Abstract Semantics of First-Order Recursive Schemes [*]

Robert Muller[†]

Aiken Computation Lab
Harvard University
Cambridge, MA, 02138, USA
muller@harvard.edu

Yuli Zhou[‡]

Lab for Computer Science
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA
zhou@abp.lcs.mit.edu

Abstract

We develop a general framework for deriving abstract domains from concrete semantic domains in the context of first-order recursive schemes and prove several theorems which ensure the correctness (safety) of abstract computations. The abstract domains, which we call *Weak Hoare powerdomains*, subsume the roles of both the abstract domains and the collecting interpretations in the abstract interpretation literature.

1 Introduction

We are often interested in abstract properties of a program that provide approximate information about its actual semantics. For example, that an expression e has *type* τ or that a function f is *strict* (i.e., $f(\perp) = \perp$). When an abstract property is computable, an algorithm for it can be embedded within a compiler and the computed information can be put to a variety of uses during program translation. For example, a type reconstruction algorithm can serve the dual roles of guaranteeing that "well-typed programs don't go wrong" and producing type information that can be used in generating object code that is more efficient than would be possible in a naive translation.

The purpose of this paper is to develop some general conditions for deriving an *abstract domain* A from a concrete semantic domain D. The conditions are intended to be as general as possible while still guaranteeing that the derived domain, what we call a *weak Hoare powerdomain*, has sufficient structure so that it can be used as a basis for computing correct information about D and functions over D. Let $\mathcal{I}(D)$ be the set of all *ideals* of D. A weak Hoare powerdomain A is any subset of $\mathcal{I}(D)$ that includes D as its top element and is closed under intersection and least upper bounds of ascending chains. (A is ordered by subset inclusion.) Examples of weak Hoare powerdomains range from the rather uninformative finite domain containing only D to the complete Hoare powerdomain $(\mathcal{I}(D), \subseteq)$. Other familiar examples include the strictness analysis domain [Myc81], Wadler's finite domain [Wad86] and most instances of Shamir-Wadge *extended domains* [SW77]. (These will be presented in Section 2.) The structure of A is sufficient to guarantee that recursive functions over D can be abstracted over A in such a way that the abstract functions are continuous and yield correct information about their concrete counterparts over D.

[*]This paper was presented at the IFIP Working Group 2.8, Workshop on Functional Programming Languages, Paris, France, April, 1991 and was subsequently published as Harvard University, CRCT Technical Report TR-08-91.

[†]Work supported in part by the US Department of Navy, Space and Naval Warfare Systems Command and Defense Advance Research Projects agency under contract N00039-88-@-0163.

[‡]Research performed in part at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Funding for the Laboratory is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-89-J-1988.

It will be helpful to contrast our approach with the closely related framework of *abstract interpretation* [CC77] and [Myc81, BHA86]. The key idea of abstract interpretation is to define a computable abstract domain which represents an abstract property of interest. Convergent computations over abstract domains will necessarily be subject to information loss. The *safety* condition that ensures that the information computed in the abstract domain is consistent with the concrete semantics is defined in terms of preserving information in a *collecting interpretation* $\mathcal{C}(D)$. In [CC77] the collecting interpretation is a powerset construction. The collecting interpretation is related to the abstract domain by an *abstraction* map $Abs : \mathcal{C}(D) \to A$. A *concretization* function $Conc : A \to \mathcal{C}(D)$ maps abstract values to elements of the collecting interpretation. The safety of a particular abstract interpretation is guaranteed if Abs and $Conc$ are an *adjoined pair* of functions. That is,

$$\begin{aligned} Abs \circ Conc &= Id_A \\ Conc \circ Abs &\sqsupseteq Id_{\mathcal{C}(D)}. \end{aligned}$$

In later work Mycroft [Myc81] retained the Cousots' general framework but replaced the powerset construction with a powerdomain construction. In [BHA86], the collecting interpretation is the Hoare powerdomain of D.

