Abstract Interpretation in Weak Powerdomains

Robert Muller*

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

Yuli Zhou†

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

Abstract

We introduce a class of semantic domains, *weak powerdomains*, that are intended to serve as value spaces for abstract interpretations in which *safety* is a concern. We apply them to the analysis of PCF programs. In the classical abstract interpretation approach, abstract domains are constructed explicitly and the abstract semantics is then related to the concrete semantics. In the approach presented here, abstract domains are *derived* directly from concrete domains. The conditions for deriving the domains are intended to be as general as possible while still guaranteeing that the derived domain has sufficient structure so that it can be used as a basis for computing correct information about the concrete semantics. We prove three main theorems, the last of which ensures the correctness of abstract interpretation of PCF programs given safe interpretations of the constants. This generalizes earlier results obtained for the special case of strictness analysis.

1 Introduction

Abstract interpretation provides a formal method for ensuring the correctness of computations of properties of programs. The idea was first proposed by the Cousots [CC77] and later applied to the problem of *strictness analysis* by Mycroft [Myc81]. Since then there have been many important developments. In [BHA86, Abr90], Mycroft's setup was extended to higher-order functions in the simply typed frame and in [Abr86, Hug88] and [AJ91] the method was extended to account for data structures and polymorphic functions. Recent work [KM89, Jen91] has studied the connection between the abstract interpretation and type inference approaches to strictness analysis.

The purpose of this paper is to propose a new class of domains, what we call *weak powerdomains*, as value spaces for certain types of abstract analysis. In the classical approach, abstract domains are constructed explicitly and the abstract semantics is then related to the concrete semantics through, for example, *collecting interpretations* [Myc81, BHA86] or *logical relations* [Abr90, HS91]. In the approach taken here, abstract semantic domains are *derived* directly from concrete domains. The conditions for deriving the domains are intended to be as general as possible while still guaranteeing that the derived domains have sufficient structure so that they can be used as a basis for computing correct information about the concrete semantics. The main advantages of the approach are its simplicity and generality. It is limited, however, to the analysis of properties of programs representable by *ideals*.

Let D be a cpo and let $\mathcal{I}(D)$ denote the set of all ideals of D. A weak powerdomain A of D is any subset of $\mathcal{I}(D)$ that includes D as its top element and is closed under intersection and least upper bounds of directed sets. (A is ordered by subset inclusion.) Examples of weak powerdomains range from the one-point domain containing only D to the complete *Hoare powerdomain* $(\mathcal{I}(D), \subseteq)$. Other familiar examples include the two-point strictness analysis domain [Myc81], Wadler's finite domain of lists [Wad86, Hug88] and most instances of Shamir-Wadge *extended domains* [SW77]. (Other examples will be presented in the sections that follow.) The key point is that 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.

Like [CC77, Myc81, MN83, Nie84] and [BHA86] we characterize *safety* from a domain theoretic rather than a relational (cf., [MJ86, Nie89, Abr90, HS91]) point of view. Unlike earlier domain theoretic frameworks, however, our approach does not make use of collecting interpretations or *concretization* maps. Rather, the roles of the collecting interpretation and the abstract domain are essentially collapsed into one — the abstract domain itself has the (minimal) essential structure inherent in the Hoare powerdomain. As we show in theorem 1, when the concrete domain D is an algebraic cpo, any weak 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.

An important property of the domains considered here is that their essential closure conditions are preserved by the standard domain constructions $+$, $\times$ and $\rightarrow$. In this paper we apply them to the analysis of PCF [Plo77]. We

*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.

1992 ACM LISP & F.P.-6/92/CA
© 1992 ACM 0-89791-483-X/92/0006/0119...$1.50

begin with a standard semantics in which terms of type σ are interpreted in the *type frame* $\{D_\sigma\}$, a collection of algebraic cpos. We then derive a weak powerdomain A_ι from the base domain D_ι for each base type ι. Next we construct an *abstract frame* $\{A_\sigma\}$, taking $A_{\sigma\to\tau} = [A_\sigma \to A_\tau]$ (i.e., the continuous functions from A_σ to A_τ). Although $A_{\sigma\to\tau}$ is not a weak powerdomain (because its components are functions rather than ideals) we show in theorem 2 that it nevertheless satisfies the required closure conditions.

In order to establish the safety of the analysis, each element of D_σ must be abstracted to an element of A_σ. The safety relation is defined inductively on types. For base type ι, $a \in A_\iota$ is safe for $d \in D_\iota$ if $d \in a$. For higher types $\sigma \to \tau$, $\hat{f} \in A_{\sigma\to\tau}$ is safe for $f \in D_{\sigma\to\tau}$ if for all $a \in A_\sigma$ safe for $d \in D_\sigma$, $\hat{f}(a) \in A_\tau$ is safe for $f(d) \in D_\tau$. We show in theorem 3 that any non-standard semantics that maps constants to safe values is safe for the standard semantics. This generalizes the result established for the special case of strictness analysis in [BHA86] and [Abr90].

