Head-strictness is not a monotonic abstract property

Samuel Kamin

Computer Science Dept.
University of Illinois at Urbana-Champaign
Urbana, IL 61801

October 23, 1992

Abstract

A property $\mathcal{P}$ of a language is said to be *definable by abstract interpretation* if there is a monotonic map **abs** from the domain of standard semantics to an abstract domain $\mathcal{A}$ of finite height, and a partition of the abstract domain into two parts $\mathcal{A}_{\mathcal{P}}$ and $\mathcal{A}_{\text{non-}\mathcal{P}}$, such that any value has property $\mathcal{P}$ if and only if **abs** maps it to an element of $\mathcal{A}_{\mathcal{P}}$. *Head-strictness* is a property of functions over lists which asserts, roughly speaking, that whenever the function looks at some prefix of a list, it looks every element in that prefix. We prove that head-strictness is not definable by abstract interpretation.

Keywords: strictness analysis, abstract interpretation

1 Introduction

Given a programming language $\mathcal{L}$ with standard semantics $[\![\cdot]\!]_{\mathcal{D}} : \mathcal{L} \to \mathcal{D}$, an *abstract interpretation* [2, 6] of $\mathcal{L}$ is a semantics $[\![\cdot]\!]_{\mathcal{A}} : \mathcal{L} \to \mathcal{A}$, together with a function **abs** $: \mathcal{D} \to \mathcal{A}$. $\mathcal{A}$ represents some property of programs in $\mathcal{L}$, $[\![\cdot]\!]_A$ is the static computation of that property, and **abs** is the definition of that property with respect to semantic values. These various functions are related by a "safety" condition asserting that $[\![\cdot]\!]_A$ approximates **abs** $\circ$ $[\![\cdot]\!]_D$; in general, only an approximation can be obtained, due to computability considerations. Despite this, the property may be considered to be subject to analysis by abstract interpretation. Strictness of functions (the property that $f\bot = \bot$) is an example of such a property [4, 5, 7].

Under what circumstances, then, can we assert that a property is *not* analyzable by abstract interpretation? If there are no constraints on $\mathcal{A}$ or **abs**, we can never say it. However, two constraints are commonly placed on them:

1. $\mathcal{A}$ is a domain of finite height. This ensures that $[\![\cdot]\!]_A$ can be computed even if it involves fixedpoints.

2. **abs** is monotonic. This permits properties of fixedpoints in $\mathcal{D}$ to be inferred from the corresponding fixedpoints in $\mathcal{A}$.

Thus, we will say:

Definition 1 *A property $\mathcal{P}$ of $\mathcal{L}$ is* definable by abstract interpretation *if there is an abstract interpretation as above, where $\mathcal{A}$ is of finite height and* **abs** *is monotonic, and a partition of $\mathcal{A}$ into $\mathcal{A}_{\mathcal{P}} \cup \mathcal{A}_{non\text{-}\mathcal{P}}$, such that for any program ℓ, ℓ has property $\mathcal{P}$ if and only if* $\mathbf{abs}([\![\ell]\!]_{\mathcal{D}}) \in \mathcal{A}_{\mathcal{P}}$.

We propose this as the most general possible reading for "definable by abstract interpretation." Note that it makes no substantive reference to $[\![\cdot]\!]_{\mathcal{A}}$: the ability to actually compute $\mathcal{P}$ is not at issue. We are simply saying that, at the minimum, we must be able to define $\mathcal{P}$ by a monotonic map and "read off" the answer from $\mathcal{A}$.

Given this definition, non-definability of property $\mathcal{P}$ can be proven by producing an infinite sequence of elements of $\mathcal{D}$ that alternate between $\mathcal{P}$ and non-$\mathcal{P}$.

