Data flow and dependence analyses for functional languages – Static analysis of Erlang programs

Theses of the Doctoral Dissertation

MELINDA TÓTH

Doctoral School of Informatics, Eötvös Loránd University
Erzsébet Csuhaı 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

The main goal of my thesis was to define data flow and dependence relations for the functional programming language, Erlang, and to make possible the various forms of static analyses to build upon the data related information.

Static analysis is the technique to gather information about the source code without executing it, thus it is often called compile-time analysis. Several kinds of static program analysers exist. Tools may support program comprehension and provide functionality to help in debugging or detecting the impact of certain changes. Other tools provide source code metrics to measure the complexity, maintainability, or quality of the software. Besides the analyses, lots of tools provide meaning preserving source code transformations, refactorings. In my thesis I have focused on the data related static analysis techniques, and the usage of these analyses in parallel pattern recognition. The presented data flow analysis is useful in code comprehension task, but several further thorough semantic analyses can be built on the top of it.

RefactorErl is a well-know static source code analysis and transformation tool for Erlang. Therefore, my goal was to extend the framework of RefactorErl with data related analyses.

In my thesis, I have defined the data related static analyses formally, using syntax driven, semantic aware, compositional rules and relations. I have developed the algorithms according to these definitions in the RefactorErl framework. Thus make the defined analysis usable on real-life projects as well.
Chapter 2

Results

The main contributions of the thesis:

- **Thesis 1.** I have defined the first-order context aware data flow graph and data flow relation for Erlang programs based on the semantics of the language. Using this relations I have defined the message sending based data flow as well. I have provided the algorithms in RefactorErl according to the definitions.

- **Thesis 2.** I have defined the behaviour dependence graphs for Erlang programs and the dependence relation on the graph. I have provided the dependence calculation algorithm in RefactorErl.

- **Thesis 3.** I have provided the rules to discover parallel pattern candidates in sequential Erlang programs based on the relations defined in Thesis 1, Thesis 2 and other static analysis techniques. The corresponding algorithms can be used in RefactorErl to identify candidates that are amenable for parallelisation.

We have defined and implemented the algorithms related to the previous theses in RefactorErl. We have used specification based testing to test them \cite{TTB+12}. On the other hand, we have also built different applications using the data related analyses. These applications satisfied the requirements during testing and large-scale usage as well.

2.1 Data Flow Analysis

Data flow analysis is a technique for gathering information about how a program manipulates its data, and what are the possible sets of values calculated at various points in a program. The goals of my research were to
define the data flow relation in Erlang, and to use this information in various other static analyses. The data flow analysis has two phases. The first one is to build the Data Flow Graph (DFG), and the second one is to calculate the data flow reaching. The Data Flow Graph is a labelled, directed graph built from the expressions of the Erlang code. The DFG represents the direct data flow among the expressions. The data flow reaching is defined as a relation on the DFG and expresses both the direct and the indirect flow among the expressions. Depending on the context the analysis takes into account we can define zeroth-order, first-order or even higher-order data flow analyses.

The definition of the DFG is based on compositional graph building rules. Each rule assigns data flow edges to expressions based on the semantics of Erlang. The algorithm to build the DFG can be expressed in the incremental semantic analyser framework of RefactorErl. Thus, there is no need to rebuild the DFG when the source code is changed. I have defined both the zeroth-order and the first-order graphs according to the calling context of the functions.

The data flow reaching is defined as a relation on the DFG satisfying certain conditions. The reaching relation determines the indirect data flow between two arbitrary elements of the data flow graph. The first-order data flow reaching extends the zeroth-order relation with calling context awareness. The $n_1 \leftarrow n_2$ relation means that the value of $n_2$ is a copy of the value of $n_1$. In other words, the value of $n_1$ may flow to $n_2$.

Using the first-order data flow reaching I have defined data flow among concurrent send and receive expressions.

The defined data flow reaching relation was built into the query language of RefactorErl. Thus, developers can ask information about the possible values of certain variables, find suspicious code in the debugging process, etc.

2.1.1 Related publications

Main publications of this thesis: [TB12b, TBHT10].
Other publications related to this thesis: [LTB18, LT18, BST18, BKT18, BT16, IM14, TB14, TB12a, TBK17, BFH+15, BFH+14, KTBH16, KTB18, TB11, TBH+10b, HBT14, BT11, TBH13, BTT+11a, BTT+11b, TTB+12, FBT17, TBH10a, HBKT11, HBTE10, TBHE11].

