Listening to the Firehose: Sonifying Z3's Behavior

Finn Hackett
University of British Columbia
Vancouver, Canada
fhackett@cs.ubc.ca

Ivan Beschastnikh
University of British Columbia
Vancouver, Canada
bestchai@cs.ubc.ca

***Abstract*—Modern formal methods rely heavily on Satisfiability Modulo Theory (SMT) solvers like Z3. Unfortunately, these solvers are complex, have unpredictable runtime behavior, and are highly sensitive to the structure of the input query. As a result, when a Z3 query runs for tens of minutes and/or times out inconclusively, there is little that an end-user can do to figure out what went wrong. They can attempt to inspect the gigabytes of logged information that these tools produce every minute. But, no existing tool provides a broad understanding of Z3 behavior.**

We propose Z3Hydrant, a scalable approach that converts Z3 logs into sound. By relying on the innate abilities of the human ear to pick out patterns, Z3Hydrant encodes raw Z3 logs into an audio stream. The result is accessible to anyone who can hear and helps to provide a general flavor of what occurred during a particular run. We describe our approach and include several example audio files that capture complex Z3 runs.

I. Introduction

Satisfiability Modulo Theory (SMT) solving is generally undecidable, with difficult to predict behavior that varies based on how a solver is configured. The solver may or may not terminate, whether or not a proof exists of a given statement. Running an SMT solver may therefore output *SAT*, *UNSAT*, or *UNKNOWN*, after running for an arbitrary period of time and perhaps timing out. During this time, the user waits for a response to their query, with little indication of progress or other feedback.

Prior to submitting an SMT query, the user cannot easily predict the outcome. This is because SMT solvers perform heuristic-guided search using multiple communicating theorem solvers, and their behavior is highly sensitive to changes in the input query. To illustrate this complexity, the widely used Z3 [1] verifier is implemented in 378,853 lines of C++[1]. Beyond the size of Z3's codebase, consider that Dafny [2], a verification-aware programming language that uses Z3 internally, loads 1,985 lines of contextual axioms before it checks any user code. Other related systems [3], [4], [5] have similarly large standard libraries of facts.

Given this inherent solver complexity, unpredictability of output, and sensitivity to input, significant prior work has gone into helping developers understand and debug SMT workloads. These efforts focus on understanding the behavior of *triggers*, a common implementation technique for logical quantifiers, which are the primary cause of undecidability in SMT. These tools are most useful to those who have deep knowledge of a tool's implementation. For instance, fine-grained dynamic analyses [6] and static guarantees [7] might help the Dafny developers debug and improve their 1,985 lines of axiom definitions. The Axiom Profiler [8] is the most general-purpose dynamic analysis tool available, whose development is the reason that modern versions of Z3 support logging of all steps taken by the solver (quantifier instantiations, theorem invocations, etc). The Axiom Profiler processes these logs, and identifies suspicious quantifier instantiation patterns that might cause poor performance or non-termination, allowing developers to refine their axiom definitions. The problem is that the output from any of these tools is only meaningful if the user already has detailed knowledge of a majority of the axioms and definitions in play. There is currently no way to represent the totality of an SMT solver's behavior to an end-user looking to understand what is happening. One simple reason for this is that Z3 logs on average 1.9 GB of information for every minute of runtime, generating enormous logs for long-running queries.

To help end-users interpret Z3 executions, we propose Z3Hydrant, a tool which *converts the Z3 log into sound*. We represent log event sequences as audio rate signals (44.1 kHz in our preliminary experiments). Z3Hydrant leverages the high-speed pattern recognition capabilities of the human ear to let developers pick out patterns and variations in SMT solver behavior, without requiring any a priori knowledge of how the underlying tool works. Our technique allows any point of interest or pattern in the sound to be precisely mapped back to log entries for further analysis. Given hybrid tooling that combines sonic exploration with existing fine-grained analysis techniques, we believe that log stream sonification has the potential to make debugging SMT constraints more accessible to end-users, and less the exclusive domain of SMT experts.

II. Background and Motivation

Our representation of SMT logs is inspired by tools like Geiger counters, which represent the complex invisible reality of ionizing radiation as clicking sounds of variable density. We find that this is a low-overhead way to convert a large quantity of data into something that can be understood by humans. Consider a digital audio stream with a common sample rate of 44.1 kHz: at an extreme, if we match every line in a log to one sample of the audio waveform (as we do in our experiments), we are able to present 44,100 log entries to a human per second. Because humans are able to summarize patterns within vibrations perceived by our eardrums, we are able to intuitively experience the structure of time-series data at a high rate.