Finally, we define some criteria for comparing the relative information content of abstract analyses that differ in their interpretations of constants. The most informative is that which maps constants to the least abstract values safe for them. An element d in the base domain D_ι is mapped to the least ideal $d^\# \in A_\iota$ containing d. An element $f \in D_{\sigma\to\tau}$ is mapped to the least monotonic function $f^\# \in A_{\sigma\to\tau}$ safe for f. Unfortunately, leastness is not preserved under function composition thus there is an inevitable loss of information from the ideal abstract semantics of PCF.

The remainder of this paper is organized as follows. In Section 2 we define the standard semantics of PCF. In section 3 we define the structure of abstract domains and consider their essential properties. In Section 4 we consider abstract mappings and safety and in section 5 we present the abstract semantics. Section 7 considers some extensions to the basis setup. Section 7 compares the present work with related work and suggests some avenues of future research.

2 Standard Semantics of PCF

In this section we define the language PCF and its standard semantics. The presentation follows that in [Plo77]. We begin with some preliminary definitions and elementary properties. We assume some familiarity with elementary domain theory.

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 directed $X \subseteq D$ has a least upper bound $\bigsqcup X \in D$. An element $d \in D$ is *compact* if for all directed $X \subseteq D$, $d \sqsubseteq \bigsqcup X \Rightarrow \exists x \in X$ such that $d \sqsubseteq x$. Let $compact(D)$ be the set of compact elements of D. D is an *algebraic* cpo if for all $d \in D$, $d = \bigsqcup X$ for some directed set X such that $X \subseteq compact(D)$. A function $f : D \to D'$ is *continuous* if for all directed $X \subseteq D$, $f(\bigsqcup_D X) = \bigsqcup_{D'}\{f(x) \mid x \in X\}$. If D and D' are cpos then $[D \to D']$, the set of continuous functions from D to D', is also a cpo under the pointwise ordering, $f \sqsubseteq g$ if $\forall x$, $f(x) \sqsubseteq g(x)$. Moreover, if D and D' are algebraic then so is $[D \to D']$. (See, for example [Sch86], for more details.)

We now define the abstract syntax of PCF type expressions.

$$\sigma ::= \iota_i \mid \sigma \to \sigma$$

where, ι_i $(i \geq 0)$ denotes a collection of *base types* with $\iota_0 = Bool$. We will use the metavariables σ and τ to range over type expressions.

Definition 1 Let the base type ι_i denote an algebraic cpo D_{ι_i} with $D_{Bool} = \{tt, ff, \bot\}$ $(\bot \sqsubseteq tt,\ \bot \sqsubseteq ff)$. A *type frame* for PCF is a set of algebraic cpos $\{D_\sigma\}$, one for each type, such that $D_{\sigma\to\tau} = [D_\sigma \to D_\tau]$.

The language PCF will include a set $Var = \{x_{\sigma_1}, x_{\sigma_2}, \ldots\}$ of *variables* each of fixed type and a set $Const = \{c_{\sigma_1}, c_{\sigma_2}, \ldots\}$ of *constants* each of fixed type. The set *Const* includes the boolean symbols *tt* and *ff* as well as the conditional $if_\sigma : Bool \to \sigma \to \sigma \to \sigma$, one for each type and the fixpoint combinator $Y_\sigma : (\sigma \to \sigma) \to \sigma$, one for each type. Here and in the treatment of expressions we will usually omit the type subscripts to avoid clutter. PCF is then given by the abstract syntax

$$e ::= c \mid x \mid e\ e \mid \lambda x.e$$

We use the symbol *Exp* to denote the set of expressions and the symbols $e, e', \ldots$ as metavariables ranging over *Exp*.

The semantics will be defined relative to an interpretation of the constants as determined by the valuation $\mathcal{S}_c : Const \to \bigcup\{D_\sigma\}$. The interpretation of the booleans is given by

$$\begin{aligned} \mathcal{S}_c[\![tt]\!] &= tt \\ \mathcal{S}_c[\![ff]\!] &= ff \end{aligned}$$

The interpretations of the conditional and fixpoint constants is given by

$$\mathcal{S}_c[\![if]\!]\ b\ d\ d' = \begin{cases} \bot & \text{if } b = \bot, \\ d & \text{if } b = tt, \\ d' & \text{if } b = ff \end{cases}$$

$$\mathcal{S}_c[\![Y]\!](f) = \bigsqcup_{n\geq 0} f^n(\bot)$$

The set of type respecting environments map variables of type σ to D_σ.

$$\rho \in Env_{\mathcal{S}} = Var \to \bigcup\{D_\sigma\}$$

We use the notation, $\rho[d/x]$ to denote the environment ρ perturbed at x.

$$\rho[d/x][\![y]\!] = \begin{cases} d & x \equiv y, \\ \rho[\![y]\!] & \text{otherwise.} \end{cases}$$

The empty environment maps any variable of type σ to $\bot_\sigma$.

