

Will this Function Terminate? An Automated Termination Analysis for Wikifunctions, a Collaborative Wikimedia Function Library

Eddie Begović¹, Axel Polleres¹

¹Vienna University of Economics and Business (WU), Welthandelsplatz 1, 1020 Vienna, Austria

Abstract

Wikifunctions is a collaborative Wikimedia project that provides an open library of functions to which anyone can contribute code. This raises a natural question, that is asking if those functions – or their respective, user-provided implementations – are guaranteed to terminate or not. In this short paper we report on a preliminary study applying five established tools from the Termination Competition (AProVE, MuVal, T2, VeryMax, KoAT) to functions extracted from Wikifunctions. From Wikifunctions’ official dump we extracted 1,968 Python implementations, categorized them by structure and translated 43 integer loop functions into the tools’ input formats, validating every translation through differential testing. The tools decided on 33 out of 43 functions and never contradicted each other. They proved to be strongly complementary with 20 functions being proven by a single tool. The analysis also uncovered a genuine non-terminating implementation on the platform. Based on these results, we propose such termination verdicts to be stored as a dedicated metadata property on Wikifunctions, as guidance for both editors and automated agents calling functions from the catalog.

Keywords

Wikifunctions, termination analysis, Wikidata, Abstract Wikipedia

1. Introduction

Besides Wikidata [1], as a community collection for structured knowledge, *Wikifunctions* [2] aims at establishing itself as another pillar for Wikimedia’s *Abstract Wikipedia* [3, 4] project that should eventually leverage automated translation, but also – generally – has the potential to power further agentic applications in a structured and collaboratively curated manner.

Within this overall bigger picture, Wikifunctions serves as a repository for functions and code, for everyone to call, write, maintain, and use.¹

In the wider context of provisioning such a repository, one of the major research problems in programming is termination analysis [5, 6], i.e. answering the question whether or not – or, resp., under which (parameter) constraints – functions terminate. While it is well known that this question cannot be answered in general [7], over the years a wide variety of termination analysis approaches and tools have been developed, cf. for instance [8].

In this paper we argue and motivate how these two worlds could be fruitfully combined by means of a preliminary study and analysis: we have studied and deployed different state-of-the-art termination tools from the Termination Portal [8] in order to investigate the feasibility of automated termination analysis for Wikifunctions; the present short paper summarizes the results of this analysis, and proposes to add termination analysis results to Wikifunctions in the form of a dedicated metadata property. We see use of this beyond a sheer experiment: (1) While (potentially) non-terminating functions are probably not at the core focus of Wikifunctions, it is a totally perceivable and valid use case to also document such functions in a generic function catalog, for instance for illustration purposes; to this end we also propose adding new, more or less trivially non-terminating functions to Wikifunctions as

Wikidata’26: Wikidata Workshop at ISWC 2026

✉ eddie.12@live.at (E. Begović); axel.polleres@wu.ac.at (A. Polleres)

ORCID 0009-0005-2740-8828 (E. Begović); 0000-0001-5670-1146 (A. Polleres)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹cf. https://meta.wikimedia.org/wiki/Abstract_Wikipedia

examples. (2) Automated agents using and calling the functions from such a catalog may benefit from documented automation guarantees and constraints as delivered by the tools we have analysed.

Overall we address the following research questions:

- RQ1** Which existing termination analysis tools are suitable for analyzing implementations extracted from Wikifunctions?
- RQ2** To what extent can these tools automatically prove termination or detect non-termination for Wikifunctions implementations?
- RQ3** How do the results of different termination analysis tools compare on the same set of Wikifunctions implementations?

The remainder of our paper is structured as follows: in Section 2 we provide a short overview on the overall mechanics of termination analysis, including methods such as ranking functions, and briefly introducing relevant tools. We also provide a justification of (a) which available tools and (b) which particular classes of functions we considered in and out of scope for our present analysis.

Section 3 gives an overview of our analysis pipeline for Wikifunctions, particularly motivating why and how we extracted, and where necessary translated, available function definitions from Wikifunctions.

Finally, Section 4 presents the results of our analysis, comparing different tools on the chosen corpus of in-scope Wikifunctions.

We conclude in section 5 with a discussion of limitations, but also a proposal to add respective termination analysis results to Wikifunctions’ metadata, both for reference and guidance of Wikifunctions’ editors and for downstream applications that make use of Wikifunctions.

