with the usual operations in the metalogic. This is why we will often not distinguish between equality in the metalogic and the object logic, because they coincide with each other and with identity as terms from Theorem 5.

Theorem 6. *If n is a numeral then $\vdash 4a+b = n$ for some numeral a (not necessarily nonzero) and some $b \in \{0,1,2,3\}$.*

Proof. If $n \in \{0,1,2,3\}$, then we can use the theorem

$$\frac{\vdash a \in \mathbb{N}_0}{\vdash 4\cdot 0 + a = a}\ \mathsf{dec0h},$$

since $n \in \mathbb{N}_0$ is a basic fact. Otherwise, $n = 4a+b$ for some nonzero decimal a and $b \in \{0,1,2,3\}$, and the goal statement is just $\vdash 4a+b = 4a+b$. □

Theorem 7. *If $m < n$ are numerals, then $\vdash m < n$.*

Proof. By induction on m; the relevant theorems are

$$\frac{\vdash a \in \mathbb{N} \quad \vdash b, c \in \mathbb{N}_0 \quad \vdash c < 4}{\vdash c < 4a+b}\ \mathsf{declti},$$

$$\frac{\vdash a, b \in \mathbb{N}_0 \quad \vdash c \in \mathbb{N} \quad \vdash b < c}{\vdash 4a+b < 4a+c}\ \mathsf{declt},$$

$$\frac{\vdash a, b, c, d \in \mathbb{N}_0 \quad \vdash b < 4 \quad \vdash a < c}{\vdash 4a+b < 4c+d}\ \mathsf{decltc}.$$

If $m, n \in \{0,1,2,3\}$, then $m < n$ is a basic fact. Otherwise if $m \in \{0,1,2,3\}$ and $n = 4a+b$, then since a is nonzero $a \in \mathbb{N}$ by Theorem 4, and $m < 4$ is a basic fact, so declti applies to give $\vdash m < n$. If $m = 4a+b$, then $m \geq 4$ so $n = 4c+d$ (because $m < n$ implies $n \notin \{0,1,2,3\}$) for nonzero numerals a, c and $b, d \in \{0,1,2,3\}$. If $a = c$, then by Theorem 5, a and c are identical, so (working in the metalogic) $4a+b < 4a+d \to b < d$, and since $b, d \in \{0,1,2,3\}$ this is a basic fact; thus declt applies and $\vdash m < n$. Again working in the metalogic, if $a > c$ then $4a+b \geq 4a \geq 4c+4 > 4c+d$ in contradiction to the assumption $m < n$, so in the other case $a < c$ and by the induction hypothesis $\vdash a < c$; and $b < 4$ is a basic fact, hence decltc applies. □

As an example of Theorem 7, we know that $\vdash 3 < 4{\cdot}3{+}1$ and $\vdash 4{\cdot}2{+}0 < 4{\cdot}2{+}1$ are theorems of `set.mm` because $3 < 13$ and $8 < 9$, respectively (and the numeral representations of these numbers are $[3] = 3$, $[13] = 4 \cdot 3 + 1$ and so on).

Next we show how to do successors, addition, and multiplication.

Definition 4. *As a variant of Definition 1, we use the notation $[x]$ to denote the unique numeral corresponding to the expression x. Thus for example* $[6{+}5] = (4 \cdot 2) + 3$.

Remark 3. Note that for $0 \le x \le 10$, the statement $[x] = \langle x \rangle$ is either an identity or one of dec4, dec5, ..., dec10, and so is provable in one step.

Theorem 8. *If n is a numeral and $\vdash n = n'$, then $\vdash [n+1] = n' + 1$.*

Proof. By induction on n; the relevant theorems are

$$\frac{\vdash a, b \in \mathbb{N}_0 \quad \vdash c = (b+1) \quad \vdash 4a + b = n}{\vdash 4a + c = n + 1}\ \textsf{decsuc},$$

$$\frac{\vdash a \in \mathbb{N}_0 \quad \vdash b = (a+1) \quad \vdash 4a + 3 = n}{\vdash 4b + 0 = n + 1}\ \textsf{decsucc2}.$$

If $n \in \{0, 1, 2\}$, then $[n+1] \in \{1, 2, 3\}$ and $[n+1] = n+1$ is a basic fact, and $\vdash n + 1 = n' + 1$. Otherwise, by Theorem 6 $n = 4a + b$ for some (not necessarily nonzero) numeral a and $b \in \{0, 1, 2, 3\}$. If $b \in \{0, 1, 2\}$, then $[n{+}1] = 4a{+}[b{+}1]$, and so the assumptions $[b+1] = b+1$, $4a + b = n'$ for decsuc are satisfied. Otherwise $b = 3$ and $[n+1] = 4[a+1] + 0$, and by the induction hypothesis $\vdash [a+1] = a+1$ (using $n, n' \mapsto a$). Then $[a+1] = a+1$ and $4a + 3 = n'$ satisfy the assumptions of decsucc2. □

Remark 4. The extra assumption $\vdash n = n'$ is not necessary (i.e. we could just have proven $\vdash [n{+}1] = n{+}1$) but makes it a little easier to work with extended numerals, because that way n can be the standard numeral while n' is the extended numeral, and the assumption is satisfied by remark 2.

To give an example, if we were able to prove (using other theorems than discussed here) that $\vdash 4 \cdot 1 + 1 = 6 - 1$, then Theorem 8 says, taking $n = [5] = 4 \cdot 1 + 1$ and $n' = 6 - 1$, that $\vdash 4 \cdot 1 + 2 = (6 - 1) + 1$ is also provable.

Theorem 9. *If m, n are numerals such that $\vdash m = m'$ and $\vdash n = n'$, then* $\vdash [m + n] = m' + n'$.

Proof. By induction on $m + n$; the relevant theorems are

$$\frac{\begin{array}{c}\vdash a, b, c, d \in \mathbb{N}_0 \quad \vdash 4a + b = m \quad \vdash 4c + d = n \\ \vdash e = a + c \quad \vdash f = b + d\end{array}}{\vdash 4e + f = m + n}\ \textsf{decadd},$$

$$\frac{\begin{array}{c}\vdash a, b, c, d, f \in \mathbb{N}_0 \quad \vdash 4a + b = m \quad \vdash 4c + d = n \\ \vdash e = (a + c) + 1 \quad \vdash 4 + f = b + d\end{array}}{\vdash 4e + f = m + n}\ \textsf{decaddc}.$$

If $m+n \in \{0,1,2,3\}$, then $[m+n] = m+n$ is a basic fact so $\vdash [m+n] = m'+n'$ follows from properties of equality. By Theorem 6 we can promote m and n to the form $m = 4a+b$, $n = 4c+d$ where $b, d < 4$ and a, c are numerals. Now if $b+d < 4$, then $\vdash b + d < 4$ follows from the basic facts $[b+d] = b+d$ and $[b+d] < 4$, and so filling the assumptions with $4a+b = m'$, $4c+d = n'$, and $[b+d] = b+d$ and using the induction hypothesis to prove $[a+c] = a+c$, we can apply decadd. Otherwise, $b+d \geq 4$, so $[b+d-4] \in \{0,1,2,3\}$ and we can use $4a+b = m'$, $4c+d = n'$, prove $4+[b+d-4] = b+d$ using the basic facts $\langle b+d\rangle = 4+[b+d-4]$ and $\langle b+d\rangle = b+d$, and prove $[a+c+1] = [a+c]+1 = (a+c)+1$ using first Theorem 8 and then the induction hypothesis; this completes the assumptions of decaddc. □

The next theorem works by double induction, so it is easier to split it into two parts.

Theorem 10. *If m, n are numerals, $p \in \{0,1,2,3\}$, and $\vdash m = m', n = n'$, then $[mp+n] = m'p+n'$.*

Proof. By induction on m, using the theorem

$$\frac{\vdash a,b,c,d,p,f,g \in \mathbb{N}_0 \quad \vdash 4a+b = m \quad \vdash 4c+d = n \qquad \vdash e = ap+(c+g) \quad \vdash 4g+f = bp+d}{\vdash 4e+f = mp+n}\ \text{decmac}.$$

If $m \in \{0,1,2,3\}$, then $[mp] = \langle mp\rangle$ is provable by remark 3 and $\langle mp\rangle = mp$ is a basic fact, so $\vdash [mp+n] = [mp]+n = m'p+n'$ by Theorem 9. Otherwise let $m = 4a+b$, and write $n = 4c+d$ and $[bp+d] = 4g+f$ for some c, d, f, g by Theorem 6. Then the assumptions to decmac are satisfied by $4a+b = m'$, $4c+d = n'$, $[ap+c+g] = ap+[c+g] = ap+(c+g)$ by the induction hypothesis and Theorem 9, and $4g+f = [bp+d] = [bp]+d = bp+d$ by Theorem 9. □

Theorem 11. *If m, n, p are numerals, and $\vdash m = m', n = n', p = p'$, then $[mp+n] = m'p'+n'$.*

Proof. By induction on p, using the theorem

$$\frac{\vdash a,b,c,d,m,f,g \in \mathbb{N}_0 \quad \vdash 4a+b = p \quad \vdash 4c+d = n \qquad \vdash e = ma+(c+g) \quad \vdash 4g+f = mb+d}{\vdash 4e+f = mp+n}\ \text{decma2c}.$$

If $p \in \{0,1,2,3\}$, then $[mp+n] = m'p+n' = m'p'+n'$ by Theorem 10. Otherwise let $m = 4a+b$, and write $n = 4c+d$ and $[mb+d] = 4g+f$ for some c, d, f, g by Theorem 6. Then the assumptions to decma2c are satisfied by $4a+b = p'$, $4c+d = n'$, $[ma+c+g] = ma+[c+g] = ma+(c+g)$ by the induction hypothesis and Theorem 9, and $4g+f = [mb+d] = mb+d$ by Theorem 10. □

Corollary 1. *If m, n are numerals, and $\vdash m = m', n = n'$, then $[mn] = m'n'$.*

Proof. Theorem 11 gives $[mn] = m'n'+0$, and $m'n'+0 = m'n'$ because $m'n' = mn \in \mathbb{N}_0$. (Slightly more efficient than this approach is an application of the theorems decmul1c, decmul2c which are essentially the same as decmac, decma2c without the addition component.) □

3 A Mathematica Implementation of the Decimal Arithmetic Algorithm

The purpose of the preceding section was not merely to prove that arithmetic operations are possible, which could be done just as easily using finite sums or unary representation. Rather, by proving the results contructively it is in effect a description of an algorithm for performing arithmetic calculations, and as such it is not difficult to implement on a computer. Due to its advanced pattern-matching capabilities, Mathematica was selected as the language of choice for the implementation.

In order to avoid Mathematica's automatic reduction of arithmetic expressions, we represent the Metamath formulas of our limited domain via the following correspondence:

- An integer n is represented as itself
- The term $a + b$ becomes `pl`$[a, b]$
- The term $a \cdot b$ becomes `tm`$[a, b]$
- $\vdash a = b$ becomes `eq`$[a, b]$
- $\vdash a < b$ becomes `lt`$[a, b]$
- $\vdash a \in \mathbb{N}$ becomes `elN`$[a]$
- $\vdash a \in \mathbb{N}_0$ becomes `elN0`$[a]$
- $\vdash a \in \mathbb{C}$ becomes `elC`$[a]$

These symbols have no evaluation semantics and so simply serve to store the shape of the target expression.

The output proof is stored as a tree of list expressions by the following correspondence:

> If the expression e is obtained by applying the theorem t to the list of expressions $e_1, \dots, e_k$ with proofs $p_1, \dots, p_k$, then the proof of e is stored as the expression $p = \{\{p_1, \dots, p_k\}, e, t\}$.

For example, the expression $4 \cdot (4 \cdot 1 + 3) + 2 = 5 \cdot 6$ is represented as

```
eq[pl[tm[4,pl[tm[4,1],3]],2],tm[5,6]]
```

and the proof of $2 \cdot (4 \cdot 1 + 1) \in \mathbb{N}_0$ (by the sequence of theorems: $2 \in \mathbb{N}_0$ by 2nn0, $1 \in \mathbb{N}_0$ by 1nn0, $4 \cdot 1 + 1 \in \mathbb{N}_0$ by decclc, $2 \cdot (4 \cdot 1 + 1) \in \mathbb{N}_0$ by nn0mulcli) is:

```
{{{{}, elN0[2], "2nn0"},
  {{{{}, elN0[1], "1nn0"},
    {{}, elN0[1], "1nn0"}},
   elN0[pl[tm[4,1],1]], "decclc"}},
 elN0[tm[2,pl[tm[4,1],1]]], "nn0mulcli"}
```

(There are more efficient storage mechanisms, but this one is relatively easy to take apart and reorganize. Furthermore, since it only needs to run once in

order to produce the proof, we are much more concerned with the length of the output proof than the speed of the proof generation itself.) For intermediate steps, we will also have use for the "proof stubs" {`Null`,e,`"?"`} (representing a proof with goal expression e that has not been completed) and {`$Failed`,e,`"?"`} (for a step that is impossible to prove or lies outside the domain of the prover).

Given a term expression, we can evaluate it easily using a pattern-matching function:

```
eval[pl[a_, b_]] := eval[a] + eval[b]
eval[tm[a_, b_]] := eval[a] eval[b]
eval[n_Integer] := n
```

Then the domain of our theorem prover will be expressions of the form `eq[`m`,`n`]` where m and n are terms built from the integers 0-10 and `pl`, `tm` which satisfy `eval[`m`] == eval[`n`]`. (As a side effect we will also support `elN0[`n`]`, and `elN[`n`]` when `eval[`n`] != 0`.) We will also need the reverse conversion:

```
bb[n_] := If[n < 4, n, pl[tm[4, bb[Quotient[n, 4]]], Mod[n, 4]]]
```

This is the equivalent of the $[x]$ function from Definition 4.

Our algorithm works in reverse from a given goal expression, breaking it down into smaller pieces until we reach the basic facts. The easiest type of proof is closure in $\mathbb{N}_0$:

```
prove[elN0[n_Integer]] :=
 With[{s = ToString[n]},
  If[n <= 4, {{}, elN0[n], s <> "nn0"}, {{prove[elN[n]]}, elN0[n],
    "nnnn0i"}]]
prove[x : elN0[pl[tm[4, a_], b_]]] := {{prove@elN0[a], prove@elN0[b]},
   x, "decclc"}
prove[x : elN0[tm[a_, b_]]] := {{prove@elN0[a], prove@elN0[b]}, x,
  "nn0mulcli"}
prove[x : elN0[pl[a_, b_]]] := {{prove@elN0[a], prove@elN0[b]}, x,
  "nn0addcli"}
prove[elN[n_Integer]] /; n > 0 := {{}, elN[n], ToString[n] <> "nn"}
prove[elN[x : pl[y : tm[4, a_], b_Integer]]] :=
 If[b === 0, {{{prove@elN0[a], eq[x, y], "dec0u"}, prove@elN[y]},
   elN[x], "eqeltri"}, {{prove@elN0[a], prove@elN[b]}, elN[x],
   "decnnclc"}]
prove[elN[x : tm[4, a_]]] := {{prove@elN[a]}, elN[x], "decnncl"}
```

This simply breaks up an expression according to its head and applies decclc to numerals, nn0addcli and nn0mulcli to integer addition and multiplication, and theorems 0nn0, 1nn0, ..., 4nn0, 5nn, ..., 10nn to the integers 0-10 (where we switch to $\mathbb{N}$ closure for numbers larger than 4 because we do not have $\mathbb{N}_0$ closure theorems prepared for these). Similarly, we can do closure for $\mathbb{N}$:

```
prove[elN[n_Integer]] /; n > 0 := {{}, elN[n], ToString[n] <> "nn"}
prove[elN[x : pl[y : tm[4, a_], b_Integer]]] :=
 If[b === 0, {{prove@elN[a]}, elN[x],
   "decnncl2"}, {{prove@elN0[a], prove@elN[b]}, elN[x], "decnnclc"}]
prove[elN[x : tm[4, a_]]] := {{prove@elN[a]}, elN[x], "decnncl"}
```

This is just an implementation of Theorem 4.