[1]Looking at the src/ directory using the cloc tool, excluding whitespace and comments.

```
[mk-app] #655 = #652 #523
[instance] 0 #655
[attach-enode] #655 0
[end-of-instance]
[mk-app] #655 = #652 #523
[mk-proof] #656 rewrite #655
[mk-app] #657 = #485 #523
[mk-proof] #658 trans #654 #656 #657
[mk-app] #659 => #523 #523
[mk-app] #660 = #486 #659
[mk-proof] #661 monotonicity #658 #660
[inst-discovered] theory-solving 0 ...
```

Fig. 1. Log fragment showing example Z3 behavior.

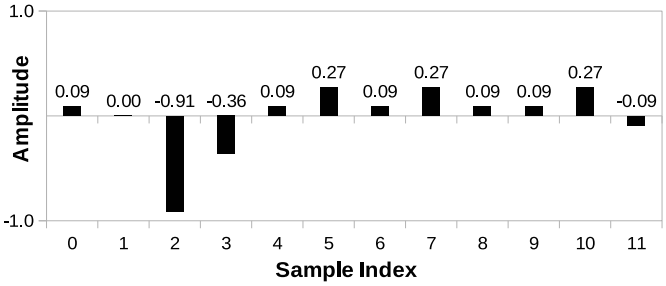

Fig. 2. Audio waveform matching the log entries in Figure 1.

Note that, in terms of human perception, a 44.1 kHz samplerate can encode frequencies just beyond the accepted upper frequency limit for human hearing, 20 kHz [9]. We chose it to explore the maximum possible sonification bandwidth. It can be progressively lowered to accommodate people with limited hearing range, or to address concerns that inaudible frequencies are present.

Compare this to our sense of sight, where we need to summarize large volumes of data to present a clear picture. General-purpose log processing techniques [10], [11], [12] rely on visual summaries. The Axiom Profiler does as well; its uses matching loop identification to summarize points of interest in the log. The visual representation used by Axiom Profiler is a graph view, representing causal relationships between quantifier instantiations made by Z3. The tool can process large logs, but when displaying its summary it shows graphs with only a few hundred nodes. Larger graphs are difficult to understand via direct inspection. There are techniques that can directly display larger graphs [13], but they work on graphs of at most tens of thousands of nodes (less than 0.1s of Z3 execution) and produce very dense images.

In summary, there is no visualization tool that can show the entire log of a non-trivial Z3 execution. If we add sonification to our toolkit, however, Z3Hydrant can represent a log of 1 million SMT events ($\approx$ 1s of Z3 runtime) as 23 seconds of audio. Since sonification requires dedicated listening time that scales with larger volumes of data, we propose that Z3Hydrant is most useful for logs that can be listened to within a few minutes, representing tens of seconds of Z3 execution. Within this sweet spot, the user can directly explore the log data by listening, identifying patterns and landmarks for further analysis.

Our goal with Z3Hydrant is to leverage the large bandwidth of the human ear, and provide a sonic replacement for the missing "view all" button, at least for logs up to $100\times$ as long as what is currently practical to visualize.

III. Sonification Design of Z3Hydrant

We now describe our approach to sonifying Z3's behavior.

Our input file looks like Figure 1. It consists of a sequence of algorithmic events that represent each step of Z3's underlying algorithm. Each log line has a label, shown as `[mk-app]` or `[instance]`, which we call the "action". The remaining information on that same line contains operational details, like term rewriting, theory applications, and so forth. This data is normally given as input to the Axiom Profiler, in which case it can be analyzed for patterns in quantifier instantiation that signify matching loops.

Our motivation for using this format is (1) it is easy to access, since Z3 produces it natively; (2) it is representative of Z3's entire decision process as it responds to a query; and (3) if we find an interesting audible pattern in the log, that pattern could be passed to the Axiom Profiler or other tools that consume the same format.

A. Chosen Representation

To keep the signal simple, since our representation should be used for quick exploration of large volumes of data, we consider the action and nothing else. Although this is a lossy encoding, it still contains the kinds of algorithmic steps Z3 has performed, if not their details. Listening to the algorithmic steps Z3 performs in a sequence allows the listener to hear the structure of the underlying computation. Looking at other log features or performing a more detailed analysis is out of scope for this encoding – we discuss some alternatives in Section V.