Our approach essentially combines the roles of the collecting interpretation $\mathcal{C}(D)$ and the abstract domain A by giving A itself the (minimal) essential structure inherent in the Hoare powerdomain. As we will show, when the concrete domain D is an ω-algebraic cpo, any weak Hoare powerdomain A derived from D forms an algebraic lattice. Moreover, since it inherits the information ordering of D, abstract computations over A can approximate concrete computations over D.

The safety of abstract computations can be characterized easily since elements of A are ideals. A function $h : A \to A'$ is *safe* for $f : D \to D'$ if, $h(t) = t'$ implies that $f(d) \in t'$ whenever $d \in t$. In order to establish safety, the elements of the concrete domain must be related to ideals in A. In particular, an element $d \in D$ is mapped under $^{\#}$ to the least ideal in A containing d. Similarly, functions $f : D \to D'$ are mapped to their least (or in the terminology of [SW77], *tight*) abstractions $f^{\#} : A \to A'$. In theorem 2, we show: *i)* that $^{\#}$ is continuous, and *ii)* that $f^{\#} : A \to A'$ is continuous whenever $f : D \to D'$ is continuous. We then show that a function $h : A \to A'$ is safe for f iff $f^{\#} \subseteq h$, thus $f^{\#}$ is the least function which is safe for f.

We then take up the question of recursively defined functions in the context of *first-order recursion schemes*. Let f be the least fixpoint of a functional $F : (D \to D') \to (D \to D')$. We show in theorem 4, that the least fixpoint of the functional $F' : (A \to A') \to (A \to A')$ constructed by replacing the primitives in F with their $^{\#}$-versions is safe for f.

We wish to point out the strong influence on our work of the general framework developed by Shamir and Wadge in [SW77]. Our conditions for abstract domains were essentially derived from their conditions for extended domains. One important advantage of the approach is that it simplifies the abstract semantics — there are two domains rather than three. Another important advantage is that any number of abstract domains can be derived directly from a single concrete semantic domain.

The remainder of this paper is organized as follows. In Section 2 we define weak Hoare powerdomains, give a number of familiar examples and state their essential properties. In Section 3 we consider general conditions for the safety of computation in the abstract domain. In section 4 we study recursively defined functions. Section 5 suggests some avenues of future research.

2 Weak Hoare Powerdomains

Preliminaries

Let $D = (U, \sqsubseteq)$ be a partially ordered set. D is a *complete partial order* (cpo) if it has a least element $\bot$, and every chain $\{x_n\} \subseteq D$ has a least upper bound $\bigsqcup_{n \geq 0} x_n \in D$. An element $d \in D$ is *compact* if for all chains $\{x_n\} \subseteq D$, $d \sqsubseteq \bigsqcup_{x \geq 0} x_n \Rightarrow \exists n \geq 0$ such that $d \sqsubseteq x_n$. Let *compact*(D) be the set of compact elements of D. D is an *algebraic* cpo if *compact*(D) is a *basis* of D, i.e., for all $d \in D$, $d = \bigsqcup_{n \geq 0} x_n$ for some chain $\{x_n\}$ such that $x_n \in$ *compact*(D), $n \geq 0$. D is ω-algebraic if it is algebraic and *compact*(D) is countable. (See [Sch86] for more details.)

We will generally use the metavariable b, c, d and e for elements of the concrete domain D and s and t for elements of the abstract domain A. We will usually use the phrase "abstract domain" rather than the more cumbersome "weak Hoare powerdomain." Depending on context, we will refer to their elements as ideals, abstractions or even types.

We now define conditions for obtaining an abstract domain from a cpo D. The elements of the abstract domains are the *weak ideals* of [MPS86].

Definition 1 (**Ideal**) Let $D = (U, \sqsubseteq)$ be a cpo. An *ideal* of D is a set $I \subseteq D$ such that

(i) $d \in I$, $d' \sqsubseteq d \Rightarrow d' \in I$ (I is downward closed).

(ii) $\forall \{x_n\} \subseteq I$ such that $\{x_n\}$ is a chain in D, $\bigsqcup\{x_n\} \in I$ (I is chain-complete).

Let $\mathcal{I}$(D) denote the set of all ideals of cpo D.