In order to do arbitrary equalities, we first ensure that both arguments are numerals, by chaining $x = [x] < [y] = y$ for inequalities and $x = [x] = y$ for equalities (where $[x]$ and $[y]$ are identical since we are assuming that the equality we are proving is in fact correct). We also allow the case "$x < 4$" where x is a numeral even though 4 is not a numeral, because it comes up often and we already have theorems for this case.

```
prove[lt[x_, y_]] /; eval[x] < eval[y] :=
 If[x === bb@eval[x],
  If[y === bb@eval[y] || y === 4,
   provelt[x, y], {{prove@lt[x, bb@eval[y]], proveeq2[y]}, lt[x, y],
    "breqtri"}], {{proveeq2[x], prove@lt[bb@eval[x], y]}, lt[x, y],
   "eqbrtrri"}]
prove[eq[x_, y_]] /; eval[x] == eval[y] :=
 If[x === bb@eval[x],
  proveeq2[y], {{proveeq2[x], proveeq2[y]}, eq[x, y], "eqtr3i"}]
proveeq2[x_] := proveeq[bb@eval[x], x]
```

It remains to define `provelt` and `proveeq`. We start with `provelt`, implementing Theorem 7.

```
provelt[x_Integer, y_Integer] := {{}, lt[x, y],
  ToString[x] <> "lt" <> ToString[y]}
provelt[x_Integer,
  y : pl[tm[4, a_], b_]] := {{prove@elN[a], prove@elN0[b],
   prove@elN0[x], provelt[x, 4]}, lt[x, y], "declti"}
provelt[x : pl[tm[4, a_], c_], y : pl[tm[4, b_], d_]] :=
 If[a === b, {{prove@elN0[a], prove@elN0[c], prove@elN[d],
    provelt[c, d]}, lt[x, y], "declt"},
  {{prove@elN0[a], prove@elN0[b], prove@elN0[c],
    prove@elN0[d], provelt[c, 4], provelt[a, b]}, lt[x, y], "decltc"}]
```

The contract of `proveeq`$[x, y]$ is such that it returns a proof of $\vdash x = y$ given an expression y, assuming $x = [y]$ (since we can calculate x from y, the left argument is not necessary, as in the variant `proveeq2`, but it simplifies pattern matching), and it is defined much the same as previous functions; we elide it here due to space constraints.

Basic facts. One fine point which may need addressing, since it was largely glossed over in Theorem 2, is the algorithm for basic facts. We define a function

`basiceq[x_]` which is valid when `x` is either `pl`$[m,n]$ or `tm`$[m,n]$ and `eval[x]`$\leq$ 10; in this case it corresponds to a "basic fact" of either addition or multiplication, and the return value is a proof of `eq[eval[x], x]`. There are many special cases, but every such statement follows from addid1i, addid2i (addition with 0), mulid1i, mulid2i (multiplication by 1), mul01i, mul02i (multiplication by 0), df-2, ..., df-10 (addition with 1), or a named theorem like 3p2e6 for $3 \cdot 2 = 6$ – these exist for each valid triple containing numbers larger than 1 – possibly followed with addcomi, mulcomi (commutation) and/or eqcomi, eqtri (symmetry/transitivity of equality) to tie the components together.

4 Results

In this section we will describe the purpose to which we applied the arithmetic algorithm.

4.1 Prime Numbers

There are several ways to prove that a number is prime, and the relative efficiency can depend a lot on the size of the numbers involved. For small numbers, especially if it is necessary to find all primes below a cutoff, the most efficient method is simple trial division. The biggest improvement in efficiency here is gained by looking only at primes less than $\sqrt{n}$ in a proof that n is prime. Starting from the two primes $2, 3$ which are proven "from first principles", we can use this to show that a number less than 25 which is not divisible by 2 or 3 is prime, and we reduce these primality/compositeness deductions to integer statements via

$$\frac{\vdash q \in \mathbb{N}_0 \quad \vdash a, r \in \mathbb{N} \quad \vdash b = aq + r \quad \vdash r < a}{\vdash a \nmid b} \text{ ndvdsi}$$

$$\frac{\vdash a, b \in \mathbb{N} \quad \vdash 1 < a \quad \vdash 1 < b \quad \vdash n = ab}{\vdash n \notin \mathbb{P}} \text{ nprmi.}$$

As a consequence of our choice of base 4, we also have easy proofs of compositeness or non-divisibility by 2:

$$\frac{\vdash a \in \mathbb{N}_0}{\vdash 2 \nmid 4a + 1} \text{ dec2dvds1} \qquad \frac{\vdash a \in \mathbb{N}_0}{\vdash 2 \nmid 4a + 3} \text{ dec2dvds3}$$

$$\frac{\vdash a \in \mathbb{N}}{\vdash 4a + 2 \notin \mathbb{P}} \text{ dec2nprm}$$

This yields proofs of $5, 7, 11, 13, 17, 19, 23 \in \mathbb{P}$. We repeat the process to show that a number less than $29^2 = 841$ and which is not divisible by $2, 3, 5, 7, 11, 13, 17, 19, 23$ is prime (the upper bound is 29^2 because 29 is the first prime larger than 23), and we use this theorem to prove primality of $37, 43, 83, 139, 163, 317, 631$. There are three more primes needed for the prime sequence in Bertrand's postulate, namely $1259, 2503, 4001$, and we would need many more primes to repeat the process with a still-larger upper bound, so we switch methods.

Pocklington's Theorem.

Theorem 12. *If $N > 1$ is an integer such that $N - 1 = AB$ with $A > B$, and for every prime factor p of A there is an a_p such that $a_p^{N-1} \equiv 1 \pmod N$ and $\gcd(a_p^{(N-1)/p} - 1, N) = 1$, then N is prime.*

This theorem has been proven in `set.mm` in decimal-friendly form (in the case when $A = p^e$ is a prime power) as

$$\frac{\vdash p \in \mathbb{P} \quad \vdash g, B, e, a \in \mathbb{N} \quad \vdash m = gp \quad \vdash N = m + 1 \quad \vdash m = Bp^e \quad \vdash B < p^e \quad \vdash a^m \equiv 1 \pmod N \quad \vdash \gcd(a^g - 1, N) = 1}{\vdash N \in \mathbb{P}} \ \textsf{pockthi.}$$

Ignoring the p^e term which is easily evaluated by writing it out as a product of p's since $p^e < N$ is relatively small, there are two new kinds of integer statements involved here, $a \equiv b \pmod N$ and $\gcd(c, d) = e$. We can further narrow our concern to statements of the form $a^m \equiv b \pmod N$ and $\gcd(c, d) = 1$ (where $c \geq d$), and we can reduce the second via Euclid's algorithm in the form

$$\frac{\vdash k, r, n \in \mathbb{N}_0 \quad \vdash m = kn + r \quad \vdash \gcd(n, r) = g}{\vdash \gcd(m, n) = g} \ \textsf{gcdi.}$$

For the power mod calculations, we used addition-chain exponentiation on manually selected chains which had good calculational properties (small intermediate calculations), since the hardest step in the calculation

$$\frac{\vdash e = b + c \quad \vdash dn + m = kl \quad \vdash a^b \equiv k,\ a^c \equiv l \pmod n}{\vdash a^e \equiv m \pmod n} \ \textsf{modxai}^2$$

is evaluating the expression $dn + m = kl$, whose proof length is driven by the size of k, l, so that exponents with smaller reduced forms yield shorter proofs. In the worst case, for $N = 4001$ we are calculating products on the order $kl \leq N^2 = 16\,008\,001$, which are roughly 12-digit numbers in base 4.

4.2 Bertrand's Postulate Gets the Last Laugh

As mentioned in Section 1.1, the framework described in this paper was developed in preparation for performing the large calculations needed to prove that numbers like 4001 are prime. After this work was completed, we discovered that there was a new proof by Shigenori Tochiori [5] (unfortunately untranslated to my knowledge) which, by strengthening the estimates in Erdős' proof, manages to prove the asymptotic part for $n \geq 64$ (instead of $n \geq 4000$), so that the explicit enumeration of primes above this became unnecessary for the completion of the proof. Nevertheless, the proof of $4001 \in \mathbb{P}$ remains as a good example of a complicated arithmetic proof, and we expect that this arithmetic system will make it much easier to handle such problems in the future, and bpos is now completed in any case.

[2] closure assumptions elided

4.3 Large Proofs

One recent formal proof which has gathered some attention is Thomas Hales' proof of the Kepler Conjecture, also known as the Flyspeck project [6], and it highlights one foundational issue regarding Metamath's prospects in the QED vision of the future [7] – *which* and *how much* resources are stressed by projects like this with a large computational component? The design of Metamath is such that it can verify a proof in nearly linear time, assuming that it can store the entire database of theorem statements in memory, because at each step it need only verify that the substitutions to the theorem statement are in fact done correctly (and the substitutions themselves are stored as part of the proof, even though they can be automatically derived with more sophisticated and slower algorithms). Thus the primary bottleneck in verification time is the length of the proof itself, and we can analyze this quite easily for our chosen algorithm.

Since we are essentially employing grade-school addition and multiplication algorithms, it is easy to see that they are $O(n)$ and $O(n^2)$ respectively, and with more advanced multiplication algorithms we could lower that to $O(n^{1.5})$ or lower. Indeed, there does not seem to be any essential difference between the number of steps in a Metamath proof and the number of cycles that a computer might go through to perform the equivalent algorithm, even though a Metamath proof doesn't "run" per se as it is a proof and not a program. (In fact a Metamath proof has at least one big advantage over a computer in that it can "guess the right answer" in the manner of a non-deterministic Turing machine.) To take this example to its conceptual extreme, we could even simulate an ALU with addition and multiplication of integers representing data values of a computer, and then the progress of the proof would directly correspond to the steps in a computer program. Of course this would introduce a ridiculously large constant, but it would suggest that any program, including a verifier for another formal system such as the HOL Light system in which Flyspeck runs, can be emulated with a proof whose length is comparable to the running time of the verifier without a change in the overall asymptotics.

Acknowledgments. The author wishes to thank Norman Megill for discussions leading to the developments of Section 2, N. Megill and Stefan O'Rear for reviewing early drafts of this work, and the many online Japanese language learning resources that assisted in translating and eventually formalizing [5].

References

1. Megill, N.: Metamath: A Computer Language for Pure Mathematics. Lulu Publishing, Morrisville, North Carolina (2007)
2. Metamath Proof Explorer, `http://us.metamath.org/mpegif/mmset.html`
3. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a Nutshell. J. of Formalized Reasoning Vol. 3, No. 2, pp. 153–245 (2010)
4. Pratt, V.: Every Prime Has a Succinct Certificate. SIAM J. Comput. 4, 214–220 (1975)

5. Tochiori S.: Considering the Proof of "There is a Prime between n and 2n": Proof by a stronger estimation than the Bertrand-Chebyshev theorem (in Japanese). Accessed from http://www.chart.co.jp/subject/sugaku/suken_tsushin/76/76-8.pdf.
6. Hales, T.: Dense Sphere Packings: A Blueprint for Formal Proofs. Cambridge University Press, Cambridge (2012)
7. The QED Manifesto. In Automated Deduction – CADE 12, Springer-Verlag, Lecture Notes in Artificial Intelligence, Vol. 814, pp. 238–251 (1994)

Auto-hyperlinking the Stacks Project

Johan Commelin[1] and Josef Urban[1]

Radboud University Nijmegen

Abstract. This paper describes an effort to automatically hyperlink mathematical terms to their definitions in the Stacks Project.
The Stacks Project is an open source, collaborative textbook on Algebraic Geometry. It covers the definition of an algebraic stack (unrelated to the notion of stack in Computer Science), and beyond; providing a technical reference of the state of the art for researchers in the field. It provides a web interface and a stand-alone PDF (currently over 4600 pages).
Throughout the project there is a thorough linking to earlier results used in proofs. However, as is customary in mathematics, nouns and symbols are not hyperlinked to their definitions. In this paper we outline our initial approach to auto-hyperlinking terminology to their definitions, and our future plans.

1 Introduction: Stacks Project

The Stacks Project[1] was initiated in 2005 by Aise Johan de Jong with the aim of collaboratively writing an online introductory text on algebraic stacks (a rather advanced topic in Algebraic Geometry). Over time it has evolved into a textbook covering the foundations of Algebraic Geometry, and serves as an online reference for algebraic geometers. The text is written in a very restricted subset of LaTeX, with a minimal amount of packages and custom commands. This allows custom translation of the text to HTML implemented by the project's PHP code, and relying on MathJaX for math-mode rendering. At the moment the compiled PDF consists of more than 4600 pages.

A crucial element of the Stacks Project is its web interface, and the concept of *tags*. Every result (theorem, lemma, equation, etc, but also chapters and sections) has a LaTeX-label, used for internal references. Besides that, each such label is assigned a *tag*, a 4-character alphanumerical string. The tag is stable, and the web interface provides an easy method to look up the mathematical statement associated with a tag. This also solves the problem of referencing results in the Stacks Project, since the tag provides a permanent URI for the mathematical statement. It should be understood that the content of a tag will not change (up to corrections of minor mathematical mistakes, typographical errors, and clarifications/expansions of proofs).

Part of the Stacks Project now also consists of several pieces of software (mostly written by Aise Johan de Jong and Pieter Belmans) covering amongst more:

[1] `http://stacks.math.columbia.edu/`

– tools to assign new tags to new results;
– scripts that parse the LaTeX source files, generating chunks of source for each tag, and converting the LaTeX to HTML (+MathJaX);
– an API that can be used to request the LaTeX source for a tag;
– dependency graphs for each tag (recursively showing which results are used in the proof).

The authors of the Stacks Project have initially decided (as is customary in pen-and-paper mathematics) that symbols in the text are not hyperlinked to their definitions.

While expert mathematicians usually understand texts written by other expert mathematicians, nonintrusive (e.g., Wikipedia-style) hyperlinking of terminology can be useful for non-experts and students. It is also a prerequisite and one of the first steps needed for making such texts – at least partially, and possibly with human assistance – computer-understandable and verifiable by formal proof assistants. Below we outline our initial approach to auto-hyperlinking symbols to their definitions. It consists of (i) heuristically collecting defined terms from the texts (Section 2) and (ii) heuristically linking symbols in an arbitrary Stacks text to their estimated definitions (Section 3).

2 Collecting definition data

The Stacks Project website employs an SQLite database to store all the information about tags that it needs. We queried this database for a list of all tags that correspond to a LaTeX `definition` environment. We then parsed the source of these tags for strings of the form `{\it foobar}`, to generate a list of all defined terminology, together with the tag where they were defined. The list consists of 2238 items, and we can already make some observations about it:

1. Certain tags define multiple terms. This is not a problem for our purposes.
2. Certain terms have definitions in multiple tags. For example, the term *flat* is defined in the following tags:

   `00HB 01U3 0251 0253 02N3 03ER 03ML 04JB 05ND 06PW.`

   Tag `00HB` defines what *flat* module over a ring is, and when a morphism of rings is *flat*. Next, `01U3` defines when a sheaf of modules is flat at some point, and when a morphism of schemes is *flat*. This list continues, and finally `06PW` defines when a morphism of algebraic stacks is *flat*.
   For (expert) mathematicians it is usually clear from the context which definition is meant. Of course for our purpose this poses a challenge, because we somehow have to take the context into account.
3. Certain terms occur as substrings of other terms. In most cases this can be solved by greedily choosing the longest matching term. Sometimes we may also need to take context into account, as in the previous point.

3 Auto-linking

While ultimately we are interested in trying as sophisticated context-based algorithms as possible,[2] initially we have tried to make the whole website work with two simple methods. A particular problem with using e.g. machine-learning approaches is that we do not have annotated training data, as for example in the Wikifier [8] project,[3] where disambiguation can be learned on the large amount of manually hyperlinked concepts, or in our related work on parsing informalized large formal corpora [7,6] with the help of strong large-theory automated reasoning "hammers" [5,1].

