Tree Kernels to Support Formal Methods-Based Testing of Evolving Specifications

Francesco Altiero[1], Anna Corazza[1], Sergio Di Martino[1], Adriano Peron[2] and Luigi Libero Lucio Starace[1]

[1]Università degli Studi di Napoli Federico II
[2]Università degli Studi di Trieste

Abstract

Tree Kernels (TKs) are a family of functions measuring the similarity between two tree-structured objects. TKs have been successfully employed in several fields of AI, including Natural Language Processing and Software Engineering (e.g.: software testing and code clone detection). A recent research line has proved that the information about source code changes captured by TKs can fruitfully be applied to select and prioritize test cases in Regression Testing, a crucial activity in modern software development processes. We suggest that a similar approach can be adopted also in the field of formal specification of safety-critical systems, whenever a structured language (e.g., a variant of Statecharts, hierarchical automata or the Promela description language) is adopted for the specification task and test cases are (semi-)automatically generated from the specifications. In evolutionary approaches to specification development, the information on changes between two consecutive versions of the specification can aid in focusing the activity of test generation and their execution on the specification modules that were mainly affected by the specification changes.

Keywords

AI, Formal Verification, Change-driven testing, Tree Kernels

1. Introduction and Background

Safety-critical systems are nowadays an integral part of our lives, impacting everything from healthcare to transportation and beyond. The consequences of a failure in these systems can range from costly inconveniences to potentially catastrophic disasters and thus ensuring their reliability and safety is of paramount importance. Formal methods have emerged as a vital tool in addressing these challenges.

In particular, hierarchical modelling languages and verification toolkits have been proposed and effectively adopted in many different contexts to handle the increasing complexity of such systems. An example of a specification environment which exploits hierarchically structured models is proposed in [1]. That work introduces a variant of Statecharts called Dynamic State Machines (DSTMs) and the encoding of DSTM models into Promela, the description language

OVERLAY 2023: 5th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 7, 2023, Rome, Italy

francesco.altiero@unina.it (F. Altiero); anna.corazza@unina.it (A. Corazza); sergio.dimartino@unina.it (S. Di Martino); adriano.peron@units.it (A. Peron); luigiliberolucio.starace@unina.it (L. L. L. Starace)
0000-0001-7090-4249 (F. Altiero); 0000-0002-9156-5079 (A. Corazza); 0000-0002-1019-9004 (S. Di Martino); 0000-0002-7111-3171 (A. Peron); 0000-0001-7945-9014 (L. L. L. Starace)

adopted by the Spin model checker, to endow the formalism with an executable semantic. The translation, which has been further refined in [2], enables both the automated verification of behavioural properties [3] on the modelled system and the automated generation of test cases, by exploiting the model checker's ability to extract counterexamples of violated properties. A case study witnessing the interest of the approach in an industrial setting has also been reported in [4]. It is worth noting that both DSTM models and their Promela executable semantics can be seen as tree-structured artifacts. Indeed, a DSTM specification has been already encoded as a tree-structured XML file in [2], and Promela code can be represented using the Abstract Syntax Tree resulting from its parsing.

In this work, we present a novel approach we are currently investigating, aimed at supporting the verification efforts for such hierarchical models in evolutionary scenarios, i.e., when changes are made to an existing specification. Inspired by the effectiveness of Tree Kernel functions in a number of Natural Language Processing and Software Engineering tasks, where they have been applied to compute meaningful structural similarity measures between tree-structured data, we envision fruitfully applying them also in the formal verification field. In particular, we aim to measure the entity of changes to a hierarchical specification by using Tree Kernels to compare a new version of a DSTM specification (or of its Promela semantics) with the previous one. Tree Kernels could serve as an effective way of identifying the most critical changes in the hierarchical specification, and this information could be crucial in directing costly test-case generation efforts towards covering the parts of the specification that have been affected by the most relevant changes.