2. Background and Related Work

Wikifunctions complements Wikidata [1] within the Abstract Wikipedia effort [3]. Every entity on the platform, whether it is a function, a type or an implementation, is represented as a *ZObject*. It is a typed JSON object identified by a ZID of the form Zn [9]. Two *ZObject* types are important for us. A *function*, which has the ZID $Z8$, defines the signature of a computation while an *implementation*, which has the ZID $Z14$, provides a way to compute it. Keys inside a *ZObject* follow the convention of $ZnKm$, so for example $Z13667K1$ denotes the first argument of the function $Z13667$. The entire content of the platform is published as periodic XML Dumps [10], which we used in this study.

Termination analysis asks whether a program halts on every input. The question is undecidable in general [7], however decades of research have produced methods that succeed on large classes of practical programs [6, 11]. The central technique is the synthesis of a *ranking function*, which is a numeric expression over program variables that monotonically decreases in every loop iteration and is bounded from below [5, 12]. Extensions of this ranking function cover lexicographic and polynomial ranking functions [13, 14] as well as dedicated methods for proving non-termination [15].

We herein evaluate five termination tools that compete in the annual International Termination Competition [16]: AProVE [17], MuVal [18] for the C Programs and T2 [19], VeryMax [20], as well as KoAT [21] for the Integer Transition Systems (ITS). All five were run in their official competition Docker Images [22].

We note that the selected tools were restricted to C Integer [23] and ITS [24], whose only parameter data types are mathematical integers. None of the tools accepts Python and the Termination Competition [16] has no Python category, so any Wikifunctions implementation has to be translated before it can be analyzed. Functions over strings, lists or floating point values as well as other recursive functions fall outside these input formats. We therefore restrict this study to Wikifunctions implementations that loop over integer variables, which is an input all five tools should in principle process. We leave full recursive and non-numeric functions to future work; e.g., we could extend the current study to simple (for instance, tail-recursive) recursive functions rewritable to loops.

3. Methodology

Our data source is the official Wikifunctions XML Dump of May 1, 2026 [10], comprising 79,340 pages. Every entity on Wikifunctions is stored as a typed JSON object, a so called ZObject [9]. The implementations are objects of type Z14 that carry executable code in a specific language. Streaming through the dump we extracted all implementations whose code block is Python, yielding 1,968 Python implementations.

Not every implementation on Wikifunctions is source code, as more than half of them are compositions built from calls to other functions. The currently supported languages for actual implementations (which also compositions can be compiled into),² are Python and JavaScript. Here, Python is the most common with 1,968 implementations, followed by JavaScript with 1,299 implementations, and many functions are implemented in both. Since none of the tools accepts a language that is used on Wikifunctions, a translation step is required for every implementation; our current pipeline is restricted to Python. Categorization parses the Python source and the differential test executes it, so both stages are tied to that language. A JavaScript front end would add 374 functions that have no Python implementation, which we leave for future work.

We further sub-categorized these implementations, cf. Table 1. We first note that out of these 1,968 implementations, 1,435 (i.e., 72.9%) do not contain loops nor recursion, i.e. trivially terminate. Each of the remaining implementations is parsed into an abstract syntax tree using Python’s built in *ast* module [25]; the result is used for static analysis and assigned to a structural category based on the kinds of loops it contains and the data types it operates on: only functions consisting of loops over integer variables can be used as the tools’ input formats (cf. Section 2). Of the 69 integer loop candidates, 43 Python implementations could be translated faithfully into the required C and ITS formats and form our analysis set. The other 26 use constructs the translation cannot express, such as a loop over a list, a call inside an expression or a variable exponent.

Table 1

Category distribution of the 1,968 extracted Python implementations. Percentages are rounded.

Category	Count	%
No loop, no recursion (trivially terminating)	1,435	72.9
Loops over non-integer data (strings, lists, floats)	288	14.6
Recursion	156	7.9
Integer loops, faithfully translated (analysis set)	43	2.2
Integer loops not faithfully translatable	26	1.3
Parse errors (Python 2 syntax)	20	1.0