The first method goes through all the defined terms (series of words) sequentially from the longest to the shortest, and in the target text it globally rewrites matched strings to special unique markers that cannot be matched by any other defined term. This way, the longest matched concepts are greedily removed, avoiding clashes in the form of possible further matches of their substrings. For example, once *flat module* has been matched at a certain position, neither of its constituent words can be matched. This is a simple longest-first greedy heuristic, which could likely be extended to Knuth-Morris-Pratt-like algorithm ensuring maximal cover by longest possible strings, using e.g. heuristics trading the average length of the matched strings for the total matched ratio of the whole text. While the efficiency of doing sequential scan with all the defined terms in SP seemed far from optimal, in practice the speed of linking turned out not to be an issue on our hardware and happens in real time.

Having the first method running allowed us to see its main deficiencies by randomly browsing dozens of the auto-linked pages. One frequent and easily removable deficiency is linking to future. The Stacks tags have a chronological ordering (as common in textbooks). Only very rarely do mathematicians allow use of concepts that have not been introduced yet, and SP explicitly forbids this. Hence our second method: when rendering a particular tag, we still go through all the defined concepts from the longest, however we only allow replacement of the matched term if it has been defined in a tag that precedes the currently rendered tag. Again, the information about the chronological ordering of the tags can be easily extracted from the SQLite database of tags. A side-by-side comparison of the second version running on our server[4] with the unmodified (slightly later) version from the Stacks website is shown on Figure 1.

4 Evaluation Methods

As mentioned above, we do not have any "ground truth" data for evaluating the quality of the autolinking and comparing different methods. Instead, we use or plan to use the following methods for evaluation:

[2] See, e.g., [2,3,4] for the decade of work on the much older PlanetMath corpus, which is however quite different in terms of the technology used, focus, and coverage of advanced topics.

[3] http://cogcomp.cs.illinois.edu/page/demo_view/Wikifier

[4] http://mws.cs.ru.nl:8008

– Random browsing through several topics, possibly using side-by-side comparison of the same text rendered with different methods. To make this easier, we use a copy-on-write filesystem (BTRFS[5]) to minimize the overhead of several simultaneous differently modified installations (each over 2GB big) of Stacks, and we optionally use javascript code that immediately previews on mouse-over the linked pages.
– We have written scripts that go through all the tags using a tracing version of the autolinking methods, resulting in a large file with the statistics of how often a particular disambiguation was used in each tag. We then compare the traces for different versions of the autolinking methods. Full tracing for all the 12500 tags takes about 30 minutes. Table 1 compares the most frequent disambiguations for the two autolinking methods explained above.
– We are working on an evaluation interface that will present readers (mathematicians, students, or just us) for each autolinked item on a page with a selection window, allowing to choose the correct disambiguation from all the options. Such choices will be stored on the server, eventually generating the ground-truth data against which we will be comparing and training the algorithms.

No position filtering			Position filtering		
count	term	tag	count	term	tag
14522	morphism	03UM	13184	finite	09G3
8941	scheme	01IJ	8984	scheme	01IJ
7793	finite	09G3	7727	morphism	03UM
7104	open	06U2	7340	algebraic	09GC
6850	algebraic	09GC	5910	category	0014
6794	flat	06PW	5225	functor	003N
6135	surjective	04ZS	4724	isomorphism	0017
5910	category	0014	3910	quasi-compact	090H
5806	affine	03WF	3396	finite type	01T1
5323	functor	003N	3225	field	09FD

Table 1: Initial statistics of the two linking methods.

5 Future Work

Even the simplest auto-linking methods described above seem to be already useful, but there is a wealth of research we can draw on. The obvious extensions include use of machine learning on the collected data, use of the(bag-of-words) context for disambiguation, stemming of the words and their permuting, detecting typing information for supplying more advanced context, etc. Since the proof

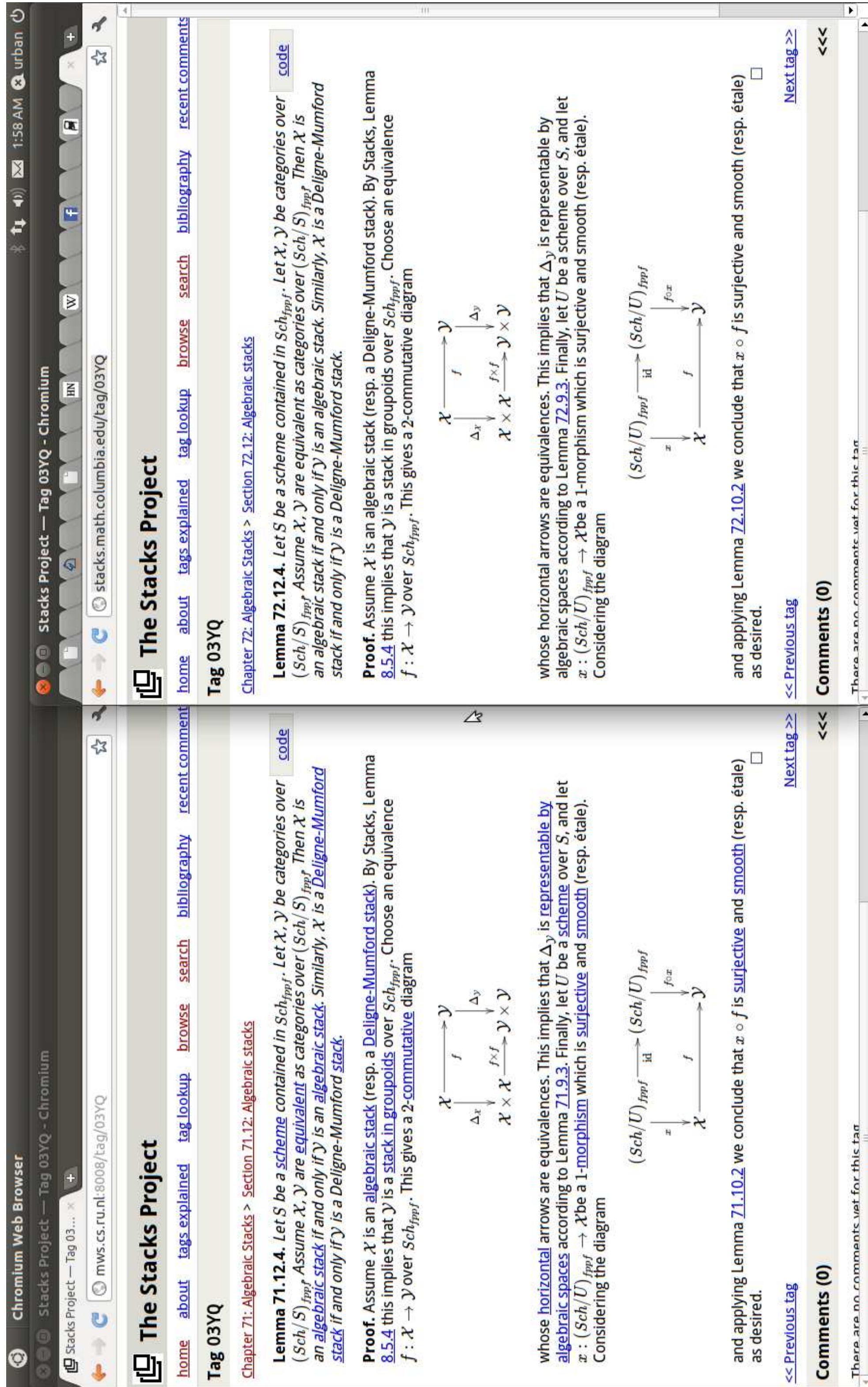
[5] `https://btrfs.wiki.kernel.org`

style is quite uniform, a distant dream is to eventually try creation of formally correct formulas and proof sketches by semi-automated methods, and attempting their discharging with strong large-theory automated reasoning tools.

References

1. J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. Accepted to Journal of Formalized Reasoning, preprint at `http://www4.in.tum.de/~blanchet/h4qed.pdf`, 2015.
2. J. J. Gardner, A. Krowne, and L. Xiong. NNexus: Towards an automatic linker for a massively-distributed collaborative corpus. In E. Blanzieri and T. Zhang, editors, *2nd International ICST Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2006, Atlanta, GA, USA, November 17-20, 2006*. IEEE Computer Society / ICST, 2006.
3. J. J. Gardner, A. Krowne, and L. Xiong. NNexus: An automatic linker for collaborative web-based corpora. *IEEE Trans. Knowl. Data Eng.*, 21(6):829–839, 2009.
4. D. Ginev and J. Corneli. NNexus reloaded. In Watt et al. [9], pages 423–426.
5. C. Kaliszyk and J. Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.
6. C. Kaliszyk, J. Urban, and J. Vyskocil. Learning To Parse on Aligned Corpora (Rough Diamond). Accepted for publication in ITP'15, preprint at `http://mws.cs.ru.nl/~urban/itp15/paper1-final.pdf`, 2015.
7. C. Kaliszyk, J. Urban, J. Vyskocil, and H. Geuvers. Developing corpus-based translation methods between informal and formal mathematics: Project description. In Watt et al. [9], pages 435–439.
8. L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to Wikipedia. In D. Lin, Y. Matsumoto, and R. Mihalcea, editors, *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 1375–1384. The Association for Computer Linguistics, 2011.
9. S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors. *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *Lecture Notes in Computer Science*. Springer, 2014.

Fig. 1: Auto-linked and original page for Tag 03YQ



The SMGloM Project and System

Deyan Ginev[1], Mihnea Iancu[1], Constantin Jucovshi[1], Andrea Kohlhase[1], Michael Kohlhase[1], Jürgen Schefter[2], and Wolfram Sperber[2]

[1] Computer Science, Jacobs University Bremen; http://kwarc.info
[2] Zentralblatt Math, Berlin; http://zbmath.org

Abstract. Mathematical vernacular – the everyday language we use to communicate about mathematics is characterized by a special vocabulary. If we want to support humans with mathematical documents, we need a resource that captures the terminological, linguistic, and ontological aspects of the mathematical vocabulary. In the SMGloM project and system, we aim to do just this. We present the glossary system prototype, the content organization, and the envisioned community aspects.

1 Introduction

One of the challenging aspects of mathematical language is its special terminology of technical terms that are defined in various mathematical documents. To alleviate this, mathematicians use special glossaries, traditionally lists of terms in a particular domain of knowledge with the definitions for those terms. Originally, glossaries appeared as alphabetical lists of new/introduced terms with short definitions in the back of books to help readers understand the contents. Another kind of resource that deals with terminology of mathematics are "dictionaries", which align mathematical terms in different languages by their meaning – originally without giving a definition.

In the last decades the term "glossary" has also been applied to digital vocabularies (online encyclopedias, thesauri, dictionaries, etc.), which have become important resources in knowledge-based systems. This is especially true for vocabularies that have a *i*) semantic aspect – i.e. some of the relations are made explicit and machine-actionable, they are also called "ontologies" – or *ii*) that are multilingual. Digital vocabularies can be hand-curated, or machine-generated/collected; an example of the former is the WordNet lexical database for English,an example of the latter is DBPedia,but they can also be hybrid, e.g. the UWN/Menta projectgenerates a multilingual WordNet by automatically adding other languages by crawling Wikipedia.

We present the SMGloM project, which aims to create a semantic, multilingual glossary for mathematics. This resource combines the characteristics of dictionaries and glossaries, with those of ontologies, but restricts the content to definitions and the relations to the lexical ones to keep the task manageable. Here we give a high-level overview over the data model, the SMGloM system, organizational and legal issues, possible applications, and the state of the effort of seeding the glossary.

2 The SMGloM System

Data Model and Encoding We build the data model of SMGloM on top of the one of OMDoc/Mmt, which provides views, statements, and theories. In a nutshell – see [Koh14] for details, a **glossary entry** consists of one **symbol**, its **definition**, and a set of **verbalizations** and **notations**. A symbol is a formal identifier of a mathematical object/concept (i.e a formal object). The verbalizations relate it to lexical entries (identified by the stem of the head), which we call **glossary terms**.

The definitions could be written down in a formal logic, but in the SMGloM, we write them down in mathematical vernacular (common mathematical language; in SMGloM natural language with sTeX annotations). Thus we consider "the definition" of a symbol to be given by a set of vernacular definitions, which are assumed to be translations of each other – an important structural invariant of the SMGloM that needs to be maintained.

Glossary entries are often grouped into a **glossary module**, which is represented as $n + 1$ OMDoc/Mmt theories: one for the language-independent part (called the **module signature**, it introduces the symbols, their dependencies, and notations), and n for the **language bindings** (which introduce the definitions and verbalizations of symbols).

Organizing a Communal Resource The ultimate cause of the SMGloM project and system is to facilitate the establishment of a knowledge resource for mathematics. We need to take appropriate organizational measures to support this. We are currently establishing a wiki-like archive submission system for glossary modules on MathHub [MH] and thinking of a quality assurance system that is based on a community/karma-driven approval system. Openness and semantic stability are ensured by a special licensing and publication regime: The SMGloM license protects symbols against non-conservative changes while allowing derived works.

3 Applications of the SMGloM

The main advantage of SMGloM over existing terminological resources for mathematics is that it makes important linguistic and ontological relations explicit that these do not. This extension makes a large variety of applications feasible without requiring full formalization, the cost of which would be prohibitive. We will sketch some of the applications here.

Glossary of Mathematical Terms An interface that presents SMGloM like a traditional glossary, i.e. as a (sorted) list of glossary entries. In addition, the semantic information in SMGloM can be used to adequately mark up references to as well as relations with (e.g. "synonym of", or "translation of") other entries. See Figure 1 for the current interface. There can be sub-glossaries, for certain areas of mathematics, for certain languages, etc.

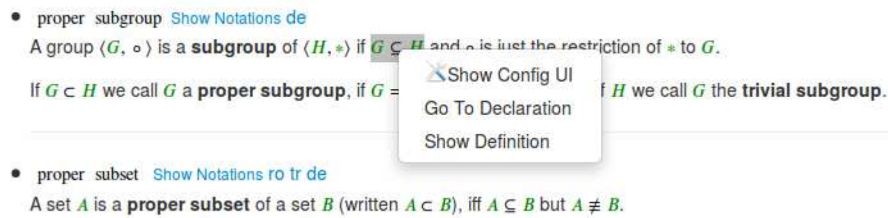


Fig. 1. The glossary interface at `https://mathhub.info/mh/glossary`

Mathematical Dictionaries The mathematical terminology is synchronized by content symbols in SMGloM, therefore a mathematical dictionary is simply an interface problem; see `https://mathhub.info/mh/dictionary`.

Flexible Styling/Presentation If we have formulae in content markup (i.e. in content MathML e.g. in OMDoc or sTeX), then we can adapt the rendering of formulae with symbols that having multiple notations in SMGloM to the user's preferences. Then, each user can state their notational preferences (in terms of SMGloM notation definitions), and the formulae in SMGloM will be rendered using these, adapting to the preferences of the reader.

Notation-Based-Parsing The notation definitions from SMGloM can be seen as user-contributed grammar rules. Therefore, they can be used for parsing formulae from presentation to content markup in the longer run. This will lead to a context-sensitive formula parser, where "context" is defined by the SMGloM glossary modules currently in focus – here the data model in term of OMDoc/Mmt theories directly contributes to the applications of the SMGloM.

More Semantic Search As SMGloM declares symbols together with notations, definitions and verbalizations it provides an unique opportunity for applying semantic search services based on it in a variety of settings:

1. notation-based parsing in the input phase could make formula entry into an interactive disambiguation process. For instance, a user enters `e^?x` (where `?x` represents a query variable), and the system ask her: "with e, do you mean Euler's number?", and also: "Is $\mathsf{e}^{?\mathsf{x}}$ a power operation?". The answers will then help refine the search.
2. Alternatively, search could use disambiguation as a facet in the search to refine the results or for clustering the results.
3. Furthermore, the SMGloM information could be used for query expansion (both visible or automatic): if the user searches for e, then the query could be expanded e.g. by *i*) the string Euler's Number (there is an interesting question about what to do with the language dependency here) and even *ii*) the formula $\lim_{?\mathsf{n}\to\infty}(1+\frac{1}{?\mathsf{n}})^{?\mathsf{n}}$.

