Static analysis and slicing of Erlang programs

Theses of the Doctoral Dissertation

István Bozó

Doctoral School of Informatics, Eötvös Loránd University
Erzsébet Csuhaj Varjú, DSc., habil
Head of School

Doctoral Programme of Foundations and Methodologies of Informatics
Zoltán Horváth, PhD, habil.
Head of Program

Department of Programming Languages and Compilers
Zoltán Horváth, PhD, habil.
Doctoral Advisor

2018

Chapter 1

Introduction

In my thesis I present new static analysis methods for functional programming language Erlang. I introduce rules for building control flow and control dependence graphs, and a method how to use these for change impact analysis. Beside this, I present a method based on static analysis how to extract the communication model of concurrent and distributed Erlang programs.

Functional programming languages are becoming more popular, thus there is a great need for tools to support development processes. Such tools may increase the productivity in development, provide help in code comprehension and debugging.

Static analysis tools only use the information that is available in the source code of the program. Such a static analysis tool is RefactorErl for the programming language Erlang. The source code in RefactorErl is represented in a so called *Semantic Program Graph*, that beside the lexical and syntactic layer, includes wide variety of semantic information. The presented methods take advantage of the facilities of this tool. The results may help in code comprehension, debugging, detecting candidates for parallelization and can be used for further analyses.

The presented static analysis methods were defined with rules, relations, patterns and algorithms. The newly developed algorithms have been introduced in RefactorErl.

Chapter 2

Results

The theses of my dissertation are:

- **1. thesis** I have defined a set of compositional rules for building control flow graphs of Erlang programs. The rules assign control flow edges to expressions according to the semantics of the language. I have developed the algorithm for calculating the execution paths based on the control flow graphs in the RefactorErl framework. The result of the analysis can be used in dependence analysis, or in paralellization techniques as well.

- **2. thesis** Based on the control flow graph, I have developed the control dependence graph for Erlang programs. I have defined a dependence graph, that includes both the control and data dependence information. I have provided an algorithm on how to use this dependence graph for program slicing. I have shown a method of how to use the resulted slice for impact analysis of refactorings, how to select the relevant test cases from a slice.

- **3. thesis** I have defined a communication model of concurrent Erlang programs and provided the static analysis algorithms required for its construction. I have extended the model and the algorithms to analyse Erlang behaviours. The developed model supports code comprehension, but it can be used to refine impact analysis of concurrent programs.

We have extended the analyzer framework of RefactorErl with the presented methods. We have validated the implementations with tests [TTB+12] and with application of the results in new analyses. Such applications are, for example, the identification of parallelizable components [TBK17], and test case selection based on impact analysis [FBT17].

We have analysed large scale applications, looking for code fragments that can be transformed to parallelable codes. Formal criterion have been defined for execution paths that must be met during the parallelable code identification. The implemented identification algorithm resulted in candidates that matched well for parallelization.

The test case selection have been applied on large scale applications, to select the relevant test cases affected by a given change. The preliminary results were appropriate, but the results showed that the analysis could be further improved by taking into account implicit dependencies between processes.

The communication model have been validated through use cases. We applied the analysis on different source codes and the results have been analysed manually.

2.1 Control flow analysis

The result of the control flow analysis is the control flow graph, which is an intermediate representation of the source code. The static execution paths can be obtained with traversal of the graph. The control flow information is widely used, for example, in compilers for optimization, identification code fragments for parallelization and impact analysis.

As the graph is defined with static analysis, it is an over approximation of the dynamic control flow, thus the traversal of the graph may result in paths that are not true execution paths.

If the scope of the analysis is a single function and does not cross the boundaries of the function, it is said to be an intrafunctional analysis. If the analysis covers the entire program, thus it follows the function calls, it is said to be an interfunctional analysis.

In the dissertation, I have defined the formal rules of the flow of control for the different syntactic categories according to the Erlang language semantics. The defined rules are intrafunctional, the function calls that can cross the function boundaries are marked by special edges. I have defined an algorithm in RefactorErl that calculates interfunctional execution paths. The algorithm uses the function call graph to resolve the special call edges. We have exploited interfunctional execution paths in identification of parallelizable pattern candidates.

2.1.1 Related publications

Main publications of the thesis: [TB12b, TB11].

