Backward Analysis for Higher–Order Functions Using Inverse Images

Tyng–Ruey Chuang
Benjamin Goldberg
Department of Computer Science
New York University*

February 1991

Abstract

We propose a method for performing backward analysis on higher–order functional programming languages based on computing inverse images of functions over abstract domains. This method can be viewed as abstract interpretation done backward. Given an abstract semantics which supports forward analysis, we can transform it into an abstract semantics which performs backward analysis. We show that if the original abstract semantics is correct and computable, then the transformed version of the abstract semantics is also correct and computable.

More specifically, given a forward abstract semantics of a higher–order functional language which is expressed in terms of Scott–closed powerdomains, we derive an backward abstraction semantics which is expressed in terms of Scott–open powerdomains. The derivation is shown to be correct and the relationships between forward analysis and backward analysis is established.

We apply this method to the classic strictness analysis in functional languages and obtain promising results. We show that the time complexity of inverse image based backward analysis is not much worse than the forward analysis from which it is derived. We then compare this work with previous works of Wadler and Hughes [17], Hughes [11], and Burn [5], showing that some special restrictions and constructions in previous works have natural interpretation in the Scott–closed/Scott–open powerdomain framework. A brief outline of applying the inverse image method to other backward semantics analysis is also given.

*The authors' address: Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA. E–mail: chuang@cs.nyu.edu, goldberg@cs.nyu.edu. This paper is available as Technical Report 620 from the Department of Computer Science, New York University, at the above address. An electronic version is also available from the ftp site cs.nyu.edu. This research has been supported, in part, by the National Science Foundation (#CCR–8909634) and DARPA (DARPA/ONR #N00014–91–J1472).

1 Introduction

Burn, Hankin, and Abramsky [6] use the Scott–closed powerdomain[1] construction to give an abstract semantics for a non–strict higher–order functional language which denotes the language's strictness property. Their construction can be regarded as an approximation to the concrete Scott–closed powerdomain semantics of the functional language. Their approximation yields an abstract semantics that is computable and safe. That is, their strictness analysis will always terminate for well–typed programs and will not give incorrect answers (although the answers may be approximations). Their method is a *forward* analysis since definedness of the result of a function call is inferred from the definedness of the arguments.

Backward methods for strictness analysis have been investigated by Wadler and Hughes [17], Hughes [11], Davis and Wadler [8] and Burn [5], which emphasize strictness analysis on functional languages with non–flat data types. Their methods are based on the *context* (or *projection*, a restricted notation of context) concept. A computation result is in a particular context (or, subject to a particular projection) if we know that the result will be used in a particular way. For example, if a computation result is to be discarded immediately after it is computed, then the result is said to be in the *ABSENT* context. Also, if we know that a computation result is in the *ABSENT* context, then we know that we need not carry out the computation at all. Context–based backward analyses infer the contexts of the arguments in a function call from the context of the function call. However, the formulation of a context–based backward analysis is quite complicated, especially in cases involving higher–order functions. Hughes [11] observes that context–based backward analysis for higher–order functions is complicated because a context–based forward analysis is also needed to obtain satisfactory answers.

Dybjer [9] uses the Scott–open powerdomain to formulate an inverse image analysis for non–strict first–order functions regarding their strictness property. Whether this method can be applied to higher–order functions remains an open question. Furthermore, it is not clear that his method can be automated.

We propose here a general method for inverse image analysis for higher–order functions. This method is also based on the Scott–open powerdomain construction. In fact, our method is inspired by the work of Burn, Hankin, and Abramsky [6] and Dybjer [9]. That is, for each function f, we will derive the Scott–open powerdomain semantics of "f^{-1}", $\mathcal{P}_O(f^{-1})$, from f's Scott–closed powerdomain semantics, $\mathcal{P}_C(f)$. If $\mathcal{P}'_C(f)$ is an approximation of $\mathcal{P}_C(f)$ which is computable and safe, then our derivation, $\mathcal{P}'_O(f^{-1})$, is also computable and safe. Since Burn, Hankin, and Abramsky's method is applicable to higher–order functions, our approach provides a method for backward analysis which is also applicable to higher–order functions. By incorporating the strictness analysis for non–flat data types by Wadler [16] and Burn [5], our method can also be applied to functional programs with non–flat data types.

After representing the idea and giving some examples in strictness analysis, we show the time complexity of doing inverse image based backward analysis and compare our work with previous works. We show that the formulation of a backward analysis is straightforward if a forward analysis is given. We also argue that answering a query in the backward analysis costs not much more than obtaining the forward analysis. A brief outline of applying inverse image analysis to other semantic analysis is also given.

2 The Main Theorem

We use bounded–complete ω–algebraic cpos as the semantic domains in our discussion of inverse image analysis. The definition and properties of bounded–complete ω–algebraic cpos can be found in Appendix A.

Scott–closed and Scott–open sets are defined as follows:

Definition 2.1 Let D be a domain. A subset $X \subseteq D$ is *Scott–closed* if

[1]The Hoare powerdomain construction, to be precise, is used in their development.

1. If $x \in X$ and $y \sqsubseteq_D x$, then $y \in X$;
2. If $Y \subseteq X$ and Y is directed, then $\bigsqcup Y \in X$.

□

We will call the condition in clause 1 of Definition 2.1 the *downward closure* property. A subset is *downward closed* if it has the downward closure property. If X is a Scott–closed subset of a domain D, then we say X *is closed in* D for short.

Definition 2.2 Let D be a domain. Define poset $\mathcal{P}_C(D)$ by

1. $\mathcal{P}_C(D) = \{S \mid S \subseteq D, S \text{ is Scott–closed}\}$;
2. $S \sqsubseteq_{\mathcal{P}_C(D)} T$ iff $S \subseteq T$.

□

We can also define the poset $\mathcal{P}_O(D)$ to be the set of all the Scott–open subsets of domain D.

Definition 2.3 Let D be a domain. A subset $X \subseteq D$ is *Scott–open* if

1. If $x \in X$ and $x \sqsubseteq_D y$, then $y \in X$;
2. If $Y \subseteq D$, Y is directed, and $\bigsqcup Y \in X$, then $Y \cap X \neq \emptyset$.

□

Again, we will call the condition in clause 1 of Definition 2.3 the *upward closure* property. A subset is *upward closed* if it has the upward closure property. If X is a Scott–open subset of a domain D, then we say X *is open in* D for short.

Definition 2.4 Let D be a domain. Define poset $\mathcal{P}_O(D)$ by

1. $\mathcal{P}_O(D) = \{S \mid S \subseteq D, S \text{ is Scott–open}\}$;
2. $S \sqsubseteq_{\mathcal{P}_O(D)} T$ iff $S \supseteq T$.

□

In Appendix A, we show that both $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are domains if D is a domain. In fact, $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are complete lattices and they are isomorphic to each other. Also note that the set union operator $\cup$ is the binary least upper bound operator in $\mathcal{P}_C(D)$, while the set intersection operator $\cap$ is the binary least upper bound operator in $\mathcal{P}_O(D)$.

An element d in a domain D has a certain property if d belongs to a particular subset of D. We will only use Scott–closed subsets and Scott–open subsets of a domain to describe the properties of the elements in the domain. This restriction may sound artificial, but is quite reasonable. A Scott–closed subset of a domain D can be viewed as a way to describe how undefined an element (or a set of elements) in D is; while a Scott–open subset of D describes how defined an element (or a set of elements) is.

We give an intuitive example here. Let D be a domain, and let $f \in D \rightarrow D$ be a continuous function from D to D. In strictness analysis, we want to know whether the function application $f\ \bot_D$ will result in $\bot_D$ or not. If we use Scott–closed sets, then we can describe strictness in the following way: If the argument x is in the Scott–closed set $\{\bot_D\}$, then the result of $f\ x$ is in the Scott–closed set $\{\bot_D\}$. The connection between strictness and Scott–closed powerdomains is well described in Burn, Hankin, and Abramsky [6].

Scott–open powerdomains can be used to perform strictness analysis in the backward direction. That is, strictness can be restated as follows: If the result of the function application $f\ x$ must be in the Scott–open set $D - \{\bot_D\}$, then x must be in the Scott–open set $D - \{\bot_D\}$.

If the argument must be defined, then a call–by–need to call–by–value transformation can be performed on function application. This backward reasoning can be viewed as partial evaluation with respect to the result, not, as it is usually done, with respect to the argument.

We will describe in this paper how a backward strictness analysis can be realized by an inverse image analysis, with the aid of an existing forward strictness analysis. The connection between backward strictness analysis and inverse image analysis is first described by Dybjer [9], although it is only done on first–order functions. Our contribution in this paper is to extend it into higher–order functions and, in our opinion, to do it in a cleaner way. We also relate forward analyses and backward analyses under the framework of Scott–closed/Scott–open powerdomains.

Recall that, given a domain D, we have defined its Scott–closed powerdomain, $\mathcal{P}_C(D)$, and Scott–open powerdomain, $\mathcal{P}_O(D)$. The next step is to define functions between these powerdomains, based on continuous functions over the original domains.