The publications listed here have a total of 35 independent citations, of which there are 8 independent citations for the main publications of this thesis (based on MTMT).
2.2 Data Dependence Analysis

The data flow relation identifies a value copying between expressions, thus it also represents a kind of dependence between them: changing the value of \( n_1 \) has effect on \( n_2 \). However, the data flow is not enough when we want to express dependence among various expressions. For example, changing a value in a guard expression may have impact on the behaviour/evaluation of the function and the expressions of the function. Therefore, we need to extend the DFG with information about the dependence and behaviour dependence among the expressions and its related subexpressions as well, and define a dependence relation. I have defined the Behaviour Dependence Graph (BDG) for Erlang programs by extending the compositional data flow rules. The BDG contains the direct dependence information among expressions.

I have defined the \( n_1 \sim n_2 \) dependence relation on the BDG. This relation holds when the evaluation/behaviour of the expression \( n_2 \) depends on the data represented by \( n_1 \).

The dependence information can be used in change impact analysis to select the dependent expressions. An other form of the usage is to check the independence of expressions. The latter can be used in parallelisable component detection.

2.2.1 Related publications

Main publications of this thesis: [TBH+10b] [TB12b].

Other publications related to this thesis: [TBK17] [BFH+15] [BFH+14] [KT18] [KTBH16] [HBT14] [BT11] [TBH13] [BTT+11a] [BTT+11b] [HBTE10] [TBHE11] [TB11] [FBT17].

The publications listed here have a total of 31 independent citations, of which there are 10 independent citations for the main publications of this thesis (based on MTMT).

2.3 Pattern Discovery

It is common to refactor the source code to adopt it to the changed hardware resources. An example is the parallelisation of the source code to utilise multi-core resources. This process can be done manually or (semi)automatically with assistance of a transformation tool.

The parallelisation has two phases. The first step is to identify the code fragments that are amenable for parallelisation, and the second one is to perform the appropriate transformation. None of the above is trivial. By
considering the increasing size of the source codes it is almost impossible to find the candidates manually. Therefore, I have provided methodologies in my thesis to help the developers to identify the source code fragments to parallelise.

I have defined rules to identify source code fragments that are good candidates to introduce well-known algorithmic skeletons provided by the skel and skel_hlp Erlang libraries. These rules are using the defined data related relations, but also build on other static analyses such as control flow analysis or syntax driven traversing. Based on the defined rules the discovery algorithms can be defined in the RefactorErl static analyser framework.

I have divided the possible candidates into two groups. Candidates belonging to the first group are library calls and iterative expressions (e.g. list comprehensions) expressing the semantics of the algorithmic skeletons. The discovery rules for this group are mainly syntax driven with built in knowledge about the semantics.

Candidates in the second group are recursive functions expressing the same semantics as the skeleton. The rules specifying the conditions are expressing special requirements about the execution paths of the function, and expressing special flow of data, independence and dependence as well.

2.3.1 Related publications

Main publications of this thesis: [TBK17, BFH+15, BFH+14, KTBH16].

Other publications related to this thesis: [KTB18, TBH+10b, TB12b, TBHT10].

The publications listed here have a total of 28 independent citations, of which there are 14 independent citations for the main publications of this thesis (based on MTMT).
Chapter 3

Summary

The importance of the tools to support software development is increasing. The size of the software products makes manual scanning for certain information almost impossible, or time consuming. Therefore tools to support code comprehension, development, maintenance, debugging, or even automatic source code transformation are really desired. We can distinguish dynamic and static tools. The former analyses the software at runtime by monitoring, instrumenting the code. The latter analyses the source code itself without executing the program. In my thesis I have developed new static analyses methods for the Erlang programming language to support code comprehension and further static analyses.

I have defined the first order data flow graph for Erlang programs, and the data flow relation among the nodes of the graph, the first order data flow reaching. The reaching relation itself is able to identify the possible values of an expression at some point in the program. It also identifies the expressions where a certain value may flow. Based on the definitions I have introduced the data flow graph building and reaching calculation algorithms using the RefactorErl framework. The data flow graph building algorithm is incremental, therefore the changes of the source code can be handled without reanalysing the whole software. The data flow reaching algorithm is interprocedural and aware of function call context. Using data flow reaching I have defined the direct data flow among asynchronous message sending and receiving expressions.

I have defined a data dependence relation among Erlang expressions based on the behaviour dependence graph that is an extension of the data flow graph. The dependence relation defines whether two expressions from the Erlang source code depends on each other. The corresponding algorithms have been defined in the framework of RefactorErl, and have been used in further static analyses, namely in change impact analysis and pattern discovery.
I have defined the behaviour of parallelisable computations, such as the elementwise processing. The definitions use the studied data flow, data dependence relations and other static analysis methods, such as control flow graphs and execution paths. Using the definitions we can identify sequential code fragments that can be replaced with parallel equivalents. The pattern discovery algorithms have been defined in the RefactorErl framework.
List of related publications


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


Static analysis of Erlang programs


