Efficient Program Extraction in Elementary Number Theory using the Proof Assistant Minlog

Franziskus Wiesnet[1]

Vienna University of Technology, Vienna, Austria, European Union
franziskus.wiesnet@tuwien.ac.at

Abstract

Overview We examine theorems of elementary number theory using a constructive, computational, and proof-theoretic approach. Key theorems include Euclid's proof of the infinitude of prime numbers, Bézout's identity, the Fundamental Theorem of Arithmetic, and Fermat's factorization method. These theorems have been formalized within the proof assistant Minlog, providing rigorous formal verification. Minlog allows the extraction of executable Haskell programs from proofs, demonstrating their computational utility. The implementations can be found in the folder `examples/arith` of the Minlog directory. Currently these files are only available and functional in the dev branch of Minlog. Instructions on how to install Minlog and switch to the dev branch can be found on the Minlog website [14]. All relevant files are also available in a GitHub repository (https://github.com/FranziskusWiesnet/MinlogArith/).

We explore interesting and instructive aspects of the proofs, the implementation in Minlog, and the extracted Haskell term. Additionally, we will highlight the unique features and advantages of Minlog as a proof assistant, offering the audience a deeper insight into its capabilities. The main focus of the presentation will be on the Fundamental Theorem of Arithmetic and Fermat's factorization method. The second one can be extended to the quadratic sieve, which is a sub-exponential factorization method. An implementation of the quadratic sieve in Minlog is the subject of future research.

While prior knowledge of Minlog, Haskell, or other programming languages can be helpful, it is not required, as we will provide all necessary explanations.

Innovations and Methodology While Minlog has primarily been used for the formalization of analysis, this study serves as a case study demonstrating that Minlog is also highly suitable for algorithmic number theory. The only article dealing with number theory (in this case, the greatest common divisor) in Minlog is [6]. However, that work primarily focused on the extraction of programs from classical proofs. Therefore, this new implementation can serve as a solid foundation for the formalization of number theory in Minlog.

Another innovation is that we have considered the efficiency of the extracted term right from the formulation of theorems and their proofs. Therefore, the proofs were formalized in two different versions. The first version is based on unary natural numbers, defined by 0 and the successor function. In contrast, the second version is based on positive binary numbers given by 1 and two successor functions, one appending 0 and the other appending 1. While the first version focuses primarily on the simplicity of the theorems and their proofs, the second version prioritizes the efficiency of the extracted term. Hence, we demonstrate how formulating theorems and structuring their proofs influence the efficiency of the extracted algorithms, bridging the gap between formal verification and practical applications. Furthermore, extracting proofs as Haskell programs (rather than as terms in Minlog) will also contribute to shortening the runtime of the extracted algorithms.

Background Minlog is a proof assistant developed in the 1990s by the logic group at the Ludwig-Maximilians-Universität München under the direction of Helmut Schwichtenberg [4, 13, 14, 15, 20, 21]. Several introductions to Minlog are available [31, 32, 33]. Recently, Minlog has predominantly been employed in the field of constructive analysis [5, 11, 16, 23, 26, 35]. However, there exists a broad spectrum of proofs in diverse domains that have been implemented in Minlog [2, 8, 22, 24, 25]. All of them share the characteristic that an implementation in Minlog not only focuses on the correctness of the proof but also on extracting programs from the proofs. As the primary goal of Minlog is to interpret proofs as programs and to work with them accordingly [1], it includes all the functions for the formal program extraction from proofs.

Minlog and the formal program extraction from proofs are based on an extension of HA^ω called *Theory of Computable Functionals* (TCF) as meta-theory. Its origins go back to Scott's seminal work on *Logic of Computable Functionals* [28] and Platek's PhD thesis [18], and it incorporates basic concepts introduced by Kreisel [10] and Troelsta [30]. In recent years the Munich team have made important progress in the development of TCF [3, 7, 9, 17, 19]. It is based on partial continuous functionals and information systems, serving as their designated model, known as *Scott model* [12, 27].