Other publications related to the thesis: [TBK17, BFH+15, BFH+14, KTB18, KTBH16, TBH+10b, HBT14, BT11, TBH13, FBT17, BTT+11a, BTT+11b, TTB+12, HBTE10, TBHE11].

The publications listed here have a total of 30 independent citations, of which there are 8 independent citations for the main publications of the thesis (MTMT information).

2.2 Impact analysis

The impact analysis calculates those parts of the source code that can be affected by a change, or it measures the effect of a particular change.

I have used dependence graph based static forward slicing to perform impact analysis. I have defined dependence graph for Erlang programs, which includes both the control and data dependence edges. The criterion of the slicing is a node (or set of nodes) that is affected by a change. The forward slice can be obtained with a traversal starting from the selected node. The nodes in the calculated slice are potentially affected by the change.

This method have been used to calculate relevant test cases. The nodes in the resulted slice have been examined to determine the functions containing the expressions. The algorithm filtered out the property-based test cases from a set of functions. The selected test cases are the potentially affected relevant test cases.

This method can be used not only for selecting property-based test cases, it can be tailored to any other test sets.

2.2.1 Related publications

Main publications of the thesis: [HBT14, TB11, BT11, TB12b].
Other publications related to the thesis: [TBH13, FBT17, TTB+12, BTT+11a, BTT+11b, HBTE10, TBHE11, TBH+10b].

The publications listed here have a total of 14 independent citations, of which there are 9 independent citations for the main publications of the thesis (MTMT information).

2.3 Communication model

Comprehending concurrent programs can be a serious challenge, which can be greatly assisted by a static analysis tool.

I have defined analysis methods that extract the static communication model of concurrent Erlang programs.

The basic analysis algorithm only detects the processes that are started in a standard way and analyses the connections and direct and hidden communication between them. In order to detect the details of processes, the analysis uses data flow information.

Another algorithm have been defined that uses application-specific information for detecting hidden processes and indirect (through interface function) communications. Concurrent processes implemented as Erlang behaviours are called hidden processes, because the spawning and the communication is hidden from the developer. Such behaviours are often used in Erlang applications.

The resulted model can be used, for example, in code comprehension and debugging, or it can be used for checking violations of some predefined properties. The information provided by the model can also be used to refine the impact analysis of concurrent programs.

2.3.1 Related publications

Main publications of the thesis: [TB14, TB12b, BT16, BKT18].
Other publications related to the thesis: [LTB18, BST18, IM14, TB12a, TBHT10].

The publications listed here have a total of 9 independent citations, of which there are 6 independent citations for the main publications of the thesis (MTMT information).

Chapter 3

Summary

Functional programming languages are getting more popular nowadays, thus there is a high demand on new tools that may support the development process. There are two main types of such tools, one is operating with dynamic information by running the code, the other is performing static analysis on the source code of the program.

Erlang is a functional programming language designed for developing real world applications. The RefactorErl tool is an extensive static analyzer framework developed for Erlang. The tool offers several refactorings and code comprehension support for developers.

In general, the focus of my research was to develop new static analysis methods. These methods extract compound semantic information from the source code, and the result could be used for other analysis methods. My results are related to control flow, control dependence, impact analysis and communication model of Erlang programs.

In my dissertation, I have presented formal control flow rules based on the semantics of the language. The rules are compositional and can be used for developing control flow graph of an expression, or a function. The results of the control flow graph have been used further analysis techniques, like discovering parallelable components in legacy source codes.

The information available in the control flow graphs can be used in many ways. It can be used for parallelization, debugging or change impact analysis. The control dependence graph is a more compact representation compared to control flow graph. It includes only direct control dependencies, while the control flow graph contains every execution path of a program. I have presented and extended the control dependence graph with data dependencies, what we call Erlang dependence graph. I have presented an impact analysis method based on the dependence graph. This method can be used for relevant test case selection. It selects only those test cases that could be affected by a

change. The presented method can not only be used for impact analysis of refactorings, but can be generalized for an arbitrary modification.