Verbalization-Based Translation One of the most tedious parts of translating mathematical documents is the correct use of technical terms. A semantically preloaded text (i.e. one that has all formulae in content markup and many semantic objects explicitly marked up) can be term-translated automatically using the translation relation induced by SMGloM. Of course, synonyms must be re-

solved consistently (there has to be an interface for this). This (and related semantic tasks) are for domain specialists. The intervening text can be done by lesser trained individuals (or even a variant of google translate). This will make translations much cheaper and will make math available in more languages.

Wikifiers like NNexus Wikifiers are systems that given a glossary of terms create definitional links in documents. A math-specific example is the NNexus system [GC14], it can already use the SMGloM glossary.

4 Conclusion & State

We have described a project to establish a public, semantic, and multilingual termbase for mathematics. We have a first prototype that supports authoring of glossary entries and glossary management at `https://mathhub.info/smglom`. The SMGloM system partially automates editing, management, refactoring, quality control, etc; for more information see `https://mathhub.info/help/main.html`.

To make public contributions to SMGloM feasible, it must already contain a nucleus of (basic) entries that can be referenced in other glossary components. The SMGloM project is currently working towards a basic inventory of glossary entries, and has almost arrived at the first milestone of 600 entries – most with two language bindings, some with 6. The current glossary contains

i) ca. 200 glossary entries from elementary mathematics, to provide a basis for further development

ii) ca. 400 are special concepts from number theory to explore the suitability of the SMGloM for more advanced areas of mathematics.

Acknowledgements Work on the SMGloM system has been partially supported by the Leibniz association under grant SAW-2012-FIZ_KA-2 and the German Research Foundation (DFG) under grant KO 2428/13-1.

References

[GC14] Deyan Ginev and Joseph Corneli. "NNexus Reloaded". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 423–426. ISBN: 978-3-319-08433-6. URL: `http://arxiv.org/abs/1404.6548`.

[Koh14] Michael Kohlhase. "A Data Model and Encoding for a Semantic, Multilingual Terminology of Mathematics". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 169–183. ISBN: 978-3-319-08433-6. URL: `http://kwarc.info/kohlhase/papers/cicm14-smglom.pdf`.

[MH] *MathHub.info: Active Mathematics*. URL: `http://mathhub.info` (visited on 01/28/2014).

[Wat+14] Stephan Watt et al., eds. *Intelligent Computer Mathematics*. LNCS 8543. Springer, 2014. ISBN: 978-3-319-08433-6.

Parsing Texts and Checking Proofs in LaTeX

Bob Neveln and Bob Alps

Widener University, Chester, PA 19013, USA,
neveln@cs.widener.edu,
2222 Simpson Street, Evanston, IL 60201, USA,
BobAlps@aol.com

Abstract. ProofCheck is a Python package which supports parsing mathematical texts and checking mathematical texts with proofs. The mathematical texts must have been written in a LaTeX or plain TeX document. By defailt, everything inside TeX dollar signs is parsed and must adhere to a very flexible syntax based on the work of A. P. Morse. The syntax for proofs consists of indications of which lines are to be checked, references to previously established results, and indication of the end of the proof. ProofCheck checks each step by searching a rules file for a rule of inference that will match up with the inidcated references. ProofCheck can be configured to work with any logic or set theory that can be expressed in the Morse syntax. The initial configuration of ProofCheck is based on a system of free logic and set theory with classes. The authors are working to adapt ProofCheck to a relevance logic, in one project, and to a constructive logic and set theory, in another. A general explanation of the use and structure of the system and programs is provided, and a sample proof is shown in detail. A description of ProofCheck can be found in the TeX User's Group Journal in 2007 and 2009 in addition to online at www.proofcheck.org. The package can be downloaded from the online site as well as from the Python Package Index (pypi.python.org.).

1 Introduction

ProofCheck is a tool for (1) checking the sytactical correctness of formal mathematical texts written in TeX and (2) checking the correctness of formal mathematical proofs. To get an understanding of of ProofCheck, it may be helpful to know its genesis. ProofCheck was not conceived as a proof checking tool. The authors were writing mathematical texts in TeX using the formal syntax of A. P. Morse [6]. As an aid to ensuring the syntactical correctness of the texts, the first author wrote a Python program to read the TeX file and check the mathematical expressions for syntactical errors, such as unbalanced parentheses. This syntax checking program deals with the syntax in a very general way, and allows definitions of new terms and formulas.

The authors found themselves writing very detailed proofs in which nearly each line had a justification citing an previous step or earlier result. It occurred to the authors that this mode of proof was close to being checkable by computer. Once again, the first author undertook the task of implementing the proof checker, which has led to what is now called ProofCheck.

The key commitment in ProofCheck is not to a certain set theory or logic, but to a set of rules of syntax. These rules, largely based on rules devised by Morse, are quite flexible and permit the development of a wide variety of mathematical foundations and systems. For example, the authors prefer a foundation using free logic with indefinite descriptions and set theory with proper classes. However, a foundation using standard logic and Zermelo-Fraenkl set theory could equally well be used.

The program was used to check all the proofs in a submission to a well-known logic journal. The file checked was the submitted LaTeX file written using the journal's own LaTeX class file. The referee commented upon but did not object to the paper's formality. The paper was rejected based on the referee's belief that it contained no new results. The program has also been used to check student proofs in an introductory discrete mathematics class. The parser is currently being used to check the syntax of a comprehensive presentation of Morse's work on logic, set theory and analysis, edited by the second author.

2 Morse-style Formalism

In this section we describe the basic Morse formalism. We show how the language is generated by the author's definitions, by way of basic forms and signatures. Morse's goal was to be completely formal and yet as convenient and readable as possible. Besides Morse's book [6], the Ph.D. dissertations of the authors [7] [1], the Ph.D. dissertation of Morse's student Robert Arnold [3], an article on the authors' logic [2], and a long article by Hewittt Kenyon and Morse on abstract measure-type derivatives [5] have all been written in Morse-style formalism.

2.1 Basic Forms

One of Morse's basic notions of mathematical language is that it is generated by an author's definitions, or more precisely, the left sides of those definitions. These are called *basic forms*. We also allow primitive or undefined forms as basic forms. Thus in set theory we might have as basic forms the primitive form '$(x \in y)$', the defined form '$(x \cap y)$' resulting from a definition such as

$$((x \cap y) = \{z : (z \in x \wedge z \in y)\})$$

and many more as well. This kind of mathematical notation is perfectly standard. On the other hand standard notation which contains bound variables is often introduced using forms with function notation mixed in such as

$$\lim x \to a \ f(x)$$

Morse's replaces the unsatisfactory function notation '$f(x)$' with what he calls a *schematic expression*, '$\underline{\text{u}}x$'. Here the symbol '$\underline{\text{u}}$' is a called *schemator*, which

is essentially a second order function symbol[1] which is allowed only free occurrences. With a schematic expression carrying the bound variable 'x' we can then define the following basic form:

$$\lim x \to a \ \underline{\mathrm{u}}x$$

using a definite description and the conventional ϵ - δ approach. With this example in view we can begin to understand the syntactic treatment of bound variables in a Morse language. In particular we have by agreement that

1. A variable is free in a basic form if and only if it occurs in it at most once.
2. A variable is bound in a basic form if and only if it occurs in it at least twice.
3. The scope of a bound variable in a basic form consists of all the schematic expressions in which it appears.

Thus in the limit form above, 'a' is free, 'x' is bound, and the schematic expression '$\underline{\mathrm{u}}x$' defines the scope of the bound variable 'x'.

Defining the roles of the variables in the basic form in this very general way allows for considerable freedom in the choice of expressions which are to be used as basic forms. In particular we see the purpose of this generality is to allow latitude to an author, by eliminating as many arbitrary constraints as possible.

2.2 Symbols

The limit form introduced above contains the constant symbols 'lim' and '$\rightarrow$', the variables 'x' and 'a', and the schemator '$\underline{\mathrm{u}}$'. The variables 'x' and 'a' are used to denote objects and are terms. In addition variables are needed which are formulas and denote sentences: '$\underline{p}$', '$\underline{q}$', etc. Using them we may form logical expressions such $(\underline{p} \wedge \underline{q})$ for conjunction. Such variables are called *sentential variables* to distinguish them from ordinary variables such as 'x' and 'a' which are also called *object variables*.

Similarly schematic expressions are needed which are formulas as well as those such as '$\underline{\mathrm{u}}x$' which are terms. We further qualify the description of '$\underline{\mathrm{u}}$' as a *term schemator* and introduce formula schemators such as '$\underline{\mathrm{p}}$' and '$\underline{\mathrm{q}}'$'. We thus have the predicate expressions '$\underline{\mathrm{p}}x$' and '$\underline{\mathrm{q}}'xy$' which are formulas. These occur for example in basic forms quantification '$\forall x \ \underline{\mathrm{p}}x$', '$\forall x, y \ \underline{\mathrm{q}}'xy$', etc.

To summarize we may describe our set of symbols Σ as

$$\Sigma = \mathcal{C} \cup \mathcal{V} \cup \mathcal{Q} \cup \mathcal{U} \cup \mathcal{P}$$

where:

- $\mathcal{C}$ is the set of constants
- $\mathcal{V}$ is the set of object variables
- $\mathcal{Q}$ is the set of sentential variables

[1] The latter usage is somewhat unfortunate due to its possible confusiton with a mathematical function which is a set of ordered pairs.

– $\mathcal{U}$ is the set of term schemators
– $\mathcal{P}$ is the set of formula schemators

These symbols make up the actual expressions of the language and and therefore constitute the set of terminal symbols of the grammar.

To build a context-free grammar for any Morse language four non-terminal symbols suffice

$$N = \{`S', `F', `T', `V'\},$$ [2]

using 'S' for "start," 'F' for "formula," 'T' for "term," and 'V' for "variable."

2.3 Signatures

Before describing the production rules of the grammar we need to build the term and formula *signatures*.

To obtain the *signature* of a schematic expression replace each variable by 'T'.

To obtain the *signature* of a basic form replace

1. Each free variable by 'T'.
2. Each sentential variable by 'F'.
3. Each schematic expression which is a term by 'T'.
4. Each schematic expression which is a formula by 'F'.
5. Each remaining variable by 'V'.

Since both basic forms and schematic expressions may be either terms or formulas we obtain a set of *term signatures* $\mathcal{S}_T$ and a set of *formula signatures* $\mathcal{S}_F$. For example we have the following signatures of several previously mentioned basic forms and schematic expressions.

Expression	Signature
$\lim x \to a\ \underline{\mathrm{u}}x$	$\lim V \to T\ T$
$\underline{\mathrm{u}}x$	$\underline{\mathrm{u}}T$
$(\underline{p} \wedge \underline{q})$	$(F \wedge F)$
$\underline{\mathrm{p}}'xy$	$\underline{\mathrm{p}}'TT$
$(x \in y)$	$(T \in T)$
$\forall x\ \underline{\mathrm{p}}x$	$\forall V\ F$

The first two of these are term signatures while the remaining four are formula signatures.

We begin by describing a Morse-like grammar as a context-free grammar.[3]

[2] Because the non-terminal symbols must be distinct from the terminal symbols, we imagine, for the purposes of this discussion, that these four symbols are not among the symobls of the mathematical language.

[3] Using a context-free grammar to describe a Morse language of necessity produces anamolous formulas with repeated bound variables such as '$\forall x, x\ \underline{\mathrm{q}}'xx$'. This requires either modifying **BF 3.** in section 2.5 so as to allow such forms to be defined, or leaving them inferentially stranded.

2.4 Context-Free Grammars

The context-free grammar (N, Σ, R, S) generates the language, where N, Σ, and S are as defined in section 2.2 and R is the set of all the following production rules:

1. $S \to T$
2. $S \to F$
3. $T \to t$, for each $t \in \mathcal{S}_T$
4. $F \to f$, for each $f \in \mathcal{S}_F$
5. $V \to v$, for each $v \in \mathcal{V}$
6. $F \to q$, for each $q \in \mathcal{Q}$
7. $T \to V$

The terms of the language are generated by the grammar (N, Σ, R, T), and the formulas by (N, Σ, R, F).

The simplicity of this prescription for the generation of a language makes it clear how ProofCheck is able to parse mathematical text which is generated almost at will by the author.

2.5 Rules for Basic Forms

Although basic forms may be constructed very freely, some restrictions are necessary. The following rules are imposed:

BF 1. No schemator or sentential variable may occur more than once in a basic form.
BF 2. Every occurrence of a schemator in a basic form is as the initial symbol of a schematic expression.
BF 3. An object variable occurring in a basic form must occur exactly once not within some schematic expression.

A constant which is the initial symbol of some basic form is called an *introductor*. A rule which is needed to show that the language is unambiguous and has the prefix property, that no term of formula is the prefix of another, is as follows.

BF 4. If s and s' are signatures of f and f' respectively where f and f' are distinct and p is the longest shared prefix of s and s', then p is not s nor is it the case that p is followed in s by an introductor and in s' by a non-terminal symbol ‘F’,‘T’, or ‘V’.

2.6 Tempering the Formalism

The fact that ProofCheck works depends on several modifications of a strict literal implementation of the above formalism. We mention what we believe are the most important of these.

Infix Notation and Precedence Standard infix notation uses operator precedence to parse a formula such as $(\underline{p} \wedge \underline{q} \rightarrow \underline{r})$ as $((\underline{p} \wedge \underline{q}) \rightarrow \underline{r})$. Reconciling this common practice with a definition-based grammar requires that definitions such as

$$((\underline{p} \wedge \underline{q} \rightarrow \underline{r}) \leftrightarrow ((\underline{p} \wedge \underline{q}) \rightarrow \underline{r}))$$

be allowed to have an "implicit" status or be generated by a definitional scheme.

ProofCheck's parser, is actually a hybrid of a left-to right recursive parser which handles explicitly defined basic forms and an infix parser which kicks in whenever an introductory left parenthesis is encountered. Default precedence values are assigned for some standard operators, but these may be changed and precedence for new operators may be set at will by the user.

Default behavior is also built in for operators having the "verbal" precedence value so that, for example, $(a, b \in c \subset d)$ is parsed as $(a \in c \wedge b \in c \wedge c \subset d)$

Scope Notation Bound variable forms whose definition has either of the forms '$\bigcap x; \underline{\mathrm{p}}x\ \underline{\mathrm{u}}x$' or '$\mathrm{The}\, x\ \underline{\mathrm{p}}x$' (whatever the initial symbol) has automatically renders varied scope notations such as '$\bigcap x \in A \cup B\ \underline{\mathrm{u}}x$' or '$\mathrm{The}\, x > 0\ (x \cdot x = 4)$'.

Polyabbreviations Many notes in checked proofs consist of single formulas which are in fact abbreviations of several. For example a formula such as:

$(\underline{p} \rightarrow \underline{q}$
$\quad \rightarrow \underline{r})$

used as a note in a proof is taken as an abbreviation of the following three formulas:

$(\underline{p} \rightarrow \underline{q})$
$(\underline{q} \rightarrow \underline{r})$
$(\underline{p} \rightarrow \underline{r})$

The first two of these are called *delta goals* and the third is called a *sigma goal*. Each line of a multi-line proof represents a delta goal and may be justified by a right-hand marginal note. In general a multi-line proof spanning n lines generates n delta goals and $(n-1)$ sigma goals. Sigma goals must of course be checked as well as delta goals.

Polyabbreviations may be nested as in the following example where a 5 line note becomes 9 separate assertions:

Line	Delta Goal	Sigma Goals
$(\underline{p} \rightarrow \underline{q}$	$(\underline{p} \rightarrow \underline{q})$	
$\rightarrow \underline{r}$	$(\underline{q} \rightarrow \underline{r})$	$(\underline{p} \rightarrow \underline{r})$
$\rightarrow a < b$	$(\underline{r} \rightarrow a < b)$	
$< c$	$(\underline{r} \rightarrow b < c)$	$(\underline{r} \rightarrow a < c)$
		$(\underline{p} \rightarrow a < c)$
$\rightarrow \underline{s})$	$(a < c \rightarrow \underline{s})$	$(\underline{p} \rightarrow \underline{s})$

Subsequent reference to a multi-line note appeals to the final sigma goal only.

Derived Rules of Inference ProofCheck obtains rules of inference other than these from a .tex file which by default is named `rules.tex`. One of the fundamental ideas on which ProofCheck is based is that this rules file needs to be big in order for the program to behave reasonably.