The remainder of this work is structured as follows. In Section 2 we provide preliminary information and some related works on Tree Kernel functions. In Section 3 we present our vision for adopting Tree Kernels to support verification efforts of hierarchical specifications in evolutionary scenarios. Lastly, in Section 4, we give some closing remarks.

2. Tree Kernels for Regression Test Prioritization

Tree-based structures can be naturally employed to model several kinds of structured information in different branches of computer science. The evaluation of similarity between tree structures is a common operation in different tasks of various domains, such as Natural Language Processing (NLP) and Software Engineering. *Tree Kernels* (TKs) [5] are a class of Kernel functions used to assess similarity between tree structures, and have been successfully applied in both NLP [6, 7] and Software Engineering [8, 9]. Given two tree-based structures, TKs evaluate their similarity by considering the number of different *tree fragments* the structures have in common. Different kinds of considered fragments give rise to different TK functions, e.g., *SubTree Kernels* takes into account the sub-trees of the original structures, while *Partial Treek Kernels* rely on parts of the original tree [10]. Some kind of TKs, such as *Smoothed Partial Tree Kernels* [11], include also *semantic* information, allowing to define a *weighting* function for the labels of tree-nodes.

All TK functions have different tunable parameters, such as a decay factor for the depth of a node in the considered fragments, or a penalization factor for gaps between the considered fragments. The main advantage of TK functions relies on their flexibility, as they can be tuned to highlight sub-structures which are specific to the problem at hand, and are often used along

other machine learning models such as *Support Vector Machine* [12]. TKs can be efficiently evaluated using dynamic programming algorithms [10].

In the Software Engineering domain, Tree Kernels have typically been employed on the *Abstract Syntax Tree* (AST) representation of the source code, in order to obtain the similarity between two code fragments. ASTs are a tree-based and natural representation of source code, highlighting both its structural properties and semantic properties. Each construct of the programming language is converted into a tree node, labeled with its semantic information (e.g.,*loop*, *statement* or *expression*), while parent-child relations reflect how the different constructs are nested within each other. Leaves of the tree represent terminal tokens such as variables or constants defined in the source code. ASTs have been exploited in several studies [13, 14, 15] due to the flexibility and the high degree of customization they hold, which makes their structure and semantics thoroughly adaptable to the problem at hand.

Our research lines focused on the application of Tree Kernels and Abstract Syntax Trees in the field of *Regression Test Prioritization* (RTP) [16], a widely-adopted practice to ease *Regression Testing*, one of the most resource-consuming activities of the software validation phase[17, 18]. RTP is applied in scenarios of limited resources allocated to the testing phase, where it is typically not possible to execute the entire test suite of the application. RTP approaches aim to find a permutation of the test cases in order to discover as many faults as possible within this limited resource window. Different techniques for RTP assign *priorities* to test cases according to some properties, e.g., *test coverage* [16], *test execution history* [19] or *code changes* [20].

Our studies on the application of TKs to RTP showed that the approach is effective in estimating the fault-proneness of source code changes [21], and its application to RTP has led to promising results. We employed Tree Kernel functions to assess the similarity of the ASTs related to the previous and current version of changed methods/functions and schedule the test cases giving higher priority to those covering the units with the higher degree of changes. The rationale of this approach is that procedures which exhibit deeper changes are more likely to show faults than functions which have been subjected to a lower degree of changes.

We implemented our technique for the Java programming language and executed it in different experimental settings, concerning both datasets of benchmark software projects with artificially injected faults and on projects with real faults. We then evaluated the technique performances according to common metrics used in prioritization (e.g., *Average Percentage of Fault Detected*). We compared our technique with several RTP approaches commonly used in practice. In particular, some of these approaches did not rely on changes between the version (e.g., only on test coverage), while other approaches were aware of changes, the latter employing a textual representation of the source code (e.g., *diff* -like analyses). The results of our experiments were promising and better than the other considered techniques for all metrics, indicating that TKs can produce a more meaningful assessment of changes within the code.