Z3Hydrant converts each action with a precomputed perfect hashing function into an equally spaced set of floating point numbers between -1 and 1. Figure 2 shows the resulting audio waveform when this process is applied to the log fragment in Figure 1. For example, the 3 bars marked 0.27 at indices 5, 7, and 10 correspond to `[mk-proof]` events. The shorter lines marked 0.9 in between them are `[mk-app]` events. We identified 23 different actions, which can be uniquely mapped onto the range of floating point values available. With this encoding, any distinct sequence of log actions corresponds to a distinct audio waveform.

These audio waveforms will sound differently based on emergent properties of the log. Based on our experience, we observed that this encoding has the following features:

Constant pitches are how our ears perceive repeating sequences of tens to hundreds of actions. The length of the sequence is the wavelength of the perceived oscillation, so at an audio samplerate of 44.1 kHz, wavelengths between 3 actions (14.7 kHz, high whistle) and around 735 actions (60 Hz, low bass note) can be easily identified as pitches. Sequences that

are not perfect repetitions but primarily follow a simple pattern will still sound like an identifiable pitch, but noisier. While not directly related to Z3's semantics, these pitches are intuitive identifiers for specific sequences of actions in a log. If the same sequence of actions re-appears, we can recognize that a log pattern occurred multiple times.

Complex tones emerge depending on the waveform implied by a sequence of actions. The closer a waveform is to a simple shape, like a square or triangle, the more it will sound like an identifiable pitch. Sequences with more complex shapes will sound metallic or bell-like, like wavetable [14] or frequency modulation synthesis [15]. They allow us to recognize the same things as constant pitches.

Slower variations, across thousands or tens of thousands of log entries, will appear as sequences of different pitches, changes in timbre over time, or rhythmic clicking sounds. Hearing this kind of sound multiple times helps us intuitively identify larger trends in Z3's behavior.

Frequency glides seem to occur when Z3 is looping over increasing/decreasing numbers of possibilities in some domain. The same pattern repeats, but one or more parts change length while preserving the overall shape. This causes a shift in the wavelength of the resulting waveform, and causes the audible frequency to glide up or down.

Pseudo-random sequences will sound similar to white noise. Perhaps because SMT solvers follow an inherent algorithmic sequence, this feature is often present but rarely dominates the listening experience. Hearing this means Z3 is doing something that is not regular enough to hear as a pitch.

Complete silence is a thousands-long sequence of the same action repeated, which is converted to a static waveform. It is not common, but we found `[mk-app]` sometimes formed such sequences in our evaluation. `[mk-app]` means that Z3 has instantiated an uninterpreted function of the form `f(a, b, c, ...)`, so these long silences imply Z3 was constructing thousands of terms.

Inaudibly quiet sounds are unlikely, given our chosen encoding. The quietest possible sound is caused by long sequences of actions that are encoded as numerically adjacent. Given a maximum amplitude of -1 to 1, splitting this across our 23 possible actions, we get a minimum amplitude of $2/23 \approx 0.09$. While human loudness perception varies, we have found a magnitude of 0.09 to be audible.

IV. Preliminary Evaluation

Our goal in evaluating Z3Hydrant is three-fold. We want (1) a representative sample of the tool's output on realistic Z3 executions; (2) a measure of how much data is logged by Z3 over time; and (3) to know how feasible it is to listen to Z3 in real-time by streaming its log to audio.

We have evaluated Z3Hydrant on a random sample of 254 problems supported by Z3[2] from the SMT-LIB 2024 non-incremental benchmark [16]. Used in yearly SMT competitions, this benchmark is representative of a variety of realistic user-submitted SMT problems. Unlike something like the Z3 test suite, we also expect that SMT-LIB is not biased toward any particular SMT solver. This means Z3 may do well on some problems and poorly on others, producing a range of behaviors for us to listen to.

[2]We discarded inputs where Z3's response included the string "unsupported", which means that an unsupported theory was requested.

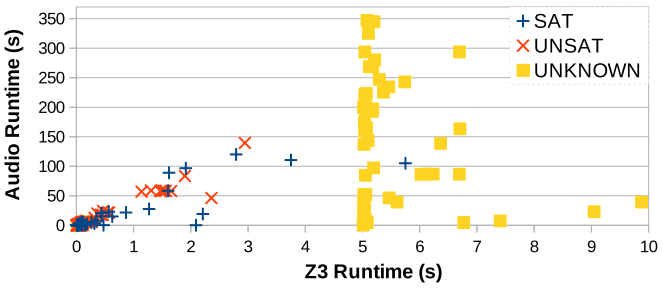

Fig. 3. Relationship between Z3 runtime (configured with a short 5s timeout) and audio runtime, annotated with the reported Z3 outcome for each run.