Definition 2 (**Weak Hoare Powerdomain**) Let $D = (U, \sqsubseteq)$ be a cpo and $S \subseteq \mathcal{I}(D)$. $A = (S, \subseteq)$ is a *weak Hoare powerdomain* of D if and only if it satisfies the following closure properties:

(i) $U \in A$.

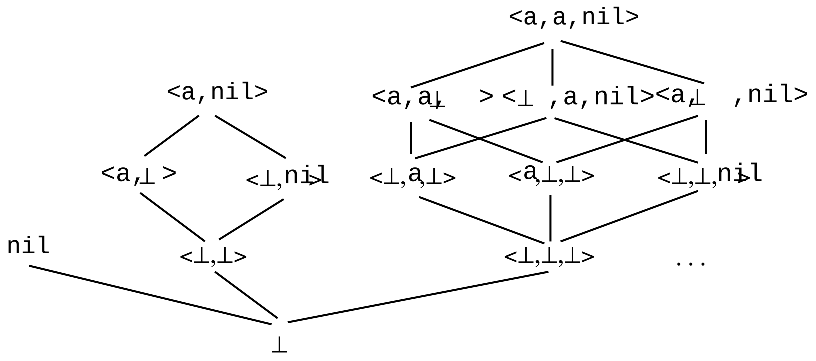
(ii) $X \subseteq A \Rightarrow \bigcap X \in A$.

(iii) $\forall \subseteq$-chains $\{x_n\} \subseteq A$, $\bigcup_{n \geq 0} x_n \in A$.

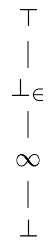
Before proceeding to consider the essential properties of these domains we consider several familiar examples.

Example 1 Let $D_1 = \mathsf{bool} + \mathsf{int}$, and let A_1 be the two point domain $\{0, 1\}$, where $0 = \{\perp\}$ and $1 = D$. A_1 is an abstract domain for strictness analysis of first order functions.

Example 2 Let D_2 be the least solution to the recursive domain equation $D = \text{nil} + (B \times D)$. For the special case where $B = \{a\}_\perp$, we can illustrate the structure of D_2 by the following diagram. D_2 is a



domain of lists. The Wadler abstract domain A_2 for strictness analysis is a four element chain



where

$$\begin{array}{rcl} \top & = & D_2 \\ \bot_{\in} & = & \{l \mid \text{some element of } l \text{ is } \bot\} \\ \infty & = & \{l \mid l \text{ is infinite, or the last element of } l \text{ is } \bot\} \\ \bot & = & \{\bot_D\}. \end{array}$$

Example 3 Let D_3 be the least solution of the domain equation

$$D = \mathsf{bool} + \mathsf{int} + (D \times D) + (D \to D).$$

The *type* domain A_3 of a Milner type system can be defined inductively as the following:

(i) bool, int, and $\top = D_3 \in A_3$.

(ii) If s, $t \in A_3$, then $s \times t$, $s \to t \in A_3$ where $s \times t$ is the usual product domain and

$$s \to t = \{f \mid \forall\, x \in s, f(x) \in t\}.$$

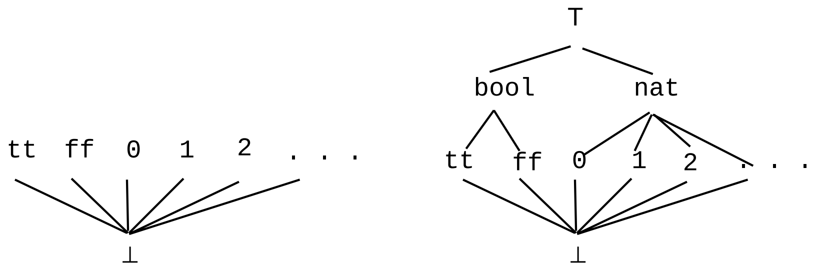
(iii) If $S \subseteq A_3$ then $\bigcap S \in A_3$.

(iv) If $\{t_n\}$ is a chain in A_3 then $\bigcup_{n \geq 0} t_n \in A_3$.

