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

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 implementation will be performed and inspected. Our conclusions are presented in the last chapter.

2 Background

2.1 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.

2.1.1 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

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

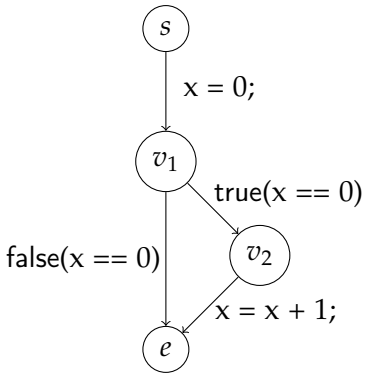

Figure 2.1: Example program (left) and corresponding CFG (right)

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 $\mathbb{D}_\mathsf{v}$ for the values-of-variables analysis should be $\mathbb{D}_\mathsf{v} = X \to 2^{\mathbb{N}}$. Finally, the resulting $\eta_\mathsf{v} : N \to \mathbb{D}_\mathsf{v}$ for this analysis describes a mapping $\eta_\mathsf{v}\ [n]$ for some program point $[n] \in N$, where $\eta_\mathsf{v}\ [n]\ x$ 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 $\eta_\mathsf{v}\ [n]\ x$ at program point $[n]$.

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

For our example values-of-variables analysis this means

2.1.3 Interprocedural analysis

2.1.4 Partial context sensitivity

2.1.5 Precision loss

3 Main Contributions

3.1 Formal description

3.1.1 Taint analysis

In this section we will propose our approach to reduce the aforementioned precision loss 2. The basic idea is to track for each procedure which variables have been written or have possibly been altered in some other way. This information is then used in the values-of-variables analysis when combining the abstract state from the caller with the abstract return state given by the callee at the end of the procedure.
In the following we will call a variable that has been written or altered in the current function context "tainted". Therefore, we introduce a new taint analysis tracking which variables have been tainted within the context of the current function.
First we formalize our taint analysis in the syntax for flow sensitive analyses we have built in the previous chapter 2:
$\mathbb{D}_\mathsf{t} = 2^X$ with $\sqsupseteq = \supseteq$
$\eta_\mathsf{t} : N \to \mathbb{D}_\mathsf{t}$
Each edge $e = (u, A, v)$ introduces the constraint $\eta_\mathsf{t}\ [v] \sqsupseteq [\![A]\!]^\#(\eta_\mathsf{t}\ [u])$

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

$$\mathsf{enter}^\#\ T = \emptyset$$
$$\mathsf{combine}^\#\ T_\mathsf{cr}\ T_\mathsf{ce} = T_\mathsf{cr} \cup T_\mathsf{ce}$$

3.1.2 Improving the values-of-variables analysis

$$\mathsf{combine}^\#\eta_\mathsf{cr}\ \eta_\mathsf{ce} = \text{let } \eta'_\mathsf{cr} = \eta_\mathsf{cr} \setminus T_\mathsf{ce} \text{ in} \tag{3.1}$$
$$\text{let } \eta'_\mathsf{ce} = \eta_\mathsf{ce} \cap T_\mathsf{ce} \text{ in} \tag{3.2}$$
$$\eta'_\mathsf{cr} \cup \eta'_\mathsf{ce} \tag{3.3}$$

3.2 Implementation

4 Evaluation

4.1 Testing

4.2 Benchmarking

5 Conclusion

Abbreviations

- CFG

List of Figures

List of Tables

Bibliography

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