Although there is nothing to prevent derived rules from embodying mathematics, ProofCheck is currently implemented using rules of inference which embody only logic. The file which has been used to do the most proofs has over 1500 rules. Most of these, far from obscure, are annoyingly transparent. An example of a fairly short derived rule would be:

$$(\underline{p} \rightarrow (\underline{q} \rightarrow \underline{r})); \underline{p}, \underline{q} \vdash \underline{r}$$

In order to provide a mechanism for preventing errors from creeping into the rules of inference file, ProofCheck can be applied to derivations of derived rules. For example, the rule just stated can be derived simply as follows

1. $(\underline{p} \rightarrow (\underline{q} \rightarrow \underline{r}))$ ‡P
2. $\underline{p}$ ‡P
3. $(\underline{q} \rightarrow \underline{r})$ ‡.1; .2
4. $\underline{q}$ ‡P

Q.E.D. .3; .4

The justification 'P' claims that the note is a premise. This rule checks easily provided that modus ponens, used to check note 3 and the Q.E.D., is in the rule of inference file.

Deduction Theorems For standard logic and several forms of non-standard logic there are deduction theorems, which state that if there is a deduction of a formula $\underline{q}$ from a premise $\underline{p}$ then there is also a proof of the formula $(\underline{p} \rightarrow \underline{q})$. A constructive proof of this theorem yields a procedure for translating the original deduction into a proof. But, if instead of using this procedure one just appeals directly to the result of the theorem, then it becomes another rule of inference.

There are of course, derived rules of this nature as well. For example if the negation of $\underline{p}$ can be deduced form $\underline{p}$ then there must be a proof of $\neg\underline{p}$. There are about 40 such derived rules of inference in ProofCheck's main rules file.

Associative and Commutative Unification Gallier has designed a unification algorithm called E-unification which recognizes equations in the unification process. ProofCheck uses a unifier which recognizes commutativity and associativity of operators so specified by the user, in the presence of commutative and associative theorems.

This feature of ProofCheck is of very great importance in shortening the length of proofs. ProofCheck's unifier is written to be quick and dirty, to succeed or fail quickly. It is not complete and is not intended to be.

3 Parsing

3.1 Definitions

In any mathematical system there are a few primitive terms and formulas from which all others are derived by definitions. Thus each basic form becomes known to ProofCheck either by way of a declaration as primitive or by way of a definition. For the purpose of parsing, we do not need to know how one basic form depends on another. It is enough to know for each basic form whether it is a basic term or a basic formula. Thus for the purpose of parsing we could declare each basic form as if it were primitive.

3.2 Parsing Setup

Several files are usually needed for parsing, beginning with the user's TEX or LATEX file. For the purpose of this discussion we will refer to this document as "myfile.tex." The parser will require information on basic forms presumed by the user in his paper, such as the basic forms of logic and set theory. Such presumed basic forms must be defined or declared in one or more auxiliary TEX files. For the purpose of discussion we will refer to this as a single file called "common.tex." This file would be listed as an input file at the beginning of myfile.tex. Special constants used in the basic forms of a file should be given TeX definitions in an auxiliary file with the same name as the file but having an extension ".tdf" for TeX or ".ldf" for LaTeX. This file may also contain primitive form declarations, and other ProofCheck macros. It needs to be listed as an input file at the beginning of <myfile>.tex. Other input ".tdf" or ".ldf" files may also be needed including always "utility.tdf". Any other files whose definitions may be relied upon, such as those in "common.tex" should have their corresponding ".tdf" or ".ldf" files input at the beginning of <myfile>.tex. Thus for example, assuming we are working in LaTeX, there would be three .ldf files listed as an input: <myfile>.ldf, common.ldf and utility.ldf.

3.3 Parsing Execution

Once all the needed files are in place and have been TEX'd, the process of parsing can begin. The user runs the parser on myfile.tex. The parser parses mathematical text found between TEX dollar signs or between double dollar sign displays. The parser stops at the first error and reports back. After the user fixes that error, the parser must be restarted from the beginning, until all errors are found and fixed.

4 Checking

4.1 File Structure

Figure 1 shows the relationships among the primary checking files. The rectangular boxes represent TEX files whereas the unifier and the rule matcher are

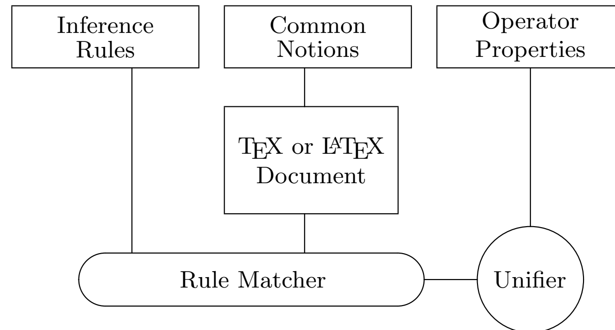


Fig. 1. Main Proof Checking Files

Python files. The default name for the Inference Rules file is "rules.tex" and the default name for the Common Notions file is "common.tex." The current compressed download file, which contains the current rules.tex and common.tex files in use by the authors, is under 300 kB.

4.2 General Structure

In ProofCheck, a proof is a sequence of noted (i.e., numbered) steps, where each step is a complete mathematical statement. Each step has one or more lines with justification. A multi-line step involves transitive relationships such as with equality, inequality, equivalence, or implication.

Each line within a step is checked individually using its justification which is an end-of-line references to theorems and/or notes. Each of these checks is done by submitting a formula and the formulas referred to in its justification to a rule matcher which conducts a simple linear search of the list of inference rules. The search must find a rule which unifies with the submitted formulas in order for the check of the assertion to succeed.

4.3 Justifications and Rules of Inference

Whatever basic forms the user creates, the rules of syntax will determine which variables are free in any basic form and which are bound. ProofCheck has rules programitically built in for substituting for a free variable, for substituting for a schematic expression, and for changing a bound variable. Other rules of inference are left to the user and stored in a file called "rules.tex". For the logic[2] used by the authors, the only additional required rule of inference is modus ponens.

From a practical perspective, ProofCheck needs many additional rules of inference, all of which are derived rules. While such derived rules could embody

mathematics, the authors have chosen to include rules that embody only logic. There are currently about 1500 rules of inference in the rules.tex file used by the authors.

When providing a justification for a line or step in ProofCheck, the user cites which previous lines or results are needed to justify the step. There is a certain syntax in how the results are cited. For example if it is a situation where there is a major premise M and a minor premise m, then the justification is written as $M;m$. If two previous results A and B have equal standing, then the citation is A, B. Rules of inference are never cited, but they will have the same syntactical structure as the coding of the justification. Thus, the entry for modus ponens is as follows:

$(\underline{p} \rightarrow \underline{q})\ ;\underline{p} \vdash \underline{q}$

There is a major premise $(\underline{p} \rightarrow \underline{q})$, a minor premise $\underline{p}$, and a conclusion $\underline{q}$.

Many references in justifications may be to facts that are not contained in the user's document. Such facts are kept in auxiliary files. The authors tend to keep such facts in a single file called "common.tex".

4.4 Deduction Theorem Rules

For most logics there are deductions theorems such as: If $\underline{q}$ can be deduced from the premise $\underline{p}$, then $(\underline{p} \rightarrow \underline{q})$ is a theorem. A constructive proof of a deduction theorem yields a procedure for constructing the proof of $(\underline{p} \rightarrow \underline{q})$ from the deduction of $\underline{q}$ from $\underline{p}$. Assured of the possibility of such a construction, the prover simply appeals directly to the deduction theorem. In this way, the deduction theorem becomes another derived rule of inference. The rule.tex file used by the authors includes about 40 such rules. For example, if $\neg\underline{p}$ can be deduced from the premise $\underline{p}$, then $\neg\underline{p}$ is a theorem.

4.5 Sample Proof: Reader View

The sample proof in figure 2 is taken from a development of the Von-Neuman model of the natural numbers, ω, in which each natural number is the set of the preceding natural numbers. The theorem asserts that if y is an element of a natural number x then y is a subset of x.

The first line of the proof defines a set A. This note should be easy to read except for possibly the quantifier notation and the classifier notation. The second note translates the definition in note 1 into a bi-conditional which is much more useful deductively. We often refer to steps that turn a definition into one or more implications as "unwrapping" steps. Explicit inclusion of such unwrapping steps is often key in getting a proof to check. The stage is set for an induction proof.

The theorem 4.7 referred to is just the standard induction theorem:

$(\emptyset \in A \wedge \bigwedge x \in A(x \in A \rightarrow \text{scsr } x \in A) \rightarrow \omega \subset A)$

where "scsr x" denotes the successor of x. Its two hypotheses are the base case which in this proof is established in note 3 and the universalization of the induction step established in note 9. This theorem is invoked in the justification

Theorem
4.8 $(y \in x \in \omega \rightarrow y \subset x)$

Proof: To prove this by induction we begin by letting A be the set of all x such that each element y of x is a subset of x. We set

.1 $(A \equiv \mathrm{E}\, x \bigwedge y \in x(y \subset x))$ ‡S

It will follow from 4.7 that ω is a subset of A. First we unwrap .1.

.2 $(x \in A \leftrightarrow \bigwedge y \in x(y \subset x) \wedge x \in \mathrm{U})$ ‡08.3;.1

Base Case $(\emptyset \in A)$.

.3 $(\emptyset \in A)$ ‡.2;09.19,09.12

Induction Step $(x \in A \rightarrow \mathrm{scsr}\, x \in A)$.

Given

.4 $(x \in A)$ ‡G

We note first that

.5 $(x \in \mathrm{U})$ ‡09.20;.4

.6 $\bigwedge y \in x(y \subset x)$ ‡.2;.4

Then we have

.7 $(y \in \mathrm{scsr}\, x \rightarrow y \in x \vee y = x$ ‡3.7

$\rightarrow y \subset x \vee y = x$ ‡.6

$\rightarrow y \subset x$ ‡011.14

$\rightarrow y \subset \mathrm{scsr}\, x)$ ‡011.10;(3.5;(09.20;.4))

So we can conclude that

.8 $(\mathrm{scsr}\, x \in A)$ ‡.2;(3.3;.5),(.7 U)

Hence

.9 $(x \in A \rightarrow \mathrm{scsr}\, x \in A)$ ‡.8 H .4

This completes the proof that

.10 $(\omega \subset A)$ ‡4.7;.3,(.9 U)

The conclusion now follows quickly.

.11 $(y \in x \wedge x \in \omega \rightarrow y \in x \wedge x \in A$ ‡011.7;.10

$\rightarrow y \in x \wedge \bigwedge y \in x(y \subset x)$ ‡.2

$\rightarrow y \subset x)$

Q.E.D. .11

Fig. 2. Sample Proof: DVI output

of note 10. In the proof of the induction step note 7 shows that any member y of scsr x is a subset of scsr x. In note 8, we conclude that scsr x is in A. In note 9 we join the hypothesis from note 4 to the conclusion obtained in note 8.

Note 11 details the step from note 10 to the theorem which is short but cannot be skipped. The "QED .11" at the end asserts that the theorem itself follows from note 11.

4.6 Sample Proof: Author View

In Figure 3 we have the LaTeX source code for the sample proof.

The first couple of lines of the sample proof begin explaining the proof. Since they are not noted they do not contribute to the check. But neither do they get in the way of the check. Unchecked text of any sort is admissible so long as it does not interrupt mathematical expressions interfere with proof specification.

Note 4 opens a Given-Hence block and establishes $(x \in A)$ as a working hypothesis. Note 4 may be referred to only within this Given-Hence block. A Hence justification may close more than one Given note, but each Given note must be explicitly closed by a Hence justification. This Given-Hence block is closed by note 9.

Note 7 is a multi-line note each line of which has a justification. A reference to note 7 accesses the telescoped result of the note which is

$$(y \in \mathrm{scsr}\ x \rightarrow y \subset \mathrm{scsr}\ x)$$

4.7 Checking a Line: An Example

Let us look at note 3 of the above proof is more detail. This note reads as follows:

.3 $(\emptyset \in A)$ ‡.2;09.19,09.12

The justification refers to three things: previous note 2, and theorems 9.19 and 9.12 from common.tex. We now list these three things.

.2 $(x \in A \leftrightarrow \bigwedge y \in x\ (y \subset x) \wedge x \in \mathrm{U})$

9.19 $(\emptyset \in \mathrm{U})$

9.12 $\bigwedge x \in \emptyset\ \underline{\mathrm{p}}x$

The justification has the form m;a,b. ProofCheck scans the rules.tex file looking for rules of this form. Each rule having this form is tested to determine whether it can be used to justify the given step. In this case, rule number 338 happens to do the trick. It goes as follows.

$$(\underline{r} \leftrightarrow \underline{p} \wedge \underline{q});\underline{p},\underline{q} \vdash \underline{r}$$

If in note 2, we replace 'x' by '$\emptyset$' we get a modified note 2 as follows.

$$(\emptyset \in A \leftrightarrow \bigwedge y \in \emptyset\ (y \subset \emptyset) \wedge \emptyset \in \mathrm{U})$$

Next consider the major premise in the rule: $(\underline{r} \leftrightarrow \underline{p} \wedge \underline{q})$. If in this major premise, we replace '$\underline{r}$' with '$(\emptyset \in A)$', '$\underline{p}$' with '$\bigwedge y \in \emptyset\ (y \subset \emptyset)$', and '$\underline{q}$' with '$(\emptyset \in \mathrm{U})$' the result, modulo removal of some parentheses, is modified note 2. The same replacement for '$\underline{q}$' gives us 9.19. Notice that 9.12 is a general statement

```
\noindent{}Theorem

\prop 4.8 $(y \in x \in \omega \c y \subset x)$
    \lineb Proof:  To prove this by induction we begin by
letting $A$ be the set of all $x$ such that each element $y$ of $x$
is a subset of $x$.  We set
\notez 1 $(A \ident \setof x \Each y\in x(y\subset x))$ \By S
\linea It will follow from 4.7 that $\omega$ is a subset of $A$. First we unwrap .1.
\notez 2 $(x \in A \Iff \Each y \in x (y \subset x) \And x \in \U)$ \By 08.3;.1
    \lineb Base Case $(\e \in A )$~.
\notez 3 $(\e \in A)$ \By .2;09.19,09.12
    \lineb Induction Step $(x \in A \c \scsr x \in A)$~.
\linea Given
\notez 4  $(x \in A )$  \By G
\linea We note first that
\notez 5 $(x \in \U)$  \By 09.20;.4
\notez 6 $\Each y \in x(y \subset  x)$\By .2;.4
\linea Then we have
\notez 7 $(y \in \scsr x \c y \in x \Or y = x$  \By 3.7
    \lined $\c y \subset x \Or y = x$  \By .6
    \lined $\c y \subset x $  \By 011.14
    \lined $\c y \subset \scsr x )$ \By 011.10;( 3.5;(09.20;.4))
\linea So we can conclude that
\notez 8 $(\scsr x \in A)$ \By .2;(3.3;.5),(.7 U)
\linea Hence
\notez 9 $(x \in A \c \scsr x \in A)$ \By .8 H .4
\linea This completes the proof that
\notez 10 $(\omega \subset A)$ \By 4.7;.3,(.9 U)
\linea The conclusion now follows quickly.
\notez 11 $(y \in x \And x \in \omega\c y \in x \And x \in A$ \By 011.7;.10
    \lined $\c y \in x \And \Each y \in x(y \subset x)$ \By .2
    \lined $\c y \subset x)$
    \lineb \Bye .11
```

Fig. 3. Sample Proof: LaTeX Input

involving a schematic expression. No matter what predicate replaces '$\underline{\text{p}}x$', for each x in the empty set, that predicate is (vacuously) true for x. First we make a change of bound variable in 9.12 so that it reads

$$\bigwedge y \in \emptyset \ \underline{\text{p}}y.$$

Next we replace '$\underline{\text{p}}y$' with '$(y \subset \emptyset)$' which gives

$$\bigwedge y \in \emptyset \ (y \subset \emptyset).$$

This is the result of replacing '$\underline{p}$' in the rule. Therefore we have the major premise, and 9.19 and 9.12 are the two minor premises. Thus this rule allows us to justify the conclusion $\underline{r}$.