Fact 2.5 Let D_α and D_β be domains and $f \in D_\alpha \to D_\beta$. Then,

1. $\{x \mid x \in D_\alpha, fx \in B\}$ is closed in D_α if B is closed in D_β;
2. $\{x \mid x \in D_\alpha, fx \in B\}$ is open in D_α if B is open in D_β.

$\square$

Corollary 2.6 Let D_α and D_β be domains. Let $f \in D_\alpha \to D_\beta$. Then,

1. $\{x \mid x \in D_\alpha, f\ x \sqsubseteq_{D_\beta} y\}$ is closed in D_α for each $y \in D_\beta$,
2. $\{x \mid x \in D_\alpha, f\ x \not\sqsubseteq_{D_\beta} y\}$ is open in D_α for each $y \in D_\beta$.

$\square$

Definition 2.7 Let D_α and D_β be domains. Let $f \in D_\alpha \to D_\beta$.
Define $\mathcal{P}_C(f) \in \mathcal{P}_C(D_\alpha) \to \mathcal{P}_C(D_\beta)$ and $\mathcal{P}_O(f^{-1}) \in \mathcal{P}_O(D_\beta) \to \mathcal{P}_O(D_\alpha)$ by

1. $(\mathcal{P}_C(f))\ X = \{f\ x \mid x \in D_\alpha, x \in X\}^c$, where $X \in \mathcal{P}_C(D_\alpha)$;
2. $(\mathcal{P}_O(f^{-1}))\ Y = \{x \mid x \in D_\alpha, f\ x \in Y\}$, where $Y \in \mathcal{P}_O(D_\beta)$.

$\square$

For a subset W of a domain D, W^c is the least Scott–closed subset of D which contains W (see Definition A.41). We can say that $\mathcal{P}_C(f)$ defines an image function of f, based on Scott–closed powerdomains; while $\mathcal{P}_O(f^{-1})$ defines an inverse image function of f, based on Scott–open powerdomains. By definition, it can be shown that $\mathcal{P}_C(f)$ is $\perp$–reflexive. That is, $(\mathcal{P}_C(f))\ A = \emptyset$ if and only if $A = \emptyset$.

Lemma 2.8 Let D_α and D_β be domains. Then both $\mathcal{P}_C(f)$ and $\mathcal{P}_O(f^{-1})$ are well–defined and continuous for every continuous function $f \in D_\alpha \to D_\beta$. $\square$

Proof. See Appendix B. $\diamond$

Furthermore, it can be shown that $\mathcal{P}_C(f)$ is *additive* if f is continuous. That is, $(\mathcal{P}_C(f))\ (X \sqcup Y) = ((\mathcal{P}_C(f))\ X) \sqcup ((\mathcal{P}_C(f))\ Y)$ for all $X, Y \in \mathcal{P}_C(D_\alpha)$. Note that additivity is a stronger condition than continuity.

Lemma 2.9 Let D_α and D_β be domains, $f \in D_\alpha \to D_\beta$, and $B \in \mathcal{P}_C(D_\beta)$. Then

$$\{x \mid x \in D_\alpha, fx \in B\} = \bigsqcup \{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))\ X \sqsubseteq_{\mathcal{P}_C(D_\beta)} B\}.$$

$\square$

Proof. See Appendix B. $\diamond$

We now prove the following main theorem.

Theorem 2.10 Let D_α and D_β be domains. Let $B \in \mathcal{P}_O(D_\beta)$ and $f \in D_\alpha \to D_\beta$. Then

$$(\mathcal{P}_O(f^{-1}))\ B = \overline{\bigsqcup \{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))\ X \sqsubseteq_{\mathcal{P}_C(D_\beta)} \overline{B}\}}.$$

$\square$

PROOF.

$$\begin{array}{rll} & (\mathcal{P}_O(f^{-1}))\ B & \\ = & \{x \mid x \in D_\alpha, f\ x \in B\} & \text{(Definition 2.7)} \\ = & \overline{\{x \mid x \in D_\alpha, f\ x \in \overline{B}\}} & \text{(Fact A.36)} \\ = & \overline{\bigsqcup\{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))\ X \sqsubseteq_{\mathcal{P}_C(D_\beta)} \overline{B}\}} & \text{(Lemma 2.9)} \end{array}$$

◇

The significant part of Theorem 2.10 is that it completely characterizes $\mathcal{P}_O(f^{-1})$ in terms of $\mathcal{P}_C(f)$, instead of in terms of function f. But from an abstract interpretation's point of view, this characterization is not very useful because $\mathcal{P}_C(f)$ in general cannot be computed in finite time and the domain $\mathcal{P}_C(D_\alpha)$ need not be finite. This makes $(\mathcal{P}_O(f^{-1}))\ B$ in general not computable in finite time.

The next step will be to give an approximation of $\mathcal{P}_O(f^{-1})$ by using an existing approximation of $\mathcal{P}_C(f)$.

The following corollary shows the relationship between the forward semantics $\mathcal{P}_C(f)$ and the backward semantics $\mathcal{P}_O(f^{-1})$.

Corollary 2.11 Let D_α and D_β be domains, $f \in D_\alpha \to D_\beta$, $A \in \mathcal{P}_C(D_\alpha)$, and $B \in \mathcal{P}_O(D_\beta)$. Then

$$\begin{array}{llll} (\mathcal{P}_O(f^{-1})) & \overline{(\mathcal{P}_C(f))A} & \sqsupseteq_{\mathcal{P}_O(D_\alpha)} & \overline{A}, \\ (\mathcal{P}_C(f)) & \overline{(\mathcal{P}_O(f^{-1}))B} & \sqsubseteq_{\mathcal{P}_C(D_\beta)} & \overline{B}. \end{array}$$

□

PROOF OUTLINE. Substitute the B in Theorem 2.10 by $\overline{(\mathcal{P}_C(f))A}$ to show that the first part is true. Also by Theorem 2.10,

$$\overline{(\mathcal{P}_O(f^{-1}))\ B} = \bigsqcup\{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))\ X \sqsubseteq_{\mathcal{P}_C(D_\beta)} \overline{B}\}.$$

Applying $\mathcal{P}_C(f)$ at both sides to show that the second part is true too. ◇

A direct consequence of the above corollary is the following.

Corollary 2.12 Let D_α and D_β be domains, $f \in D_\alpha \to D_\beta$, $A \in \mathcal{P}_C(D_\alpha)$, and $B \in \mathcal{P}_C(D_\beta)$. Then

1. if $(\mathcal{P}_O(f^{-1}))\ \overline{B} \sqsupseteq_{\mathcal{P}_O(D_\alpha)} \overline{A}$, then $(\mathcal{P}_C(f))\ A \sqsubseteq_{\mathcal{P}_C(D_\beta)} B$;
2. if $(\mathcal{P}_C(f))\ A \sqsubseteq_{\mathcal{P}_C(D_\beta)} B$, then $(\mathcal{P}_O(f^{-1}))\ \overline{B} \sqsupseteq_{\mathcal{P}_O(D_\alpha)} \overline{A}$.

□

3 An Abstract Semantics and its Inverse

In Burn, Hankin, and Abramsky [6], they develop an approximation based on the Scott–closed powerdomain construction. They use this approximation to define an abstract semantics for determining the strictness of higher–order functions. In this section, after a brief introduction to their approach and results, we show how to take their approximation and transform it to give an approximation based on the Scott–open powerdomain construction. We then use this newly defined approximation as an abstract semantics for determining strictness, only that it now works backward.

The following formulation is a little different from Burn, Hankin, and Abramsky [6] to fit the current discussion. Suppose that we are given an approximation domain $\mathcal{P'}_C(D)$ for $\mathcal{P}_C(D)$, where D is a domain, such that $\mathcal{P'}_C(D)$ is a finite complete sub–lattice of $\mathcal{P}_C(D)$. Note that both $\emptyset$ and D are in $\mathcal{P'}_C(D)$. That is, $\{\emptyset, D\} \subseteq \mathcal{P'}_C(D) \subseteq \mathcal{P}_C(D)$, $\mathcal{P'}_C(D)$ has only a finite number of elements, and for all subsets $\mathcal{X} \subseteq \mathcal{P'}_C(D)$, the least upper bound of $\mathcal{X}$ in $\mathcal{P'}_C(D)$ is the same as the least upper bound of $\mathcal{X}$ in $\mathcal{P}_C(D)$. This property also holds for the greatest lower bound. For a continuous function $f \in D_\alpha \to D_\beta$, where D_α and D_β are domains, we then give an approximation function $\mathcal{P'}_C(f)$ for $\mathcal{P}_C(f)$ such that $\mathcal{P'}_C(f)$ is computable in finite time and $(\mathcal{P'}_C(f))\ A \supseteq (\mathcal{P}_C(f))\ A$ for each $A \in \mathcal{P'}_C(D_\alpha)$.