The standard semantics is then given by the valuation $\mathcal{S} : Exp \to Env_{\mathcal{S}} \to \bigcup\{D_\sigma\}$ which is defined in the usual way by induction on the structure of e.

$$\begin{aligned} \mathcal{S}[\![c]\!]\rho &= \mathcal{S}_c[\![c]\!] \\ \mathcal{S}[\![x]\!]\rho &= \rho[\![x]\!] \\ \mathcal{S}[\![e\ e']\!]\rho &= \mathcal{S}[\![e]\!]\rho(\mathcal{S}[\![e']\!]\rho) \\ \mathcal{S}[\![\lambda x.e]\!]\rho &= \lambda d.\mathcal{S}[\![e]\!](\rho[d/x]) \end{aligned}$$

It is well known that the semantics is well-defined in the sense that any expression $e \in Exp$ is assigned a meaning in the appropriate domain D_σ.

3 Abstract Domains

Given a type frame $\{D_\sigma\}$ for PCF, we wish to construct an abstract frame $\{A_\sigma\}$ over which an abstract semantics of PCF can be defined. Naturally, each abstract domain A_σ should represent a class of properties on the corresponding concrete domain D_σ. The abstract domains we shall construct are all algebraic lattices that are (at base types), or are isomorphic to (at function types), subsets of the Hoare powerdomain of the corresponding concrete cpo's. The construction will be carried out inductively, first for the base types (weak powerdomains) and then for the function types.

We recall from domain theory some basic concepts about lattices. A *complete lattice* A is a cpo in which $\bigsqcup X$ and $\bigsqcap X$ exist for every subset $X \subseteq A$. An *algebraic lattice* is a complete lattice which is algebraic as a cpo. A subset $E \subseteq A$ is called a *sub-basis* of A if $x = \bigsqcup\{e \mid e \in E \text{ and } e \sqsubseteq x\}\ \forall\ x \in A$. E is a *basis* of A if it is moreover closed under the lub of all its finite subsets, i.e., $\forall\ E' \subseteq E$, E' finite, $\bigsqcup E' \in E$. Let *compact*(A) be the set of compact elements of A, then it is closed under the lub of all its finite subsets. Thus A is algebraic if *compact*(A) is a basis of A.

3.1 Weak Powerdomains

Given a concrete domain D, the kind of properties we are interested in are represented extensionally by ideals of D.

Definition 2 An *ideal* of D is a subset $I \subseteq D$ s.t.

(i) $x \in I, y \sqsubseteq x \implies y \in I$,

(ii) $X \subseteq I$ directed $\implies \bigsqcup X \in I$.

An abstract domain for D will be a collection of ideals of D, thus a subset of the Hoare (lower) powerdomain of D. Clearly, not every subset can be an abstract domain as it may lack the structure needed for abstract semantics. On the other hand, the full Hoare domain has many ideals which are of no conceivable interest. We therefore introduce the following:

Definition 3 Let D be a cpo. A *weak powerdomain* (wpd) of D is $(A, \subseteq)$ where A is any collection of ideals of D satisfying the following closure conditions:

(i) $D \in A$ (we shall call D the *top element* of A and denote it by $\top_A$),

(ii) $X \subseteq A \implies \bigcap X \in A$,

(iii) For all directed $X \subseteq A$, $\bigsqcup X \in A$.

Example 1 Let int be the domain of integers. The familiar abstract domain for the rule of signs

$$S = \{\{\bot\}, \mathbf{neg}, \mathbf{zero}, \mathbf{pos}, \mathbf{int}\}$$

is a wpd of int, where

$$\begin{aligned}\mathbf{neg} &= \{\bot, \ldots, -2, -1\}\\ \mathbf{zero} &= \{\bot, 0\}\\ \mathbf{pos} &= \{\bot, 1, 2, \ldots\}.\end{aligned}$$

Example 2 Let D be any domain. The two point abstract domain $\{0, 1\}$ for strictness analysis is a wpd of D, where $0 = \{\bot\}$ and $1 = D$.

As it will be shown, closure conditions (i) and (ii) make A a complete lattice, and condition (iii) in addition guarantees A to be algebraic. In fact, A has a much nicer structure than what is implied here, which will be revealed as we proceed to prove the foregoing claims. We first define an abstraction mapping $^\# : D \to A$ that relates concrete and abstract elements:

Definition 4 $d^\# = \bigcap\{a \in A \mid d \in a\}$.

Intuitively, $^\#$ maps $d \in D$ to the least element in A that contains d. This fact gives us the following obvious lemma:

Lemma 1 $d \in a \iff d^\# \subseteq a$.

Now the main theorem:

Theorem 1 *Let A be a weak powerdomain of an algebraic cpo. Then A is an algebraic lattice.*

Proof We first show that A is a complete lattice and then show that it is also algebraic. Let $X \subseteq A$, since A is closed under intersection, $\bigsqcap X = \bigcap X$. $\bigsqcup X = \bigcap\{x \mid \forall y \in X\ y \subseteq x\}$ is just the ordinary definition of lub in A. Therefore A is a complete lattice. To show that A is algebraic we require the following lemmas.

Lemma 2 *For all $d \in D$, $d^\# \in A$ is compact.*

Proof Let $d^\# \subseteq \bigsqcup X$, where $X \subseteq A$ is directed. By lemma 1, $d \in \bigsqcup X$. Since $\bigsqcup X = \bigcup X$, there is some $x \in X$, s.t. $d \in x$. It then follows that $d^\# \subseteq x$, i.e., $d^\#$ is compact. □

Remark 1 Note that $d^\#$ is always compact in A, even if d is not compact in D. Therefore the wpd A of an algebraic cpo D is generally not ω-algebraic as *compact*(A) can have uncountably many elements.