3. Tree Kernels for Hierarchical Specifications

As the source code changes during an evolutionary step in a software lifecycle, so does the formal specification of a complex safety critical system during its design process. In this scenario, we will focus on changes in both DSTM specifications and their Promela executable semantics. In

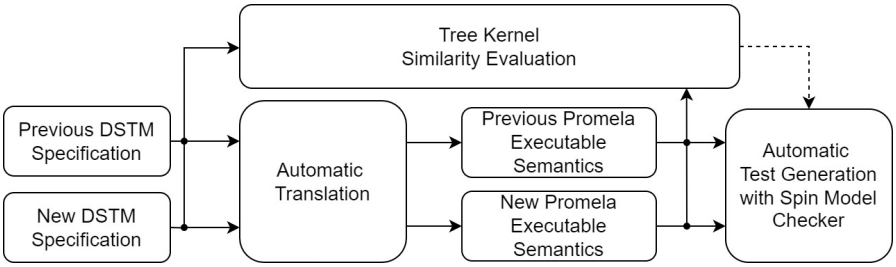

Figure 1: Envisioned pipeline of the steps involved in the evaluation of Tree Kernels to generate test cases from DSTM specifications and Promela semantics.

such evolutionary contexts, to validate the new model one should: (i) execute all the previously assessed simulations (test cases); and (ii) generate new test cases to improve coverage metrics of the new specification (state/transition coverage of DSTM specification or Promela line code coverage for the executable semantics). Following the encouraging experience in the case of software regression testing guided by code churn, we are confident that benefits can be derived in the validation cost of the specification evolution: (i) simulations priority can be rearranged according to their impact on changed parts; (ii) costly automatic test generation routines based on model checking techniques can be invoked only for changed parts not covered by simulations at point (i). Notice that in the context of DSTM specifications, changes can be evaluated at two different detail levels: either at the more abstract detail of DSTM specification or at a more detailed level considering Promela code automatically generated from DSTM specifications. A graphical sketch of the outlined process is depicted in Figure 1.

We are currently working on the application of Tree Kernel functions to evaluate the similarity, and consequentially the degree of changes, of the tree XML representation of the DSTM model and of ASTs of the executable semantic given by Promela. As Tree Kernels proved useful in capturing the similarity of HTML documents [9], we are confident that they can produce fruitful results also in the analysis of DSTM models serialization (e.g., in XML format). Furthermore, as the Promela encoding of a model is a fully-qualified programming language, it is possible to extract AST representations from the encoding and thus analyze their similarity using the same promising approach we designed for RTP.

4. Conclusion

In this paper, we discussed an application of similarity measures to aid formal verification activities in an evolving software, which we are currently investigating. We are designing and implementing methods to use Tree Kernels on hierarchical models produced by DSTM specifications and on the artifacts produced by the description language Promela, both having a natural tree-based representation. Our research is motivated by our previous and promising studies of Tree Kernels to measure code similarity in the Regression Test Prioritization domain.

The analysis of changes in the DSTM specifications can indeed drive the test generation to stress the most critically changed parts in the software. As formal methods-based test generation is a costly task, focusing on the critical sections could save important resources and - we envision - lead to an earlier discovery of faults.

Acknowledgements

This work was partially funded by the PNRR MUR project PE0000013-FAIR.

References

[1] M. Benerecetti, R. D. Guglielmo, U. Gentile, S. Marrone, N. Mazzocca, R. Nardone, A. Peron, L. Velardi, V. Vittorini, Dynamic state machines for modelling railway control systems, Sci. Comput. Program. 133 (2017) 116–153. doi:10.1016/j.scico.2016.09.002.

[2] M. Benerecetti, U. Gentile, S. Marrone, R. Nardone, A. Peron, L. L. Starace, V. Vittorini, From dynamic state machines to promela, in: Model Checking Software: 26th International Symposium, SPIN 2019, Beijing, China, July 15–16, 2019, Proceedings 26, Springer, 2019, pp. 56–73.