Definition 3.13 Let D be a domain and $\mathcal{P}'_C(D)$ be a finite complete sub–lattice of $\mathcal{P}_C(D)$. The abstraction function $Abs_C \in \mathcal{P}_C(D) \rightarrow \mathcal{P}'_C(D)$ and the concretization function $Conc_C \in \mathcal{P}'_C(D) \rightarrow \mathcal{P}_C(D)$ are defined by

$$\begin{array}{rcl} Abs_C\ X & = & \bigsqcap\{Y \mid Y \in \mathcal{P}'_C(D), X \sqsubseteq_{\mathcal{P}_C(D)} Y\},\ \text{where } X \in \mathcal{P}_C(D); \\ Conc_C\ Y & = & Y,\ \text{where } Y \in \mathcal{P}'_C(D). \end{array}$$

□

An interesting consequence of the above definition is that both Abs_C and $Conc_C$ are $\perp$–reflexive and additive.

Fact 3.14 Let D be a domain. Let $\mathcal{P}'_C(D)$ be a finite complete sub–lattice of $\mathcal{P}_C(D)$; and let $X \in \mathcal{P}_C(D)$ and $X' \in \mathcal{P}'_C(D)$. Then,

$$\begin{array}{rcl} (Conc_C \circ Abs_C)\ X & \supseteq & X; \\ (Abs_C \circ Conc_C)\ X' & = & X'. \end{array}$$

□

Definition 3.15 Let D_α and D_β be domains. Let $f \in D_\alpha \rightarrow D_\beta$. Define

$$\mathcal{P}'_C(f) = Abs_C \circ \mathcal{P}_C(f) \circ Conc_C.$$

□

$\mathcal{P}'_C(f)$ is $\perp$–reflexive and additive because $Abs_C, \mathcal{P}_C(f)$, and $Conc_C$ are $\perp$–reflexive and additive.

Fact 3.16 Let D_α and D_β be domain. Let $f \in D_\alpha \rightarrow D_\beta$ and $A \in \mathcal{P}_C(D_\alpha)$. Then

$$(\mathcal{P}_C(f))\ A \subseteq (Conc_C \circ \mathcal{P}'_C(f) \circ Abs_C)\ A.$$

□

In short, $\mathcal{P}'_C(D)$ is an abstract domain for $\mathcal{P}_C(D)$ and $\mathcal{P}'_C(f)$ is an abstract interpretation of $\mathcal{P}_C(f)$. Fact 3.16 asserts that such an abstract interpretation is safe.

Suppose that $\mathcal{P}'_C$ is an approximation based on the Scott–closed powerdomain construction as defined above. We show in the following how to turn $\mathcal{P}'_C$ into $\mathcal{P}'_O$, an approximation based on the Scott–open powerdomain construction.

Definition 3.17 Let D be a domain. Define $\mathcal{P}'_O(D) = \{\overline{X} \mid X \in \mathcal{P}'_C(D)\}$. □

Definition 3.18 Let D be a domain. Define the abstraction function $Abs_O \in \mathcal{P}_O(D) \rightarrow \mathcal{P}'_O(D)$ and the concretization function $Conc_O \in \mathcal{P}'_O(D) \rightarrow \mathcal{P}_O(D)$ by

$$\begin{array}{rcl} Abs_O\ X & = & \bigsqcup\{Y \mid Y \in \mathcal{P}'_O(D), Y \sqsubseteq_{\mathcal{P}_O(D)} X\},\ \text{where } X \in \mathcal{P}_O(D); \\ Conc_O\ Y & = & Y,\ \text{where } Y \in \mathcal{P}'_O(D). \end{array}$$

□

Fact 3.19 Let D be a domain, $X \in \mathcal{P}_O(D)$, and $X' \in \mathcal{P}'_O(D)$. Then

$$\begin{array}{rcl} (Conc_O \circ Abs_O)\ X & \supseteq & X \quad \text{and} \\ (Abs_O \circ Conc_O)\ X' & = & X'. \end{array}$$

□

For each function f, we can *define* $\mathcal{P}'_O(f^{-1})$ in terms of $\mathcal{P}'_C(f)$, as we did in Theorem 2.10.

Definition 3.20 Let D_α and D_β are domains and let $f \in D_\alpha \to D_\beta$. Define

$$(\mathcal{P}'_O(f^{-1}))\ Y = \overline{\bigsqcup\{X \mid X \in \mathcal{P}'_C(D_\alpha), (\mathcal{P}'_C(f))\ X \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ \overline{Y}\}},$$

where $Y \in \mathcal{P}'_O(D_\beta)$. □

Note that, in the above definition, if $\mathcal{P}'_C(f)$ is computable in finite time, then $\mathcal{P}'_O(f^{-1})$ is computable in finite time since $\mathcal{P}'_C(D_\alpha)$ is a finite set. The following theorem states that the above definition is safe.

Theorem 3.21 Let D_α and D_β be domain. Let $f \in D_\alpha \to D_\beta$ and $B \in D_\beta$. Then

$$(\mathcal{P}_O(f^{-1}))\ B \subseteq (Conc_O \circ \mathcal{P}'_O(f^{-1}) \circ Abs_O)\ B.$$

□

Proof. We want to show that, for every $B \in \mathcal{P}_O(D_\beta)$,

$$\begin{array}{ll} & \overline{\bigsqcup\{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))X \ \sqsubseteq_{\mathcal{P}_C(D_\beta)}\ \overline{B}\}} \\ \subseteq & Conc_O\ \overline{\bigsqcup\{X \mid X \in \mathcal{P}'_C(D_\alpha), (\mathcal{P}'_C(f))\ X \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ \overline{Abs_O\ B}\}}. \end{array}$$

That is,

$$\begin{array}{ll} & \bigsqcup\{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))X \ \sqsubseteq_{\mathcal{P}_C(D_\beta)}\ \overline{B}\} \\ \supseteq & \bigsqcup\{X \mid X \in \mathcal{P}'_C(D_\alpha), (\mathcal{P}'_C(f))\ X \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ \overline{Abs_O\ B}\}. \end{array}$$

Note that $\overline{B} \supseteq \overline{Abs_O\ B}$ for every $B \in \mathcal{P}_O(D_\beta)$. Therefore, for each

$$A' \in \{X \mid X \in \mathcal{P}'_C(D_\alpha), (\mathcal{P}'_C(f))\ X \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ \overline{Abs_O\ B}\}$$

there exists a

$$A \in \{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))X \ \sqsubseteq_{\mathcal{P}_C(D_\beta)}\ \overline{B}\}$$

such that $A \supseteq A'$.

We complete the proof by observing that $\mathcal{P}'_C(D_\alpha)$ is a finite complete sub–lattice of $\mathcal{P}_C(D_\alpha)$.◇

The following corollary shows the relationship between the forward abstract semantics $\mathcal{P}'_O(f^{-1})$ and the backward abstract semantics $\mathcal{P}'_C(f)$.

Corollary 3.22 Let D_α and D_β be domains. Let $f \in D_\alpha \to D_\beta$, $A \in \mathcal{P}'_C(D_\alpha)$, and $B \in \mathcal{P}'_O(D_\beta)$. Then

$$\begin{array}{lll} (\mathcal{P}'_O(f^{-1})) & \overline{(\mathcal{P}'_C(f))\ A} & \sqsupseteq_{\mathcal{P}'_O(D_\alpha)}\ \overline{A}, \\ (\mathcal{P}'_C(f)) & \overline{(\mathcal{P}'_O(f^{-1}))\ B} & \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ \overline{B}. \end{array}$$

□

The following is a direct consequence of the above corollary.

Corollary 3.23 Let D_α and D_β be domains, $f \in D_\alpha \to D_\beta$, $A \in \mathcal{P}'_C(D_\alpha)$, and $B \in \mathcal{P}'_C(D_\beta)$. Then

1. if $(\mathcal{P}'_O(f^{-1}))\ \overline{B} \ \sqsupseteq_{\mathcal{P}'_O(D_\alpha)}\ \overline{A}$, then $(\mathcal{P}'_C(f))\ A \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ B$;
2. if $(\mathcal{P}'_C(f))\ A \ \sqsubseteq_{\mathcal{P}'_C(D_\beta)}\ B$, then $(\mathcal{P}'_O(f^{-1}))\ \overline{B} \ \sqsupseteq_{\mathcal{P}'_O(D_\alpha)}\ \overline{A}$.

□

The reader may like to compare the above corollary with the result in Burn [5, Theorem 3.1]. The following lemma describes some degenerate cases.

Lemma 3.24 Let D_α and D_β be domains. Let $f \in D_\alpha \to D_\beta$. Then,

1. $\mathcal{P}'_C(f)\ \emptyset = \emptyset$,
2. $\mathcal{P}'_C(f)\ D_\alpha \sqsubseteq_{\mathcal{P}'_C(D_\beta)} D_\beta$,
3. $\mathcal{P}'_O(f^{-1})\ \emptyset = \emptyset$,
4. $\mathcal{P}'_O(f^{-1})\ D_\beta = D_\alpha$.

□

PROOF. The proofs follow immediately from Theorem 2.10, Definition 3.15, and Definition 3.20. ◇

We now relate Burn, Hankin, and Abramsky's forward strictness analysis to a backward strictness analysis.