Lemma 3 $B = \{d^\# \mid d \in D\}$ *is a sub-basis of A.*

Proof The statement in the lemma means that for each element $x \in A$,

$$x = \bigsqcup\{d^\# \mid d^\# \subseteq x\},$$

which according to lemma 1 is equivalent to

$$x = \bigsqcup\{d^\# \mid d \in x\}.$$

Let $x' = \bigsqcup\{d^\# \mid d \in x\}$. Since $d^\# \subseteq x$ if $d \in x$, we have $x' \subseteq x$. On the other hand, every element $d \in x$ is contained in $d^\#$, thus in x', therefore $x \subseteq x'$. □

By the definition of a basis, the equation

$$B^* = \{\bigsqcup E \mid E \subseteq B \text{ and } E \text{ is finite}\}$$

defines a basis of A, where all element of B^* are compact.

Lemma 4 compact$(A) = B^*$.

Proof Clearly $B^* \subseteq$ *compact*(A), we now show that the reverse inclusion also holds. Let x be any compact element of A. According to lemma 3,

$$x = \bigsqcup\{d^\# \mid d^\# \subseteq x\} = \bigsqcup\{u \mid u \subseteq x \text{ and } u \in B^*\}.$$

Since x is compact, there is some $u \in B^*$ s.t. $x \subseteq u$. Moreover since $u \subseteq x$, we have $x = u \in B^*$. □

Lemmas 3 and 4 together prove theorem 1. □

3.2 Abstract Frames

In order to define an abstract semantics of PCF, we need to set up a collection of abstract domains $\{A_\sigma\}$ for each type frame $\{D_\sigma\}$. Such a collection of abstract domains will be called an abstract frame.

Definition 5 Let $\{D_\sigma\}$ be a type frame for PCF, where the base domains are D_{Bool}, D_{ι_1}, ... Given wpd's A_{Bool}, A_{ι_1}, ..., we define an *abstract frame* to be $\{A_\sigma\}$, where $A_{\sigma\to\tau} = [A_\sigma \to A_\tau]$, the set of all continuous functions from A_σ to A_τ.

Example 3 For the purpose of strictness analysis, let us abstract each base domain, such as int, into two points $\{0, 1\}$ with the usual meaning. This induces an abstract frame according to our definition. As an example, the abstract domain $A_{\mathsf{int}\to\mathsf{int}} = [A_{\mathsf{int}} \to A_{\mathsf{int}}]$ has three functions (represented by their graphs): $\{0 \mapsto 0, 1 \mapsto 0\}$, $\{0 \mapsto 0, 1 \mapsto 1\}$ and $\{0 \mapsto 1, 1 \mapsto 1\}$, with the obvious ordering among them.

In view of theorem 1, the basic results of domain theory gives us the following

Theorem 2 *Every abstract domain A_σ is an algebraic lattice.*

The induced structure on $A_{\sigma\to\tau}$ is defined by the ordering

$$f \subseteq g \quad \text{if } \forall\, x \in A_\sigma, \quad f(x) \subseteq g(x),$$

and the operations

$$(\bigsqcap F)(x) = \bigsqcap\{f(x) \mid f \in F\},$$
$$(\bigsqcup F)(x) = \bigsqcup\{f(x) \mid f \in F\}.$$

In case F is directed, we also have

$$(\bigsqcup F)(x) = (\bigcup F)(x) = \bigcup\{f(x) \mid f \in F\}.$$

Note that at a function type, $A_{\sigma\to\tau}$ is not a wpd of $D_{\sigma\to\tau}$ as elements of the former are not ideals of the latter, although it has exactly the structure required of wpd's in view of the equations above. In fact, we can define the following wpd's inductively

(i) $\mathring{A}_\iota = A_\iota$,

(ii) $\mathring{A}_{\sigma\to\tau} = \{h \in D_{\sigma\to\tau} \mid d \in \mathring{A}_\sigma \implies h(d) \in \mathring{A}_\tau\}$,

and show that $\mathring{A}_{\sigma\to\tau}$ is isomorphic to $A_{\sigma\to\tau}$.

4 Abstraction and Its Safety

Given a type frame $\{D_\sigma\}$ and an abstract frame $\{A_\sigma\}$, we need a way of relating elements in A_σ to elements in D_σ as the basis for relating an abstract semantics to the concrete semantics. Usually, the desired relationship is one of safety, defined inductively on the type as follows:

Definition 6 (i) $a \in A_{\iota_1}$ is *safe for* $d \in D_{\iota_1}$ if $d \in a$.

(ii) $f \in A_{\sigma\to\tau}$ is *safe for* $h \in D_{\sigma\to\tau}$ if $a \in A_\sigma$ is safe for $d \in D_\sigma \Rightarrow f(a)$ is safe for $h(d)$.

