REFERENCES

[1] P. Purdom, “A sentence generator for testing parsers,” *Bit*, vol. 12, no. 3, pp. 366–375, 1972, doi: 10.1007/BF01932308.

[2] J. Dongarra, M. Furtney, S. Reinhardt, and J. Russell, “Parallel loops - A test suite for parallelizing compilers: Description and example results,” *Parallel Comput.*, vol. 17, no. 10–11, pp. 1247–1255, 1991, doi: 10.1016/S0167-8191(05)80036-5.

[3] K. H. Wolf and M. Klimek, “A conformance test suite for Arden syntax compilers and interpreters,” *Stud. Health Technol. Inform.*, vol. 228, pp. 379–383, 2017, doi: 10.3233/978-1-61499-678-1-379.

[4] R. Report, “Rapport report,” no. January, 1993.

[5] F. Bazzichi and I. Spadafora, “An Automatic Generator for Compiler Testing,” *IEEE Trans. Softw. Eng.*, vol. SE-8, no. 4, pp. 343–353, 1982, doi: 10.1109/TSE.1982.235428.

[6] S. V. Zelenov, S. A. Zelenova, A. S. Kossatchev, and A. K. Petrenko, “Test generation for compilers and other formal text processors,” *Program. Comput. Softw.*, vol. 29, no. 2, pp. 104–111, 2003, doi: 10.1023/A:1022904917707.

[7] D. E. Knuth, “Semantics of context-free languages,” *Math. Syst. Theory*, vol. 2, no. 2, pp. 127–145, 1968, doi: 10.1007/BF01692511.

[8] A. S. Boujarwah and K. Saleh, “Compiler test case generation methods: A survey and assessment,” *Inf. Softw. Technol.*, vol. 39, no. 9, pp. 617–625, 1997, doi: 10.1016/S0950-5849(97)00017-7.

[9] M. Amodio, S. Chaudhuri, and T. W. Reps, “Neural Attribute Machines for Program Generation,” 2017, [Online]. Available: http://arxiv.org/abs/1705.09231

[10] C. H. A. Koster, “Affix grammars for programming languages,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 545 LNCS, pp. 358–373, 1991, doi: 10.1007/3-540-54572-7_13.

[11] HANFORD KV, “Automatic Generation of Tests Cases,” *IBM Syst. J.*, vol. 9, no. 4, pp. 242–257, 1970, doi: 10.1147/sj.94.0242.

[12] N. B. Bershad and E. Sirer Gun, “Using Production Grammars in Software Testing Department of Computer Science,” *SIGPLAN Not.*, vol. 35, no. Jan. 2000, pp. 1--13, 1999, [Online]. Available: http://doi.acm.org/10.1145/331963.331965

[13] C. Holler, K. Herzig, and A. Zeller, “Fuzzing with code fragments,” *Proc. 21st USENIX Secur. Symp.*, pp. 445–458, 2012.

[14] A. S. Boujarwah, K. Saleh, and J. Al-Dallal, “Testing syntax and semantic coverage of Java language compilers,” *Inf. Softw. Technol.*, vol. 41, no. 1, pp. 15–28, 1999, doi: 10.1016/S0950-5849(98)00075-5.

[15] S. Park, W. Xu, I. Yun, D. Jang, and T. Kim, “Fuzzing JavaScript engines with aspect-preserving mutation,” *Proc. - IEEE Symp. Secur. Priv.*, vol. 2020-May, pp. 1629–1642, 2020, doi: 10.1109/SP40000.2020.00067.

[16] Y. Koroglu and F. Wotawa, “Fully automated compiler testing of a reasoning engine via mutated grammar fuzzing,” *Proc. - 2019 IEEE/ACM 14th Int. Work. Autom. Softw. Test, AST 2019*, pp. 28–34, 2019, doi: 10.1109/AST.2019.00010.

[17] V. Le, M. Afshari, and Z. Su, “Compiler validation via equivalence modulo inputs,” *ACM SIGPLAN Not.*, vol. 49, no. 6, pp. 216–226, 2014, doi: 10.1145/2594291.2594334.

[18] C. Lidbury, A. Lascu, N. Chong, and A. F. Donaldson, “Many-core compiler fuzzing,” *Proc. ACM SIGPLAN Conf. Program. Lang. Des. Implement.*, vol. 2015-June, pp. 65–76, 2015, doi: 10.1145/2737924.2737986.

[19] C. Sun, V. Le, and Z. Su, “Finding compiler bugs via live code mutation,” *ACM SIGPLAN Not.*, vol. 51, no. 10, pp. 849–863, 2016, doi: 10.1145/2983990.2984038.