$(\emptyset \in A)$.

5 ProofCheck Characteristics and Issues

5.1 Checking Derived Rules of Inference

The main rules file used by the authors contains over 1500 rules of inference. All but a handful of these rules are derived rules, with most of them being annoyingly transpararent. The following is an example of such a rule.

$$(\underline{p} \rightarrow (\underline{q} \rightarrow \underline{r}));\underline{p},\underline{q} \vdash \underline{r}$$

ProofCheck may be used to verify the derivation of derived rules. A derivation for the above rule goes as follows.

.1	$(\underline{p} \rightarrow (\underline{q} \rightarrow \underline{r}))$	‡P
.2	$\underline{p}$	‡P
.3	$(\underline{q} \rightarrow \underline{r})$	‡.1; .2
.4	$\underline{q}$	‡P
		Q.E.D. .3; .4

The justification 'P' claims that the note is a premise. This rule checks easily provided that modus ponens is in the rule of inference file.

5.2 Associative and Commutative Unification

Gallier designed a unification algorithm called E-unification which recognizes equations in the unification process. ProofCheck uses a unifier which recognizes commutativity and associativity of operators so specified by the user, in the presence of commutative and associative theorems.

This feature of ProofCheck is of very great importance in shortening the length of proofs. ProofCheck's unifier is written to be quick and dirty, to succeed or fail quickly.

5.3 Completeness

Based on its current design, it will always be possible to construct a correct proof that ProofCheck will not check. Each rule involves reference to one or more theorems. Suppose that the largest number of theorems referred to by any rule in the rules file is n. Then a justification that references $n+1$ theorems cannot be matched to any rule in the file. In general, the unification algorithm is designed with speed in mind rather than completeness. As the authors have worked through proofs, additional rules of inference have frequently been added. At this point, new rules are seldom added.

6 Conclusion

We find the parsing capability of ProofCheck to be an extremely valuable tool in preparing manuscripts. Of course, that implies that the manuscripts contain mathematical text conforming to the syntactical rules of ProofCheck.

ProofCheck has been used to check about 500 proofs, including about 400 proofs in a work-in-progress by the authors on combinatorial topology.

A checkable proof is usually at least twice as long as a usual proof and may be as much as ten times as long. Even with the added length, a checkable proof can be readily followed by a human reader familiar with the syntax. We are seeking ways to shorten checkable proofs.

References

[1] Alps, R. A.: *A Translation Algorithm for Morse Systems*. Ph. D. Dissertation, Northwestern University (1979)

[2] Alps, R. A., Neveln, R. C.: *A Predicate Logic Based on Indefinite Descriptions and Two Notions of Identity. Notre Dame Journal of Formal Logic* (1981)

[3] Arnold, R. S.: *Plus and Times*. Ph.D. Dissertation, University of California Berkeley (1969)

[4] Bledsoe, W. W., Gilbert, E. J.: *Automatic Theorem Proof-Checking in Set Theory*. Sandia Labarotories Research Report SC-RR-67-525 (July 1967)

[5] Kenyon, H., Morse, A. P., *Web Derivatives*. Memoirs of the American Mathematical Society (1973)

[6] Morse, A. P.: *A Theory of Sets* Second Edition. Academic Press (1986)

[7] Neveln, R.C.: *Basic Theory of Morse Languages*. Ph.D. Dissertation, Northwestern University (1975)

[8] Neveln, Bob, Alps, Bob: Writing and Checking Complete Proofs in TEX. *TUGboat*, Volume 28 (2007), No. 1

[9] Neveln, Bob, Alps, Bob: Writing and Checking Complete Proofs in LATEX. *TUGboat*, Volume 28 (2009), No. 2

A Web Environment for Geometry

Pedro Quaresma[1,2], Vanda Santos[2,3], and Milena Marić[4]

[1] Department of Mathematics, University of Coimbra, Portugal
[2] CISUC, Coimbra, Portugal
[3] University National of Timor Lorosa'e, East-Timor
[4] Faculty of Mathematics, University of Belgrade, Serbia
pedro@mat.uc.pt, vsantos7@gmail.com, milena.maric.f@gmail.com

Abstract. The Web Geometry Laboratory, *WGL*, is a blended-learning, collaborative and adaptive, Web environment for geometry. It integrates a well known dynamic geometry system.

In a collaborative session, exchange of geometrical and textual information between the user engaged in the session is possible.

In a normal work session (stand-alone mode), all the geometric steps done by the students are recorded, alongside the navigation information, allowing, in a latter stage, their teachers to "play back" the students sessions, using that info to assert the students level and adjust the teaching strategies to each individual student.

Teachers can register and begin using one of the public servers, defining students, preparing materials to be released to the students, open collaborative sessions, etc.

Using an action research methodology the *WGL* system is being developed, validated through case-studies, and further improved, in a cycle where the implementation steps are intertwined with case studies.

Keywords: adaptive learning, collaborative learning, blended-learning, dynamic geometry

1 The Web Geometry System

The *Web Geometry Laboratory* (v1.4) is a Web client/server application; the server must be hosted by a Web-server (e.g. Apache server) the clients may use any Web-browser available. The database (to keep: constructions; users information, constructions permissions, etc.); the dynamic geometry system (DGS), JavaScript applet; the synchronous and asynchronous interaction, are all implemented using free cross-platform software, namely GeoGebra, PHP, JavaScript, AJAX, JSON, JQuery and MySQL. Also Web-standards like HTML5, CSS stylesheets and XML. The *WGL* is an internationalised system with the English language as the default language and already localised to the Portuguese and Serbian languages. It is an open-source system[5], versions of the server are available to be installed on Linux systems (or other systems through virtual machines).

[5] http://webgeometrylab.sourceforge.net/

The last version of WGL (1.4) introduces a total separation between development branches: stable; testing; unstable (development), given a added stability to the public (stable) server and allowing a public availability of the code. Apart many small improvements the major new features are: the text chat; the exchange of geometric information between the group and individual windows and the saving of the students work to their own scrapbook, when in a collaborative sessions; the “record & play” of student’s sessions, i.e. the adaptive module (at a prototype stage in [8]); the JavaScript/HTML5 DGS applet (instead of the Java applet).

Two instances of the *WGL* server are available, one in Portugal,[6] another in Serbia.[7] Users can log on to the system using the anonymous student-level user, but without access to collaborative sessions. For more advanced use, a user must register and then be confirmed by the administrator. During the last three years these systems have been intensively used, e.g., for testing collaborative learning in teaching geometryr [2,7,8,10]. Please feel free to contact the authors if you want to use the *WGL* platform, accessing the platform as a teacher.

In any *WGL* server there are four distinct types of users: administrators, teachers, students and anonymous visitors. The administrator(s) main role is the administration of teachers. They have also access to the log-in information off all users, information that can be used to streamline the server.

The teachers are privileged users in the sense that they will be capable of defining other users; their students. In the beginning of each school year the teachers should define all their classes, the students in each class and, if needed, the aggregation of the students into groups

The students, each linked to a given teacher, are able to work in the platform, performing tasks created by their teachers and/or pursuing their own work. The students are unable to create other users.

Finally, the anonymous visitor is a student-type user, not linked to any teacher and because of that, unable to participate in collaborative sessions. The purpose of this type of user is solely to allow unregistered users to test the *WGL* platform.

There are two distinct modes for the students to interact with the *WGL* system. The collaborative sessions and the regular (stand-alone) sessions. These two distinct modes are controlled by the teachers. In a collaborative session the students, working in groups, have some specific assignment to fulfil and they will do it in a collaborative way, exchanging geometric and textual information to reach the common goal. In a regular session the students will be working alone, they can share constructions with the other users of the platform but all this exchange of information will be asynchronous.

---

[6] `http://hilbert.mat.uc.pt/WebGeometryLab/`

[7] `http://jason.matf.bg.ac.rs/wgl/`

2 The Collaborative Module

Planning a collaborative working session the teacher has to decide how to group the students and the design of the tasks to be solved collaboratively, i.e., prepare a set of geometric constructions, starting points for tasks to be completed during the class; illustrative cases; etc.

In a *WGL* collaborative session the students will solve the tasks proposed by their teachers, being able to exchange geometric and textual information, producing the geometric constructions in a collaborative fashion.

The students engaged in a collaborative session will always be in working groups, with access to the material prepared by the teacher and with access to two DGS applets. One of those DGS applets is for their own work, the other is where the group construction is being done. The *group-construction* is shared by all the members of a given group, one of the students will have the lock over the construction, all the other group members will see the work being done (synchronised every 20s). At any given moment the student can release the lock, which can be claimed by any student in the group.

At the same time, the students has their own work-space, this can be used to: follow the work that is being done by the group representative; develop their own constructions; to anticipate the group construction; to develop auxiliary constructions. In this work-space the saving of the work being done is the responsibility of the student.

The students have the possibility of exchange constructions between DGS workspace windows. The students without the lock should be able to "import" the group construction to his/her own work-space. The student with the lock adds to that, the possibility of exporting the construction to the group work-space. A chat is provided to allow the exchange of short messages between all the members of the group, including the teacher.

Apart from being responsible for setting the collaborative session and being able to assess its results at the end, the teacher has also access to a DGS work-space window where he/she can follow the work of all the groups and all the individual students in each group.

3 The Adaptive Module

To be able to build individual student's profiles and/or individual learning paths, the system collects information about the student's interactions when in the stand-alone mode, i.e., in a regular work session.

The system records navigation and also geometric information for each student. The navigation information is a plain list of all the pages visited with enter and exit time-stamps. The geometric information is recorded when the student is using the DGS applet, using JavaScript listeners of the DGS application programming interface. We record every step done by the students.

At a later stage the student's teacher is able to see the work done by the student, play step by step, play in a regular speed, play in a fast forward fashion.

In this way the teacher can analyse the path used by the students to solve a given task, getting information that can be used to assert the student's van Hiele level [1].

4 Access to the System

The *WGL* public servers can be used by any interested teacher. The International/Portuguese server is `http://hilbert.mat.uc.pt/WebGeometryLab`, the Serbian server is `http://jason.matf.bg.ac.rs/wgl`. After registration (subject to validation) a teacher can create classes and use the system as a geometry laboratory or as platform for homework tasks. In a stand-alone fashion or in collaborative sessions.

We performed two different set of studies, one in Portugal, in classroom mode, and another in Serbia, in remote access mode[8]. The first set of studies was done using *WGL* version 1.2, still without the group-wise communication channel (chat). The second set of studies were done using *WGL* version 1.3, already with the chat communication channel, among other developments done in the platform. The platform was positively received by teachers and students, improving their learning experience [10]. The case-studies were/are used to improve the system but also to publicise the system, training teachers in its use.

A forum (phpBB forum) is provided to allow the exchange of information between users.

5 Conclusions and Future Work

Related Systems There are several DGS available (see [11] for a comprehensive list) but none of them defines an environment where the DGS is integrated into a learning platform with collaborative and adaptive features. In [5,6,9] we can find accounts of DGSs and geometric automated theorem provers (GATPs) integration and the integration of those tools in learning environments but always partial integrations not building any kind of collaborative, adaptive blended-learning platform. Some learning environments in the area of geometry have been developed, e.g. Tabulae [3] and GeoThink [4]. The *WGL* distinguishes itself relying on an external DGS, allowing in this way to possess a full fledged DGS, well known by its users and supported by its developers. The well grounded permissions system and the capability that this opens for a personalised contact with the platform, is also something in favour of *WGL*. The many case-studies already conducted, validating the *WGL* goals, and the internationalisation, i.e. the ability to receive translations into different languages (Tabulæ lacks this feature), are also positive points for *WGL*.

---

[8] *Web Geometry Laboratory: Case Studies in Portugal and Serbia*, submitted to *Educational Technology Research and Development*, May 2015

Conclusions and Future Work At the moment the adaptive module only collects the student's information and allow the teachers to "play" that information. A first step ahead, already planned, will give the teachers the possibility of building students profiles or individualised learning paths. A second, more ambitious, step would give the system some capabilities of automatic construction of those profiles and/or learning paths.

A second development planned is the integration of a GATP. To be able to provide a formal validation of geometric properties, e.g. "*two lines are perpendicular, because ...*" and also to support the automatic or semi-automatic adaptive features, e.g. one-step guidance, formal reasoning and visual proofs.

The Web Geometry Laboratory is a blended-learning, collaborative, adaptive, Web environment for geometry already being used by teachers in Portugal and Serbia and we expect that its user base can grown not only in those countries but also in other countries.

Acknowledgments

The first author is partially supported by the iCIS project (CENTRO-07-ST24-FEDER-002003), co-financed by QREN, in the scope of the Mais Centro Program and European Union's FEDER.

References

1. Mary L. Crowley. The van Hiele Model of the Development of Geometric Thought. *Learning and Teaching Geometry, K12*, Yearbook of the National Council of Teachers of Mathematics, chapter 1, pages 9–23. National Council of Teachers of Mathematics, Reston, VA, USA, 1987.
2. Milena Marić. The Web Geometry Laboratory - mogućnosti i primene. In *Korelacija matematike sa drugim nastavnim predmetima*, pages 248–257, Pula, Croatia, 2013.
3. Thiago Guimaraes Moraes, Flávia Maria Santoro, and Marcos R.S. Borges. Tabulæ: educational groupware for learning geometry. In *Advanced Learning Technologies, 2005. ICALT 2005. Fifth IEEE International Conference on*, pages 750 – 754, july 2005.
4. R Moriyn, F Saiz, and M Mora. *GeoThink: An Environment for Guided Collaborative Learning of Geometry*, volume 4 of *Nuevas Ideas en Informática Educativa*, pages 198–206. J. Snchez (ed), Santiago de Chile, 2008.
5. Pedro Quaresma and Predrag Janičić. Integrating dynamic geometry software, deduction systems, and theorem repositories. *Mathematical Knowledge Management*, volume 4108 of *LNAI*, pages 280–294. Springer, 2006.
6. Pedro Quaresma and Predrag Janičić. GeoThms – a Web System for Euclidean constructive geometry. *Electronic Notes in Theoretical Computer Science*, 174(2):35 – 48, 2007.
7. Pedro Quaresma, Vanda Santos, and Seifeddine Bouallegue. The Web Geometry Laboratory project. In *CICM 2013*, volume 7961 of *LNAI*, pages 364–368. Springer, 2013.

8. Pedro Quaresma, Vanda Santos, and Juan Moral. Reproducing a geometric working session. *Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM*, number 1186 in CEUR Workshop Proceedings, Aachen, 2014.
9. Vanda Santos and Pedro Quaresma. eLearning course for Euclidean Geometry. *Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies, July 1st- July 5th, 2008, Santander, Cantabria, Spain*, pages 387–388, 2008.
10. Vanda Santos and Pedro Quaresma. Collaborative environment for geometry. *2nd Experiment@ International Conference (exp.at'13), 2013*, pages 42 – 46. IEEEXplore, Sept. 2013. INSPEC Accession Number: 14027552.
11. Wikipedia. List of interactive geometry software. `http://en.wikipedia.org/wiki/List_of_interactive_geometry_software`, (last accessed, 2015-04-07).

Automatic and Transparent Transfer of Theorems along Isomorphisms in the Coq Proof Assistant

Theo Zimmermann[1] and Hugo Herbelin[2]

[1] École Normale Supérieure, Paris, France
theo.zimmermann@ens.fr
[2] Inria Paris-Rocquencourt, Paris, France
hugo.herbelin@inria.fr

Abstract. In mathematics, it is common practice to have several constructions for the same objects. Mathematicians will identify them modulo isomorphism and will not worry later on which construction they use, as theorems proved for one construction will be valid for all.
When working with proof assistants, it is also common to see several data-types representing the same objects. This work aims at making the use of several isomorphic constructions as simple and as transparent as it can be done informally in mathematics. This requires inferring automatically the missing proof-steps.
We are designing an algorithm which finds and fills these missing proof-steps and we are implementing it as a plugin for Coq[3].