Example 4 Consider $+ \in \mathsf{int} \to \mathsf{int} \to \mathsf{int}$. Suppose we have a safe abstract function $\hat{+}$ for $+$, it means that if $x \hat{+} y = z$, then $m \in x$ and $n \in y \implies m+n \in z$. Thus safety corresponds to our intuitive notion of the correctness of abstraction. In the context of strictness analysis, the equation $0\hat{+}1 = 0$ means $\bot + m = \bot$ for all $m \in \mathsf{int}$, i.e., $+$ is strict in its first argument.

The following technical lemmas will be required in Section 5.

Lemma 5 *If a is safe for d and a' is safe for d' then $a \sqcup a'$ is safe for d and d'.*

We now show that safety is an order-preserving property on cpo's, i.e.,

Lemma 6 *Let $E \subseteq D_\sigma$, $X \subseteq A_\sigma$ be directed. If for every $d \in E$ there is some $x \in X$ s.t. x is safe for d, then $\bigsqcup X$ is safe for $\bigsqcup E$.*

Proof By induction on the type.

$\sigma = \iota_1$. If x is safe for d, then $d \in x$. Therefore for every $d \in E$ there is some $x \in X$ s.t. $d \in x$. It then follows that $E \subseteq \bigcup X = \bigsqcup X$. Since $\bigsqcup X$ is an ideal, $\bigsqcup E \in \bigsqcup X$, therefore $\bigsqcup X$ is safe for $\bigsqcup E$.

$\sigma = \tau \to \tau'$. For clarity, let us substitute F for X and H for E. Let $x \in A_\tau$ and $d \in D_\tau$ be s.t. x is safe for d. Consider the equations

$$(\bigsqcup F)(x) = \bigsqcup\{f(x) \mid f \in F\}$$
$$(\bigsqcup H)(d) = \bigsqcup\{h(d) \mid h \in H\}.$$

For every $h(d) \in \{h(d) \mid h \in H\}$, due to the existence of an abstract function $f \in F$ safe for h, there is an element $f(x) \in \{f(x) \mid f \in F\}$ safe for $h(d)$. Thus by the induction hypothesis $\bigsqcup\{f(x) \mid f \in F\}$ is safe for $\bigsqcup\{h(d) \mid h \in H\}$. By definition, this means $\bigsqcup F$ is safe for $\bigsqcup H$. □

Since A_σ is a complete lattice, every concrete $d \in D_\sigma$ has the safe abstraction $\top_{A_\sigma}$. However, one is clearly more interested in $\bigcap\{x \mid x \in A_\sigma \text{ is safe for } d\}$. The latter is the least element in A_σ that is safe for d. Note that at base types This element is exactly $d^\#$, therefore we extend the definition of $^\#$ to all types:

Definition 7 Let $d \in D_\sigma$. Define

$$d^\# = \bigsqcap\{x \mid x \in A \text{ and } x \text{ is safe for } d\}.$$

Remark 2 A more constructive way of extending $^\#$ to function types is to define for $h \in D_{\sigma\to\tau}$,

$$h^\#(x) = \bigsqcup\{h(d) \mid x \text{ is safe for } d\}.$$

We can actually show that the two definition gives the same abstract function due to the algebraicity of A. In case A is not algebraic, the latter definition may give a monotonic but discontinuous function strictly below that given by the former in terms of the ordering $\subseteq$.

5 Abstract Semantics

We now define the abstract interpretation $\mathcal{A}$ which maps an expression $e \in Exp$ to a value in $\{A_\sigma\}$, where the frame $\{A_\sigma\}$ is derived from the $\{D_\sigma\}$ according to the conditions prescribed in Section 3. Our principal aim is to prove that the abstract interpretation is safe whenever the constants are interpreted safely and the abstract environment maps variables to safe values. We will then define a method for comparing different abstract semantics and characterize a least (i.e., most informative) abstract semantics.

5.1 Semantics

We first require an interpretation of the constants $\mathcal{A}_c$: Const $\to \bigcup\{A_\sigma\}$. This must include interpretations of all of the constants of PCF including $\{tt, ff, if, Y\}$. In general we will only be interested in safe interpretations of constants.

Definition 8 An abstract interpretation of constants $\mathcal{A}_c$ is safe for a standard interpretation $\mathcal{S}_c$ if for all constants c, $\mathcal{A}_c[\![c]\!]$ is safe for $\mathcal{S}_c[\![c]\!]$.

Example 5 Let $\{A_\sigma\}$ be an abstract frame derived from $\{D_\sigma\}$. Define $\mathcal{A}_{c\#}$ by

$$\begin{aligned}
\mathcal{A}_{c\#}[\![tt]\!] &= \bigcap\{b \in A_{Bool} \mid tt \in b\} \\
\mathcal{A}_{c\#}[\![ff]\!] &= \bigcap\{b \in A_{Bool} \mid ff \in b\} \\
\mathcal{A}_{c\#}[\![if]\!]b\ a\ a' &= \begin{cases} \bot_{A_\sigma} & \text{if } b = \bot_{A_{Bool}}, \\ a & \text{if } b = \{tt\}_\bot, \\ a' & \text{if } b = \{ff\}_\bot, \\ a \sqcup a' & \text{otherwise.} \end{cases} \\
\mathcal{A}_{c\#}[\![Y]\!](h) &= \bigsqcup_{n\geq 0} h^n(\bot)
\end{aligned}$$