Fact 3.25 (Burn, Hankin, and Abramsky) Let $\mathcal{P}'_C$ be the Scott–closed powerdomain approximation of Burn, Hankin, and Abramsky. Let D_α and D_β be domain, and let $f \in D_\alpha \rightarrow D_\beta$. Then

1. $(\mathcal{P}'_C(f))\ \{\perp_{D_\alpha}\} \sqsupseteq_{\mathcal{P}'_C(D_\beta)} \{\perp_{D_\beta}\}$;
2. if $(\mathcal{P}'_C(f))\ \{\perp_{D_\alpha}\} = \{\perp_{D_\beta}\}$, then $f\ \perp_{D_\alpha} = \perp_{D_\beta}$. (That is, f is strict.)

□

Theorem 3.26 Let $\mathcal{P}'_C$ be the Scott–closed powerdomain approximation of Burn, Hankin, and Abramsky. Let $\mathcal{P}'_O$ be defined as in Definition 3.20. Let D_α and D_β be domains, and let $f \in D_\alpha \rightarrow D_\beta$. If

$$(\mathcal{P}'_O(f^{-1}))\ \overline{\{\perp_{D_\beta}\}} \sqsupseteq_{\mathcal{P}'_O(D_\alpha)} \overline{\{\perp_{D_\alpha}\}},$$

then f is strict. □

PROOF OUTLINE. By Corollary 3.23 and Fact 3.25. ◇

4 Examples

In this section we demonstrate how to use our inverse image analysis on higher–order functions. More specifically, we show how a backward strictness analysis can be done within the framework of inverse image analysis. Given a function and an expected property of its result, we infer the property of the function's arguments which will yield the expected result. For instance, if we expect a function application resulting a defined value, then we would like to use our inverse image analysis to determine whether the argument shall be defined or not. If the argument is shown to be defined, then the argument can be evaluated by call–by–value instead of call–by–need.

We start with a modified representation of Burn, Hankin, and Abramsky's strictness analysis. After showing the forward approximation scheme, $\mathcal{P}'_C$, we then reverse it into a backward approximation scheme, $\mathcal{P}'_O$, which can be used in backward strictness analysis.

Let $\langle A,\ B\rangle$ denote an element of the smashed product $\mathcal{P}'_C(D_\alpha) \otimes \mathcal{P}'_C(D_\beta)$, where $A \in \mathcal{P}'_C(D_\alpha)$, $B \in \mathcal{P}'_C(D_\beta)$, and $\mathcal{P}'_C(D)$ denoting a complete sub–lattices of the Scott–closed powerdomain $\mathcal{P}_C(D)$. Then, by the definition of smashed product, we have $\langle A,\ \emptyset\rangle = \langle \emptyset,\ B\rangle = \emptyset$ for every $A \in \mathcal{P}'_C(D_\alpha)$ and $B \in \mathcal{P}'_C(D_\beta)$. Note that, by notation, $\langle A,\ B\rangle$ represents a pair of Scott–closed subsets. However, what we have in mind is to use it to represent the Scott–closed subset of the domain $D_\alpha \times D_\beta$ such that

$$\langle A,\ B\rangle = \{\langle a,\ b\rangle \mid \langle a,\ b\rangle \in D_\alpha \times D_\beta, a \in A, b \in B\}.$$

We will overload $\langle A,\ B\rangle$ to the above denotation.

Likewise, we overload $\mathcal{P}'_C(D_\alpha) \otimes \mathcal{P}'_C(D_\beta)$ to denote the subset of the Scott–closed powerdomain $\mathcal{P}_C(D_\alpha \times D_\beta)$ such that it now includes exactly those Scott–closed subsets denoted by $\langle A,\ B\rangle$ (where $A \in \mathcal{P}'_C(D_\alpha)$ and $B \in \mathcal{P}'_C(D_\beta)$).

We run into the same notation problem in the $\perp$–reflexive additive function space $\mathcal{P}'_C(D_\alpha) \mathrel{\bullet\!\!\rightarrow} \mathcal{P}'_C(D_\beta)$. We will use

$$[A_1 \mapsto B_1,\ A_2 \mapsto B_2,\ \ldots, A_n \mapsto B_n],$$

where $A_1, A_2, \ldots, A_n \in \mathcal{P}'_C(D_\alpha)$ and $B_1, B_2, \ldots, B_n \in \mathcal{P}'_C(D_\beta)$, not only as a notation for a $\perp$–reflexive additive map from $\mathcal{P}'_C(D_\alpha)$ to $\mathcal{P}'_C(D_\beta)$, but also as a notation for the Scott–closed subset

$$\{f \mid f \in D_\alpha \rightarrow D_\beta, f\ a \in B_1 \text{ for all } a \in A_1, f\ a \in B_2 \text{ for all } a \in A_2, \ldots, \text{ and } f\ a \in B_n \text{ for all } a \in A_n\}$$

of the domain $D_\alpha \rightarrow D_\beta$.

If n is finite, it can be shown that, by the new denotation, $[A_1 \mapsto B_1, A_2 \mapsto B_2, \ldots, A_n \mapsto B_n] \in \mathcal{P}_C(D_\alpha \rightarrow D_\beta)$. For the special case of $n = 0$, it is written as $\emptyset$, which is also an element in $\mathcal{P}_C(D_\alpha \rightarrow D_\beta)$.

Likewise, $\mathcal{P}'_C(D_\alpha) \mathrel{\bullet\!\!\rightarrow} \mathcal{P}'_C(D_\beta)$ is overloaded to denote the subset of the Scott–closed powerdomain $\mathcal{P}_C(D_\alpha \rightarrow D_\beta)$ such that it now includes exactly those Scott–closed subsets denoted by the $\perp$–reflexive additive maps from $\mathcal{P}'_C(D_\alpha)$ to $\mathcal{P}'_C(D_\beta)$.

Definition 4.27 Let D be a domain. $\mathcal{P}'_C(D)$ is defined by

- if D is a primitive domain B, then
$$\mathcal{P}'_C(D) = \{B,\ \{\perp_B\},\ \emptyset\}.$$
- if $D = D_\alpha \times D_\beta$, then
$$\mathcal{P}'_C(D) = \{\bigcup \mathcal{X} \mid \mathcal{X} \subseteq \mathcal{P}'_C(D_\alpha) \otimes \mathcal{P}'_C(D_\beta)\}.$$
- if $D = D_\alpha \rightarrow D_\beta$, then
$$\mathcal{P}'_C(D) = \{\bigcup \mathcal{X} \mid \mathcal{X} \subseteq \mathcal{P}'_C(D_\alpha) \mathrel{\bullet\!\!\rightarrow} \mathcal{P}'_C(D_\beta)\}.$$

□

The definition for $\mathcal{P}'_C(D_\alpha \times D_\beta)$ and $\mathcal{P}'_C(D_\alpha \rightarrow D_\beta)$ deserves some attention. The definition is to make them each a complete sub–lattice of $\mathcal{P}_C(D_\alpha \times D_\beta)$ and $\mathcal{P}_C(D_\alpha \rightarrow D_\beta)$ respectively. If we simply define $\mathcal{P}'_C(D_\alpha \times D_\beta)$ as $\mathcal{P}'_C(D_\alpha) \otimes \mathcal{P}'_C(D_\beta)$ and $\mathcal{P}'_C(D_\alpha \rightarrow D_\beta)$ as $\mathcal{P}'_C(D_\alpha) \mathrel{\bullet\!\!\rightarrow} \mathcal{P}'_C(D_\beta)$, they will not be complete sub–lattices. Before going into the examples, recall that $\mathcal{P}'_O(D)$ is defined as $\{\overline{X} \mid X \in \mathcal{P}'_C(D)\}$for every domain D.

Example 4.28 Let N be the flat domain of all natural numbers. Then, we have

1. - $\mathcal{P}'_C(N) = \{N, \{\perp_N\}, \emptyset\}$,
   - $\mathcal{P}'_C(N \times N) = \begin{array}{lllll} \{ & \langle N, N\rangle, & \langle N, \{\perp_N\}\rangle \cup \langle\{\perp_N\}, N\rangle, & \langle N, \{\perp_N\}\rangle, & \\ & \langle\{\perp_N\}, N\rangle, & \langle\{\perp_N\}, \{\perp_N\}\rangle, & \emptyset & \}, \end{array}$
   - $\mathcal{P}'_C(N \rightarrow N) = \begin{array}{llll} \{ & [N \mapsto N, & \{\perp_N\} \mapsto N, & \emptyset \mapsto \emptyset], \\ & [N \mapsto N, & \{\perp_N\} \mapsto \{\perp_N\}, & \emptyset \mapsto \emptyset], \\ & [N \mapsto \{\perp_N\}, & \{\perp_N\} \mapsto \{\perp_N\}, & \emptyset \mapsto \emptyset], \\ & \emptyset & & \}. \end{array}$

   We will write, for example, $[N \mapsto N,\ \{\perp_N\} \mapsto N,\ \emptyset \mapsto \emptyset]$ as $\lambda X\,.\,N$ without explicitly mentioning that $X \in \mathcal{P}'_C(N)$ and this function is $\perp$–reflexive.