Head-strictness [3, 11] is a property of functions of lists that has for some time been thought not to be definable by abstract interpretation. What it says, intuitively, is that if the function examines any prefix of its argument, it evaluates all the elements of that prefix. Specifically:

Definition 2 *Function f is* head-strict, *or* $\mathcal{H}$-strict, *if f is strict and $f \circ \mathcal{H} = f$, where $\mathcal{H}$ is defined by:*

$$\mathcal{H} : \begin{array}{lcl} \bot & \mapsto & \bot \\ nil & \mapsto & nil \\ \bot.y & \mapsto & \bot \\ x.y & \mapsto & x.\mathcal{H}(y), \ \ \textit{if}\ x \neq \bot \end{array}$$

Our main theorem is:

Theorem 1 *Head-strictness is not definable by abstract interpretation.*

By an easy extension of the proof of theorem 1, we can obtain a proof of undefinability of a useful variant of head-strictness:

Definition 3 *f is* head-strict in a head-strict context *if f is strict and $\mathcal{H} \circ f \circ \mathcal{H} = \mathcal{H} \circ f$.*

Theorem 2 *Head-strictness in a head-strict context is not definable by abstract interpretation.*

Section 2 contains the proofs of theorems 1 and 2, section 3 explains why our definition of head-strictness is slightly different from that in [11], and section 4 discusses "undefinability by BHA-style analysis" [1, 4].

2 Head-strictness is not definable

$\mathcal{H}$ is an example of a *projection*, that is, an idempotent function below the identity in the function space ordering. In [11], strictness properties of functions on lists are formulated using projections; head-strictness and head-strictness in a head-strict context are examples.

Definition 4 *[11] Given projections α and β and function f, we say f* has property $\alpha \Rightarrow \beta$, *or write $f : \alpha \Rightarrow \beta$, if f is strict and $\alpha \circ f = \alpha \circ f \circ \beta$. (By abuse of notation, we also write $\alpha \Rightarrow \beta$ to denote $\{f \mid f : \alpha \Rightarrow \beta\}$.)*

Thus, head-strictness is property $\mathcal{I} \Rightarrow \mathcal{H}$, where $\mathcal{I}$ is the identity function, and head-strictness in a head-strict context is property $\mathcal{H} \Rightarrow \mathcal{H}$.

Many projections used in strictness analysis are *smash projections* [4], which means they map every element either to $\perp$ or to itself. Such projections lead to properties having a simple structure:

Fact 1 *If α and β are smash projections, then $\alpha \Rightarrow \beta$ is downward closed.*

Proof Our assumptions are that f is strict, $\alpha \circ f = \alpha \circ f \circ \beta$, and $g \sqsubseteq f$. Obviously, g is strict. Then, for any x, there are two possibilities:

- $\beta x = \perp$. Then $g \sqsubseteq f$ implies $\alpha(gx) \sqsubseteq \alpha(fx) = \alpha(f(\beta x)) = \alpha(f\perp) = \perp$, so $\alpha(gx) = \perp$. But $\alpha(g(\beta x)) \sqsubseteq \alpha(gx)$, so $\alpha(g(\beta x)) = \perp$ also.
- $\beta x = x$. In this case, $\alpha(g(\beta x)) = \alpha(gx)$ is immediate.

The simple structure of $\alpha \Rightarrow \beta$ in this case allows it to be captured by a monotonic function. The peculiar feature of $\mathcal{H}$ is that it is not a smash projection. In fact, if we define a value x to be *α-strict* when $\alpha(x) = x$, we can construct an infinite sequence of lists that are alternately $\mathcal{H}$-strict and non-$\mathcal{H}$-strict:

$$1.\perp \;\sqsubseteq\; 1.\perp.\perp \;\sqsubseteq\; 1.1.\perp \;\sqsubseteq\; 1.1.\perp.\perp \;\sqsubseteq\; \ldots$$

$\mathcal{H}$-strict not $\mathcal{H}$-strict where 1 is some value of $\mathcal{D}$ other than $\perp$. This observation can be parlayed into a proof of theorem 1:

Proof of theorem 1 As stated earlier, it suffices to present an increasing sequence of functions $< f_i >_{i \geq 0}$, such that for all i, f_i is $\mathcal{H}$-strict *iff* f_{i+1} is not $\mathcal{H}$-strict. The result follows from the observation that these functions are computable and therefore contained in $\mathcal{D}$.