One can verify that A_3 is an abstract domain. Note that polymorphic functions have intersection types, e.g., the type of the identity function $\lambda x.x$ has the type $\bigcap_{t \in A_3} t \to t$. Note also that A_3 contains more ideals than is needed for Milner-style type reconstruction.

Our final example is drawn from [SW77] which defined a closely related notion of extended domains.

Example 4 Let $D_4 = \mathsf{bool} + \mathsf{nat}$. For $d \in D$, let $\tilde{d} = \{d' \mid d' \sqsubseteq d\}$ (This left closure of d is given by $\{d\}$ in [BHA86]) and $\tilde{D} = \{\tilde{d} \mid d \in D\}$. Then $A_4 = \tilde{D_4} \cup \{\mathsf{bool}, nat, D_4\}$ ordered in the usual way.



It is easy to see that A_4 is an abstract domain. Note that A_4 contains an isomorphic copy $\tilde{D}$ of D. Thus, A_4 can be thought of as D_4 extended with type objects.

2.1 Properties of Weak Hoare Powerdomains

We now develop the properties of abstract domains which ensure that their structure is suitable for computation. First note that for domain A, and arbitrary $X \subseteq A$, the greatest lower bound of X is the intersection of X, but the least upper bound of X is generally not the union of X. However, due to condition (iii) of definition 2 we have:

Lemma 1 *If $X \subseteq A$ is chain, then $\bigsqcup X = \bigcup X$.*

Note also that conditions (i) and (ii) guarantee A to be a complete lattice under $\subseteq$ with $\bigcap A$ as its least element, D as its greatest element, $\bigcap$ as $\sqcap$ and $\bigsqcup X = \bigcap\{t \mid t \in A, t \text{ is an upper bound of } X\}$. Moreover, we have

Theorem 1 *Any weak Hoare powerdomain A of an ω-algebraic cpo D is an algebraic lattice.*

Proof By the above remarks, any abstract domain A is a complete lattice. It remains to show that it is also algebraic. The proof requires the abstraction map ${}^{\#} : D \to A$, to be defined in Section 3. $d^{\#}$ maps d to the least ideal containing d.

$$d^{\#} = \bigcap\{t \in A \mid d \in t\}.$$

The crucial property of ${}^{\#}$ that we require is that $d \in t \iff d^{\#} \subseteq t$, (lemma 4).

We shall show that each element $t \in A$ is the l.u.b. of some chain $\{t_n\}$ of compact elements by actually constructing $\{t_n\}$ for each t. Given $t \in A$, let $B_t = \{b \mid b \in t \text{ and } b \text{ is compact}\}$ be the set of compact elements of t. Since D is ω-algebraic, we can enumerate the elements of B_t as $b_0, b_1, \ldots$. The following two lemmas will be needed.

Lemma 2 $t = \bigsqcup_{n\geq 0} b_n^{\#}$.

Proof Clearly $b_n \in t$, thus $b_n^{\#} \subseteq t$, $n \geq 0$ and $\bigsqcup_{n\geq 0} b_n^{\#} \subseteq t$.

We now show the reverse inclusion also holds. Since D is ω-algebraic, each $d \in t$ is the limit of some chain $\{c_n\}$ where $c_n \in B_t$, $n \geq 0$. Therefore

$$c_n \in \bigsqcup_{n\geq 0} b_n^{\#}.$$

Since the right-hand side is an ideal, which is closed under the l.u.b. of chains, we have

$$d = \bigsqcup_{n\geq 0} c_n \in \bigsqcup_{n\geq 0} b_n^{\#}.$$

□

Lemma 3 *If $e_1, e_2, \ldots, e_n$ are compact then $\bigsqcup\{e_1^{\#}, e_2^{\#}, \ldots, e_n^{\#}\}$ is compact.*

Proof Let $t = \bigsqcup\{e_1^{\#}, e_2^{\#}, \ldots, e_n^{\#}\}$, we show for all chain $\{s_m\}$ in A

$$t \subseteq \bigsqcup_{m\geq 0} s_m \implies \exists\, N \geq 0 \text{ s.t. } t \subseteq s_N.$$