   Note that $\lambda X\,.\,N$ denotes the set of all the continuous functions from N to N; $\lambda X\,.\,X$ denotes the set of all the strict continuous functions from N to N; and $\lambda X\,.\,\{\perp_N\}$ denotes the set containing only the everywhere undefined function from N to N.

2. - $\mathcal{P}'_O(N) = \{\overline{N}, \overline{\{\perp_N\}}, \overline{\emptyset}\} = \{\emptyset, N - \{\perp_N\}, N\}$,

- $$\begin{array}{rclllll} \mathcal{P}'_O(N \times N) & = & \{ & \overline{\langle N, N\rangle}, & \overline{\langle N, \{\bot_N\}\rangle \cup \langle \{\bot_N\}, N\rangle}, & \overline{\langle N, \{\bot_N\}\rangle}, & \\ & & & \overline{\langle \{\bot_N\}, N\rangle}, & \overline{\langle \{\bot_N\}, \{\bot_N\}\rangle}, & \overline{\emptyset} & \} \\ & = & \{ & \emptyset, & \overline{\langle N, \{\bot_N\}\rangle} \cap \overline{\langle \{\bot_N\}, N\rangle}, & \overline{\langle N, \{\bot_N\}\rangle}, & \\ & & & \overline{\langle \{\bot_N\}, N\rangle}, & \overline{\langle \{\bot_N\}, \{\bot_N\}\rangle}, & \langle N, N\rangle & \}, \end{array}$$
- $\mathcal{P}'_O(N \to N) = \{\overline{\lambda X \,.\, N},\ \overline{\lambda X \,.\, X},\ \overline{\lambda X \,.\, \{\bot_N\}},\ \overline{\emptyset}\}$.
  Note that $\overline{\lambda X \,.\, N}$ denotes the empty set; $\overline{\lambda X \,.\, X}$ denotes the set of continuous functions from N to N which are non–strict (that is, in the $N \to N$ case, the non–$\bot_N$ constant functions); $\overline{\lambda X \,.\, \{\bot_N\}}$ denotes the set of continuous functions from N to N which are somewhere defined; and $\overline{\emptyset}$ denotes, of course, the set of all the continuous functions from N to N.

□

Lemma 4.29 Let D be a domain constructed from a finite set of primitive domains by applying a finite number of product and function space construction. (We will call D *finite constructible*.) Let $\mathcal{P}'_C$ be the approximation scheme in Definition 4.27. Then,

1. $\mathcal{P}'_C(D)$ is a finite complete sub–lattice of $\mathcal{P}_C(D)$; and $\mathcal{P}'_O(D)$ is a finite complete sub–lattice of $\mathcal{P}_O(D)$. In particular, $\bot_{\mathcal{P}'_C(D)} = \emptyset$ and $\top_{\mathcal{P}'_C(D)} = D$. Also, $\bot_{\mathcal{P}'_O(D)} = D$ and $\top_{\mathcal{P}'_O(D)} = \emptyset$.
2. if $\mathcal{C} \subseteq \mathcal{P}'_C(D)$ and $\mathcal{O} \subseteq \mathcal{P}'_O(D)$, then $\bigsqcup \mathcal{C} = \bigcup \mathcal{C}$ and $\bigsqcup \mathcal{O} = \bigcap \mathcal{O}$.

□

Proof Outline. A structural induction on the construction of D. ◇

We omit the construction of $\mathcal{P}'_C(f)$, where $f \in D_\alpha \to D_\beta$, D_α and D_β finite constructible, to save space. The construction of $\mathcal{P}'_C(f)$ is done by a structural buildup, according to the program text defining f, from approximations of primitive functions. In cases where f is recursive defined by a functional F (*i.e.*, $f = \mathit{fix}\ F$), then $\mathit{fix}\ (\mathcal{P}'_C(F))$ is used as a safe approximation for $\mathcal{P}'_C(\mathit{fix}\ F)$.[2] The details can be found in Burn, Hankin, and Abramsky [6]. In all cases, the approximation $\mathcal{P}'_C(f)$ will be a computable continuous function if f is a continuous function.

We observe that, if D_α is not a primitive domain, an approximation $\mathcal{P}'_C(f) \in \mathcal{P}'_C(D_\alpha) \bullet\!\!\rightarrow \mathcal{P}'_C(D_\beta)$ can be characterized by a simpler approximation without losing any accuracy. This is because $\mathcal{P}'_C(f)$ is additive. As an example, let $D_\alpha = N \times N$ and $D_\beta = N$. Then we can characterize

$$\mathcal{P}'_C(f) \in \mathcal{P}'_C(N \times N) \bullet\!\!\rightarrow \mathcal{P}'_C(N)$$

by a

$$\mathcal{P}'_C(f') \in \mathcal{P}'_C(N) \otimes \mathcal{P}_C(N) \bullet\!\!\rightarrow \mathcal{P}'_C(N)$$

which is only defined on $\mathcal{P}'_C(N) \otimes \mathcal{P}_C(N)$, instead of $\mathcal{P}'_C(N \times N)$, but have the same functionality of $\mathcal{P}'_C(f)$. This causes no problem because every $X \in \mathcal{P}'_C(N \times N)$ can be expressed as $X = \bigsqcup \mathcal{X}$, where $\mathcal{X} \subseteq \mathcal{P}'_C(N) \otimes \mathcal{P}'_C(N)$, and we have $(\mathcal{P}'_C(f)) \bigsqcup \mathcal{X} = \bigsqcup((\mathcal{P}'_C(f'))\ \{\mathcal{X}\})$.

We now give some examples.

Example 4.30 Function *apply* $\in (N \to N) \times N \to N$ is defined as

$$\mathit{apply}\ \langle f,\, x\rangle = f\ x.$$

By Burn, Hankin, and Abramsky's abstract interpretation, we obtain a function

$$\mathcal{P}'_C(\mathit{apply}) \in \mathcal{P}'_C((N \to N) \times N) \bullet\!\!\rightarrow \mathcal{P}'_C(N),$$

which is characterized as

$$(\mathcal{P}'_C(\mathit{apply}))\ \langle F,\, X\rangle = F\ X$$

[2] Strictly speaking, we do not use the least fixed point of $\mathcal{P}'_C(F)$ as an safe approximation for $\mathcal{P}'_C(\mathit{fix}\ F)$. Otherwise we will always end up with $\emptyset$ as approximations since all functions between Scott–closed powerdomains is $\bot$–reflexive (*i.e.*, $\emptyset$–reflexive). We will use the least fixed point *above* $\emptyset$ of $\mathcal{P}'_C(F)$ as an safe approximation for $\mathcal{P}'_C(\mathit{fix}\ F)$.

where

$$\langle F,\, X\rangle \in \mathcal{P}'_C(N \to N) \otimes \mathcal{P}'_C(N)$$

and

$$F\ X = \{f\ x \mid f \in F, x \in X\}.$$

Note that $(\mathcal{P}'_C(\mathit{apply}))\ \emptyset = \emptyset$ by Lemma 3.24.

We can characterize $\mathcal{P}'_C(\mathit{apply})$ by the following table:

F \ X: $(\mathcal{P}'_C(\mathit{apply}))\ \langle F,\, X\rangle$	$\{\bot_N\}$	N
$\lambda X\,.\,\{\bot_N\}$	$\{\bot_N\}$	$\{\bot_N\}$
$\lambda X\,.\,X$	$\{\bot_N\}$	N
$\lambda X\,.\,N$	N	N

Now, suppose that we expect the result of *apply* $\langle f,\, x\rangle$ to be defined, *i.e.* it cannot be $\bot_N$. We use our inverse image analysis as follows:

$$\begin{array}{rl} & (\mathcal{P}'_O(\mathit{apply}^{-1}))\ (N - \{\bot_N\}) \\ = & \overline{\bigsqcup\{\langle F,\, X\rangle \mid \langle F,\, X\rangle \in \mathcal{P}'_C(N \to N) \otimes \mathcal{P}'_C(N), (\mathcal{P}'_C(\mathit{apply}))\ \langle F,\, X\rangle \ \sqsubseteq_{\mathcal{P}'_C(N)}\ \overline{N - \{\bot_N\}}\}} \\ = & \overline{\bigcup\{\langle F,\, X\rangle \mid \langle F,\, X\rangle \in \mathcal{P}'_C(N \to N) \otimes \mathcal{P}'_C(N), (\mathcal{P}'_C(\mathit{apply}))\ \langle F,\, X\rangle \subseteq \{\bot_N\}\}}. \end{array}$$

Therefore, using the definition of $\mathcal{P}'_C(\mathit{apply})$,

$$\begin{array}{rl} & (\mathcal{P}'_O(\mathit{apply}^{-1}))\ (N - \{\bot_N\}) \\ = & \overline{\bigcup\{\emptyset,\ \langle \lambda X\,.\,\{\bot_N\},\, N\rangle,\ \langle \lambda X\,.\,X,\, \{\bot_N\}\rangle\}} \\ = & \bigcap\{\overline{\langle \lambda X\,.\,\{\bot_N\},\, N\rangle},\ \overline{\langle \lambda X\,.\,X,\, \{\bot_N\}\rangle}\}. \end{array}$$

Summarized in plain English,

- f cannot be the everywhere undefined function,
- if f is a strict function, then x cannot be $\bot_N$, and
- if x is $\bot_N$, then f must be a (non–$\bot_N$) constant function,

which is no worse than our intuition. □

Example 4.31 Function $\mathit{compose} \in (N \to N) \times (N \to N) \to (N \to N)$ is defined as

$$\mathit{compose}\ \langle f,\, g\rangle = \lambda\, x\,.\, f\ (g\ x).$$

By Burn, Hankin, and Abramsky's abstract interpretation, we obtain a function

$$\mathcal{P}'_C(\mathit{compose}) \in \mathcal{P}'_C((N \to N) \times (N \to N)) \bullet\!\!\to \mathcal{P}'_C(N \to N),$$

which is characterized by

$$(\mathcal{P}'_C(\mathit{compose}))\ \langle F,\, G\rangle = \lambda\, X\,.\, F\ (G\ X),$$

where $\langle F,\, G\rangle \in \mathcal{P}'_C(N \to N) \otimes \mathcal{P}'_C(N \to N)$.