Notation $x^i.y$ is the list $\underbrace{x.x.\ldots.x}_{i\ times}.y$

Continuing to assume 1 is an element of $\mathcal{D}$ other than $\perp$, define the function g from lists to lists:

$$\begin{array}{llll} g: & \perp & \mapsto & \perp \\ & x & \mapsto & 1.\perp, \ \ x \neq \perp \end{array}$$

Now define the sequence $< f_i >_i$ by, for all $i \geq 0$:

$$\begin{array}{rlll} f_{2i}: & \perp & \mapsto & \perp \\ & \perp.y & \mapsto & \perp \\ & x.y & \mapsto & 1^i.\perp, \ \ x \neq \perp \\ f_{2i+1}: & \perp & \mapsto & \perp \\ & \perp.y & \mapsto & \perp \\ & x.y & \mapsto & 1^i.g(y), \ \ x \neq \perp \end{array}$$

In other words, $f_{2i}(x.y)$ inspects x and, if the inspection ever terminates, returns the list $1^i.\perp$. $f_{2i+1}(x.y)$ inspects x and return either $1^i.\perp$ or $1^{i+1}.\perp$, depending upon the result of inspecting y; in programming terms, it returns $1^i.t$, where t is a thunk which, when evaluated, inspects y and, if that inspection terminates, returns $1.\perp$. Clearly, all these functions are computable.

That $< f_i >_i$ forms a non-decreasing sequence is obvious. We need only:

1. f_{2i} is $\mathcal{H}$-strict. Consider the cases:

$$\begin{array}{lll} f_{2i}(\perp) & = & f_{2i}(\mathcal{H}(\perp)) \\ f_{2i}(\perp.y) & = & \perp \ = \ f_{2i}(\perp) \ = \ f_{2i}(\mathcal{H}(\perp.y)) \\ f_{2i}(x.y) & = & 1^i.\perp \ = \ f_{2i}(x.\mathcal{H}(y)) \quad \text{(since } f_{2i}(x.y) \text{ does not depend on } y\text{)} \\ & = & f_{2i}(\mathcal{H}(x.y)) \end{array}$$

2. f_{2i+1} is not $\mathcal{H}$-strict. Consider the list $1.\perp.\perp$:

$$\begin{array}{lll} f_{2i+1}(1.\perp.\perp) & = & 1^{i+1}.\perp \\ f_{2i+1}(\mathcal{H}(1.\perp.\perp)) & = & f_{2i+1}(1.\perp) \ = \ 1^i.\perp \end{array}$$

Thus, $f_{2i+1} \neq f_{2i+1} \circ \mathcal{H}$.

From this proof, we easily obtain:

Proof of theorem 2 Recall that $f : \mathcal{H} \Rightarrow \mathcal{H}$ if f is strict and $\mathcal{H} \circ f \circ \mathcal{H} = \mathcal{H} \circ f$. Then simply note that $\mathcal{H} \circ f_i = f_i$ for all i in the proof of theorem 1. It follows that for all i, f_{2i} is head-strict in a head-strict context and f_{2i+1} is not.

3 About our definition of $\alpha \Rightarrow \beta$

In [11], the definition of $\alpha \Rightarrow \beta$ does not require strictness:

Definition 5 *$f : \alpha \Rightarrow_{wh} \beta$ if $\alpha \circ f = \alpha \circ f \circ \beta$.*

By lifting D and defining new projections, the strictness requirement can be expressed. (For example, $\mathcal{I} \Rightarrow \mathcal{H}$ is equivalent to $STR \Rightarrow_{wh} \mathcal{H}'$ in [11].) We added the strictness requirement just to avoid the lifting operation, but this leaves open the question of definability of $\mathcal{I} \Rightarrow_{wh} \mathcal{H}$, i.e. head-strictness without strictness.

In fact, it is undefinable, but so are many other properties that are definable when strictness is required. The basic problem is that Fact 1 fails for $\Rightarrow_{wh}$, so even smash projections lead to undefinable properties.

Consider this sequence of functions:

$$\begin{array}{rccl} f_{2i}: & x & \mapsto & \perp^i.\perp \\ f_{2i+1}: & \perp & \mapsto & \perp^i.\perp \\ & x & \mapsto & \perp^{i+1}.\perp, \ \ x \neq \perp \end{array}$$

Suppose α is a projection, and $\alpha(\perp.\perp) = \perp$. Obviously, $f_{2i} = f_{2i} \circ \alpha$ for all i; and $f_{2i+1}(\perp.\perp) = \perp^{i+1} \neq \perp^i = f_{2i+1}(\alpha(\perp.\perp))$. It is also easy to show that the sequence $< f_i >_{i \geq 0}$ is increasing and that all f_i are computable.