Proposition 1 *$\mathcal{A}_{c\#}$ is safe for $\mathcal{S}_c$.*

Proof The first two cases are obvious and the third follows immediately from lemma 5. That

$$\mathcal{A}_{c\#}[\![Y]\!](h) = \bigsqcup_{n\geq 0} h^n(\bot)$$

is safe for

$$\mathcal{S}_c[\![Y]\!](f) = \bigsqcup_{n\geq 0} f^n(\bot),$$

follows from lemma 6 and the fact that $\bot_{A_\sigma}$ is safe for $\bot_{D_\sigma}$. □

Abstract environments are defined analogously to the definition of concrete environments given in Section 2.

$$\eta \in Env_A = Var \to \bigcup\{A_\sigma\}$$

$$\eta[a/x][\![y]\!] = \begin{cases} a & x \equiv y, \\ \eta[\![y]\!] & \text{otherwise.} \end{cases}$$

Definition 9 An abstract environment η is safe for a concrete environment ρ if $\mathrm{Dom}(\rho) \subseteq \mathrm{Dom}(\eta)$ and $\forall x \in \mathrm{Dom}(\rho)$, $\eta(x)$ is safe for $\rho(x)$.

Given an interpretation of constants $\mathcal{A}_c$ and an environment η, the abstract semantics is then given by the valuation $\mathcal{A} : Exp \to Env_A \to \bigcup\{A_\sigma\}$. Its definition is schematically identical to the definition of $\mathcal{S}$ given in Section 2.

$$\begin{aligned}
\mathcal{A}[\![c]\!]\eta &= \mathcal{A}_c[\![c]\!] \\
\mathcal{A}[\![x]\!]\eta &= \eta[\![x]\!] \\
\mathcal{A}[\![e\ e']\!]\eta &= \mathcal{A}[\![e]\!]\eta(\mathcal{A}[\![e']\!]\eta) \\
\mathcal{A}[\![\lambda x.e]\!]\eta &= \lambda a.\mathcal{A}[\![e]\!](\eta[a/x])
\end{aligned}$$

By theorem 2 the above semantics is well-defined in the sense that every term of type σ is assigned a meaning in $\bigcup\{A_\sigma\}$. It remains to establish the connection between the abstract semantics and the standard semantics defined in Section 2. The essential connection is given in the following theorem.

Theorem 3 *Let $\mathcal{A}_c$ be any safe interpretation for $\mathcal{S}_c$. Then for all e, ρ and η safe for ρ, $\mathcal{A}[\![e]\!]\eta$ is safe for $\mathcal{S}[\![e]\!]\rho$.*

Proof By induction on e. □

Example 6 Let us take the abstract frame of example 3. Consider the function

$$\mathrm{Apply}_{(\mathsf{int}\to\mathsf{int})\to\mathsf{int}\to\mathsf{int}} = \lambda f x.fx.$$

Below is the table for the graph of Apply in the abstract domain $A_{(\mathsf{int}\to\mathsf{int})\to\mathsf{int}\to\mathsf{int}}$:

	$\lambda x.0$	$\lambda x.x$	$\lambda x.1$
0	0	0	1
1	0	1	1

This gives the desired strictness information: Apply is strict in the first argument, but not in the second (actually, it is strict in the second argument when its first argument is a strict function).

5.2 Relating Abstract Interpretations

We would now like to compare two abstract semantics $\mathcal{A}$ and $\mathcal{A}'$ over an abstract frame $\{A_\sigma\}$ which differ in their interpretations of constants $\mathcal{A}_c$ and $\mathcal{A}_{c'}$ and possibly in their environments η and η'.

Definition 10 (i) Define the relations $\preceq$ (same notation) such that $\mathcal{A}_c \preceq \mathcal{A}_{c'}$ iff $\forall c \in Const$, $\mathcal{A}_c[\![c]\!] \sqsubseteq \mathcal{A}_{c'}[\![c]\!]$, $\eta \preceq \eta'$ iff $\forall x$, $\eta(x) \sqsubseteq \eta'(x)$ and $\mathcal{A} \preceq \mathcal{A}'$ iff for all e, $\mathcal{A}[\![e]\!]\eta \sqsubseteq \mathcal{A}'[\![e]\!]\eta'$.

(ii) Define $\mathcal{A}_c \simeq \mathcal{A}_{c'}$ if $\mathcal{A}_c \preceq \mathcal{A}_{c'}$ and $\mathcal{A}_{c'} \preceq \mathcal{A}_c$ and likewise for η and $\mathcal{A}$.

The following proposition confirms that the abstract semantics is determined by its interpretation of constants.

Proposition 2 $\mathcal{A}_c \preceq \mathcal{A}_{c'} \wedge \eta \preceq \eta' \Rightarrow \mathcal{A} \preceq \mathcal{A}'$.

Proof By induction on e. □

Let $\mathcal{A}_{c\#}$ be as in example 5 and let $\mathcal{A}^\#$ denote the abstract semantics obtained using $\mathcal{A}_{c\#}$. It can be shown that this is the least semantics which is safe for the standard semantics.

