Search-Based Generation of Complex Inputs with FANDANGO

José Antonio Zamudio Amaya, Marius Smytzek, and Andreas Zeller

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{jose.zamudio, marius.smytzek, andreas.zeller}@cispa.de

Abstract. Generating complex, semantically valid test inputs remains a central challenge in search-based software testing. FANDANGO addresses this challenge by combining context-free grammars with fully specified Python constraints and using evolutionary search to satisfy them. Compared with the symbolic state of the art, FANDANGO achieves speedups of up to three orders of magnitude while maintaining 100% input validity and equal or higher grammar coverage. FANDANGO is under active development: recent work explores whitebox fuzzing and stateful fuzzing through interaction grammars that model multi-turn protocol behavior. With more than 100,000 PyPI downloads, adoption in industry (including Volkswagen and Bosch), and coverage in the science press, FANDANGO has established itself as a practical input generator.

Keywords: Language-based testing · search-based software testing · fuzzing · test generation · evolutionary algorithms

1 Introduction and Relevance to SSBSE

Language-based testing combines grammars with semantic constraints to generate inputs that are not only syntactically well formed, but also accepted by downstream processors. This is valuable for complex formats such as CSV, TAR, XML, and binary image formats, where simple random or grammar-only generation is often insufficient. The original FANDANGO paper [7], published at ISSTA 2025, addresses a central bottleneck of language-based testing: how to satisfy semantic constraints efficiently.

Existing tools such as ISLa [5] rely on symbolic constraint solving. This approach is precise and elegant, but it can be slow and ties the specification language to solver-supported operations. FANDANGO explores a different point in the design space. Instead of solving constraints symbolically, it interprets them concretely and searches for valid inputs using an evolutionary algorithm. This shift directly connects language-based testing [6] with search-based software engineering.

The work is particularly relevant to the SSBSE community for three reasons. First, it reformulates valid-input generation as search over structured individuals. Second, it shows how representation and operators matter: derivation trees preserve syntax, while crossover and mutation are specialized to the parts of an

input responsible for failed constraints. Third, it provides an empirical comparison against the solver-based state of the art, showing that heuristic search can outperform exact solving in throughput without giving up the practical benefits of valid-input generation.

2 Search-Based Language-Based Testing with FANDANGO

FANDANGO takes as input a context-free grammar together with semantic constraints. The grammar captures syntax; the constraints capture dependencies between grammar elements. For instance, in GIF, a size field determines how many color-table entries are valid. In FANDANGO, such relations can be written directly in Python instead of being encoded into an SMT-friendly mini-language. This is one of the core ideas of the paper: concrete evaluation makes the specification language far more expressive while keeping the generation process declarative at the syntactic level.

Inputs are represented as derivation trees. This choice is crucial for search because every node is associated with the grammar rule that produced it, allowing FANDANGO to identify the parts of an input that contribute to a constraint violation. Fitness is computed as a weighted sum of per-constraint satisfaction scores. Fully satisfied constraints contribute the maximum score, while violated constraints contribute lower scores; for numerical constraints, FANDANGO can also assign intermediate scores based on distance to the target value. These partial scores provide the gradient-like signals that random generation lacks.

Search proceeds with a standard evolutionary loop. The initial population is generated from the grammar, or alternatively derived from existing sample inputs. Selection uses elitism and tournament selection to balance exploitation and diversity. Crossover chooses a *failing subtree* in one parent and exchanges it with a matching subtree from another parent. Mutation regenerates a failing subtree from the same nonterminal. Because both operators are grammar-aware, offspring remain syntactically valid by construction. Unlike byte-level fuzzers, the search therefore spends its effort on satisfying semantics rather than repeatedly repairing broken syntax.

The paper also demonstrates that concrete constraint evaluation broadens the space of useful input specifications. Constraints can call arbitrary Python code and libraries, including hashing functions, external data collections, and statistical distributions. Beyond the standard benchmarks, the original paper reports case studies on GIF and SVG, showing that the approach scales to binary and highly structured formats and is not limited to toy grammars.

3 Evaluation Summary

The evaluation in the original paper compares FANDANGO with ISLa on five benchmarks from the ISLa suite: CSV, reStructuredText, ScriptSizeC, TAR,

Table 1. Summary of one-hour runs on the five ISLa benchmark subjects. Coverage reports 4-path coverage for FANDANGO and ISLa.

Subject	Inputs/s	Speedup	Precision	4-path cov. (F/I)
CSV	2792.39	1355×	99.74%	100/100
reST	1195.34	146×	99.27%	100/95.13
ScriptSizeC	200.75	29×	99.51%	96.00/55.67
TAR	13.33	48×	100.00%	74.00/87.42
XML	433.70	183×	99.67%	100/90.50

and XML. To keep the comparison fair, the authors adapted ISLa's benchmark specifications and ran both tools for one hour per subject on the same machine and with the same subject-specific parsers. The reported metrics are throughput, parser acceptance (precision), grammar coverage using 4-path coverage [3], and input length. Table 1 summarizes the central quantitative results.