Thus, $\mathcal{I} \Rightarrow_{wh} \alpha$ is undefinable for any such α.[1] This includes tail strictness ($\mathcal{I} \Rightarrow_{wh} \mathcal{T}$) [11] and simple head-strictness ($\mathcal{I} \Rightarrow_{wh} \mathcal{H}_B$) [3], although their strict versions ($\mathcal{I} \Rightarrow \mathcal{T}$ and $\mathcal{I} \Rightarrow \mathcal{H}_B$) are definable [3].

In summary: we required strictness in our definition to avoid having to lift D, even though not all $\alpha \Rightarrow_{wh} \beta$ are expressible using $\Rightarrow$. We were justified in doing this because the inexpressible $\alpha \Rightarrow_{wh} \beta$ (namely, those that include non-strict functions) are undefinable by abstract interpretation.

4 BHA-style analysis

One formulation of strictness analysis that is widely followed is the one presented in [1, 4]. There, the Scott-closed subsets of $\mathcal{D}$ — the relevant property of these subsets is that they are downward-closed — play a leading role. In particular, BHA-style analyses can only characterize properties that are downward-closed, such as strictness. Thus, the existence of the chain $f_1 \sqsubseteq f_2$ suffices to prove the undefinability of head-strictness, since f_2 is head-strict and f_1 is not; adding f_0 demonstrates the undefinability of *non*-head-strictness.

It is probably fair to say that most abstract interpretations of interest are either downward-closed or upward-closed. That is, most program transformations are either equivalence-preserving in the presence of non-termination or equivalence-preserving in the presence of guaranteed termination. ([1] extends the BHA-style analysis to upward-closed properties.) However, this still leaves many properties occupying an undefinable middle ground — most obviously, any product of a downward-closed and an upward-closed analysis. Our result is stronger than undefinability via BHA-style analysis, since it precludes analysis by any method whatsoever.

Finally, we note that recently Hunt [8], building on [1], has presented a non-monotonic abstract interpretation for head-strictness, basing his safety proof on logical relations.

[1] I thank Marc Neuberger for this observation.

Acknowledgements

A number of people suggested improvements to an earlier version of this paper. Aside from the referees, who were most helpful, I'd like to thank Geoffrey Burn, Marc Neuberger, and Prateek Mishra. Thanks also to Uday Reddy for many interesting conversations about strictness.

References

[1] S. Abramsky, "Abstract interpretation, logical relations, and Kan extensions," J. Logic and Computation 1(1), July 1990, 5–40.

[2] S. Abramsky, C.L. Hankin (eds.), *Abstract Interpretation of Declarative Languages*, Ellis Horwood, 1987,

[3] G. Burn, "A relationship between abstract interpretation and projection analysis (extended abstract)," 17th POPL, 1990, 151–156.

[4] G. Burn, C.L. Hankin, S. Abramsky, "Strictness analysis of higher-order functions," *Science of Computer Programming* 7, Nov. 1986, 249–278.

[5] C. Clack, S.L. Peyton Jones, "Strictness analysis — a practical approach," Conf. on Functional Programming Languages and Computer Architecture, 1985, LNCS 201, 35–49.

[6] P. Cousot, R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixed points," 4th POPL, 1977, 238–252.

[7] P. Hudak, J. Young, "Higher-order strictness analysis in untyped lambda calculus," 13th POPL, 1986, 97–109.

[8] S. Hunt, "PERs generalize projections for strictness analysis," Glasgow Workshop on Functional Programming, Ullapool, in *Workshops in Computing*, Springer-Verlag, 1991.

[9] S. L. Peyton Jones, *The Implementation of Functional Programming Languages,* Prentice-Hall International, New York, 1987.

[10] P. Wadler, "Strictness analysis on non-flat domains," Chapter 12 in [2], 266–275.

[11] P. Wadler, R.J.M. Hughes, "Projections for strictness analysis," Conf. on Functional Programming Languages and Computer Arch., 1987, LNCS 274, 385–407.