Since $t \subseteq \bigsqcup_{m\geq 0} s_m$, we have $e_1, e_2, \ldots, e_n \in \bigsqcup_{m\geq 0} s_m = \bigcup_{m\geq 0} s_m$. It follows that $b_i \in s_{m_i}$ for some m_i. Let $N = \max\{m_i \mid 1 \leq i \leq n\}$, then $e_1, e_2, \ldots, e_n \in s_N$, which means $b_i^{\#} \subseteq s_N$, $1 \leq i \leq n$. Therefore

$$t = \bigsqcup\{e_1^{\#}, e_2^{\#}, \ldots, e_n^{\#}\} \subseteq s_N.$$

□

Resuming our proof of theorem 1. Let us define

$$t_n = \bigsqcup\{b_1^{\#}, b_2^{\#}, \ldots, b_n^{\#}\}, \quad n \geq 0.$$

$\{t_n\}$ is a chain, whose elements are compact according to lemma 3. More over by lemma 2

$$\bigsqcup_{n\geq 0} t_n = \bigsqcup_{n\geq 0} b_n^{\#} = t,$$

thus the theorem is proved. □

In the next section we develop one of the important consequences of the algebraicity of A: that every continuous function $f \in [D \to D']$ has a continuous abstraction $f^{\#} \in [A \to A']$.

3 Abstraction and Its Safety

Let D be a domain, A be any abstract domain of D. We first define the mapping ${}^{\#} : D \to A$ such that

$$d^{\#} = \bigcap \{t \in A \mid d \in t\}.$$

Intuitively, $^{\#}$ maps every element d to the least ideal in A which contains it. Note that $d^{\#}$ always exists since A is a complete lattice. Therefore $^{\#}$ is well-defined. Moreover, we have the following,

Lemma 4 *$d \in t \iff d^{\#} \subseteq t$.*

We now extend $^{\#}$ to functions. Let A, A' be abstract domains for D, D' respectively. Given $f : D \to D'$, define $f^{\#} : A \to A'$ such that

$$f^{\#}(t) = \bigsqcup_{d \in t} f(d)^{\#}.$$

We return to the examples from the preceding section.

Example 1 (*continued*) The abstraction map ${}^{\#} : D_1 \to A_1$ is defined by

$$d^{\#} = \begin{cases} 0 & \text{if } d = \bot \\ 1 & \text{if } d \neq \bot. \end{cases}$$

Example 2 (*continued*) (cf. [Wad86] where this abstraction map was originally defined)

$$l^{\#} = \begin{cases} \bot & \text{if } l = \bot \\ \infty & l \text{ is infinite, or the last element of } l \text{ is } \bot \\ \bot_{\in} & l \text{ is finite, some element of } l \text{ is } \bot \\ \top & l \text{ is finite and no element of } l \text{ is } \bot. \end{cases}$$

Example 3 (*continued*) For the non-functional values in D_3, $\bot^{\#} = \{\bot\}$, $d^{\#} = \mathsf{bool}$ if $d \in \mathsf{bool}$, and $d^{\#} = \mathsf{int}$ if $d \in int$. For functions in D_3 we only show a few special cases. Clearly $+^{\#} = \mathsf{int} \times \mathsf{int} \to \mathsf{int}$, and cond$^{\#} = \bigcap_{t \in A_3} \mathsf{bool} \times t \times t \to t$. Note that cond is thus polymorphic. Another example of a polymorphic function is $\lambda x.x$, which gets mapped to $\bigcap_{t \in A_3} t \to t$.

Example 4 (*continued*) For $d \in D$, $d^{\#} = \tilde{d}$.

Theorem 2 *(i) ${}^{\#} : D \to A$ is continuous.*

(ii) $f^{\#} : A \to A'$ is continuous whenever $f : D \to D'$ is continuous.