Each of the 43 functions is converted into a single control flow graph representation from which all tool specific input formats are generated, i.e., C integer programs [23] for MuVal and AProVE, and ITS [24] in the supported *.t2* and *.koat* syntaxes for T2, VeryMax and KoAT. All five tools therefore analyze the same underlying functions. Since no off-the-shelf translator from Python to these formats exists, a custom translator was implemented and Anthropic’s LLM Claude [26] assisted in writing that code. Each generated graph is checked by a differential test against the original Python function. Every function is tested on up to 21 inputs, ten of them random with values between 0 and 9 drawn from a fixed seed and eleven that give every parameter the same value from 0 to 8 and also -1 and -2. Both sides run under a budget of 40,000 steps and one single disagreement rejects the translation, which never occurred. If both sides run past the budget on the same input, then we count this as an agreement, which is why the one implementation that does not terminate on a particular input stayed in the analysis set. While this approach assures a “best effort” approach to achieve faithful translation, which we consider sufficient for the small, tested functions in our setting, we note that – as a current limitation of our work – this LLM translation does provide correctness guarantees as such.

All tools were run in their official Termination Competition Docker images [22] with a 60 second

²cf. Wikifunctions FAQ, <https://www.wikifunctions.org/wiki/Wikifunctions:FAQ> last accessed September 2026.

timeout per function and every output is stored. The outputs are mapped to five verdict categories: *terminating*, *non-terminating*, *unknown*, *timeout* and *tool error*. All results, translations and the analysis pipeline are publicly available.³

4. Results

All five tools could be configured and executed on all 43 functions, but how far each tool got with its input varied widely. The two C tools never rejected a .c input with a parse error, however AProVE produced no output on 20 runs and ran out of memory on one, so successful processing can only be confirmed for the remaining 22 runs which it returned as a result. T2 accepted 41 out of 43, rejecting two with a parse error. VeryMax processed only 14 and rejected most of them with a semantic error. KoAT parsed only 6 of its 43 inputs.

Table 2 shows the per tool verdicts. Combined, 32 of the 43 functions were proven terminating by at least one tool and one was proven non-terminating, which means that 33 out of 43 functions were proven to terminate or non-terminate and 10 remain undecided. MuVal is the strongest single tool with 30 conclusive verdicts, followed by AProVE with 15, T2 and VeryMax with 11 each, and finally KoAT with 6.

Table 2

Verdicts per tool over the 43 translated functions. The last row aggregates over the tools, counting each function once. A function counts as decided if at least one tool proved it terminating or non-terminating. The three failure columns are not aggregated, since the tools fail in different ways on the same function.

Tool	Input	Term.	Non-term.	Unknown	Timeout	Error
MuVal	.c	29	1	0	0	13
AProVE	.c	15	0	7	0	21
T2	.t2	11	0	30	0	2
VeryMax	.t2	11	0	2	1	29
KoAT	.koat	6	0	0	0	37
Combined		32	1	10 undecided		

The inconclusive outcomes align with one feature of the source code. Of the 43 functions, 37 use integer division or modulo and every inconclusive outcome falls inside that group, while the six functions that use neither are proven to be terminating by all five tools.

Whenever two tools decided the same function they agreed. Across all tool pairs not a single contradiction occurred. At the same time the tools are strongly complementary, as 20 of the 33 proven functions were covered by exactly one tool. The single non-terminating verdict concerns Z14962, which is a Wikifunctions implementation of whether an argument is a factorial number by repeatedly dividing it by $i = 1, 2, 3, \dots$ while divisible. On input 0 however the loop never exits, as zero is divisible by every i . MuVal produced a non-terminating proof and manual inspection of the Python source confirms the verdict. The remaining tools did not produce any verdict on this function so they neither confirmed its non-termination nor did they deny it. This case is precisely the kind of finding a termination metadata flag in Wikifunctions would surface to users and agents.

5. Discussion and Future Work

Our study gives a qualified positive answer to our paper’s underlying question. Where the tools apply, they are trustworthy since no contradiction has occurred and they are best used as a portfolio. The bottleneck is not proving power but reach, as only 43 out of 1,968 implementations survived the translation and even on those coverage is limited by the tools’ front ends rather than the analyses behind them. The applicability of the tools differed far more than we expected. KoAT rejects 37 of its

³<https://github.com/16x29/Master-Thesis---Certainly-Terminating-Wikifunctions>

43 inputs already in its parser because its syntax has no division or modulo and VeryMax aborts on 27 of them because the denominator is not a constant. T2 reads the same operators without a problem and only gives up later during its proof search. Most of the functions we analyze compute primes, divisors or greatest common divisors, and that is where the division and modulo come from. We note that the complementary results also place Wikifunctions as a potentially interesting benchmark for the research community working on Termination tools.