Our artifact [17] contains our experimental results (including the audio files) and all the code to reproduce them. We used Z3 4.8.12, and all experiments were done on a machine with a 13th Gen Intel Core i9-13900HX CPU and 64 GB of RAM.

A. Examining the Output

Figure 3 summarizes the outcomes of running Z3 on all 254 input files. Overall, Z3 responded to 83 inputs with `SAT`, 110 inputs with `UNSAT`, and 57 inputs with `UNKNOWN`. In 4 cases, Z3 ran normally but did not output a conclusion.

Note that the cluster of results at 5s is because we gave Z3 a 5s timeout to keep a manageable overall runtime. The cluster is cases where Z3 would have taken longer, but timed out and exited on time while returning `UNKNOWN`. Longer times are due to Z3 missing its timeout and exiting late. In our experience, longer-running Z3 queries will run for a very long time if unchecked.

We listened to all the generated audio files while making notes summarizing what we heard. Here are highlights of these notes, alongside representative examples from our artifact in the `wavsDir/` folder.

A large majority of the audio was highly structured. Many sequences of events translated into repetitive cycles. Textural changes along these cycles allowed us to identify repeated behaviors. See `bench_11793.wav`, where starting around 8-9s in, there is a repeating chirping sound. It represents Z3 repeatedly discovering facts via theory solving. None of these facts are necessarily related, but we can detect a correlation in the type of work that goes into instantiating them. We think that variation in the pitch content of the chirps identifies facts that take more operations to instantiate. `156_gcc.wav` and `nlzbe256.wav` exhibit similar features.

We were able to identify behaviors with similar filenames (under the same SMT-LIB group) because they sounded similar. See collections like `toIntegral-has-*`, `ball_count_*`, and `From_AProve_2014__*`. Exploring these

logs directly and via textual diffing, we notice that they are not identical, but they share similar structural features. There are similar groups of operations in the same places, and each log contains many "rewrite" and "monotonicity" steps.

Some outputs are monotonous. They consist of long held tones, long silences, or long homogeneous-sounding blasts of noise. This is a true indication that Z3 is doing similar things for a long time. In `rem-has-no-other-solution-13376.wav`, the silence we are hearing reflects that the entire log is exclusively `mk-app` operations (Z3 is constructing terms). The noisy `gensys_icl584.wav`, on the other hand, represents Z3 constantly oscillating between relatively long sequences, including 100s of `[assign]` and tens of `[mk-proof]` actions. This creates a noisy low rumble due to a dominant pattern in the log having an unusually long wavelength. The first minute of `vlsat3_b81.wav` consists of a continuous high-pitched whine, which represents a uniform cycle of 12 repeating actions.

B. Log File Volume

The logs we collected for each Z3 run were 63 MB on average. The smallest was 4 KB, and the largest was 624 MB. Dividing by the runtime of the associated Z3 execution, the average rate at which Z3 logged data was 32 MB/s (or 1.9 GB/min), with a standard deviation of 31 MB/s. There is significant variation in how much data Z3 logs, depending on what it is doing. It could be due to variations in log line length, how long different actions take to perform, or both.

C. Relationship with Real Time: Live Listening not Possible

To evaluate how close to real-time we could sonify a log, we computed the ratios between Z3 runtime and audio runtime. On average, the audio would play for $20\times$ as long as it took to generate the log, with a standard deviation of $18\times$. Since we used the highest humanly perceptible data rate, this shows that Z3Hydrant cannot do real-time sonification while Z3 runs. Buffering and skipping of data would be necessary.

D. Discussion

We have found that sampling the action labels from Z3's operational logs directly into the audio amplitude domain is effective in providing a high-level overview of the solver's behavior. This overview allows us to notice similarities and differences between log segments thousands of events long. For example, we can confirm via inspection that the repeated chirping in `bench_11793.wav` indicates structurally similar cases where Z3 infers facts. Even for sonifications like `bench_5085.wav` that are less repetitive, clicks, impacts, and pitch changes identify transition points. These could mean that Z3 has finished instantiating a set of axioms, that it has exhausted a certain search space, or something else. This result is encouraging, in that these landmarks could help developers qualitatively guide and scope further analysis.