The table characterizing $\mathcal{P}'_C(\mathit{compose})$ is the following:

F \ G: $(\mathcal{P}'_C(\mathit{compose}))\ \langle F,\, G\rangle$	$\lambda X\,.\,\{\bot_N\}$	$\lambda X\,.\,X$	$\lambda X\,.\,N$
$\lambda X\,.\,\{\bot_N\}$	$\lambda X\,.\,\{\bot_N\}$	$\lambda X\,.\,\{\bot_N\}$	$\lambda X\,.\,\{\bot_N\}$
$\lambda X\,.\,X$	$\lambda X\,.\,\{\bot_N\}$	$\lambda X\,.\,X$	$\lambda X\,.\,N$
$\lambda X\,.\,N$	$\lambda X\,.\,N$	$\lambda X\,.\,N$	$\lambda X\,.\,N$

Suppose that the result of the function application *compose* $\langle f,\, g\rangle$ is not the everywhere undefined function (*i.e.* *compose* $\langle f,\, g\rangle \notin \lambda X\,.\,\bot_N$). Then, what can we say regarding f and g? Furthermore, suppose that the result is a (non–$\bot_N$) constant function (*i.e.* *compose* $\langle f,\, g\rangle \notin \lambda X\,.\,X$). Then, what should f and g be?

We simply compute $(\mathcal{P}'_O(compose^{-1}))\ \overline{\lambda X\,.\,\{\bot_N\}}$ and $(\mathcal{P}'_O(compose^{-1}))\ \overline{\lambda X\,.\,X}$ to get the answers.

$$(\mathcal{P}'_O(compose^{-1}))\ \overline{\lambda X\,.\,\{\bot_N\}} = \bigcap\{\overline{\langle \lambda X\,.\,\{\bot_N\},\, \lambda X\,.\,N\rangle},\ \overline{\langle \lambda X\,.\,X,\, \lambda X\,.\,\{\bot_N\}\rangle}\},$$

and

$$(\mathcal{P}'_O(compose^{-1}))\ \overline{\lambda X\,.\,X} = \bigcap\{\overline{\langle \lambda X\,.\,\{\bot_N\},\, \lambda X\,.\,N\rangle},\ \overline{\langle \lambda X\,.\,X,\, \lambda X\,.\,X\rangle}\}.$$

That is, if the result of *compose* $\langle f,\, g\rangle$ is not the everywhere undefined function, then

- f cannot be the everywhere undefined function,
- if f is strict, then g cannot be the everywhere undefined function, and
- if g is the everywhere undefined function, then f must be a (non–$\bot_N$) constant function.

Also, if the result of *compose* $\langle f,\, g\rangle$ is a (non–$\bot_N$) constant function, then

- f cannot be the everywhere undefined function,
- if f is strict, then g must be a (non–$\bot_N$) constant function, and
- if g is strict, then f must be a (non–$\bot_N$) constant function.

□

It is interesting to note that Hughes [11] cites the above two examples to show why backward analysis is difficult. The difficulty is to capture the interdependency of the two parts in an input argument (f and x in *apply* $\langle f,\, x\rangle$; and f and g in *compose* $\langle f,\, g\rangle$). By using our inverse image analysis, we are able to give a satisfactory account of the interdependency.

We proceed in analyzing some examples involving non-flat domains. The following examples are taken from Dybjer [9].

Example 4.32 Function *length* $\in \mathrm{L}(N) \to N$ is defined as

$$\begin{array}{lcl} length\ \ [\,] & = & 0, \quad \text{and} \\ length\ \ (x :: l) & = & 1 + length\ l, \end{array}$$

where $\mathrm{L}(N)$ is the non-flat domain of lazy lists of natural numbers.

We define $\mathcal{P}'_C(\mathrm{L}(N))$ as $\{\emptyset, \mathrm{L}_{inf}(N), \mathrm{L}(N)\}$, where $\mathrm{L}_{inf}(N)$ is the Scott–closed subset of $\mathrm{L}(N)$ which contains all the lists with undefined tails. In particular, $\mathrm{L}(N)$ includes the lists in N^ω to make $\mathrm{L}(N)$ Scott–closed. Also, $\bot_{\mathrm{L}(N)} \in \mathrm{L}_{inf}(N)$. As usual, $\mathcal{P}'_O(\mathrm{L}(N)) = \{\overline{X} \mid X \in \mathcal{P}'_C(\mathrm{L}(N))\}$.

Using Wadler's technique [16], we obtain

$$\mathcal{P}'_C(length) \in \mathcal{P}'_C(\mathrm{L}(N)) \bullet\!\!\rightarrow \mathcal{P}'_C(N)$$

which is defined by

$$\begin{array}{lcl} (\mathcal{P}'_C(length))\ \ \mathrm{L}_{inf}(N) & = & \{\bot_N\}, \quad \text{and} \\ (\mathcal{P}'_C(length))\ \ \mathrm{L}(N) & = & N. \end{array}$$

Note that $(\mathcal{P}'_C(length))\ \emptyset = \emptyset$ by Lemma 3.24.

Suppose that the result of *length* l is not $\bot_N$, then

$$\begin{array}{cl} & (\mathcal{P}'_O(length^{-1}))\ \overline{\{\bot_N\}} \\ = & \overline{\bigsqcup\{L \mid L \in \mathcal{P}'_C(\mathrm{L}(N)), (\mathcal{P}'_C(length))\ L\ \sqsubseteq_{\mathcal{P}'_C(N)}\ \overline{\overline{\{\bot_N\}}}\}} \\ = & \overline{\bigcup\{L \mid L \in \mathcal{P}'_C(\mathrm{L}(N)), (\mathcal{P}'_C(length))\ L \subseteq \{\bot_N\}\}} \\ = & \overline{\bigcup\{\emptyset, \mathrm{L}_{inf}(N)\}} \\ = & \overline{\mathrm{L}_{inf}(N)}. \end{array}$$

That is, for the result of *length* l to be defined, l cannot have an undefined tail. □

Example 4.33 Function *last* $\in \mathrm{L}(N) \rightarrow N$ is defined as

$$\begin{array}{lllll} \mathit{last} & (x :: [\,]) & = & x, & \text{and} \\ \mathit{last} & (x :: l) & = & \mathit{last}\ l. & \end{array}$$

$\mathcal{P}'_C(\mathit{last}) \in \mathcal{P}'_C(\mathrm{L}(N)) \bullet\!\!\rightarrow \mathcal{P}'_C(N)$, is defined by

$$\begin{array}{lllll} \mathcal{P}'_C(\mathit{last}) & \mathrm{L}_{\mathit{inf}}(N) & = & \{\bot_N\}, & \text{and} \\ \mathcal{P}'_C(\mathit{last}) & \mathrm{L}(N) & = & N. & \end{array}$$

In the case that the result of *last* l is not $\bot_N$, $(\mathcal{P}'_O(\mathit{last}^{-1}))\ \overline{\{\bot_N\}} = \overline{\mathrm{L}_{\mathit{inf}}(N)}$. That is, l cannot have an undefined tail.

Note that this answer is safe but not totally accurate. A list l with an undefined tail will make *last* l undefined, but not all finite–length lists (*i.e.* the lists without undefined tails) will make *last* l defined. In particular, if $l = [\,]$, then $\mathit{last}\ [\,] = \bot_N$. Can we do better than what we have done?