Example 7 Let $\mathcal{A}_{c'}$ be the interpretation of constant obtained from $\mathcal{A}_{c\#}$ by replacing the interpretation of *if* with

$$\mathcal{A}_{c'}[\![if\ b\ a\ a']\!] = a \sqcup a'.$$

Then $\mathcal{A}'$ is clearly correct in the sense of being safe for $\mathcal{S}$. However, it is also true that $\mathcal{A}^\# \preceq \mathcal{A}'$.

Unfortunately the least (i.e., most informative) abstract semantics is, in general, uncomputable as leastness is not preserved under composition (i.e., $(f \circ g)^\# \sqsubseteq (f^\# \circ g^\#)$). This kind of information loss can be ameliorated by *case analysis* which we discuss in the next section.

6 Extensions to the Basic Scheme

Data Structures

The basic scheme for abstract interpretation as presented in Sections 2 – 5 can be easily extended to accommodate non-recursive datatypes. Let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be types, the expressions $\sigma_1 + \cdots + \sigma_n$ and $\sigma_1 \times \cdots \times \sigma_n$ then denote respectively the n-ary (coalesced) sum and (cartesian) product types for the concrete domain:

$$D_{\sigma_1+\cdots+\sigma_n} = D_{\sigma_1} + \cdots + D_{\sigma_n}$$
$$D_{\sigma_1\times\cdots\times\sigma_n} = D_{\sigma_1} \times \cdots \times D_{\sigma_n}.$$

Abstract domains for sum and product types can be derived inductively:

$$A_{\sigma_1+\cdots+\sigma_n} = A_{\sigma_1} \times \cdots \times A_{\sigma_n}$$
$$A_{\sigma_1\times\cdots\times\sigma_n} = A_{\sigma_1} \times \cdots \times A_{\sigma_n}.$$

Note that the abstract domain for sum becomes a product, since otherwise an abstract value can not contain elements from both A_{σ_i} and A_{σ_j} for distinct i and j.

For recursive data types the abstract domains derived according the equations above will be infinite. Therefore, various methods of flattening the domain may be useful. For example, in [Wad86, Hug88] a recursive domain of lists is abstracted to a four point domain

$$\bot \sqsubseteq \infty \sqsubseteq \bot_\in \sqsubseteq \top.$$

Here, $\bot$ denotes $\{\bot\}$, ∞ denotes the set of all lists not ending in *nil*, $\bot_\in$ denotes the set of lists containing at least one $\bot$ and $\top$ is the set of all lists. It is clear that this domain of coalesced elements is itself a weak powerdomain and thus the results developed in this paper ensure the correctness of the analysis.

Case Analysis

Let $f : \sigma \to \tau$ be a function and $f^\#$ be an abstract function safe for f such as obtained by our previous construction. Given an abstract value u, suppose it can be decomposed into $u = u_1 \sqcup \cdots \sqcup u_n$, where $u_i, 1 \le i \le n$ are abstract values in the same domain. we can show that $\bigsqcup_{i=1}^n f^\#(u_i)$ is also safe for $f(u)$. This is essentially the *case analysis* proposed in [SW77].

Case analysis is most useful for tightening up the interpretation of conditionals. Given the abstract domain $\{\bot, \mathsf{zero}, \mathsf{nonzero}, \mathsf{nat}\}$ where $\mathsf{zero} = \{0\}_\bot$, $\mathsf{nonzero} = \{1, 2, \ldots\}_\bot$ and $\mathsf{nat} = \{0, 1, \ldots\}_\bot$, consider the program

$$\mathrm{fac}\ x \equiv \text{if } x = 0 \text{ then } 1 \text{ else } x * \mathrm{fac}(n-1),$$

Since fac always computes a nonzero value, we would like to have $\mathrm{fac}^\#(\mathsf{nat}) = \mathsf{nonzero}$. However, this is impossible under our simple scheme since it gives

$$\mathrm{fac}^\#\ \mathsf{nat} = \quad (1)$$
$$\mathsf{nonzero} \sqcup \mathsf{nat} *^\# \mathrm{fac}^\#(\mathsf{nat} -^\# \mathsf{nonzero}),$$

Using case analysis, let us decompose

$$\mathsf{nat} = \mathsf{zero} \sqcup \mathsf{nonzero},$$

thus obtaining the following equation

$$\mathrm{fac}^\#\ \mathsf{nat} = \quad (2)$$
$$\mathsf{nonzero} \sqcup \mathsf{nonzero} *^\# \mathrm{fac}^\#(\mathsf{nonzero} -^\# \mathsf{nonzero}),$$

which gives us the desired result. The improvement obtained by case analysis is easily seen by comparing equations (1) and (2).

7 Related Work

Our approach has been strongly influenced by the general framework developed by Shamir and Wadge in [SW77] — our conditions for weak powerdomains are essentially derived from their conditions for *extended domains*. There are several differences. First, as a technical matter, Shamir-Wadge extended domains are not directed complete and thus there are continuous functions over concrete domain which do not have continuous abstractions over the extended domain. In our setting, any continuous concrete function is guaranteed to have a continuous abstraction. Second, every Shamir-Wadge extended domain contains an isomorphic image of its concrete domain. This is not required in our setting. Finally, their analysis applies to first-order functions in an untyped setting while ours treats higher-order functions in the simply typed frame. Shamir and Wadge also emphasized the tightness of abstract functions to a greater extent than we have done here.

