DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Combatting the Precision Loss of Partial Contexts in Abstract Interpretation

Felix Sebastian Krayer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Combatting the Precision Loss of Partial Contexts in Abstract Interpretation

Bekämpfung des Präzisionsverlust durch partielle Kontexte in Abstrakter Interpretation

Author:	Felix Sebastian Krayer
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	15th of February 2023

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15th of February 2023 Felix Sebastian Krayer

Acknowledgments

Abstract

Contents

1 Introduction

"[GOBLINT is]a static analyzer for multithreaded C programs, specializing in finding concurrency bugs." (Copy paste from https://goblint.in.tum.de/) Analyzing interprocedural programs poses a certain difficulty. A function can be called in many different places in a program, where the (possible) values of not only formal arguments but also the global variables can be very different. Even a call in the same line of code can happen in very different contexts of such parameters and globals. Therefore, one would like to analyze the function multiple times, each time with a different starting state of formal arguments and global variables or 'contexts'. However, due to the high or potentially infinite amount of different calling contexts, this can be very costly.
An approach to reduce this cost is to only analyze the function in question for some few joint contexts, where multiple possible starting states are joined into a single context representing all of them. This however comes with the price of precision, especially when tracking values or relations between variables: The joint context needs to represent multiple different states. This means that the resulting context needs to represent everything that the states describe about each variable, leading to a less precise state. For example (assuming an interval analysis), if it is known in some context that x = 5 and in another that x = 3 before the call, then the starting state of the context needs to map x to the Interval [3, 5].
Now even if a variable is not changed during the call and therefore also its state in the bigger state has not changed, the precision loss is still propagated to the state after the call. This is because when combining caller and callee state, it is not known whether this particular variable was updated or not. For unreachable variables by the callee the caller state can be kept, however this does not apply to globals and variables reachable by the callee. In the above example, assuming x is a global or reachable variable, the state of x after the call will be the interval [3, 5], when x has not been altered in the call.
To reduce this loss of precision, it would be helpful to know which variables have been altered by the call. Then, when combining the states, only the states of variables which have been altered need to be updated with the state returned after the call, while the states of other variables can be kept from the caller state before the call.
To gain the information of which variables have been altered, we will present a new analysis keeping track of this in chapter 3.

Structure: First we will introduce the basics of static analysis. This will go by introducing constraint systems and how these are used to gain information about the program statically. It will be accompanied by an example of a value-of-variables analysis acting on a toy language we will use for examples in this thesis. This will be extended to an interprocedural approach where partial context sensitivity will be introduced. Here the source of the precision loss will also be pointed out. We then will propose an approach to combat this precision loss. The approach will first be introduced theoretically, after which we also present the challenges and results of implementing it in the Goblint analyzer. To give an evaluation to the proposed approach, a benchmark of the implementation will be performed and inspected. Our conclusions are presented in the last chapter.

2 Background

2.1 Related Work

2.2 Static Analysis

Static analysis is defined by Rival [RY20] as "[...]an automatic technique that approximates in a conservative manner semantic properties of programs before their execution". This means that the program is analyzed just by the given source code without execution. The goal is to prove certain properties about the program in a "sound" manner i.e. any property that is proven to hold actually does hold. However, from failing to prove a property one cannot conclude that the given property does not hold.
A useful tool within static analysis is a values-of-variables analysis that maps a set of unknowns X to abstractions of their possible values at any given point within the program. This allows for example to find guards (e.g. if-statements) which for any program execution will definitely hold or not hold, identifying dead code.
In the scope of this thesis we will focus on abstracting integer values by sets of integers. Thereby our Domain of abstract values will be $\mathbb{D} = 2^{\mathbb{N}}$

(* TODO *)

2.3 Constraint systems

To find a mapping of unknowns to abstractions of possible values the following approach is used: First a control flow graph ("CFG") is created, where program points are connected by edges of actions (u, A, v). This example denotes an action A (e.g. an assignment of $A = (x = 7;)$) where u us the point immediately before this action and v the one immediately after. Using this CFG, a system of constraints for each program point is created. We can then compute a fix point of the system of constraints to get a solution there of (since everything is monotonic).

3 Main Contributions

3.1 Taint analysis

3.1.1 Formal description

$\mathbb{D} = 2^{\{\mathsf{lval}\}}$
$[u] \in \mathbb{D}$
Edge $e = (u, A, u')$ introduces the constraint $[u'] \sqsupseteq [\![A]\!]^\#(\mathsf{get}\ [u])$

$$[\![x = y]\!]^\# \ \mathsf{lv} = \mathsf{lv} \cup \{x\}$$
$$[\![{}^*x = y]\!]^\# \ \mathsf{lv} = \mathsf{lv} \cup \mathsf{MayPointTo}(x)$$

$$\mathsf{enter}^\# \ \mathsf{lv} = \emptyset$$
$$\mathsf{combine}^\# \ \mathsf{lv}_{\mathsf{cr}} \ \mathsf{lv}_{\mathsf{ce}} = \mathsf{lv}_{\mathsf{cr}} \cup \mathsf{lv}_{\mathsf{ce}}$$

3.1.2 Implementation

3.2 Benefiting other Analyses

In this section we will use the new taint analysis to improve a context insensitive analysis. For this let's choose an analysis that maps Lvalues to Rvalues. When combining the contexts of the caller before the call with the one returned by the callee there a few aspects to keep in mind:

- All mappings of Lvalues, which are not tracked in the caller (i.e. map to top), but have a concrete value within the callee need to be added to the combined context. This is for Lvalues which are newly initialized inside the caller.
- (All mappings which are not in the callee context but have been in the caller context need to be removed. This can happen in multithreaded programs, if in the caller a mutex was held, that then was unlocked by the callee, deleting the information protected by the mutex)

- for all other Lvalues present in both contexts, the Rvalues mapped to by Lvalues not in the tainted set can be kept. We are sure that these variables are unchanged, even if they have a less precise record in the callee's context. For Lvalues present in the tainted set, it is necessary to take the Rvalue from the callee context, as the old Rvalue mapped to by the caller is incorrect.

Formal:

$$\begin{aligned}
\mathsf{combine}^{\#} \eta_{\mathsf{cr}}\ \eta_{\mathsf{ce}} = & \text{let } \eta'_{\mathsf{cr}} = \eta_{\mathsf{cr}} \setminus \mathsf{lv}_{\mathsf{ce}} \text{ in} && (3.1)\\
& \text{let } \eta'_{\mathsf{ce}} = \eta_{\mathsf{ce}} \cap \mathsf{lv}_{\mathsf{ce}} \text{ in} && (3.2)\\
& \eta'_{\mathsf{cr}} \cup \eta'_{\mathsf{ce}} && (3.3)
\end{aligned}$$

4 Evaluation

4.1 Testing

4.2 Benchmarking

5 Conclusion

List of Figures

List of Tables

Bibliography

[RY20] X. Rival and K. Yi. *Introduction to static analysis: an abstract interpretation perspective*. Mit Press, 2020.