Proof (i) We need to verify that for every chain $\{d_n\}$ in D,

$$(\bigsqcup_{n \geq 0} d_n)^{\#} = \bigsqcup_{n \geq 0} d_n^{\#}. \tag{1}$$

Note that $d_n \sqsubseteq \bigsqcup_{n \geq 0} d_n$ and $^{\#}$ is monotonic, therefore $d_n^{\#} \subseteq (\bigsqcup_{n \geq 0} d_n)^{\#}$, and thus

$$\bigsqcup_{n \geq 0} d_n^{\#} \subseteq (\bigsqcup_{n \geq 0} d_n)^{\#}. \tag{2}$$

On the other hand, by definition of $^{\#}$ we have

$$\begin{aligned} (\bigsqcup_{n \geq 0} d_n)^{\#} &= \bigcap \{t \mid \bigsqcup_{n \geq 0} d_n \in t\} \\ &= \bigcap \{t \mid \forall\, n \geq 0 d_n \in t\} \qquad \text{since } t \text{ is an ideal} \\ &\subseteq \bigsqcup_{n \geq 0} d_n^{\#}, \end{aligned} \tag{3}$$

where the last inclusion holds since clearly $\forall\, n \geq 0\ d_n \in \bigsqcup_{n\geq 0} d_n^{\#}$. Combining equation (2) and (3) we know (1) holds.

(ii) It is easy to see that $f^{\#}$ is monotonic when f is continuous, therefore clearly

$$\bigsqcup_{n\geq 0} f^{\#}(t_n) \subseteq f^{\#}(\bigsqcup_{n\geq 0} t_n). \tag{4}$$

To see the reverse inclusion also holds, we expand $f^{\#}(\bigsqcup_{n\geq 0} t_n)$ according to definition to get

$$\begin{aligned} f^{\#}(\bigsqcup_{n\geq 0} t_n) &= \bigsqcup\{f(d)^{\#} \mid d \in \bigsqcup_{n\geq 0} t_n\} \\ &= \bigsqcup\{f(d)^{\#} \mid d \in \bigcup_{n\geq 0} t_n\} \qquad \text{since } \{t_n\} \text{ is a chain} && (5) \\ &= \bigsqcup_{\substack{d\in t_n \\ n\geq 0}} f(d)^{\#} && (6) \end{aligned}$$

Note that closure condition (iii) in the definition of abstract domains is crucial to obtaining (5). Also note that if $d \in t$, then $f(d)^{\#} \subseteq f^{\#}(t)$. Thus for every $d \in t_n$, $n \geq 0$ we have $f(d)^{\#} \subseteq f^{\#}(t_n)$. It then follows easily that

$$\bigsqcup_{\substack{d\in t_n \\ n\geq 0}} f(d)^{\#} \subseteq \bigsqcup_{n\geq 0} f^{\#}(t_n). \tag{7}$$

Equations (4), (6) and (7) together validates our claim. □

3.1 Safety

Intuitively, we would like $f^{\#}$ to correspond to f, in the sense that if $f^{\#}(t) = t'$, then it is guaranteed that f maps every element in t to some element in t'. Obviously, there are many functions $A \to A'$ satisfying this condition, which is known in the abstract interpretation literature as *safety*. More specifically, we shall say that a function $h : A \to A'$ is *safe for* $f : D \to D'$ if $h(t) = t'$ implies that $f(d) \in t'$ whenever $d \in t$. Our definition for $f^{\#}$ will be justified shortly when we show it is in fact the least function safe for f, but before we proceed, we first give the familiar characterization for safety in the following proposition.

Proposition 1 *If $f : D \to D'$ and $h : A \to A'$ are continuous, then the following conditions are equivalent:*

(i) h is safe for f.

(ii) For all d, $f(d)^{\#} \subseteq h(d^{\#})$, i.e., the following diagram commutes:

$$\begin{tikzcd} \mathbf{D} \arrow[r, "\mathbf{f}"] \arrow[d, "\#"] & \mathbf{D'} \arrow[d, "\#"] \\ \mathbf{T} \arrow[r, "\mathbf{h}"] & \mathbf{T'} \end{tikzcd}$$

Proof (i) $\Longrightarrow$ (ii). Since $d \in d^{\#}$, we know $f(d) \in h(d^{\#})$ by the safety assumption, and $f(d)^{\#} \subseteq h(d^{\#})$ follows from lemma 4.