Several limitations should be kept in mind when reading these results. Our differential test on automated function translations gives good practical confidence but it is not a proof of equivalence as the verdict applies to the translated program rather than to the original function itself. The two sets of translations also differ in their input domain, because we restricted the C translations to non-negative integers, whereas the ITS contain no such restriction. All arguments of these functions are declared as natural numbers on Wikifunctions, so negative inputs are outside what they accept. Python and C agree on division and modulo only for non-negative values, since Python rounds a division down while C rounds it towards zero, so on negative inputs the two would compute different results. The restriction is therefore what keeps the translation faithful. Finally, the tools proved non-termination only once, for Z14962. Despite the in our opinion interesting finding, we shy away from drawing general conclusions from that.

We manually inspected the ten implementations on which no tool reached a verdict. Seven of them clearly terminate, as every iteration either grows the divisor or shrinks the number. Two of them, Z14065 and Z14069, compute Collatz stopping times, where termination is a long standing problem [27]. The last one, Z26991, does not always terminate, as it loops forever when its first argument is 0 or its second is 1.

We (rather) see these results as support for the proposal made in the introduction. A dedicated metadata property on Wikifunctions could record tools' verdicts for user-contributed implementations. As such, editors would see defects like the aforementioned Z14962 before users run into them and automated agents calling functions from the catalog could retrieve machine-readable hints about termination before use. Making this happen would require Wikifunctions to integrate the analysis and record its results. Our pipeline could serve as a starting point here.

For future work, the most direct extension is towards the function classes we had to exclude. Full recursive functions cannot be expressed by our loop based translation, because recursion repeats through a call rather than through a loop, and would instead have to be analyzed by other methods such as term rewriting systems [28]. Loops over non-integer data such as strings and lists would for example have to rank structural measures such as the length of a string or a list instead of the loop variables themselves. Recursion is also not confined to a single implementation, since a function can be defined as a composition of calls to other functions. In the dump 393 functions take part in such a call cycle, either by calling themselves or through a cycle running over several functions, so their termination cannot be decided from one definition on its own. A further extension would be to also consider JavaScript implementations, which would need an adaptation of the translation pipeline, but no fundamental change to the analysis behind it. At the moment, we only considered Python functions faithfully translatable to *both* C and ITS; another option would be to translate functions to C only. The C format is far more expressive than ITS, so a C only pipeline could cover more functions, though at the price of giving up the cross format comparison that this study was built on. Finally, only one function in our analysis set was proven non-terminating. Adding documented non-terminating function examples deliberately to Wikifunctions would therefore allow future evaluations to test the non-termination detection more systematically.

Finally, we note that the Termination Portal itself is a MediaWiki powered wiki, apparently without any Semantic Wiki capabilities [29] or extensions of its own. We would be delighted if this small pre-study could spark discussions with the termination analysis community on linking the portal and its documented tools with Wikifunctions in a more integrated manner.

Declaration on Generative AI

During the preparation of this work the authors used Anthropic's LLM Claude in order to: Paraphrase and reword, Improve writing style, Grammar and spelling check. Beyond that, Claude Code was used to assist in implementing the analysis pipeline described in this paper as disclosed in section 3. The pipeline itself runs no model, as the translation from Python to the tool formats is performed by the code alone.

Acknowledgments

This work was conducted as part of Eddie Begović's master thesis [30] which contains further details on the herein reported results. Axel Polleres' research was funded in part by the Austrian Science Fund (FWF) [10.55776/COE12].