Ideals ordered by inclusion also figure prominently in the ideal model of polymorphic types presented in [MPS86]. In that setting, each type term σ denotes an ideal in the Hoare powerdomain induced by the concrete domain D_∞. A syntactic inference algorithm is presented which infers some type σ for any term e. The soundness of the algorithm is then proved by showing that the denotation of the term is always contained in the ideal denoted by the inferred type term. Our approach suggests generating a much sparser weak powerdomain based on the domain constructions.

Along this line, Ernoult and Mycroft [EM91] have recently proposed *uniform ideals* to model strictness analysis. These are subsets of the Hoare powerdomain tailored at base type for strictness analysis (i.e., containing $\{\bot\}$ and D) and closed under $\times$ and $\boxed{\to}$ (see [MPS86]). Uniform ideals are shown to be isomorphic to the Burn-Hankin-Abramsky *strictness types* developed in [BHA86]. This isomorphism is a special case of that cited in Section 3 between our abstract domains of higher type, $A_{\sigma\to\tau}$ and the weak powerdomain $\mathring{A}_{\sigma\to\tau}$. Thus, uniform ideals can be seen as the special case of weak powerdomains suitable for strictness analysis. While our work has been more general in this sense, it does not stress effectiveness to the same extent.

Conclusions and Future Work

We have introduced a new class of semantic domains, which are intended to serve as value spaces for abstract semantics. These domains are derived under fairly general conditions directly from concrete domains. In this paper we applied them to the analysis of PCF and proved several theorems which ensure the correctness of abstract computations.

The domain theoretic approach to abstract interpretation is usually very costly, especially on higher types, as the computation constructs the graph of the function (although some efficient algorithms have been proposed, see for example [JC87, JM86, HH91]), even though most of the information there is not needed. A computationally more effective approach is type inference [KM89, Jen91]. The semantic domains presented here provide an explicit link between the concrete and abstract domains which was left implicit in [Jen91]. It is then natural to consider what logical systems arise from the constructions developed here. We expect that the general theory developed in [Abr91] is relevant.

Finally, we believe that the approach presented here could be profitably applied to the static analysis of untyped languages such as LISP. The essential technique used in this paper, that of constructing abstract domains following the inductive structure of the concrete domains, is applicable to the untyped setting as well, where domains are defined as limits.

Acknowledgements

The authors wish to thank Mitch Wand for helpful feedback on this work.

References

[Abr86] S. Abramsky. Strictness analysis and polymorphic invariance. In *Programs as Data Objects (H. Ganzinger and N. Jones editors)*, pages 1–23. Springer-Verlag LNCS Vol. 217, 1986.

[Abr90] S. Abramsky. Abstract interpretation, logical relations and kan extensions. *Journal of Logic and Computation*, 1, 1990.

[Abr91] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[AJ91] S. Abramsky and T. P. Jensen. A relational approach to strictness analysis for higher-order polymorphic functions. In *Proceedings of the Eighteenth ACM Symposium on Principles of Programming Languages*, pages 49–54, 1991.

[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.

[EM91] C Ernoult and A Mycroft. Uniform ideals and strictness analysis. In *Proceedings of ICALP'91*. Springer-Verlag Lecture Notes in Computer Science, 1991.

[HH91] S. Hunt and C. Hankin. Fixed points and frontiers: A new perspective. *Journal of Functional Programming*, 1:91–120, 1991.

[HS91] S. Hunt and D. Sands. Binding time analysis: A new perspective. In *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 154–165, 1991.

[Hug88] J. Hughes. Abstract interpretation of first-order polymorphic functions. Technical Report 89/R4, University of Glasgow, Department of Computing Science, 1988.

[JC87] S. P. Jones and C. Clack. Finding fixpoints in abstract interpretation. In *Abstract Interpretation of Declarative Languages*, pages 246–265. Ellis Horwood Limited, 1987.

[Jen91] T. P. Jensen. Strictness analysis in logical form. In *Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture*, pages 352–366, 1991.

[JM86] N. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs. In *Proceedings of the Thirteenth ACM Symposium on Principles of Programming Languages*, pages 296–306, 1986.

[KM89] T. Kuo and P. Mishra. Strictness analysis: A new perspective based on type inference. In *Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture*, pages 260–272, 1989.

[MJ86] A. Mycroft and N. Jones. A relational framework for abstract interpretation. In *Programs as Data Objects (H. Ganzinger and N. Jones editors)*, pages 156–171. Springer-Verlag LNCS Vol. 217, 1986.

[MN83] A. Mycroft and F. Nielson. Strong abstract interpretation using power domains (extended abstract). *International Colloquim on Automata, Languages and Programming, Springer-Verlag, Lecture Notes in Computer Science*, 154:336–547, 1983.

[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.

[Nie84] F. Nielson. *Abstract Interpretation of Denotational Definitions.* PhD thesis, University of Edinburgh, 1984.

[Nie89] F. Nielson. Two-level semantics and abstract interpretation. *Theoretical Computer Science*, 69:117–241, 1989.

[Plo77] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[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.

[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.