I have also presented a method for extracting the communication model from Erlang source codes. I have described the algorithms that can be used to identify the processes and the possible communication between them. I have extended the model with the analysis results of hidden communication. The *Erlang Term Storage (ets)* tables can be used as shared memory between processes. Any reading or writing operation is taken as an interaction with other processes accessing the same table. I have added analysis of generic server behaviors as another extension to the process model. This introduces another type processes and the hidden communication. The results can be used in code comprehension techniques, but the results from the communication model could be used in impact analysis as well.

List of related publications

[BFH+14] István Bozó, Viktória Fördős, Zoltán Horváth, Melinda Tóth, Dániel Horpácsi, Tamás Kozsik, Judit Köszegi, Adam Barwell, Christopher Brown, and Kevin Hammond. Discovering parallel pattern candidates in Erlang. In *Proceedings of the Thirteenth ACM SIGPLAN Workshop on Erlang*, Erlang '14, pages 13–23, New York, NY, USA, 2014. ACM.

[BFH+15] István Bozó, Viktória Fördős, Dániel Horpácsi, Zoltán Horváth, Tamás Kozsik, Judit Kőszegi, and Melinda Tóth. Refactorings to enable parallelization. In Jurriaan Hage and Jay McCarthy, editors, *Trends in Functional Programming*, pages 104–121, Cham, 2015. Springer International Publishing.

[BKT18] István Bozó, Mátyás Béla Kuti, and Melinda Tóth. Analysing and visualising callback modules of Erlang generic server behaviours. In *Proceedings of the 11th Joint Conference on Mathematics and Computer Science, CEUR Workshop Proceedings*, volume 2046, pages 23–41, 2018.

[BST18] István Bozó, Bence János Szabó, and Melinda Tóth. Analysing the hierarchical structure of Erlang applications. In *Proceedings of the 11th Joint Conference on Mathematics and Computer Science, CEUR Workshop Proceedings*, volume 2046, pages 42–55, 2018.

[BT11] István Bozó and Melinda Tóth. Selecting Erlang test cases using impact analysis. In Simos BE, editor, *International Conference on Numerical Analysis and Applied Mathematics: ICNAAM 2011, AIP Conference Proceedings, 1389*, pages 802–805, Melville (NY), 2011. American Institute of Physics.

[BT16] István Bozó and Melinda Tóth. Analysing and visualising Erlang behaviours. In Theodore Simos and Charalambos Tsitoura, editors, *International Conference on Numerical Analysis and Applied Mathematics: 2015 ICNAAM, AIP Conference Proceedings, 1738*, Melville (NY), 2016. AIP Publishing. Art. No.: 240004.

[BTT+11a] István Bozó, Melinda Tóth, Máté Tejfel, Dániel Horpácsi, Róbert Kitlei, Judit Kőszegi, and Zoltán Horváth. Using impact analysis based knowledge for validating refactoring steps. In Militon Frentiu, Horia F. Pop, and Simona Motogna, editors, *International Conference on Knowledge Engineering, Principles and Techniques (KEPT 2011): Selected papers, 407 p.*, pages 325–336, Cluj-Napoca, 2011. Presa Universitara Clujeana.

[BTT+11b] István Bozó, Melinda Tóth, Máté Tejfel, Dániel Horpácsi, Róbert Kitlei, Judit Kőszegi, and Zoltán Horváth. Using impact analysis based knowledge for

validating refactoring steps. *STUDIA UNIVERSITATIS BABES-BOLYAI SERIES INFORMATICA*, 56(3):57–64, 2011.

[FBT17] Viktória Fördős, István Bozó, and Melinda Tóth. Towards change-driven testing. In Natalia Chechina and Scott Lystig Fritchie, editors, *Erlang 2017: Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang*, pages 64–65, New York, 2017. ACM Press.

[HBT14] István Bozó, Melinda Tóth and Zoltán Horváth. Reduction of regression tests for Erlang based on impact analysis. *STUDIA UNIVERSITATIS BABES-BOLYAI SERIES INFORMATICA*, 59(Special Issue 1):31–46, 2014.

[HBTE10] Zoltán Horváth, István Bozó, Melinda Tóth, and Attila Erdődi. Dependency graphs for parallelizing Erlang programs. In Hage Jurriaan, editor, *Preproceedings of the 22nd Symposium on Implementation and Application of Functional Languages: IFL 2010, 423 p.*, pages 180–186, Utrecht, 2010. Utrecht University.