1 Introduction

With examples such as the well-known relation between linear maps and matrices, the various constructions of real numbers (equivalence classes of Cauchy sequences, Dedekind cuts, infinite sequences of digits, subset of complex numbers), we see that there are a great many cases when identifying several constructions of the same objects can be useful in mathematics. In particular, proofs are then done on the most convenient one but theorems apply to all.

In formal systems like Coq [3], a canonical example is the various constructions available for natural numbers. The most natural construction and the closest to the mathematical view is unary (`0`, `S 0`, `S (S 0)` and so on) while the more efficient binary construction is closest to what is available in most programming languages.

When several constructions coexist, they often share an axiomatic representation, abstracting away from the internal details. In Coq, it is possible to do proofs directly on the axiomatic representation thanks to the module and functor system [1]. While this has the advantage of factoring proofs, it also makes

[3] This plugin introduces a new tactic called `exact modulo`. Its most recent version is available on the web at https://github.com/Zimmi48/transfer.

the proof harder as it does not allow taking advantage of the specifics of the implementation.

The purpose of this work is to make easy to transport theorems to all isomorphic constructions even when the proof relies on one particular such construction. In an informal setting, the mathematician would declare that "we can identify the two structures" once she has proved they were isomorphic and would proceed from there. Our goal is to justify that claim because it will be that missing justification that the proof checker will ask for. Moreover, we need to determine when this justification is missing and insert it automatically.

Although we focus on isomorphic structures in our description of the problem and in our examples, we want to emphasize that we thrive to be as general as possible and require as little as possible to allow the automatic transfer of a theorem. Sometimes an isomorphism is required but sometimes a weaker correspondence is sufficient. Our algorithm will typically allow the following transfer:

Example 1. Take two sets A and A'. If we have the following result on the first set:

Axiom 1 (A is empty).

$$\forall x \in A, \bot \ .$$

then a surjective function $f : A \to A'$ is all we need to transfer the result and get:

Theorem 1 (A' is empty).

$$\forall x' \in A', \bot \ .$$

Here is the complete corresponding Coq development (using our plugin – although in that case, it is extremely easy to build the proof by hand):

```
Parameter A A' : Set.
Axiom emptyA : ∀ x : A, False.
Parameter f : A → A'.
Parameter g : A' → A.
Axiom surjf : ∀ x' : A', f (g x') = x'.
Declare Surjection f by (g, surjf).
Theorem emptyA' : ∀ x' : A', False.
  exact modulo emptyA.
Qed.
```

In the remainder of this text, we will start by presenting our current algorithm which is able to transfer a limited but already interesting set of theorems. Then, we will detail our ideas to generalize it. Finally, we will compare our approach to previous related works.

2 How to Transfer a Theorem

To start, we are limiting ourselves to transferring first-order formulas containing only universal quantifiers, implication and relations.

2.1 User-provided declarations

We only require from the user to provide a set of surjective functions between related data-types, along with a proof of surjectivity, and transfer lemmas. That is, we can relate two data-types A and A' by producing a function $f : A \to A'$ and a proof that f is surjective. To ease our task, we will require that the proof that f is surjective be given by producing a right-inverse[4] g and a proof that

$$\forall x' \in A', f(g(x')) = x' \ .$$

If the user wishes to transfer a relation $R \in A \times A \times \ldots \times A$ to a relation $R' \in A' \times A' \times \ldots \times A'$, she must provide a transfer lemma of the form

$$\forall x_1 \ldots x_n \in A, R(x_1, \ldots, x_n) \Rightarrow R'(f(x_1), \ldots, f(x_n))$$

where f is called the transfer function between R and R'.

The declared surjections and transfer lemmas will be stored in tables (maps). A given surjection can be retrieved by looking for a pair of data-types while a given transfer lemma can be retrieved by looking for a pair of relations. There can be only one stored item for each key which prevents defining several distinct isomorphisms between two structures.

Example 2 shows how this is enough for transferring interesting theorems from one data-type to another.

Example 2. Suppose we are given two data-types to represent $\mathbb{N}$, called nat and N together with two relations $\leq_{\text{nat}}$ and $\leq_{\text{N}}$.

We know nothing of their implementation but we are also given two functions N.to_nat : N $\to$ nat and N.of_nat : nat $\to$ N and the four accompanying axioms:

Axiom 2 (Surjectivity of N.to_nat).

$$\forall x \in \text{nat}, \text{N.to_nat}(\text{N.of_nat}(x)) = x \ .$$

Axiom 3 (Surjectivity of N.of_nat).

$$\forall x' \in \text{N}, \text{N.of_nat}(\text{N.to_nat}(x')) = x' \ .$$

Axiom 4 (Transfer from $\leq_{\text{N}}$ to $\leq_{\text{nat}}$ by N.to_nat).

$$\forall x', y' \in \text{N}, x' \leq_{\text{N}} y' \Rightarrow \text{N.to_nat}(x') \leq_{\text{nat}} \text{N.to_nat}(y') \ .$$

Axiom 5 (Transfer from $\leq_{\text{nat}}$ to $\leq_{\text{N}}$ by N.of_nat).

$$\forall x, y \in \text{nat}, x \leq_{\text{nat}} y \Rightarrow \text{N.of_nat}(x) \leq_{\text{N}} \text{N.of_nat}(y) \ .$$

Finally, we are given the following result to transfer:

---

[4] In other words, using terminology of category theory, we ask that g be a section of f and f be a retraction of g.

Axiom 6 (Transitivity of $\leq_{\text{nat}}$).

$$\forall x, y, z \in \text{nat}, x \leq_{\text{nat}} y \Rightarrow y \leq_{\text{nat}} z \Rightarrow x \leq_{\text{nat}} z \ .$$

All these results enable us indeed to transfer Axiom 6 into Theorem 6.

Theorem 6 (Transitivity of $\leq_{\text{N}}$).

$$\forall x', y', z' \in \text{N}, x' \leq_{\text{N}} y' \Rightarrow y' \leq_{\text{N}} z' \Rightarrow x' \leq_{\text{N}} z' \ .$$

Proof. Let $x', y', z' \in \text{N}$ and assume that the following two hypotheses hold:

$$x' \leq_{\text{N}} y' \ , \tag{1}$$

$$y' \leq_{\text{N}} z' \ . \tag{2}$$

From (1) (respectively (2)) and Axiom 4, we draw

$$\text{N.to_nat}(x') \leq_{\text{nat}} \text{N.to_nat}(y') \ , \tag{3}$$

$$\text{N.to_nat}(y') \leq_{\text{nat}} \text{N.to_nat}(z') \ . \tag{4}$$

We can now apply Axiom 6 to $\text{N.to_nat}(x')$, $\text{N.to_nat}(y')$ and $\text{N.to_nat}(z')$ and conclude

$$\text{N.to_nat}(x') \leq_{\text{nat}} \text{N.to_nat}(z') \ . \tag{5}$$

We then apply Axiom 5 to get

$$\text{N.of_nat}(\text{N.to_nat}(x')) \leq_{\text{N}} \text{N.of_nat}(\text{N.to_nat}(z')) \ . \tag{6}$$

That is (rewriting with Axiom 3):

$$x' \leq_{\text{N}} z' \ . \tag{7}$$

□

You will have noticed that Axiom 2 has not been useful here. It would have been if there had been a quantification to transfer inside one of the hypotheses. This suggests a similar example where Axiom 2 would not hold, thus where there would be no isomorphism between the two related data-types. Such an example is provided in the repository containing the plugin: we transfer various theorems (such as transitivity of $\leq$) from $\mathbb{Z}$ to $\mathbb{N}$.

2.2 Preliminaries in type-theory-based logic

Understanding the proposed algorithm will not require much knowledge about the internals of Coq:

- Dependent products are the way in which the Calculus of Inductive Constructions [3, Ch. 4], the logical base of Coq, models both universal quantification and implication. The implication is just the degenerate non-dependent case, i.e. $A \Rightarrow B$ is just an abbreviation for $\forall x : A, B$ when x does not appear in B.
- In the Calculus of Inductive Constructions as well as in any other type-theory-based logic, proofs can be viewed as programs, and in particular the proof $\rho_{A \Rightarrow B}$ of an implication $A \Rightarrow B$ can be viewed as a function that takes a proof ρ_A of A as argument and produces a proof $\rho_{A \Rightarrow B}(\rho_A)$ of B.

2.3 The algorithm

Algorithm 1 takes as input two formulas (called *theorem* and *goal*) differing only in the data-types that are quantified over and in the relations they contain, as well as a proof of *theorem*. It outputs a proof of *goal* provided that the differences between the two formulas all correspond to previously declared surjections and transfer lemmas.

The algorithm is recursive over the structure of the two formulas (which must be the same). There are two main cases: when the formulas are atoms (i.e. in our case, relations applied to arguments) or dependent products.

You will have noticed, at line 25 of Algorithm 1, the strange choice of substituting x' with $f(g(x'))$ only in covariant places. As $x' = f(g(x'))$, we could have done the substitution wherever we liked. We do it only in covariant places so that the formulas in the recursive calls will have exactly the right form when reaching the atomic case (relations). One can convince oneself that substituting in covariant places is enough by observing what it gives on the last example (transitivity of $\leq_{\mathrm{N}}$) while remembering that the right-hand side of an implication is covariant while the left-hand side is contravariant.

We could add support for logical connectives such as $\land$ and $\lor$ or the existential quantifier $\exists$ but as they play no specific role in the Calculus of Inductive Constructions (unlike universal quantification and implication), we rather want a more general way of treating any such addition. As for the negation $\neg A$, in Coq it is defined as $A \Rightarrow \bot$ so it is already supported provided we unfold its definition first.

3 Generalizing

Algorithm 1 has quite a lot of limitations at the moment which we plan to lift.

Functions. So far we have considered only relations. Even though any function can be expressed as a relation, this path would require a lot of preliminary rewriting steps; thus it would be a lot more convenient to be able to transfer functions directly. Given that relations are represented as functions to the special sort `Prop` in Coq, what we need is a generalization where functions to any type, as well as internal operators, would be supported.

New connectives. We want to be able to handle logical connectives such as $\land$ and $\lor$ but also various other combinators and non-propositional functions. For instance, we should be able to transfer theorems involving equality.

Other equivalence relations. Currently, Leibniz (structural) equality plays a special role as it has to appear in the surjection lemmas. Leibniz equality has the advantage of allowing rewriting in any subterm. But techniques have already been devised [8] to allow rewriting with other equivalence relations and we plan to inspire from them.

Algorithm 1 Transfer a Theorem

Precondition: In the environment Γ, F and F' are two well-defined formulas and ρ_F is a proof of F.

Postcondition: $\textsc{ExactModulo}(\Gamma, F, F', \rho_F)$ is a proof of F' in environment Γ or it is a failure.

function $\textsc{ExactModulo}(\Gamma, F, F', \rho_F)$
  if $F = F'$ **then**
    return ρ_F
5: **else if** $F = R(t_1, \ldots, t_n)$ **and** $F' = R'(t'_1, \ldots, t'_n)$ **then**
    $f \leftarrow$ transfer function between R and R' ▷ **return** failure if it does not exist
    $\rho_{\text{transfer}} \leftarrow$ proof of compatibility of f with respect to R and R'
    for $i \leftarrow 1$ to n **do**
10:       **if** $t'_i \neq f(t_i)$ **then**
        return failure
    return $\rho_{\text{transfer}}(t_1, \ldots, t_n, \rho_F)$
  else if $F = \forall x : A, B$ **and** $F' = \forall x' : A', B'$ **then**
    $\Gamma \leftarrow \Gamma, x' : A'$
15:     $t \leftarrow \textsc{ExactModulo}(\Gamma, A', A, x')$
    if $t \neq$ failure **then**
      $\rho_{\text{rec}} \leftarrow \textsc{ExactModulo}(\Gamma, B, B', \rho_F(t))$ ▷ **return** failure if $\rho_{\text{rec}} =$ failure
      return $\lambda x' : A'. \rho_{\text{rec}}$
20:     **else**
      $f \leftarrow$ surjection from A to A' ▷ **return** failure if it does not exist
      $g \leftarrow$ right-inverse of f
      $\rho_{\text{surjection}} \leftarrow$ proof that g is a right-inverse of f
      $B_{\text{subst}} \leftarrow B$ where x was replaced by $g(x')$
25:       $B'_{\text{subst}} \leftarrow B'$ where x' was replaced by $f(g(x'))$ in covariant places
      $\rho_{\text{rec}} \leftarrow \textsc{ExactModulo}(\Gamma, B_{\text{subst}}, B'_{\text{subst}}, \rho_F(g(x')))$ ▷ **return** failure if $\rho_{\text{rec}} =$ failure
      Now $\lambda x' : A'. \rho_{\text{rec}}$ is a proof of $\forall x' : A', B'_{\text{subst}}$. With the help of $\rho_{\text{surjection}}$ we can transform it into $\rho_{F'}$ a proof of $\forall x' : A', B'$.
      return $\rho_{F'}$
30: **else**
    return failure

No right-inverse. For simplicity, we have asked so far for proofs of surjectivity which involved producing a right-inverse. This has a major drawback. Indeed, surjectivity is equivalent to having a right-inverse only if we admit the Axiom of Choice. We want our algorithm to be as general as possible, therefore we will work to remove that requirement.

3.1 Generalizing Declarations

Transfer lemmas. The Coq Morphisms library[5] introduces a new notion of respectful morphisms for a binary homogeneous relation. We draw from [2] the idea of using the generalized heterogeneous version for our transfer declarations. Heterogeneous relations bring us the ability to relate objects from one data-type with objects from another data-type.

We will note

```
(R ##> R') f g := ∀ (x : X) (y : Y), R x y → R' (f x) (g y) .
```

This can also be seen as a (commutative) diagram.

$$\begin{array}{ccc} X & \overset{R}{\longleftrightarrow} & Y \\ f\downarrow & & \downarrow g \\ X' & \overset{R'}{\longleftrightarrow} & Y' \end{array}$$

It is easy to show that this corresponds precisely to a very general notion of homomorphism that can be found in mathematics textbooks such as [7, Ch. 5.7]. The pair of mappings (f, g) is a homomorphism between the two "structures" $(X \times Y, R)$ and $(X' \times Y', R')$ if the following holds:

$$R \circ g \subseteq f \circ R'$$

where $\circ$ is the relational composition, i.e.

$$\forall x \in X, y' \in Y', [(R \circ g)(x, y') \Leftrightarrow \exists y \in Y, R(x, y) \wedge g(y) = y'] \quad ,$$

$$\forall x \in X, y' \in Y', [(f \circ R')(x, y') \Leftrightarrow \exists x' \in Y, f(x) = x' \wedge R'(x', y')] \quad .$$

It will be possible to declare all sorts of transfer lemmas thanks to the respectful arrow as can be seen in the following example.

Example 3. Let us consider a heterogeneous binary relation `natN` relating elements of `nat` with elements of `N`. One possible definition would be:

```
Definition natN x x' := N.of_nat x = x'.
```

Then, we can declare how to transfer various functions and relations:

[5] The Coq Morphisms library is part of the work of Matthieu Sozeau [8] to generalize rewriting for equivalence relations that are not Leibniz equality. Its documentation is available online at https://coq.inria.fr/library/Coq.Classes.Morphisms.html.

```
Theorem le_transfer : (natN ##> natN ##> impl) le N.le.
```

where `le` represents $\leq_{\text{nat}}$, `N.le` represents $\leq_{\text{N}}$ and `impl` is a relation corresponding to the implication (also, note that `##>` is right-associative). That is, after unfolding the definitions of `natN`, `##>` and `impl`:

```
Theorem le_transfer :
    ∀ (x : nat) (x' : N), N.of_nat x = x' →
    ∀ (y : nat) (y' : N), N.of_nat y = y' → le x y → N.le x' y'.
```

Considering two new Boolean functions `iszero_nat` and `iszero_N`, we can make explicit how they relate in the following way:

```
Theorem iszero_transfer : (natN ##> @eq bool) iszero_nat iszero_N.
```

where `@eq bool` is the Boolean equality.

