A Posteriori Soundness for Non-deterministic Abstract Interpretations

Matthew Might\textsuperscript{1} and Panagiotis Manolios\textsuperscript{2}

\textsuperscript{1} University of Utah, Salt Lake City, Utah, USA
might@cs.utah.edu
\textsuperscript{2} Northeastern University, Boston, Massachusetts, USA
pete@ccs.neu.edu

Abstract. An abstract interpretation’s resource-allocation policy (\textit{e.g.}, one heap summary node per allocation site) largely determines both its speed and precision. Historically, context has driven allocation policies, and as a result, these policies are said to determine the “context-sensitivity” of the analysis. This work gives analysis designers newfound freedom to manipulate speed and precision by severing the link between allocation policy and context-sensitivity: abstract allocation policies may be unhinged not only from context, but also from even a predefined correspondence with a concrete allocation policy. We do so by proving that abstract allocation policies can be made non-deterministic without sacrificing correctness; this non-determinism permits precision-guided allocation policies previously assumed to be unsafe. To prove correctness, we introduce the notion of \textit{a posteriori} soundness for an analysis. A proof of \textit{a posteriori} soundness differs from a standard proof of soundness in that the abstraction maps used in an \textit{a posteriori} proof cannot be constructed until \textit{after} an analysis has been run. Delaying construction allows them to be built so as to justify the decisions made by non-determinism. The crux of the \textit{a posteriori} soundness theorem is to demonstrate that a justifying abstraction map can \textit{always} be constructed.

1 Introduction

When engineering a static analysis, better speed and higher precision are principal goals. In abstract interpretation, speed and precision are a function of the abstract allocation policy. By \textit{abstract allocation policy}, we mean the procedure by which an abstract interpretation chooses a resource from a pool of abstract resources during the transition from one abstract state to another. To ground this discussion with some specifics, examples of an abstract resource include abstract environment bindings (for environment analyses \cite{17,18,19,21}), abstract heap addresses (for alias and shape analyses \cite{2,4,22}), abstract contours (for flow analyses \cite{11,15,16,19,23,24,25}), and abstract time-stamps (for frame-string analyses \cite{11,17,18,20}).

\textsuperscript{*} This research was funded in part by NASA Cooperative Agreement NNX08AE37A and NSF grants CCF-0429924, IIS-0417413, and CCF-0438871.
The abstract allocation policy determines how the abstract state-space partitions the concrete state-space; likewise, a given partitioning uniquely determines an abstract allocation strategy. (More directly, this policy determines the relationship between the concrete and abstract instances of objects like stores, heaps and environments.) A fast, precise analysis needs an allocation policy which summarizes concrete resources that behave alike to the same abstract resource, but which summarizes concrete resources that behave differently to separate abstract resources.

The original motivation for this work came from frustration with context-sensitive policies: for the best result, one must choose the context-sensitive policy whose heuristic is geared toward the behavior of the program under consideration. We wanted to know whether abstract allocation policies could be made sensitive to precision rather than context—whether it is sound to make allocations based purely on precision, or not. Thus, this work begins by asking a general question: what are the fundamental, necessary constraints which all abstract allocation policies must obey. Specifically, we want to know whether an abstract allocation policy must directly simulate the concrete allocation policy in order to prove soundness.

Our pursuit of the constraints on abstract allocation policies ends with an unanticipated result: there are no such constraints. To demonstrate completeness, we prove that even the allocation policy which is fully non-deterministic is safe. Central to this result is the criterion of and a proof technique for “a posteriori soundness.”

1.1 A Priori Soundness and Context-Sensitivity

Frequently, abstract resources are selected as a function of the context of the current state, e.g., the current program counter [3], the last $k$ call sites [24], the let-polymorphism of the current function [25], the Cartesian product of argument types [1]. Context has been popular in designing allocation policies because context serves as a reasonably good heuristic for data usage: data structures allocated in the same context (e.g., the same call site, the same stack frame) tend to have similar usage patterns. Context-sensitive policies bleed precision and speed to the extent that they tend to split like resources across several abstract resources, e.g., 1CFA [24], or tend to associate unlike resources as a single abstract resource, e.g., CPA [1].

In an attempt at better performance, one might ask whether allocation policies can be hybridized and proven sound, so that the strengths of the two policies can be combined. For simplistic hybrids, such as the natural “Cartesian product” of two policies, the answer is yes; however, such a hybridization also combines their weaknesses—the splitting tendencies of the two policies multiply: precision goes up, but so does analysis time. If an analysis designer were to compensate for this splitting by making the allocation policy adaptive, then proving soundness is suddenly murkier, and the answer seems to be no. By adaptive, we mean that the allocation policy is allowed to directly consider the ramifications upon the precision or speed of the analysis when selecting an abstract resource to allocate.