That said, we need further study to better understand the relationship between Z3Hydrant's sounds, Z3's semantics, and other tools like the Axiom Profiler.

V. Future Plans

This paper introduces the concept of log sonification for SMT solvers, but it leaves open multiple avenues for development.

Alternative encodings. Our simplified log label-based encoding seems effective in bringing out the high-level structure of what a solver is doing, but it is limited. As is, Z3Hydrant is unable to distinguish between the same action applied to different parameters. We plan to investigate encodings that take parameters into account. One alternative could be to hash more of the log lines together while keeping the rest of Z3Hydrant the same. Another approach could be to investigate frequency-domain synthesis using Fast Fourier Transforms [18].

Integration into analysis tools. Listening to a log is an initial step that allows us to identify points of interest for further analysis. Actually performing that analysis requires switching to more detail-oriented tools, such as the Axiom Profiler. Following successful examples from prior work [19], we plan to bring sonification and visualization together by integrating Z3Hydrant with tools like Axiom Profiler, and evaluate the result on realistic SMT debugging tasks.

VI. Related Work

There is a broad range of sonification techniques [20], [21]. Z3Hydrant maps a sequence of events into the amplitude domain at audio rate. Historically, this has happened coincidentally at the hardware level. Consider the dial-up modem handshake [22], or AM radio interference from mainframes [23].

Work that we are aware of focuses on fine-grained sonification. Notable examples are sonifying program slices [24], enabling accessible AST-based programming [25], sonification of runtime information [26], [23], debug-time sonification [27], [28], and sonification of version control histories [29]. This prior work sonifies hundreds or thousands of data points at a time, not the millions we aim for.

Z3Hydrant uses the same data as Axiom Profiler [8]. The two techniques are complementary: Z3Hydrant targets open-ended exploration, while the profiler isolates potential matching loops and can only represent small portions of a Z3 log at once. Other work to assist developers in working with SMT solvers is similarly orthogonal and complementary [7], [6].

More broadly, there are a variety of techniques for processing logs [10], [11], [12]. These approaches target systems whose logs contain distinct anomalies or failures. For SMT solvers, all user-visible failures occur during normal operation. As such, existing log analysis techniques are less applicable to SMT solver logs.

VII. Conclusion

SMT executions can take a long time, are difficult to interpret, and can produce massive amounts of log data. We propose a scalable approach that converts logs from the Z3 SMT solver into sound. Our sonification strategy relies on the ability of the human ear to pick out patterns. We sketched out the design of our proposed tool, Z3Hydrant, which complements existing tools and shows promise as a comprehension tool.

References

[1] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[2] K. R. M. Leino, "Dafny: An Automatic Program Verifier for Functional Correctness," in *Logic for Programming, Artificial Intelligence, and Reasoning*, E. M. Clarke and A. Voronkov, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 348–370.

[3] P. Müller, M. Schwerhoff, and A. J. Summers, "Viper: A Verification Infrastructure for Permission-Based Reasoning," in *Verification, Model Checking, and Abstract Interpretation*, B. Jobstmann and K. R. M. Leino, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 41–62.

[4] N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoue, and S. Zanella-Béguelin, "Dependent Types and Multi-Monadic Effects in F*," in *POPL '16 Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, January 2016, pp. 256–270. [Online]. Available: https://www.microsoft.com/en-us/research/publication/dependent-types-multi-monadic-effects-f/

[5] L. d. Moura and S. Ullrich, "The Lean 4 Theorem Prover and Programming Language," in *Automated Deduction – CADE 28*, A. Platzer and G. Sutcliffe, Eds. Cham: Springer International Publishing, 2021, pp. 625–635.

[6] A. Bugariu, A. Ter-Gabrielyan, and P. Müller, "Identifying Overly Restrictive Matching Patterns in SMT-based Program Verifiers (Extended Version)," *Form. Asp. Comput.*, vol. 35, no. 2, jun 2023. [Online]. Available: https://doi.org/10.1145/3571748

[7] R. Ge, R. Garcia, and A. J. Summers, "A Formal Model to Prove Instantiation Termination for E-matching-Based Axiomatisations," in *Automated Reasoning*, C. Benzmüller, M. J. Heule, and R. A. Schmidt, Eds. Cham: Springer Nature Switzerland, 2024, pp. 419–438.