Illustrative examples. The **natural numbers** $\mathbb{N}$ are given by the constructors

$$0 : \mathbb{N}, \quad \mathrm{S} : \mathbb{N} \to \mathbb{N},$$

and the **positive binary numbers** $\mathbb{P}$ are given by

$$1 : \mathbb{P}, \quad \mathrm{S}_0 : \mathbb{P} \to \mathbb{P}, \quad \mathrm{S}_1 : \mathbb{P} \to \mathbb{P}.$$

Note that the binary representation is reversed. For example, $\mathrm{S}_0\,\mathrm{S}_1\,1$ represents the number 6, which corresponds to binary 110. The reason for starting with 1 instead of 0 is primarily to ensure that each number is uniquely determined by its constructors. Otherwise, for example 0 and $\mathrm{S}_0\,0$ would represent the same number, violating this uniqueness.

Already the definition of the **greatest common divisor** clearly illustrates the computational difference between the number types. On natural numbers, the greatest common divisor can be defined by the euclidean algorithm:

$$\begin{aligned} \gcd(0,n) &:= \gcd(n,0) := n \\ \gcd(\mathrm{S}\,m, \mathrm{S}\,n) &:= \begin{cases} \gcd(\mathrm{S}\,m, n-m) & \text{if } m < n \\ \gcd(m-n, \mathrm{S}\,n) & \text{otherwise} \end{cases} \end{aligned}$$

It should be noted that using division with remainder on the natural numbers would not make this algorithm more efficient, because the computation of division with remainder results essentially in an iterated application of the step case in the definition above.

On positive binary numbers, however, the greatest common divisor can be defined much more efficiently by *Stein's algorithm* (named after Josef Stein [29]):

$$\begin{aligned} \gcd(1,p) := \gcd(\mathrm{S}_1\,p, 1) &:= \gcd(\mathrm{S}_0\,p, 1) := 1 \\ \gcd(\mathrm{S}_0\,p, \mathrm{S}_0\,q) &:= \mathrm{S}_0(\gcd(p,q)) \\ \gcd(\mathrm{S}_0\,p, \mathrm{S}_1\,q) := \gcd(\mathrm{S}_1\,p, \mathrm{S}_0\,q) &:= \gcd(p, \mathrm{S}_1\,q) \\ \gcd(\mathrm{S}_1\,p, \mathrm{S}_1\,q) &:= \begin{cases} \gcd(\mathrm{S}_1\,p, q-p) & \text{if } p < q \\ \gcd(p-q, \mathrm{S}_1\,q) & \text{if } q < p \\ \mathrm{S}_1\,p & \text{otherwise.} \end{cases} \end{aligned}$$

One can see that in Stein's algorithm, at each step, at least one argument is reduced by one digit and is therefore at least halved. Noting that the subtraction of two binary numbers has approximately linear runtime in the number of digits, Stein's algorithm therefore has quadratic runtime in the number of digits. In contrast, the Euclidean algorithm on natural numbers has quadratic runtime only with respect to the absolute size of the arguments.

Bézout's identity is known as the statement that the greatest common divisor is a linear combination of the two arguments. That is, for integers a, b there are integers u, v with $\gcd(a,b) = u \cdot a + v \cdot b$. During the formalization, we wanted to remain within the respective number system, so the theorem had to be reformulated as

$$\forall_{n,m} \exists_{l_0} \exists_{l_1} . \quad \gcd(n,m) + l_0 \cdot n = l_1 \cdot m \quad \vee \quad \gcd(n,m) + l_0 \cdot m = l_1 \cdot n$$

for natural numbers, and as

$$\begin{aligned} \forall_{p_0,p_1} . \quad & \exists_q \; q \cdot p_0 = p_1 \quad \vee \quad \exists_q \; q \cdot p_1 = p_0 \\ \vee \quad & \exists_{q_0} \exists_{q_1} \; \gcd(p_0,p_1) + q_0 \cdot p_0 = q_1 \cdot p_1 \\ \vee \quad & \exists_{q_0} \exists_{q_1} \; \gcd(p_0,p_1) + q_1 \cdot p_1 = q_0 \cdot p_0 \end{aligned}$$

for positive binary numbers. Note that, since integers in Minlog are defined as a type sum of negative binary numbers, zero, and positive binary numbers, the case distinction would be implicitly present in the proof anyway.