(ii) $\implies$ (i). If $d \in t$, then $d^{\#} \subseteq t$, therefore by assumption

$$f(d)^{\#} \subseteq h(d^{\#}) \subseteq h(t).$$

This implies that $f(d) \in h(t)$, i.e., h is safe for f. □

We state an obvious lemma which will be needed later, which says that safety is closed under functional composition.

Lemma 5 *If h, h' are safe for f, f' respectively, then $h' \circ h$ is safe for $f' \circ f$.*

$f^{\#}$ is the least function safe for f. This fact is stated in the following theorem.

Theorem 3 *h is safe for f iff $f^{\#} \subseteq h$ (inclusion taken to be point-wise).*

Proof (if) Assuming that h is safe for f, then for all $d \in t$ $f(d) \in h(t)$, which means $f(d)^{\#} \subseteq h(t)$. From this we can verify that

$$f^{\#}(t) = \bigsqcup_{d \in t} f(d)^{\#} \subseteq h(t).$$

(only-if) By the definition of $f^{\#}$ it is clear that $f^{\#}$ is safe for f. If $f^{\#} \subseteq h$, then for all $d \in t$

$$f(d) \in f^{\#}(t) \subseteq h(t),$$

i.e., h is safe for f. □

Unfortunately, $^{\#}$ is not closed under functional composition, i.e.,

$$(f \circ g)^{\#} \sqsubseteq f^{\#} \circ g^{\#}.$$

Thus, composite functions will be subject to information loss. In [SW77] this problem is addressed using *case analysis*.

4 Recursively Defined Functions

We now consider the question of how to compute abstract information for a recursively defined function in the corresponding abstract domains. The functions we study in this section are those definable using first-order recursion schemes [Vui74, DS76]. As in both of those papers we restrict our attention to a single recursively defined function, extension to a group of simultaneous recursive definitions being straightforward.

Specifically, let e be an expression which is either a constant d, the variable x, $p(e)$ or $f(e)$ where p is a primitive function and f a function variable. Given an expression e containing variable x and function variable f, the equation

$$f(x) = e$$

defines a unique function f_0 as the least fixpoint of the functional

$$F = \lambda f.\lambda x.e.$$

Our intention is to transform F into F' by replacing the constants and base functions in F by their respective $^{\#}$-versions, and to show the least fixpoint of F' is safe for f_0. To this end let us define the map $\hat{}$ s.t.

(i) $\hat{d} = d^{\#}$, $\hat{x} = x$.

(ii) $\widehat{p(e)} = p^{\#}(\hat{e})$.

(iii) $\hat{f(e)} = f(\hat{e})$.

We can now define

$$F' = \lambda f.\lambda x.\hat{e}$$

whose least fixpoint gives us a function $h : A \to A'$.

Theorem 4 *$h = \mathrm{fix}(\lambda f.\lambda x.\hat{e})$ is safe for $f_0 = \mathrm{fix}(\lambda f.\lambda x.e)$.*

We shall prove the theorem by an induction on the construction of the two fixpoints, where the following two lemmas establish the crucial connections between them.

Lemma 6 *If h is safe for p, then $\lambda x.\hat{e}[h/f]$ is safe for $\lambda x.e[p/f]$.*

Proof We prove the lemma by induction on the formation of e. There are four cases according to the formation rules.

(1) $e \equiv d$. $\lambda x.\hat{d} = \lambda x.d^{\#}$ is clearly safe for $\lambda x.d$.

(2) $e \equiv x$. Obvious.

(3) $e \equiv p(e')$. We know

$$\begin{array}{rcl}\lambda x.p(e') & = & p \circ \lambda x.e', \\ \lambda x.\hat{p(e')} & = & p^{\#} \circ \lambda x.\hat{e'}.\end{array}$$

Since by the induction hypothesis $\lambda x.\hat{e'}$ is safe for $\lambda x.e'$, from lemma 6 it then follows easily that $\lambda x.\hat{p(e')}$ is safe for $\lambda x.p(e')$.