[8] N. Becker, P. Müller, and A. J. Summers, "The Axiom Profiler: Understanding and Debugging SMT Quantifier Instantiations," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 99–116.

[9] S. Rosen and P. Howell, *Signals and systems for speech and hearing*. Brill, 2011, vol. 29.

[10] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A Survey on Automated Log Analysis for Reliability Engineering," *ACM Comput. Surv.*, vol. 54, no. 6, jul 2021. [Online]. Available: https://doi.org/10.1145/3460345

[11] A. Rabkin, W. Xu, A. Wildani, A. Fox, D. Patterson, and R. Katz, "A graphical representation for identifier structure in logs," in *Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML 10)*, 2010.

[12] A. Frei and M. Rennhard, "Histogram Matrix: Log File Visualization for Anomaly Detection," in *2008 Third International Conference on Availability, Reliability and Security*, 2008, pp. 610–617.

[13] K.-L. Ma and C. W. Muelder, "Large-Scale Graph Visualization and Analytics," *Computer*, vol. 46, no. 7, pp. 39–46, 2013.

[14] U. Andresen, "A New Way in Sound Synthesis," *Journal of the Audio Engineering Society*, no. 1434, march 1979.

[15] J. M. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," *Journal of the Audio Engineering Society*, vol. 21, pp. 526–534, September 1973.

[16] M. Preiner, H.-J. Schurr, C. Barrett, P. Fontaine, A. Niemetz, and C. Tinelli, "SMT-LIB release 2024 (non-incremental benchmarks)," Apr. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11061097

[17] F. Hackett and I. Beschastnikh, "Z3Hydrant Preliminary Evaluation," Sep. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13786261

[18] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[19] K. J. North, A. Sarma, and M. B. Cohen, "Understanding Git history: a multi-sense view," in *Proceedings of the 8th International Workshop on Social Software Engineering*, ser. SSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 1–7. [Online]. Available: https://doi.org/10.1145/2993283.2993285

[20] T. Hermann, A. Hunt, and J. G. Neuhoff, *The Sonification Handbook*. Berlin, Heidelberg: Logos Publishing House, 2011. [Online]. Available: https://sonification.de/handbook

[21] K. Enge, E. Elmquist, V. Caiola, N. Rönnberg, A. Rind, M. Iber, S. Lenzi, F. Lan, R. Höldrich, and W. Aigner, "Open Your Ears and Take a Look: A State-of-the-Art Report on the Integration of Sonification and Visualization," *Computer Graphics Forum*, vol. 43, no. 3, p. e15114, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.15114

[22] L. Peskoe-Yang, "Analyzing Every Second of the Classic Dial-Up Modem Sound," https://www.popularmechanics.com/science/a29611456/internet-dialup-modem-sounds/, 2022, accessed 11/09/2024.

[23] P. Vickers and J. L. Alty, "Siren songs and swan songs debugging with music," *Commun. ACM*, vol. 46, no. 7, p. 86–93, jul 2003. [Online]. Available: https://doi.org/10.1145/792704.792734

[24] L. Berman and K. Gallagher, "Listening to Program Slices," *International Conference on Auditory Display*, pp. 172–175, 2006. [Online]. Available: https://durham-repository.worktribe.com/output/1679273

[25] E. Schanzer, S. Bahram, and S. Krishnamurthi, "Accessible AST-Based Programming for Visually-Impaired Programmers," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 773–779. [Online]. Available: https://doi.org/10.1145/3287324.3287499

[26] P. Vickers and J. L. Alty, "When bugs sing," *Interacting with Computers*, vol. 14, no. 6, pp. 793–819, 12 2002. [Online]. Available: https://doi.org/10.1016/S0953-5438(02)00026-7

[27] A. Stefik, R. Alexander, R. Patterson, and J. Brown, "WAD: A Feasibility study using the Wicked Audio Debugger," in *15th IEEE International Conference on Program Comprehension (ICPC '07)*, 2007, pp. 69–80.

[28] A. Stefik, C. Hundhausen, and R. Patterson, "An empirical investigation into the design of auditory cues to enhance computer program comprehension," *International Journal of Human-Computer Studies*, vol. 69, no. 12, pp. 820–838, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581911000814

[29] K. J. North, S. Bolan, A. Sarma, and M. B. Cohen, "GitSonifier: using sound to portray developer conflict history," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 886–889. [Online]. Available: https://doi.org/10.1145/2786805.2803199