A Modular Monadic Framework for Abstract Interpretation (Draft)

David Darais

August 4, 2014

Abstract

The design and implementation of static analyses is a difficult process. Verifying the correctness of an implementation is often the most painful step. We present Monadic AAM–an extension of the Abstracting Abstract Machines methodology introduced by Van Horn and Might [**?**]–which captures a large class of both automatically derivable and correct by construction abstract interpreters. Monadic AAM is part methodology and part toolkit. In the methodology, semantics are designed in a monadic extension of traditional AAM. Once designed, a large class of known analyses can be automatically recovered using our language agnostic toolkit. Both the computational and correctness properties of our framework are realized through a restricted class of monad transformers. Our framework enjoys the benefits of being highly compositional and placing a minimal burden of proof on the analysis designer.

1 Introduction **2**

2 Background **2**

3 AAM By Example **2**
3.1 CPS . . . 2
3.2 Concrete Interpreter . . . 3
3.3 State Space Abstraction . . . 4
3.3.1 Cutting Recursion . . . 5
3.3.2 Address and Time Abstraction . . . 5
3.3.3 Introducing Nondeterminism . . . 6
3.3.4 Delta Abstraction . . . 6
3.4 Abstract Semantics . . . 7
3.5 Recovering the Concrete Interpreter . . . 9
3.6 Recovering 0CFA . . . 10
3.7 Recovering kCFA . . . 10
3.8 Optimizations . . . 11

3.8.1 Heap Widening 11
3.8.2 Abstract Garbage Collection 12
3.8.3 Composing Heap Widening and Abstract GC 12

4 Monadic AAM **13**
4.1 AAM in Monadic Style 13
4.1.1 Nondeterminism 13
4.1.2 State 14
4.2 Generalizing the Monad 17
4.3 Recovering a Concrete Interpreter 19
4.4 Recovering kCFA 20
4.5 Optimizations 20
4.5.1 Heap Widening 20
4.5.2 Abstract Garbage Collection 22

5 Correctness **22**
5.1 Abstract Semantics 22
5.2 0CFA 22
5.3 kCFA 22
5.4 Widening 22
5.5 GarbageCollection 22
5.6 LatticeOfAnalyses 22

6 Conclusion **22**