(4) $e \equiv f(e')$. Similar to (3). □

Lemma 7 *If $\{p_n\}$, $\{h_n\}$ are two chains of functions s.t. h_n is safe for p_n, $n \geq 0$, then $\bigsqcup_{n\geq 0} h_n$ is safe for $\bigsqcup_{n\geq 0} p_n$.*

Proof Let $t \in A$ and $d \in t$. If h_n is safe for p_n, then $p_n(d) \in h_n(t)$. It follows that

$$p_n(d) \in \bigsqcup_{n\geq 0} h_n(t), \quad n \geq 0.$$

since the right-hand side is an ideal and is thus closed under the l.u.b. of chains, we have

$$\bigsqcup_{n\geq 0} p_n(d) \in \bigsqcup_{n\geq 0} h_n(t),$$

thus the lemma holds. □

Proof of theorem 4: Let

$$h_0 = \lambda x.\{\perp\}$$
$$h_n = \lambda x.\hat{e}[h_{n-1}/f],$$

then $h = \bigsqcup_{n\geq 0} h_n$. Similarly let

$$p_0 = \lambda x.\perp$$
$$p_n = \lambda x.e[p_{n-1}/f],$$

and we have $f_0 = \bigsqcup_{n\geq 0} p_n$. By lemma 6 we know h_n is safe for p_n, $n \geq 0$. It then follows easily from lemma 7 that h is safe for f_0. □

5 Conclusions

We have developed a general framework for deriving abstract domains from concrete semantic domains and several theorems which ensure the correctness of abstract computations. We believe that the framework is simpler than abstract interpretations which employ collecting interpretations.

There are a number of interesting questions which require further study. We are particularly interested in developing methods to improve the approximations of recursive functions. (That is, to reduce the information loss by bringing the least fixpoint in the abstract domain as close to $f^{\#}$ as possible.) Shamir and Wadge developed a general *case analysis* method which may be useful when the abstract domain is not too sparse.

We are also interested in surveying other applications of abstract interpretation to see if they might be usefully recast in the current framework. We believe that the generality of the framework makes it particularly well suited for *type analysis* which is the subject of increasing interest in the literature [YO88, RW91, FP91].

If the framework is to be generally applicable to functional programming languages then it will have to be extended analogously to [BHA86] to accommodate higher order functions. Unfortunately it seems that the techniques in [BHA86] will not carry over. We expect this to be a major undertaking.

We are also interested in studying the computational aspects of fixpoint computations in abstract domains along the lines of [Vui74, DS76]. A related problem is to develop good hueristic methods for fixpoint computations over abstract domains that have infinite chains. Some ideas along these lines have been suggested in [YO88].

References

[BHA86] G. Burns, C. Hankin, and S. Abramsky. Strictness analysis for higher-order functions. *Science of Computer Programming*, 7:249–278, 1986.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixpoints. In *Proceedings of the Fourteenth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[DS76] P. Downey and R. Sethi. Correct computation rules for recursive languages. *SIAM Journal of Computing*, 5:378–401, 1976.

[FP91] T. Freeman and F. Pfenning. Refinement types for ML (extended abstract) (to appear). In *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*, 1991.

[MPS86] D. MacQueen, G. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. *Information and Computation*, **71**, No. 1/2:95–130, 1986.

[Myc81] A. Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs.* PhD thesis, University of Edinburgh, 1981.

[RW91] E. Ruf and D. Weise. Using types to avoid redundant specialization (to appear). In *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, 1991.

[Sch86] D. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, 1986.

[SW77] A. Shamir and W. Wadge. Data types as objects. In *4th Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Volume 52*, pages 465–479, 1977.

[Vui74] J. Vuillemin. Correct and optimal implementation of recursion in a simple programming language. *Journal of Computer and System Science*, 9:332–334, 1974.

[Wad86] P. Wadler. Strictness analysis on non-flat domains (by abstract interpretation over finite domains). In *Abstract Interpretation of Declarative Languages*, pages 266–275. Ellis Horwood Limited, 1986.

[YO88] J. Young and P. O'Keefe. Experience with a type evaluator. In *Partial Evaluation and Mixed Computation, North Holland*, pages 573–581, 1988.