The most striking result is throughput. FANDANGO achieves between 29× and 1355× more accepted inputs per second than ISLa. At the same time, accepted-input precision remains between 99.27% and 100%, which means the speedup is not obtained by relaxing validity. Diversity likewise remains competitive: FANDANGO reaches full 4-path coverage on CSV, reST, and XML, substantially improves over ISLa on ScriptSizeC, and trails ISLa only on TAR. The original paper also notes that FANDANGO typically produces shorter inputs than ISLa. From an SBSE perspective, this is interesting because it suggests that the search is not merely exploiting larger individuals to satisfy constraints, but often finds compact valid solutions.

Language-based testing has often been viewed as a trade-off between expressive validity constraints and practical throughput. The evaluation shows that a search-based formulation can shift this trade-off. Once constraints are encoded as fitness functions, the cost of symbolic reasoning is replaced by many cheap concrete evaluations, enabling hundreds or thousands of valid inputs per second on several subjects.

4 Lessons, Impact, and Follow-up

For SSBSE, the original paper contributes more than a speedup table. It offers a reusable design pattern for search over structured artifacts. The representation must preserve the invariants that are hard to rediscover by mutation; the fitness function must provide partial progress signals; and the operators should target the syntactic regions most likely to affect semantic validity. In FANDANGO, all three elements are aligned. That alignment helps explain why a comparatively classical genetic algorithm performs well in a domain where unguided mutations would mostly create invalid inputs.

However, search performs best when constraints admit informative intermediate scores. Non-monotonic constraints, narrow feasible regions, or unsatisfiable specifications can leave the algorithm with little guidance. Unlike an exact solver,

FANDANGO cannot prove that no solution exists, nor can it guarantee globally optimal inputs. These limitations identify the boundary where hybrid techniques, combining search with symbolic reasoning, are likely to be most promising.

This line of work has already opened several directions beyond the original ISSTA contribution. Recent work based on FANDANGO explores higher-performance generation of constrained inputs [2], coverage-aware and white-box fitness signals, and stateful protocol fuzzing via interaction grammars [4]. These extensions push the original ideas beyond file formats toward complex, security-critical, and often stateful protocols, uncovering vulnerabilities in targets such as TLS and FTP. This illustrates that valid structured interactions are essential for exposing deeper flaws. In parallel, advances in automatic grammar inference [1] lower the cost of obtaining the structured specifications that tools like FANDANGO rely on. Taken together, these developments suggest a broader research agenda at the intersection of SBSE, fuzzing, and input-language engineering.

In summary, FANDANGO shows that search-based optimization can serve as a practical replacement for symbolic solving in language-based testing when the search space is structured and the fitness function reflects semantic distance. That combination makes the work a natural fit for SSBSE and a useful case study in how classical evolutionary ideas can unlock high-throughput generation of complex valid inputs. FANDANGO is open source and available at:

`https://github.com/fandango-fuzzer/fandango`.

References

1. Bettscheider, L., Zeller, A.: Inferring input grammars from code with symbolic parsing. ACM Trans. Softw. Eng. Methodol. (Nov 2025). https://doi.org/10.1145/3776743, https://doi.org/10.1145/3776743, just Accepted
2. Crump, A., Turcotte, A., Amaya, J.A.Z., Zeller, A.: High-performance generation of constrained inputs (2025), https://arxiv.org/abs/2511.05987
3. Havrikov, N., Zeller, A.: Systematically covering input structure. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. p. 189–199. ASE '19, IEEE Press (2020). https://doi.org/10.1109/ASE.2019.00027, https://doi.org/10.1109/ASE.2019.00027
4. Liggesmeyer, A., Amaya, J.A.Z., Zeller, A.: Language-based protocol testing (2026), https://arxiv.org/abs/2509.20308
5. Steinhöfel, D., Zeller, A.: Input invariants. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 583–594. ESEC/FSE 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3540250.3549139, https://doi.org/10.1145/3540250.3549139
6. Steinhöfel, D., Zeller, A.: Language-based software testing. Commun. ACM **67**(4), 80–84 (Mar 2024). https://doi.org/10.1145/3631520, https://doi.org/10.1145/3631520
7. Zamudio Amaya, J.A., Smytzek, M., Zeller, A.: Fandango: Evolving language-based testing. Proc. ACM Softw. Eng. **2**(ISSTA) (Jun 2025). https://doi.org/10.1145/3728915, https://doi.org/10.1145/3728915