Let us define a new powerdomain $\mathcal{P}''_C(\mathrm{L}(N)) = \{\emptyset, \{\bot_{\mathrm{L}(N)}\}, \{\bot_{\mathrm{L}(N)}, [\,]\}, \mathrm{L}_{\mathit{inf}}(N), \mathrm{L}(N)\}$. Also, $\mathcal{P}''_O(\mathrm{L}(N)) = \{\overline{X} \mid X \in \mathcal{P}''_C(\mathrm{L}(N))\}$. $\mathcal{P}''_C(\mathit{last})$ is characterized by

X	$(\mathcal{P}''_C(\mathit{last}))\ X$
$\emptyset$	$\emptyset$
$\{\bot_{\mathrm{L}(N)}\}$	$\{\bot_N\}$
$\{\bot_{\mathrm{L}(N)}, [\,]\}$	$\{\bot_N\}$
$\mathrm{L}_{\mathit{inf}}(N)$	$\{\bot_N\}$
$\mathrm{L}(N)$	N

It is not difficult to see that

$$\begin{array}{ll} & (\mathcal{P}''_O(\mathit{last}^{-1}))\ \overline{\{\bot_N\}} \\ = & \overline{\bigcup\{X \mid X \in \mathcal{P}''_C(\mathrm{L}(N)), (\mathcal{P}''_C(\mathit{last}))\ X \subseteq \{\bot_N\}\}} \\ = & \overline{\bigcup\{\emptyset,\ \{\bot_{\mathrm{L}(N)}\},\ \{\bot_{\mathrm{L}(N)}, [\,]\},\ \mathrm{L}_{\mathit{inf}}(N)\}} \\ = & \overline{\mathrm{L}_{\mathit{inf}}(N) \cup \{[\,]\}} \\ = & \overline{\mathrm{L}_{\mathit{inf}}(N)} \cap \overline{\{[\,]\}} \end{array}$$

That is, if *last* l is defined, then l cannot be the list with an undefined tail and l cannot be the null list. □

It is clear from the above example that the accuracy of our inverse image analysis will depend on the accuracy of the forward analysis from which the inverse image analysis is derived. How precise a forward analysis will suffice? The answer depends on the problem being studied and the programs being analyzed. So far, we have only discussed *tail strictness*, without mentioning *head strictness*. In the *last* program, we would like to know that in order for *last* l to be defined, the last element of l cannot be undefined. In order to do this, we would have to define a powerdomain more refined than $\mathcal{P}''_C(\mathrm{L}(N))$.

In order to capture head strictness, we can define a Scott–closed subset $\mathrm{L}_{\exists\bot_N}(N)$ to include all the lists which either containing $\bot_N$ or in N^ω. We then have $\mathrm{L}_{\mathit{inf}}(N) \subseteq \mathrm{L}_{\exists\bot_N}(N)$. Also, all elements in the Scott–open set $\overline{\mathrm{L}_{\exists\bot_N}(N)}$ are of finite length and containing no $\bot_N$.

Two more examples from Dybjer [9].

Example 4.34 Function *from* $\in N \rightarrow \mathrm{L}(N)$ is defined as

$$\mathit{from}\quad x \quad = \quad x :: (\mathit{from}\ (x+1)).$$

$\mathcal{P}'_C(\mathit{from}) \in \mathcal{P}'_C(N) \bullet\!\!\rightarrow \mathcal{P}'_C(\mathrm{L}(N))$, is defined by

X	$(\mathcal{P}'_C(\mathit{from}))\ X$
$\emptyset$	$\emptyset$
$\{\bot_N\}$	$\mathrm{L}_{\mathit{inf}}(N)$
N	$\mathrm{L}_{\mathit{inf}}(N)$

Suppose we expect the result of *from* x to be a finite–length list. We calculate

$$\begin{array}{rl} & (\mathcal{P}'_O(\mathit{from}^{-1}))\ \overline{\mathrm{L}_{inf}(N)} \\ = & \overline{\bigcup\{X \mid X \in \mathcal{P}'_C(N), (\mathcal{P}'_C(\mathit{from}))\ X \subseteq \mathrm{L}_{inf}(N)\}} \\ = & \overline{\bigcup\{\emptyset,\ \{\bot_N\},\ N\}} \\ = & \emptyset. \end{array}$$

This means that there is no $x \in N$ which will make *from* x a finite–length list. □

Example 4.35 Function $\mathit{append} \in \mathrm{L}(N) \times \mathrm{L}(N) \to \mathrm{L}(N)$ is defined as

$$\begin{array}{lcl} \mathit{append}\ \langle [\,],\ g\rangle & = & g, \\ \mathit{append}\ \langle x :: l,\ g\rangle & = & x :: (\mathit{append}\ l\ g). \end{array}$$

$\mathcal{P}'_C(\mathit{append}) \in \mathcal{P}'_C(\mathrm{L}(N) \times \mathrm{L}(N)) \bullet\!\!\to \mathcal{P}'_C(\mathrm{L}(N))$, is characterized by

F \ G: $\mathcal{P}'_C(\mathit{append})\ \langle F,\ G\rangle$	$\mathrm{L}_{inf}(N)$	$\mathrm{L}(N)$
$\mathrm{L}_{inf}(N)$	$\mathrm{L}_{inf}(N)$	$\mathrm{L}_{inf}(N)$
$\mathrm{L}(N)$	$\mathrm{L}_{inf}(N)$	$\mathrm{L}(N)$

Suppose we expect the result of *append* $\langle f,\ g\rangle$ to be a finite–length list.
We calculate

$$\begin{array}{rl} & (\mathcal{P}'_O(\mathit{append}^{-1}))\ \overline{\mathrm{L}_{inf}(N)} \\ = & \overline{\bigcup\{\langle F,\ G\rangle \mid \langle F,\ G\rangle \in \mathcal{P}'_C(\mathrm{L}(N)) \otimes \mathcal{P}'_C(\mathrm{L}(N)), (\mathcal{P}'_C(\mathit{append}))\ \langle F,\ G\rangle \subseteq \mathrm{L}_{inf}(N)\}} \\ = & \overline{\bigcup\{\langle \mathrm{L}_{inf}(N),\ \mathrm{L}(N)\rangle,\ \langle \mathrm{L}(N),\ \mathrm{L}_{inf}(N)\rangle\}} \\ = & \overline{\langle \mathrm{L}_{inf}(N),\ \mathrm{L}(N)\rangle} \cap \overline{\langle \mathrm{L}(N),\ \mathrm{L}_{inf}(N)\rangle}. \end{array}$$

That is, for the result of *append* $\langle f,\ g\rangle$ to be a finite–length list, f cannot have an undefined tail and g cannot have an undefined tail. □

5 Some Remarks

How expensive is it to carry out the inverse image analysis described here? Also, how expressive is this analysis? In this section, We would like to make some remarks regarding these two questions. We will also state the relationships between this work and previous works, and outline other possible applications of this work.

5.1 The Time Complexity of Inverse Image Analysis

At first glance it seems that backward analysis by inverse images is very expensive. It is not so. The cost of approximating an inverse image of a specific function f is only as expensive as computing $\mathcal{P}'_C(f)$ from the program text defining f.

To verify this claim, let us first estimate how expensive it is to get $\mathcal{P}'_C(f)$. Suppose f is a function from domain D_α to D_β. Then $\mathcal{P}'_C(f)$ is a continuous function from $\mathcal{P}'_C(D_\alpha)$ to $\mathcal{P}'_C(D_\beta)$. In the cases that the program defining f is recursive, we will need to find an element in the abstract domain $\mathcal{P}'_C(D_\alpha) \to \mathcal{P}'_C(D_\beta)$ which serves as the least fixed point of a particular function in the abstract domain $(\mathcal{P}'_C(D_\alpha) \to \mathcal{P}'_C(D_\beta)) \to (\mathcal{P}'_C(D_\alpha) \to \mathcal{P}'_C(D_\beta))$. The worst cases time complexity for this least–fixed–point–finding procedure is proportional to the height of the domain $\mathcal{P}'_C(D_\alpha) \to \mathcal{P}'_C(D_\beta)$. It can be shown that the worst cases will need $O(|\mathcal{P}'_C(D_\alpha)| \cdot \mathit{height}(\mathcal{P}'_C(D_\beta)))$ time, where $|\mathcal{P}'_C(D_\alpha)|$ is the size of $\mathcal{P}'_C(D_\alpha)$ and $\mathit{height}(\mathcal{P}'_C(D_\beta))$ is the height of $\mathcal{P}'_C(D_\beta)$.

By Definition 3.20, the formulation of an abstract inverse image function of f is essential free if $\mathcal{P}'_C(f)$ is given. A naive computation of an inverse image will involve the enumeration of all the elements in the abstract domain ${\mathcal{P}'}_C(D_\alpha)$, which will take time $O(|{\mathcal{P}'}_C(\alpha)|)$. But this is only as expensive as (or, less expensive than) the $O(|{\mathcal{P}'}_C(D_\alpha)| \cdot \mathit{height}({\mathcal{P}'}_C(D_\beta)))$ time to compute ${\mathcal{P}'}_C(f)$.