References

- [1] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* 57 (2014) 78–85.
- [2] D. Vrandečić, Building a multilingual wikipedia, *Commun. ACM* 64 (2021) 38–41. URL: <https://doi.org/10.1145/3425778>. doi:10.1145/3425778.
- [3] D. Vrandečić, Capturing meaning: Toward an abstract wikipedia., in: *ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks*, volume 2180, CEUR.org, 2018. URL: https://ceur-ws.org/Vol-2180/ISWC_2018_Outrageous_Ideas_paper_6.pdf.
- [4] D. Vrandečić, L. Pintscher, M. Krötzsch, Wikidata: The making of, in: *Companion Proceedings of the ACM Web Conference 2023, WWW '23 Companion*, Association for Computing Machinery, New York, NY, USA, 2023, p. 615–624. URL: <https://doi.org/10.1145/3543873.3585579>. doi:10.1145/3543873.3585579.
- [5] A. Podelski, A. Rybalchenko, A complete method for the synthesis of linear ranking functions, in: B. Steffen, G. Levi (Eds.), *Verification, Model Checking, and Abstract Interpretation*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 239–251.
- [6] B. Cook, A. Podelski, A. Rybalchenko, Proving program termination, *Communications of the ACM* 54 (2011) 88–98.
- [7] A. M. Turing, et al., On computable numbers, with an application to the entscheidungsproblem, *J. of Math* 58 (1936) 5. doi:10.1112/plms/s2-42.1.230.
- [8] J. G. et al., Termination portal, since 2008. URL: https://termination-portal.org/wiki/Termination_Portal.
- [9] Wikimedia, ZObjects, https://meta.wikimedia.org/wiki/Abstract_Wikipedia/ZObject, 2026. Accessed 23 July 2026.
- [10] Wikimedia, Data dumps, https://meta.wikimedia.org/wiki/Data_dumps, 2026. Accessed 23 July 2026.
- [11] B. Cook, A. Podelski, A. Rybalchenko, Termination proofs for systems code, *ACM Sigplan Notices* 41 (2006) 415–426.
- [12] M. A. Colón, H. B. Sipma, Synthesis of linear ranking functions, in: T. Margaria, W. Yi (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 67–81.
- [13] A. R. Bradley, Z. Manna, H. B. Sipma, The polyranking principle, in: L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), *Automata, Languages and Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 1349–1361.
- [14] A. R. Bradley, Z. Manna, H. B. Sipma, Linear ranking with reachability, in: K. Etessami, S. K. Rajamani (Eds.), *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 491–504.

- [15] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, R.-G. Xu, Proving non-termination, in: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2008, pp. 147–158.
- [16] Termination-portal.org, Termination Competition, https://termination-portal.org/wiki/Termination_Competition, 2026. Accessed 16-06-2026.
- [17] J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, et al., Proving termination of programs automatically with aprove, in: International Joint Conference on Automated Reasoning, Springer, 2014, pp. 184–191.
- [18] H. Unno, T. Terauchi, Y. Gu, E. Koskinen, Modular primal-dual fixpoint logic solving for temporal verification, Proceedings of the ACM on Programming Languages 7 (2023) 2111–2140.
- [19] M. Brockschmidt, B. Cook, S. Ishtiaq, H. Khlaaf, N. Piterman, T2: temporal property verification, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2016, pp. 387–393.
- [20] C. Borralleras, D. Larraz, A. Oliveras, J. M. Rivero, E. Rodríguez-Carbonell, A. Rubio, Verymax: tool description for termcomp 2016, in: 15th International Workshop on Termination, volume 18, 2016.
- [21] N. Lommen, J. Giesl, Aprove (koat + loat), in: A. Gurfinkel, M. Heule (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Nature Switzerland, Cham, 2025, pp. 205–211.
- [22] Termination Competition, TermComp, <https://hub.docker.com/u/termcomp>, 2025. Docker Hub. Accessed 23 July 2026.
- [23] Termination-portal.org, C integer programs, https://termination-portal.org/wiki/C_Integer_Programs, 2026. Accessed 23 July 2026.
- [24] Termination-portal.org, Integer transition systems, https://termination-portal.org/wiki/Integer_Transition_Systems, 2026. Accessed 23 July 2026.
- [25] Python Software Foundation, Abstract syntax trees, <https://docs.python.org/3/library/ast.html>, 2026. Accessed 23 July 2026.
- [26] Anthropic, Claude, www.anthropic.com, 2026. Accessed 23-07-2026.
- [27] T. Tao, Almost all orbits of the collatz map attain almost bounded values, Forum of Mathematics, Pi 10 (2022) e12. doi:10.1017/fmp.2022.8.
- [28] Termination-portal.org, TRS standard, https://termination-portal.org/wiki/TRS_Standard, 2026. Accessed 23 July 2026.
- [29] M. Krötzsch, D. Vrandečić, M. Völkel, Semantic mediawiki, in: International semantic web conference, Springer, 2006, pp. 935–942.
- [30] E. Begovic, Certainly Terminating Wikifunctions, Master’s thesis, Vienna University of Economics and Business, 2026. URL: http://www.polleres.net/supervised_theses/Eddie_Begovic_MSc_2026.pdf.