[20] W. M. McKeeman, “Differential Testing for Software,” *Digit. Tech. J.*, vol. 10, no. 1, pp. 100–107, 1998, [Online]. Available: http://www.cs.dartmouth.edu/~mckeeman/references/DifferentialTestingForSoftware.pdf

[21] F. Sheridan, “Practical testing of a C99 compiler using output comparison,” *Softw. - Pract. Exp.*, vol. 37, no. 14, pp. 1475–1488, 2007, doi: 10.1002/spe.812.

[22] M. Sassa, “Experience in Testing Compiler Optimizers Using Comparison Checking . Experience in Testing Compiler Optimizers Using Comparison Checking,” no. June, 2014.

[23] C. Hawblitzel *et al.*, “Will you still compile me tomorrow?: Static cross-version compiler validation,” *2013 9th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. ESEC/FSE 2013 - Proc.*, pp. 191–201, 2013, doi: 10.1145/2491411.2491442.

[24] G. Ofenbeck, T. Rompf, and M. Püschel, “RandIR: Differential testing for embedded compilers,” *SCALA 2016 - Proc. 2016 7th ACM SIGPLAN Symp. Scala*, pp. 21–30, 2016, doi: 10.1145/2998392.2998397.

[25] A. F. Donaldson and A. Lascu, “Metamorphic testing for (graphics) compilers,” *Proc. - Int. Conf. Softw. Eng.*, vol. 16, pp. 44–47, 2016, doi: 10.1145/2896971.2896978.

[26] D. Xiao, Z. Liu, Y. Yuan, Q. Pang, and S. Wang, “Metamorphic Testing of Deep Learning Compilers,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 1, 2022, doi: 10.1145/3508035.

[27] J. Liu *et al.*, “NNSmith: Generating Diverse and Valid Test Cases for Deep Learning Compilers,” *Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS*, vol. 2, pp. 530–543, 2023, doi: 10.1145/3575693.3575707.

[28] C. Sun, V. Le, and Z. Su, “Finding and analyzing compiler warning defects,” *Proc. - Int. Conf. Softw. Eng.*, vol. 14-22-May-2016, pp. 203–213, 2016, doi: 10.1145/2884781.2884879.

[29] G. Barany, F. Missed, C. Optimizations, T. Cc, and G. Barany, “Finding Missed Compiler Optimizations by Differential Gergö Barany To cite this version : HAL Id : hal-01682683 Finding Missed Compiler Optimizations by Differential Testing ∗,” 2018.

[30] F. Engel, R. Leupers, G. Ascheid, M. Ferger, and M. Beemster, “Enhanced structural analysis for C code reconstruction from IR code,” *Proc. 14th Int. Work. Softw. Compil. Embed. Syst. SCOPES 2011*, pp. 21–27, 2011, doi: 10.1145/1988932.1988936.

[31] V. Le, C. Sun, and Z. Su, “Finding deep compiler bugs via guided stochastic program mutation,” *ACM SIGPLAN Not.*, vol. 50, no. 10, pp. 386–399, 2015, doi: 10.1145/2858965.2814319.

[32] C. Sun, V. Le, Q. Zhang, and Z. Su, “Toward understanding compiler bugs in GCC and LLVM,” *ISSTA 2016 - Proc. 25th Int. Symp. Softw. Test. Anal.*, no. 2, pp. 294–305, 2016, doi: 10.1145/2931037.2931074.

[33] A. Gül and V. Zaytsev, “Mutative fuzzing for an assembler compiler,” *CEUR*

Workshop Proc., vol. 2605, pp. 1–7, 2019.
[34] X. Li, X. Liu, L. Chen, R. Prajapati, and D. Wu, "FuzzBoost : Reinforcement Compiler Fuzzing," pp. 1–17.
[35] M. Wu, M. Lu, and J. Chen, "JITfuzz : Coverage-guided Fuzzing for JVM Just-in-Time Compilers".
[36] X. Liu, X. Li, R. Prajapati, and D. Wu, "DeepFuzz : Automatic Generation of Syntax Valid C Programs for Fuzz Testing," 2015.
[37] J. Chen, A. F. Donaldson, and A. Zeller, "Edited by," vol. 7, no. 12, pp. 50–65.
[38] Y. Li and F. Wotawa, "On Using Ontologies for Testing Compilers," pp. 181–184, 2020, doi: 10.1109/ICSTW50294.2020.00039.
[39] R. Mcnally, K. Yiu, and D. Grove, "Fuzzing : The State of the Art".
[40] S. Chaliasos, A. Gervais, and B. Livshits, "Finding Typing Compiler Bugs," pp. 183–198, 2022.
[41] V. Livinskii, D. Babokin, and J. Regehr, "Random Testing for C and C ++ Compilers with YARPGen," vol. 4, no. November, 2020, doi: 10.1145/3428264.