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

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Background **3**
2.1 Related Work 3
2.2 Static Analysis 3
2.2.1 Flow sensitive analysis 4
2.2.2 Constraint systems 4

3 Main Contributions **6**
3.1 Taint analysis 6
3.1.1 Formal description 6
3.1.2 Implementation 6
3.2 Benefiting other Analyses 6

4 Evaluation **8**
4.1 Testing 8
4.2 Benchmarking 8

5 Conclusion **9**

List of Figures **10**

List of Tables **11**

Bibliography **12**

1 Introduction

```
1  int function (int a) {
2    //a = [0, 12]; y = [1, 2]
3    a = a * 2;
4    return a; //x_f = [0, 24]; y = [1, 2]
5  }
6
7  int y; //global
8
9  int main() {
10   int x; //local
11   x = 0;
12
13   y = 1;
14   x = function(0); // a = x = [0, 0]; y = [1, 1]
15   // x = [0, 24], y = [1, 1]
16
17   //...
18
19   y = 2;
20   x = function(12); // a = x = [12, 12]; y = [2, 2]
21   // x = [0, 24], y = [2, 2]
22 }
```

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 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 imple-

mentation 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.
In order to prove properties, e.g. finding that a program does not contain races or identifying dead code, we need to gain information about the program. This is done by performing various kinds of analyses. We will focus on flow sensitive analyses from now on i.e. analyses which find properties of the program dependent on the location within it. The semantic of these will be introduced in the following chapters.

```
1 int main() {
2   int x;
3   x = 0;
4   if (x == 0)
5     x = x + 1;
6 }
```

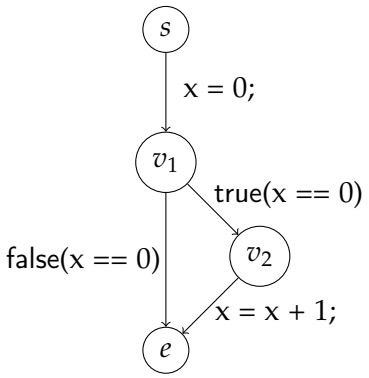

Figure 2.1: Example program (left) and corresponding CFG (right)

2.2.1 Flow sensitive analysis

As noted above flow sensitive analyses find properties of the program dependent on the point within the program. Expressed differently this means a flow sensitive analysis will find an overapproximation of states the program may be in for any given point within the program or "program point". This state can describe many things dependent on the analysis performed.
First let us define what a program point is: Imagine a control flow graph (CFG), where nodes represent points between instructions within the program. Edges are labeled with instructions or checks and describe the transitions between these points (see example 2.1). Then any node on this CFG would be what we call a program point.
Concretely let N be the set of all program points. Furthermore, let $\mathbb{D}$ be a Domain containing abstract states describing concrete states of the program. This means that some $d \in \mathbb{D}$ can describe many states the program can be in.
Then an analysis is expected to find a mapping $\eta : N \to \mathbb{D}$ which maps program points to abstract states describing that location within the program i.e. for $[n] \in N$, $\eta\ [n]$ should be an abstract state describing all possible states (and possibly more) the program can be in at program point $[n]$.
As an example we will introduce a values-of-variables analysis for integers. This analysis finds a mapping from a set of variables X to abstractions of their possible values at any given program point. In the scope of this thesis we will focus on abstracting integer values by sets of integers. Thereby the goal of our values-of-variables analysis is to find a mapping $X \to 2^{\mathbb{N}}$ for each program point.
Combining this with the semantic of flow sensitive analysis from before, we get that the Domain $\mathbb{D}_v$ for the values-of-variables analysis should be $\mathbb{D}_v = X \to 2^{\mathbb{N}}$. Finally, the resulting $\eta_v : N \to \mathbb{D}_v$ for this analysis describes a mapping $\eta_v\ [n]$ for some program point $[n] \in N$, where $\eta_v\ [n]\ x$ is a set containing all values $x \in X$ may possibly hold at $[n]$. From this we can conclude that x cannot hold any value outside $\eta_v\ [n]\ x$ at program point $[n]$.

2.2.2 Constraint systems

We now formulate a way in which we can describe an analysis in the form of constraints. For this we need a partial ordering $\sqsupseteq$ on the domain $\mathbb{D}$.
Then we create a system of constraints which can be solved for a solution. Consider the edges (u, A, v) of the CFG, where each edge denotes a transition from program point $[u]$ to program point $[v]$ via the instruction A. Now let each of these edges give raise to a constraint $\eta\ [v] \sqsupseteq [\![A]\!]^{\#}\ (\eta\ [u])$, where $[\![A]\!]^{\#}$ denotes the abstract effect of the instruction or check A defining our analysis. In addition we need a start starte. this

is usually the maximal element according to our ordering, giving raise to the start constraint $\eta\ [s] \sqsupseteq \mathsf{max}(\sqsupseteq, \mathbb{D})$.

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:

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

4 Evaluation

4.1 Testing

4.2 Benchmarking

5 Conclusion

Abbreviations

- CFG

List of Figures

2.1 Example program (left) and corresponding CFG (right) 3

List of Tables

Bibliography

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