The proofs were carried out by induction over an upper bound of the numbers, which is seen as a natural number. In the induction step itself, a case distinction was made based on the structure of the numbers. Specifically for positive binary numbers, this means that the statement

$$\begin{aligned} \forall_{l,p_0,p_1} . \quad & p_0 + p_1 < l \quad \to \\ & \exists_q \; q \cdot p_0 = p_1 \quad \vee \quad \exists_q \; q \cdot p_1 = p_0 \\ \vee \quad & \exists_{q_0} \exists_{q_1} \; \gcd(p_0,p_1) + q_0 \cdot p_0 = q_1 \cdot p_1 \\ \vee \quad & \exists_{q_0} \exists_{q_1} \; \gcd(p_0,p_1) + q_1 \cdot p_1 = q_0 \cdot p_0 \end{aligned}$$

was proven by induction on $l : \mathbb{N}$ and case distinction on $p_0 \in \{1, \mathrm{S}_0\, q_0, \mathrm{S}_1\, q_0\}$ and $p_1 \in \{1, \mathrm{S}_0\, q_1, \mathrm{S}_1\, q_1\}$. In particular, properties of both natural numbers and positive binary numbers were combined. The individual cases are then quite straightforward, but in many instances, a further case distinction according to the four cases in the statement is required.

A similar approach was also chosen for the Fundamental Theorem of Arithmetic and Fermat's factorization method. A detailed presentation can be found in [34].

Acknowledgements. The research for this document was funded by the Austrian Science Fund (FWF) **10.55776/ESP576**.

Many thanks to *Helmut Schwichtenberg* for providing valuable tips on writing the Minlog code and for integrating it into the official Minlog version.

I am grateful to the anonymous reviewers for their thoughtful suggestions. Although space and time constraints prevented me from including all of them, their input was highly appreciated.

References

[1] Holger Benl and Helmut Schwichtenberg. Formal Correctness Proofs of Functional Programs: Dijkstra's Algorithm, a Case Study. In U. Berger and H. Schwichtenberg, editors, *Computational*

Logic, volume 165 of *Series F: Computer and Systems Sciences*, pages 113–126. Proceedings of the NATO Advanced Study Institute on Computational Logic, held in Marktoberdorf, Germany, July 29 – August 10, 1997, Springer Berlin Heidelberg, 1999.

[2] Ulrich Berger, Stefan Berghofer, Pierre Letouzey, and Helmut Schwichtenberg. Program Extraction from Normalization Proofs. *Studia Logica*, 82(1):25–49, February 2006.

[3] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114(1–3):3–25, April 2002.

[4] Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger. *Minlog - A Tool for Program Extraction Supporting Algebras and Coalgebras*, pages 393–399. Springer Berlin Heidelberg, 2011. 4th International Conference, CALCO 2011, Winchester, UK, August 30 – September 2, 2011. Proceedings.

[5] Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Hideki Tsuiki. Logic for Gray-code Computation. In D. Probst and P. Schuster, editors, *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, pages 69–110. De Gruyter, July 2016.

[6] Ulrich Berger and Helmut Schwichtenberg. The greatest common divisor: A case study for program extraction from classical proofs. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, volume 1158, pages 36–46. Springer Berlin Heidelberg, 1996. International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers.

[7] Simon Huber, Basil A. Karádais, and Helmut Schwichtenberg. Towards a Formal Theory of Computability. In R. Schindler, editor, *Ways of Proof Theory: Festschrift for W. Pohlers*, pages 257–282. De Gruyter, December 2010.

[8] Hajime Ishihara and Helmut Schwichtenberg. Embedding classical in minimal implicational logic. *Mathematical Logic Quarterly*, 62(1–2):94–101, January 2016.

[9] Basil A. Karádais. *Towards an Arithmetic for Partial Computable Functionals*. PhD thesis, Ludwig-Maximilians University Munich, 2013.

[10] Georg Kreisel. Interpretation of Analysis by Means of Constructive Functionals of Finite Types. In A. Heyting, editor, *Constructivity in mathematics*, pages 101–128. North-Holland Pub. Co., 1959.

[11] Nils Köpp and Helmut Schwichtenberg. Lookahead analysis in exact real arithmetic with logical methods. *Theoretical Computer Science*, 943:171–186, January 2023.

[12] Kim Guldstrand Larsen and Glynn Winskel. Using information systems to solve recursive domain equations. *Information and Computation*, 91(2):232–258, April 1991.