Finally, considering two operations `Nat.add` and `N.add`:

```
Theorem plus_transf : (natN ##> natN ##> natN) Nat.add N.add.
```

Surjection lemmas. That very same idea of respectful morphisms can be used to replace the surjection declarations we used so far. Just as we had replaced the implication → by a new relation `impl`, we will use a new relation `@all` to represent ∀ :

```
@all A (λ x : A, B) := ∀ x : A, B .
```

Any surjection declaration in the style of Sec. 2:

```
Declare Surjection f by (g, proof).
```

can be equivalently replaced by the following three declarations:

```
Theorem R_surj : ((R ##> impl) ##> impl) (@all A) (@all A').
Theorem R_tot : ((R⁻¹ ##> impl) ##> impl) (@all A') (@all A).
Theorem R_func : (R ##> R ##> impl) (@eq A) (@eq A').
```

where `R x x' := f x = x'` and `R`$^{-1}$ `x' x := R x x'` .

The first declaration corresponds to the surjectivity of relation R (also called right-totality). The second and third declaration express the fact that R is a mapping. More precisely, the second declaration corresponds to the surjectivity of the inverse relation, that is the (left-)totality of R. The third declaration expresses the knowledge that R is functional (also called univalent in [7, Ch. 5.1] or right-unique elsewhere).

The three declarations provide interesting "point-free" formulations of a relation totality and unicity properties. Let us unfold two of them to give more intuition on what they mean:

```
Theorem R_surj :
    ∀ P P', (∀ (x : A) (x' : A'), R x x' → P x → P' x') →
    (∀ x : A, P x) → ∀ x' : A', P' x'.
Theorem R_func :
    ∀ (x : A) (x' : A'), R x x' →
    ∀ (y : A) (y' : A'), R y y' → x = y → x' = y'.
```

We immediately see that `R_func` indeed expresses that R is functional (each input has at most one output). As for `R_surj`, while it is clearly a necessary condition for surjectivity, we will have to instantiate the theorem with `P = λ _ : A, True` and `P' = λ x' : A', ∃ x : A, R x x'` to see that it is sufficient.

We can already foresee two advantages of this new formulation of surjectivity lemmas. First, it is more general as it will allow considering data-types which are related by a non-functional or non-total relation. Second, we can already imagine replacing `@eq` by any equivalence relation and `@all` by any bounded quantification, thus allowing to relate two partial quotients and not only classic data-types.

3.2 Transfer to the context

In [8], Matthieu Sozeau gives a set of inference rules to find where a rewrite can occur and the proof that the rewrite is correct. Building the proof will sometimes require prior declarations that some functions are respectful morphisms for some homogeneous relations. For our purpose, we need to generalize these rules to heterogeneous relations.

As before, we take a theorem and a goal as arguments and we must produce a proof of `thm → goal`, that is `impl thm goal`. We borrow the notation

$$\Gamma \vdash \tau \rightsquigarrow_p^R \tau'$$

which means that given an environment Γ in which τ and τ' are well-defined, p is a proof of $R(\tau, \tau')$.

Initially, given a theorem $\Gamma \vdash \tau$ and a goal $\Gamma \vdash \tau'$, we want to derive a judgment of the form:

$$\Gamma \vdash \tau \rightsquigarrow_p^{\texttt{impl}} \tau'$$

Rules. We give in Fig. 1 the rules to get to that judgment, adapted from [8]. We have dropped the Unify rule as it was used for rewriting but does not apply in our case. To avoid unnecessary complexity, we have also chosen to drop the Sub rule in a first version.

From these rules, we plan to derive a deterministic algorithm, which we will implement and test.

We will now illustrate each of these rules by a few examples, taken from the transfer of Axiom 6 (transitivity of $\leq_{\text{nat}}$) to Theorem 6 (transitivity of $\leq_{\text{N}}$).

$$\frac{p : R(\tau,\tau') \in \Gamma}{\Gamma \vdash \tau \leadsto_p^R \tau'}\ \textsc{Env} \qquad \frac{p : R(\tau,\tau') \in \text{Tables}}{\Gamma \vdash \tau \leadsto_p^R \tau'}\ \textsc{Table}$$

$$\frac{\Gamma, x : \tau_1, x' : \tau_1', H : R(x,x') \vdash \tau_2 \leadsto_p^S \tau_2'}{\Gamma \vdash \lambda x : \tau_1 . \tau_2 \leadsto_{\lambda x:\tau_1, x':\tau_1', H:R(x,x').p}^{R\ \#\#>\ S} \lambda x' : \tau_1' . \tau_2'}\ \textsc{Lambda}$$

$$\frac{\Gamma \vdash f \leadsto_{p_{\mathtt{f}}}^{R\ \#\#>\ S} f' \quad \Gamma \vdash e \leadsto_{p_{\mathtt{e}}}^{R} e'}{\Gamma \vdash f(e) \leadsto_{p_{\mathtt{f}}\,(e,e',p_{\mathtt{e}})}^{S} f'(e')}\ \textsc{App}$$

$$\frac{\Gamma \vdash @\mathtt{all}\ \tau_1\ (\lambda x : \tau_1 . \tau_2) \leadsto_p^R @\mathtt{all}\ \tau_1'\ (\lambda x' : \tau_1' . \tau_2')}{\Gamma \vdash \forall x : \tau_1, \tau_2 \leadsto_p^R \forall x' : \tau_1', \tau_2'}\ \textsc{Forall}$$

$$\frac{\Gamma \vdash \mathtt{impl}\ \tau_1\ \tau_2 \leadsto_p^R \mathtt{impl}\ \tau_1'\ \tau_2'}{\Gamma \vdash \tau_1 \to \tau_2 \leadsto_p^R \tau_1' \to \tau_2'}\ \textsc{Arrow}$$

Fig. 1. `exact modulo` inference rules.

Example 4. Initially, we want to find a judgment of the form

$$\begin{array}{rl} \vdash & \forall x,y,z \in \text{nat},\ x \leq_{\text{nat}} y \ \Rightarrow\ y \leq_{\text{nat}} z \ \Rightarrow\ x \leq_{\text{nat}} z \\ \leadsto^{\mathtt{impl}} & \forall x',y',z' \in \text{N},\ x' \leq_{\text{N}} y' \ \Rightarrow\ y' \leq_{\text{N}} z' \ \Rightarrow\ x' \leq_{\text{N}} z'\ . \end{array}$$

By rule Forall, this reduces to

$$\begin{array}{rl} \vdash & @\mathtt{all}\ \text{nat}\ (\lambda x : \text{nat},\ \forall y,z \in \text{nat},\ x \leq_{\text{nat}} y \Rightarrow\ y \leq_{\text{nat}} z \Rightarrow\ x \leq_{\text{nat}} z) \\ \leadsto^{\mathtt{impl}} & @\mathtt{all}\ \ \text{N}\ (\lambda x' : \text{N},\ \ \forall y',z' \in \text{N},\ x' \leq_{\text{N}} y' \Rightarrow\ y' \leq_{\text{N}} z' \Rightarrow\ x' \leq_{\text{N}} z')\ . \end{array}$$

By rule App, this reduces to

$$\begin{array}{rl} \vdash & \lambda x : \text{nat},\ \forall y,z \in \text{nat},\ x \leq_{\text{nat}} y \Rightarrow\ y \leq_{\text{nat}} z \Rightarrow\ x \leq_{\text{nat}} z \\ \leadsto^{R} & \lambda x' : \text{N},\ \ \forall y',z' \in \text{N},\ x' \leq_{\text{N}} y' \Rightarrow\ y' \leq_{\text{N}} z' \Rightarrow\ x' \leq_{\text{N}} z'\ , \end{array} \tag{8}$$

$$\vdash \quad @\mathtt{all}\ \text{nat} \leadsto^{R\ \#\#>\ \mathtt{impl}} \quad @\mathtt{all}\ \text{N}\ . \tag{9}$$

Then (9) is solved by applying rule Table. We get $R = \mathtt{natN\ \#\#>\ impl}$. Finally, we can report the value of R in (8) and apply rule Lambda and thus our initial problem reduces to

$$\begin{array}{l} x : \text{nat}, x' : \text{N}, H : \mathtt{natN}\ x\ x' \vdash \forall y,z \in \text{nat},\ x \leq_{\text{nat}} y \ \Rightarrow\ y \leq_{\text{nat}} z \ \Rightarrow\ x \leq_{\text{nat}} z \\ \qquad\qquad\qquad\qquad \leadsto^{\mathtt{impl}} \forall y',z' \in \text{N},\ x' \leq_{\text{N}} y' \ \Rightarrow\ y' \leq_{\text{N}} z' \ \Rightarrow\ x' \leq_{\text{N}} z'\ . \end{array}$$

From now on,

$$\begin{array}{rl} \Gamma \ = & x : \text{nat},\ x' : \text{N},\ H : \mathtt{natN}\ x\ x', \\ & y : \text{nat},\ y' : \text{N},\ H_1 : \mathtt{natN}\ y\ y', \\ & z : \text{nat},\ z' : \text{N},\ H_2 : \mathtt{natN}\ z\ z'\ \ . \end{array}$$

We now consider the problem of finding a judgment of the form

$$\begin{array}{rl} \Gamma \vdash & x \le_{\text{nat}} y \Rightarrow\ y \le_{\text{nat}} z \Rightarrow\ x \le_{\text{nat}} z \\ \leadsto^{\texttt{impl}} & x' \le_{\text{N}} y' \Rightarrow\ y' \le_{\text{N}} z' \Rightarrow\ x' \le_{\text{N}} z' \ . \end{array}$$

By rule Impl, this reduces to

$$\begin{array}{rl} \Gamma \vdash & \texttt{impl}\ (x \le_{\text{nat}} y)\ (y \le_{\text{nat}} z \Rightarrow\ x \le_{\text{nat}} z) \\ \leadsto^{\texttt{impl}} & \texttt{impl}\ (x' \le_{\text{N}} y')\ (y' \le_{\text{N}} z' \Rightarrow\ x' \le_{\text{N}} z') \ . \end{array}$$

By rule App, this reduces to

$$\Gamma \vdash y \le_{\text{nat}} z \Rightarrow x \le_{\text{nat}} z \leadsto^{R} y' \le_{\text{N}} z' \Rightarrow x' \le_{\text{N}} z' \ , \tag{10}$$

$$\Gamma \vdash \texttt{impl}\ (x \le_{\text{nat}} y) \leadsto^{R \#\#> \texttt{impl}} \texttt{impl}\ (x' \le_{\text{N}} y') \ . \tag{11}$$

By rule App, (11) reduces again to

$$\Gamma \vdash x \le_{\text{nat}} y \leadsto^{S} x' \le_{\text{N}} y' \ , \tag{12}$$

$$\Gamma \vdash \texttt{impl} \leadsto^{S \#\#> R \#\#> \texttt{impl}} \texttt{impl} \ . \tag{13}$$

We will make sure that the tables are pre-filled so that judgments such as (13) can be solved with rule Table. In that case, we will get $S = \texttt{impl}^{-1}$ and $R = \texttt{impl}$. Now by rule App, (12) reduces to

$$\Gamma \vdash y \leadsto^{T} y' \ , \tag{14}$$

$$\Gamma \vdash \texttt{le}\ x \leadsto^{T \#\#> \texttt{impl}^{-1}} \texttt{N.le}\ x' \ . \tag{15}$$

Rule Env allows us to derive (14) with $T = \texttt{natN}$.

As for (15), it can be solved after a few more steps by using the knowledge that `(natN ##> natN ##> impl`$^{-1}$`) le N.le`, which is equivalent to `(natN`$^{-1}$ `##> natN`$^{-1}$ `##> impl) N.le le`, which will be one of the user-provided transfer lemmas (it corresponds to Axiom 4). Therefore, there only remains to solve (10) in ways similar to this example.

4 Related work

4.1 Proof reuse

More than ten years ago, Nicolas Magaud [6] proposed an extension of Coq that seemed to share our objectives. Notably, he was able to transfer all the theorems that were, at the time, in the standard Arith library, from `nat` to `N`.

The approach was quite intricate because it was able to transfer proofs, and not just theorems. Given two isomorphic data-types, one will be considered as the *origin type* and the other one as the *target type*. The first step is to define functions to model the origin constructors within the target type. Moreover, new

recursion operators behaving like the ones of the origin type are added to the target type.

With such a projection of the origin type into the target type, it is easy to project operators and relations. Proofs are transferred in the same way. The last step is to establish extensional equality between projected operators and the corresponding native operators of the target type.

While interesting, we do not need to take such a complicated path for our objective which is only *theorem reuse*. Using Magaud's approach requires much more work in establishing the relations between the two data-types. Moreover, our approach is more powerful in a sense: we can transfer properties between two data-types even if we know nothing of their content and the transfer lemmas where provided as axioms.

4.2 Algorithm reuse

A much more recent work by Cohen et al. [2] has been of much inspiration to us. However, the focus is not the same. In the context of program verification, the authors propose a general method for algorithm reuse through parametricity when refining proof-oriented data-types into efficient computation-oriented data-types. Parametricity then enables the automatic transfer of algorithm correctness proofs. Although they give this general method, they explain why they do not provide a plugin. Our focus being on transparency and usability by mathematicians, we decided to create such a plugin.

An other inspiring characteristic of their work lies in that they typically allow refined types to contain more objects, including objects which would have no meaning (no specification). Although we currently require precisely the opposite so as to be able to translate theorems stating properties *for all* elements, including unicity properties, we could quite easily add support for bounded quantification. Bounded quantification would be useful for transferring theorems from a subset type to the corresponding elements of a larger type (for instance from $\mathbb{N}$ to non-negative elements of $\mathbb{Z}$). Similarly, the new way to declare links between two data-types presented in Sec. 3.1 makes it easy to use other equivalence relations than just Leibniz equality.

4.3 Other works proposing a heterogeneous respectful arrow

While Cohen et al. [2] inspired us to use a generalized heterogeneous respectful arrow to allow for more precise transfer declarations and remove the limitations of Algorithm 1, there are many other (and sometimes older) works proposing the same definition. One example of such a work is [4, Def. 13]. But this is not surprising as we have remarked in Sec. 3.1 that this arrow just encodes for an already existing mathematical notion of homomorphism.

Huffman and Kunčar [5] go further as they also show how the relational unicity and totality properties can be expressed in terms of the respectful arrow. They produced a Transfer package for Isabelle/HOL with comparable objectives to ours, and their `transfer` tactic is based on a two-step algorithm sharing

many ideas with Matthieu Sozeau's [8]. Nothing going as far as their Transfer package has yet been created for Coq.

5 Conclusion

In this paper, we have shown how a simple algorithm can make use of a few initial declarations to ease the reuse of results from one data-type to another.

As we improve our algorithm and become able to transfer more theorems, we will still have a lot to do in order to make our plugin as simple-to-use as possible. A first easy step will be to transform our `exact modulo` tactic into an `apply modulo` tactic. Then, we will need to allow for compositionality in ways similar to [2] and [5]. First, by allowing and handling transfer declarations for parametrized types. Then, by finding paths from one type to another, even when the relation between the two was not declared, but can be established by going through a sequence of transfers.

We view this work as a little but quite interesting step in the enormous task of making the use of a formal proof system as easy as a pen-and-paper proof.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments.

References

1. Jacek Chrzaszcz. Implementing modules in the Coq system. In *Theorem Proving in Higher Order Logics*, pages 270–286. Springer, 2003.
2. Cyril Cohen, Maxime Dénes, and Anders Mörtberg. Refinements for free! In *Certified Programs and Proofs*, pages 147–162. Springer, 2013.
3. Coq development team. *The Coq proof assistant reference manual.* Inria, 2015. Version 8.5.
4. Peter V Homeier. A design structure for higher order quotients. In *Theorem Proving in Higher Order Logics*, pages 130–146. Springer, 2005.
5. Brian Huffman and Ondřej Kunčar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In *Certified Programs and Proofs*, pages 131–146. Springer, 2013.
6. Nicolas Magaud. Changing data representation within the Coq system. In *Theorem Proving in Higher Order Logics*, pages 87–102. Springer, 2003.
7. Gunther Schmidt. *Relational mathematics*, volume 132 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 2011.
8. Matthieu Sozeau. A new look at generalized rewriting in type theory. *J. Formalized Reasoning*, 2(1):41–62, 2009.