zbMATH Open
THE FIRST RESOURCE FOR MATHEMATICS

Juma, Nahid; **Dietl, Werner**; **Tripunitara, Mahesh**

A computational complexity analysis of tunable type inference for Generic Universe Types.

(English) Zbl 1435.68059
Theor. Comput. Sci. 814, 189-209 (2020).

Summary: We address questions related to the computational complexity of inferring types for a particular type system, Generic Universe Types (GUT), for Java programs. GUT is useful for many applications, such as program verification, thread synchronization, and memory management. However, requiring the programmer to explicitly provide type information is onerous, which motivates the problem of automatically inferring types. In contrast to classical type systems, ownership type systems such as GUT may have multiple typings that satisfy the type system's rules. It is therefore appropriate for the inference to be tunable – that is, the programmer can indicate preferences for certain typings via breakable constraints and/or by partial annotations. A question then is whether efficient algorithms exist for the type inference problem. In this work we establish the following results for the type inference problem for GUT [*W. Dietl* et al., Lect. Notes Comput. Sci. 6813, 333–357 (2011; Zbl 1435.68061)]. (1) The tunable type inference problem that allows breakable constraints is **NP**-hard, (2) an encoding of the problem as Boolean satisfiability (SAT), as in prior work, is indeed a polynomial-time reduction, (3) $\mathbf{P} \neq \mathbf{NP}$ implies that the problem is not approximable in polynomial time within an approximation ratio of $n^{1-\epsilon}$ for any $\epsilon > 0$, and (4) while some restricted versions of the problem of practical interest, such as when breakable constraints are forbidden, are in **P**, others remain **NP**-hard. Our results justify the prior approach to the problem that is based on reduction to SAT. Apart from these results, given the observation in prior work that instances of the problem that arise in practice appear to be easy, we address the natural question as to what hard instances may look like, and whether they may arise in practice. We identify a class of hard instances of the problem by devising a method to generate such instances starting at instances of the Vertex Cover problem, which is known to be **NP**-hard. We then analyze the structural properties of such instances as compared to easy instances of similar size. We find that for the classes of instances we consider, certain SAT structural parameters may be predictive of empirical hardness.

MSC:

68N15 Theory of programming languages
68N19 Other programming paradigms (object-oriented, sequential, concurrent, automatic, etc.)
68Q17 Computational difficulty of problems (lower bounds, completeness, difficulty of approximation, etc.)
68Q25 Analysis of algorithms and problem complexity
68R07 Computational aspects of satisfiability

Keywords:

computational complexity; programming languages; Java; type inference; type systems; object ownership; NP-hardness

Software:

Lingeling; Plingeling; Treengeling

Full Text: DOI

References:

[1] Dietl, W.; Drossopoulou, S.; Müller, P., Generic Universe Types, (European Conference on Object-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP (2007)), 28-53

[2] Dietl, W.; Ernst, M. D.; Müller, P., Tunable static inference for Generic Universe Types, (European Conference on Object-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP (2011)), 333-357

[3] Dietl, W.; Müller, P., Object ownership in program verification, (Clarke, D.; Noble, J.; Wrigstad, T., Aliasing in Object-Oriented Programming. Aliasing in Object-Oriented Programming, Lecture Notes in Computer Science (2012))

© 2021 FIZ Karlsruhe GmbH

[4] Bracha, G., Pluggable type systems, (Workshop on Revival of Dynamic Languages (2004))

[5] Huang, W.; Dietl, W.; Milanova, A.; Ernst, M. D., Inference and checking of object ownership, (European Conference on Object-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP (2012)), 181-206

[6] Dietl, W.; Müller, P., Universes: lightweight ownership for JML, J. Object Technol., 4, 8, 5-32 (2005)

[7] Karp, R. M., Reducibility among combinatorial problems, (Complexity of Computer Computations (1972)), 85-103

[8] Arora, S., The approximability of NP-hard problems, (Symposium on Theory of Computing (1998), ACM), 337-348

[9] Håstad, J., Clique is hard to approximate within \(n^{1 - \epsilon}\), (Foundations of Computer Science (1996)), 627-636

[10] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C., Introduction to Algorithms (2009)

[11] SHARCNET

[12] Biere, A., Lingeling, plingeling and treengeling entering the SAT competition 2013, (Proceedings of SAT Competition 2013, vol. B-2013-1 (2013)), 51-52, available from

[13] Nudelman, E.; Leyton-Brown, K.; Hoos, H.; Devkar, A.; Shoham, Y., Understanding random SAT: beyond the clauses-to-variables ratio, (Principles and Practice of Constraint Programming, vol. 3258 (2004)), 438-452

[14] Selman, B.; Mitchell, D. G.; Levesque, H. J., Generating hard satisfiability problems, Artif. Intell., 81, 1-2, 17-29 (1996)

[15] Flanagan, C.; Freund, S. N., Type inference against races, (Static Analysis Symposium (2004)), 116-132

[16] Flanagan, C.; Freund, S. N., Detecting race conditions in large programs, (Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (2001), ACM), 90-96

[17] Elder, M.; Liblit, B., Heap typability is NP-complete (Oct. 2007), University of Wisconsin-Madison, Tech. Rep. 1618

[18] Polishchuk, M.; Liblit, B.; Schulze, C. W., Dynamic Heap Type Inference for Program Understanding and Debugging, ACM SIGPLAN Notices, vol. 42, 39-46 (2007), ACM

[19] Gagnon, E.; Hendren, L. J.; Marceau, G., Efficient inference of static types for Java bytecode, (Static Analysis Symposium (2000)), 199-219

[20] Tiuryn, J., Subtype inequalities, (Logic in Computer Science (1992)), 308-315

[21] Palsberg, J.; Zhao, T., Type inference for record concatenation and subtyping, Inf. Comput., 189, 1, 54-86 (2004)

[22] Palsberg, J.; Jim, T., Type inference with simple selftypes is NP-complete, Nord. J. Comput., 4, 259-286 (1997)

[23] Lincoln, P.; Mitchell, J. C., Algorithmic aspects of type inference with subtypes, (Principles of Programming Languages. Principles of Programming Languages, POPL (1992)), 293-304

[24] Hoang, M.; Mitchell, J. C., Lower bounds on type inference with subtypes, (Principles of Programming Languages. Principles of Programming Languages, POPL (1995)), 176-185

[25] Haque, R.; Palsberg, J., Type inference for place-oblivious objects, (European Conference on Object-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP, vol. 37 (2015)), 371-395

This reference list is based on information provided by the publisher or from digital mathematics libraries. Its items are heuristically matched to zbMATH identifiers and may contain data conversion errors. It attempts to reflect the references listed in the original paper as accurately as possible without claiming the completeness or perfect precision of the matching.