[IM14] Bozó István and Tóth Melinda. Erlang folyamatok és a köztük lévő kapcsolatok elemzése. In Csörnyei Zoltán, editor, *Tízéves az ELTE Eötvös József Collegium Informatikai Műhelye: 2004-2014*, pages 79–92, Budapest, 2014. ELTE Eötvös József Collegium.

[KTB18] Tamás Kozsik, Melinda Tóth, and István Bozó. Free the conqueror! refactoring divide-and-conquer functions. *Future Generation Computer Systems*, 79:687 – 699, 2018.

[KTBH16] Tamás Kozsik, Melinda Tóth, István Bozó, and Zoltán Horváth. Static analysis for divide-and-conquer pattern discovery. *Computing and Informatics*, 35:764–791, 01 2016.

[LTB18] Dániel Lukács, Melinda Tóth, and István Bozó. Transforming Erlang finite state machines. In *Proceedings of the 11th Joint Conference on Mathematics and Computer Science, CEUR Workshop Proceedings*, volume 2046, pages 197–218, 2018.

[TB11] Melinda Tóth and István Bozó. Building dependency graph for slicing Erlang programs. *PERIODICA POLYTECHNICA-ELECTRICAL ENGINEERING*, 55(3-4):133–138, 2011.

[TB12a] Melinda Tóth and István Bozó. Detecting process relationships in Erlang programs. In Ralf Hinze, editor, *Draft Proceedings of the 24th Symposium on Implementation and Application of Functional Languages*, pages 494–508, Oxford, 2012.

[TB12b] Melinda Tóth and István Bozó. Static analysis of complex software systems implemented in Erlang. In Viktória Zsók, Zoltán Horváth, and Rinus Plasmeijer, editors, *Central European Functional Programming School: 4th Summer School, CEFP 2011, Budapest, Hungary, June 14-24, 2011, Revised Selected Papers*, pages 440–498. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[TB14] Melinda Tóth and István Bozó. Detecting and visualising process relationships in Erlang. *PROCEDIA COMPUTER SCIENCE*, 29:1524–1534, 2014.

[TBH+10b] Melinda Tóth, István Bozó, Zoltán Horváth, László Lövei, Máté Tejfel, and Tamás Kozsik. Impact analysis of Erlang programs using behaviour dependency graphs. In Zoltán Horváth, Rinus Plasmeijer, and Viktória Zsók, editors, *Central European Functional Programming School: Third Summer School, CEFP 2009, Budapest, Hungary, May 21-23, 2009 and Komárno, Slovakia, May 25-30, 2009, Revised Selected Lectures*, pages 372–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[TBH13] Melinda Tóth, István Bozó, and Zoltán Horváth. Reduction of regression tests for Erlang based on impact analysis. In Peter Achten, editor, *Draft Proceedings of the 25th Symposium on Implementation and Application of Functional Languages*, pages 1–7, 2013.

[TBHE11] Melinda Tóth, István Bozó, Zoltán Horváth, and Attila Erdődi. Static analysis and refactoring towards Erlang multicore programming. In Vivek Sarkar and Vasco T. Vasconcelos, editors, *Pre-Proceedings of the Fourth Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES'11*, pages 43–50, Saarbrücken, Germany, 2011.

[TBHT10] Melinda Tóth, István Bozó, Zoltán Horváth, and Máté Tejfel. First order flow analysis for Erlang. In *8th Joint Conference on Mathematics and Computer Science, Selected papers, ISBN 978-963-9056-38-1*, pages 403–416, Komárno, Slovakia, July 2010.

[TBK17] Melinda Tóth, István Bozó, and Tamás Kozsik. Pattern Candidate Discovery and Parallelization Techniques. In *IFL 2017: 29th Symposium on the Implementation and Application of Functional Programming Languages*, p. 1-26, New York, NY, USA, 2017. ACM Press.

[TTB+12] Máté Tejfel, Melinda Tóth, István Bozó, Dániel Horpácsi, and Zoltán Horváth. Improving quality of software analyser and transformer tools using specification based testing. *ANNALES UNIVERSITATIS SCIENTIARUM BUDAPESTINENSIS DE ROLANDO EOTVOS NOMINATAE SECTIO COMPUTATORICA*, 37:355–368, 2012.