[3] M. Benerecetti, F. Mogavero, A. Peron, L. L. L. Starace, Expressing structural temporal properties of safety critical hierarchical systems, in: Quality of Information and Communications Technology: 14th International Conference, QUATIC 2021, Algarve, Portugal, September 8–11, 2021, Proceedings 14, Springer, 2021, pp. 356–369.

[4] R. Nardone, S. Marrone, U. Gentile, A. Amato, G. Barberio, M. Benerecetti, R. D. Guglielmo, B. D. Martino, N. Mazzocca, A. Peron, G. Pisani, L. Velardi, V. Vittorini, An oslc-based environment for system-level functional testing of ERTMS/ETCS controllers, J. Syst. Softw. 161 (2020). doi:10.1016/j.jss.2019.110478.

[5] D. Haussler, Convolution Kernels on Discrete Structures, Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999. URL: http://citeseer.ist.psu.edu/haussler99convolution.html.

[6] D. Liga, M. Palmirani, Classifying argumentative stances of opposition using tree kernels, in: Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, ACAI '19, Association for Computing Machinery, New York, NY, USA, 2020, p. 17–22. doi:10.1145/3377713.3377717.

[7] D. Hovy, S. Srivastava, S. Jauhar, M. Sachan, K. Goyal, H. Li, W. Sanders, E. Hovy, Identifying metaphorical word use with tree kernels, in: Proceedings of the First Workshop on Metaphor in NLP, Association for Computational Linguistics, 2013, pp. 52–57.

[8] A. Corazza, S. Di Martino, V. Maggio, G. Scanniello, A tree kernel based approach for clone detection, in: 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1–5. doi:10.1109/ICSM.2010.5609715.

[9] A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Web application testing: Using tree kernels to detect near-duplicate states in automated model inference, in: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), ESEM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–6. doi:10.1145/3475716.3484187.

[10] A. Moschitti, Efficient convolution kernels for dependency and constituent syntactic trees, in: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 318–329.

[11] D. Croce, A. Moschitti, R. Basili, Semantic convolution kernels over dependency trees:

Smoothed partial tree kernel, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 2013–2016. doi:10.1145/2063576.2063878.

[12] A. Moschitti, Making tree kernels practical for natural language learning., in: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics, 2006, p. 113–120.

[13] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, L. Bier, Clone detection using abstract syntax trees, in: Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), 1998, pp. 368–377. doi:10.1109/ICSM.1998.738528.

[14] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu, A novel neural source code representation based on abstract syntax tree, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 783–794. doi:10.1109/ICSE.2019.00086.

[15] B. Cui, J. Li, T. Guo, J. Wang, D. Ma, Code comparison system based on abstract syntax tree, in: 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), 2010, pp. 668–673. doi:10.1109/ICBNMT.2010.5705174.

[16] G. Rothermel, R. Untch, C. Chu, M. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on Software Engineering 27 (2001) 929–948. doi:10.1109/32.962562.

[17] F. Altiero, G. Colella, A. Corazza, S. Di Martino, A. Peron, L. L. Starace, Change-aware regression test prioritization using genetic algorithms, in: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2022, pp. 125–132.

[18] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. Starace, Recover: A curated dataset for regression testing research, in: Proceedings of the 19th International Conference on Mining Software Repositories, 2022, pp. 196–200.

[19] J.-M. Kim, A. Porter, A history-based test prioritization technique for regression testing in resource constrained environments, in: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, Association for Computing Machinery, New York, NY, USA, 2002, p. 119–129. doi:10.1145/581339.581357.

[20] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Inspecting code churns to prioritize test-cases, in: Testing Software and Systems: 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9–11, 2020, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2020, p. 272–285. doi:10.1007/978-3-030-64881-7_17.

[21] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, AI-Based fault-proneness metrics for source code changes, in: International Conference on Software Process and Product Measurement (IWSM/MENSURA) 23, 2023.