[13] Kenji Miyamoto. *Program extraction from coinductive proofs and its application to exact real arithmetic*. PhD thesis, Ludwig-Maximilians University Munich, 2013.

[14] Kenji Miyamoto. The Minlog System. https://www.mathematik.uni-muenchen.de/~logik/minlog/, 2024.

[15] Kenji Miyamoto, Fredrik Nordvall Forsberg, and Helmut Schwichtenberg. Program Extraction from Nested Definitions. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 370–385. Springer Berlin Heidelberg, 2013. 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings.

[16] Kenji Miyamoto and Helmut Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structures in Computer Science*, 25(8):1692–1704, November 2014.

[17] Iosif Petrakis. Advances in the Theory of ComputableFunctionals TCF+ due to its Implementation, 2013. online: https://www.math.lmu.de/~petrakis/TCF+.pdf.

[18] Richard Alan Platek. *Foundations of recursion theory*. PhD thesis, Stanford University, 1966.

[19] Helmut Schwichtenberg. Primitive Recursion on the Partial Continuous Functionals. In M. Broy, editor, *Informatik und Mathematik*, pages 251–268. Springer Berlin Heidelberg, 1991.

[20] Helmut Schwichtenberg. Proofs as Programs. In P. Aczel, H. Simmons, and S. Wainer, editors, *Proof Theory: A selection of papers from the Leeds Proof Theory Programme 1990*, page 79–114,

Cambridge, 1993. Cambridge University Press. Title from publisher's bibliographic system (viewed on 24 Feb 2016).

[21] Helmut Schwichtenberg. Minlog. In Freek Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600, pages 151–157. Springer Berlin Heidelberg, 2006.

[22] Helmut Schwichtenberg. Program Extraction from Proofs: The Fan Theorem for Uniformly Coconvex Bars. In S. Centrone, S. Negri, D. Sarikaya, and P. Schuster, editors, *Mathesis Universalis, Computability and Proof*, volume 412 of *Synthese Library*, pages 333–341. Springer International Publishing, 2019.

[23] Helmut Schwichtenberg. Logic for Exact Real Arithmetic: Multiplication. In *Mathematics for Computation (M4C)*, chapter 3, pages 39–69. World Scientific, April 2023.

[24] Helmut Schwichtenberg, Monika Seisenberger, and Franziskus Wiesnet. Higman's Lemma and its Computational Content. In R. Kahle, T. Strahm, and T. Studer, editors, *Advances in Proof Theory*, pages 353–375. Springer International Publishing, 2016.

[25] Helmut Schwichtenberg and Stanley S. Wainer. Tiered Arithmetics. In G. Jäger and W. Sieg, editors, *Feferman on Foundations*, volume 13, pages 145–168. Springer International Publishing, 2017.

[26] Helmut Schwichtenberg and Franziskus Wiesnet. Logic for exact real arithmetic. *Logical Methods in Computer Science*, 17(2):7:1–7:27, April 2021.

[27] Dana S. Scott. *Domains for denotational semantics*, pages 577–610. Springer Berlin Heidelberg, 1982.

[28] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1):411–440, 1993.

[29] Josef Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1(3):397–405, February 1967.

[30] Anne Sjerp Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. Springer Berlin Heidelberg, 1973.

[31] Franziskus Wiesnet. Konstruktive Analysis mit exakten reellen Zahlen. Master's thesis, Ludwig-Maximilians Universität München, September 2017.

[32] Franziskus Wiesnet. Introduction to Minlog. In Klaus Mainzer, Peter Schuster, and Helmut Schwichtenberg, editors, *Proof and Computation*, pages 233–288. World Scientific, May 2018.

[33] Franziskus Wiesnet. Minlog-Kurs. https://www.youtube.com/playlist?list=PLD87fNDrm1skaFxUA-ArQjmqj50IARRPf, 2024. YouTube Playlist.

[34] Franziskus Wiesnet. Verified Program Extraction in Number Theory: The Fundamental Theorem of Arithmetic and Relatives. *arXiv*, April 2025. https://arxiv.org/abs/2504.03460, (under review).

[35] Franziskus Wiesnet and Nils Köpp. Limits of real numbers in the binary signed digit representation. *Logical Methods in Computer Science*, 18(3), August 2022.