Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Mining inline cache data to order inferred types in dynamic languages

Nevena Milojković [a,*], Clément Béra [b], Mohammad Ghafari [a], Oscar Nierstrasz [a]

[a] Software Composition Group, University of Bern, Switzerland
[b] RMOD-INRIA Lille Nord Europe, France

ARTICLE INFO

Article history:
Received 13 January 2017
Received in revised form 3 November 2017
Accepted 9 November 2017
Available online xxxx

Keywords:
Type inference
Dynamically-typed languages
Inline caches

ABSTRACT

The lack of static type information in dynamically-typed languages often poses obstacles for developers. Type inference algorithms can help, but inferring precise type information requires complex algorithms that are often slow.

A simple approach that considers only the locally used interface of variables can identify potential classes for variables, but popular interfaces can generate a large number of false positives. We propose an approach called *inline-cache type inference* (ICTI) to augment the precision of fast and simple type inference algorithms. ICTI uses type information available in the inline caches during multiple software runs, to provide a ranked list of possible classes that most likely represent a variable's type. We evaluate ICTI through a proof-of-concept that we implement in Pharo Smalltalk. The analysis of the top-$n+2$ inferred types (where n is the number of recorded run-time types for a variable) for 5486 variables from four different software systems shows that ICTI produces promising results for about 75% of the variables. For more than 90% of variables, the correct run-time type is present among first six inferred types. Our ordering shows a twofold improvement when compared with the unordered basic approach, *i.e.*, for a significant number of variables for which the basic approach offered ambiguous results, ICTI was able to promote the correct type to the top of the list.

1. Introduction

Static type information has shown to be of crucial importance to developers during software maintenance [1,2]. Inferring type information from source code in dynamically-typed languages has been extensively researched over the past decades [3–12]. While some approaches rely only on the available static information [3,4,13,8], others use dynamic execution to collect run-time type information and feed it back to the algorithm [14,5,10].

For static type analysis to be precise, it must closely track control and data flow. However, reliable results are usually achieved by analysing the whole program which is very expensive. Besides, modern software systems not only depend heavily on libraries, but are often part of a distributed system which may not be available for analysis. Simpler type inference analyses [8,4] statically track variable assignments and the set of messages sent[1] to a variable in order to determine which

* Corresponding author.
E-mail address: nevena@inf.unibe.ch (N. Milojković).
URL: http://scg.unibe.ch/staff/Milojkovic (N. Milojković).

[1] The terms "message" and "method" originate from Smalltalk, where one "sends a message" to an object, and the receiver then selects a "method" to respond to that message.

https://doi.org/10.1016/j.scico.2017.11.003
0167-6423/

classes either implement those methods, or inherit them from a superclass. Since they are neither control- nor data-flow sensitive, these approaches tend to be less precise, but very fast. Nevertheless, the problem with these simple approaches in dynamically-typed languages is that they provide a developer with the set of unordered classes that represent possible type for a variable. Unfortunately, this leaves the burden of looking for the correct type on the developer. To alleviate this issue, we have investigated a way of ordering the results of a simple type inference algorithm by statically analysing the frequency of class usages and class instantiations in the source code [15]. In that work, the focus was only on static data, which results in missing certain types when dynamic class loading or reflection are used [16].

Nowadays many virtual machines for dynamic languages include Just-in-Time compilers that use inline caches [17,18] to achieve high performance. Inline caches have been exploited for compiler optimisation purposes [14]. Besides the information about methods that were previously selected to respond to a message send, these caches also contain receiver type information for the message send, which could be easily exploited in order to improve current development tools. This run-time information about the type of the receiver has already been used to feed the type back to code, and in case of successful type checking at run time, to inline the message send, and execute code faster. However, to the best of our knowledge, it has still not been used to improve static type information for other message sends, for which the receiver type has not been collected from inline caches. We believe that this information collected during execution of any program written in the same language would add productively to the statically collected knowledge used for inferring a variable's type. As run-time information has been read from the virtual machine, no instrumentation is required.

We present an approach called *inline cache type inference* (ICTI) to exploit type information collected from inline caches during program runs from different systems written in the same language in order to improve static type inference. We employ a simple static analysis algorithm to infer types of variables. Type information collected from inline caches during execution of other programs written in the same language is then used to order the types of these variables. This means that the possible classes for a variable are ordered based on the class usage frequency during program runs.

We have implemented a proof-of-concept prototype for Pharo,[2] a dialect of Smalltalk, a highly reflective dynamically-typed language, which enables fast and easy implementation of analysis tools [19]. We have used this implementation to evaluate our claim that the frequency of class usage as the type of a receiver at run time can serve as a reliable proxy to statically identify the type of a variable. We have used a basic static analysis algorithm, *i.e.*, RoelTyper [8], to collect static type information based on the messages sent to variables and from assignments to them. We then augmented the results with the help of the inline cache information. The results show that the implemented heuristic is reasonably precise for more than 75% of the variables, and compared to the basic algorithm, ICTI more than doubled the number of correctly guessed types for a variable. We believe the improvement achieved by our heuristic can boost the performance of simple static type inference algorithms, regardless of their various applications in the field.

This article extends our previous work [20] as follows: (i) we present a motivating example for ICTI, (ii) we provide a thorough discussion of related work, (iii) we explain in detail how we gather type information from the runtime, (iv) we improve ICTI with a heuristic that collects type hints from method argument names, (v) we evaluate ICTI on 15% larger set of variables, (vi) we compare ICTI with the basic algorithm, and (vii) we discuss the results.

Structure of the paper. We start by giving an overview of the problem in section 2. We discuss the related work in the field in section 3. Section 4 explains the virtual machine used for run-time data collection. Next we define the used terminology and the implemented heuristic in section 5. Section 6 shows results of the evaluation of the prototype. We then describe potential threats to validity in section 7 before concluding in section 8.

2. Motivation

To explain the contribution of this paper, let us consider the example in Listing 1. The example[3] is written in Pharo Smalltalk.

Lines 1–4 define a class named MethodBrowser used to browse methods of a given class, and lines 6–18 define a method named initializePresenter to initialise variables of the MethodBrowser class. Suppose that a developer needs to know the type of the block argument item in the last line of the method initializePresenter, either to understand which method with selector *i.e.*, the name of the method, methodClass will be invoked, or to understand the behaviour of the method. The only available information is that this variable needs to understand messages with selectors methodClass and selector. Polymorphic selectors are frequently used in Smalltalk [21] which means that a large number of methods implement these selectors. In the Pharo image[4] we used for our experiments, there are 20 methods with selector methodClass and 67 methods with selector selector. A simple analysis, such as that offered by *RoelTyper*, which is used as the basic approach in the paper, uses the information about messages sent to the variable, and presents the developer with fourteen classes whose instances understand both messages. RoelTyper also uses the information about assignments to the variables, which is missing in this case. Hence, these fourteen classes are presented to the developer without any particular order. This

[2] http://www.pharo.org Pharo is a Smalltalk IDE, including a large library that contains the core of the Smalltalk system itself.

[3] This code snippet is actual code from the SPEC system: http://www.smalltalkhub.com/#!/~Pharo/Pharo60/packages/Spec-Examples.

[4] Pharo 6.0 version 60324.

Please cite this article in press as: N. Milojković et al., Mining inline cache data to order inferred types in dynamic languages, Sci. Comput. Program. (2017), https://doi.org/10.1016/j.scico.2017.11.003