Let us draw a table to compare the time complexity of an abstract interpretation and its inverse image analysis. The "preprocessing" time in the following table refers to the time complexity to compute ${\mathcal{P}'}_C(f)$, or ${\mathcal{P}'}_O(f^{-1})$, given the program text which defines f. The "query" time is the time complexity to compute $({\mathcal{P}'}_C(f))\ A$, or $({\mathcal{P}'}_O(f^{-1}))\ B$, given $A \in {\mathcal{P}'}_C(D_\alpha)$ and $B \in {\mathcal{P}'}_O(D_\beta)$.

	abstract interpretation	inverse image analysis
preprocessing time	$O(\lvert{\mathcal{P}'}_C(D_\alpha)\rvert \cdot \mathit{height}({\mathcal{P}'}_C(D_\beta)))$	$O(\lvert{\mathcal{P}'}_C(D_\alpha)\rvert \cdot \mathit{height}({\mathcal{P}'}_C(D_\beta)))$
query time	$O(1)$	$O(\lvert{\mathcal{P}'}_C(D_\alpha)\rvert)$

Note that if an abstract interpretation is performed before an inverse image analysis, then the preprocessing time for the inverse image analysis is reduced to $O(1)$. Also note that we have greatly simplified the computation model to make comparison in the above table. For example, the query time for an abstract interpretation is not necessary $O(1)$ because it depends on the structure of ${\mathcal{P}'}_C(f)$. However, since both the queries in abstract interpretation and in inverse image analysis use the same approximation ${\mathcal{P}'}_C(f)$, it is reasonable to take the $O(1)$ assumption to make the comparison clear. We also implicitly assume that the ordering predicate, $\sqsubseteq$, takes only $O(1)$ time for a fixed finite abstract domain.

5.2 The Expressive Power of Inverse Image Analysis

How expressive is the inverse image analysis described here? Before answering this question, let us first make a brief review of backward analysis as in literature so far.

A *context* of a domain D as described in Hughes [11] can be viewed as a continuous function from D to D. Given a program P which defines a continuous function f from domain D_α to domain D_β, and given a context c_β of D_β, a backward analysis based on context will try to infer from program P, as precise as possible, a context c_α of D_α such that

$$c_\beta \circ f = c_\beta \circ f \circ c_\alpha.$$

The *projection* notation as described in Wadler and Hughes [17] is similar to the context notation, except that projections are required to be idempotent and less defined than the identity context. Burn [5] further restricts projection to *smash projection*, which either maps an element in a domain D to the element itself or maps the element to $\bot_D$, to conclude some interesting relationships between abstract interpretation and projection analysis. It is clear that context notation is more expressive than projection notation, and projection more expressive than smash projection.

One important characterization of previous works is that they often operate on lifted domains and often require the functions between these lifted domain to be strict in the newly added least defined element. Also note that for a context–based (or projection–based, or smash–projection–based) backward analysis to work effectively, we must restrict the collection of contexts (or projections, or smash projections) to a finite set. We must also order these contexts properly and approximate any give context to an restricted context in the safe direction.

It turns out that Scott–closed/Scott–open subsets have the exactly expressive power of strict smash projections.[3] Let domain lifted(D) be the lifted domain of a domain D. Then the poset of all the strict smash projections from lifted(D) to lifted(D) can be shown to be isomorphic to $\mathcal{P}_C(D)$. That is, each strict smash projection maps, beside $\bot_{\text{lifted}(D)}$, the elements in a lifted Scott–closed subset of D to $\bot_{\text{lifted}(D)}$, and maps each element of the lifted complement Scott–open subset to itself. Likewise, each Scott–closed subset of D defines a strict smash projection from lifted(D) to lifted(D) which maps, beside $\bot_{\text{lifted}(D)}$, those elements in the lifted Scott–closed subset to $\bot_{\text{lifted}(D)}$, and maps each element in the lifted complement Scott–open subsets to itself.

[3] We would like to thank an anonymous functional programming researcher for pointing out this connection. Burn probably knows this, although it is not stated in [5]. This is a direct consequence of $\mathcal{P}_C(D) \cong D \to \mathit{2}$ (Barendregt [3, page 20, exercise 1.3.11 (ii)]), where $\mathit{2}$ is the two element domain.

Although the Scott–closed/Scott–open powerdomain possess the same expressive power of strict smash projections, it does have its advantage. One of the advantages is it leads to a natural extension to higher–order cases. Previous works have not be able to give a successfully account of higher–order backward analysis based on context (or projection, or smashed projection).

By using Scott–closed/Scott–open powerdomains, we are also able to explain some special constructions/restrictions in previous works. For examples, previous works lift a domain to include a new least defined element which symbolizes the computation which leading to an "abort". And the functions between these lifted domains must be strict (or $\perp$–reflexive in some cases) on this newly added least defined element. By using Scott–closed powerdomains, it becomes clear that the empty set element $\emptyset$ in domain $\mathcal{P}_C(D)$ and $\mathcal{P}'_C(D)$ states exactly the "abort" situation; which simply says that no element in D is appropriate. Also, for a continuous function f from domain D_α to D_β, it is shown that $\mathcal{P}_C(f)$ and $\mathcal{P}'_C(f)$ are $\emptyset$–reflexive (Lemma 3.24). It is a natural *consequence* in the theory of Scott–closed/Scott–open powerdomain, not as a restriction in the settings of previous works.

The relationship between forward analysis and backward analysis is clearer too. For example, both Burn [5, page 154, Theorem 3.1] and we (Corollary 2.12 and 3.23) state the relationships between forward analysis and backward analysis. We think our representation is simpler. The restriction of $\perp$–reflexive abstraction maps and strict functions are gone too in our development. They are natural consequences of the theory of Scott–closed/Scott–open powerdomain.

Let us draw some tables to illustrate some simple contexts and their counter–parts in Scott–closed/Scott–open powerdomains. The following table lists four contexts and their effects on the elements in a lifted domain.

α	*FAIL*	*ABSENT*	*STRICT*	*IDENTICAL*
e / $\alpha\ e$				
lifted $d, d \neq \perp_D$	$\perp_{\text{lifted}(D)}$	*lifted* $\perp_D$	*lifted* d	*lifted* d
lifted $\perp_D$	$\perp_{\text{lifted}(D)}$	*lifted* $\perp_D$	$\perp_{\text{lifted}(D)}$	*lifted* $\perp_D$
$\perp_{\text{lifted}(D)}$	$\perp_{\text{lifted}(D)}$	$\perp_{\text{lifted}(D)}$	$\perp_{\text{lifted}(D)}$	$\perp_{\text{lifted}(D)}$

The following table shows how the four contexts are described by the approximating Scott–closed powerdomain $\mathcal{P}'_C(D) = \{D, \{\perp_D\}, \emptyset\}$ and its complement Scott–open powerdomain.

	Scott–closed	Scott–open
FAIL	D	$\emptyset$
ABSENT	$-$	$-$
STRICT	$\{\perp_D\}$	$D - \{\perp_D\}$
IDENTICAL	$\emptyset$	D

Note that *ABSENT* is not a smash projection and it cannot be described by Scott–closed/Scott–open subsets either.

5.3 Beyond Strictness Analysis

The inverse image analysis, as described so far in this paper, always assume that the abstract interpretation is done on Scott–closed powerdomains; hence, the backward analysis is done on Scott–open powerdomains. However, it is not difficult to develop an abstract interpretation based on Scott–open powerdomains, and to have the backward analysis done on Scott–closed powerdomains. This should not come as a surprise since a Scott–closed powerdomain $\mathcal{P}_C(D)$ is isomorphic to the complement Scott–open powerdomain $\mathcal{P}_O(D)$. We should emphasize that a Scott–open based abstract interpretation is also a useful development. For example, it is quite useful in the constant–propagation problem.

They are many possible applications of inverse image analysis, besides backward strictness analysis, since it is a generic scheme for doing backward analysis. Given any correct and computable abstract semantic of a non–strict higher–order functional programming language, we are able to transform it into a backward version that is also correct and computable. It is not difficult to image many *non–standard* semantics which correspond to some specific optimization opportunities. If there is a correct (Scott–closed/Scott–open powerdomain based) abstraction of an specific non–standard semantics, then, by using inverse image analysis, a correct backward version of the abstraction comes for free.

6 Conclusion

We have proposed a method for performing backward analysis based on inverse images of abstract higher–order functions. Our method differs from previous works on backward analysis [8] [11] [17] in two major ways. First, our method deals with a program's *extensional* representation, rather than its *intensional* (textual) representation. That is, for a program P, we analyze the inverse image of the function f_P, which is the semantic denotation of P, as well as approximation versions of f_P. This approach leads to a very clean concept for backward analysis. It frees us from the necessity of analyzing the text of a program. However, a forward analysis of a program P, based on its text, is needed before we perform backward analysis.

Secondly, we are able to do backward analysis on higher–order functions in a very natural way. Also, we are able to capture the interdependency of arguments of a program regarding their effects on the result of the program. We view this an improvement over previous works, which blend abstract interpretation and backward analysis at the same time to get these results. Again, our approach (performing forward analysis to get an extensional representation of a program, then performing a backward analysis on the extensional representation) seems to be cleaner.

7 Acknowledgments

The authors wish to thank Fritz Henglein, Peter Sestoft, Kei Davis and John Hughes for their assistance.

A Bounded–Complete ω–Algebraic CPOs and Scott–Closed/Scott–Open Powerdomains

A cpo (*complete partial order*) D is *bounded–complete* if, for all $a, b \in D$, if there exists a $c \in D$ such that $a \sqsubseteq_D c$ and $b \sqsubseteq_D c$, then $a \sqcup b$ exists in D. If D is bounded–complete, it follows that every countable non–empty subset of a directed subset of D has the least upper bound in D.

A cpo D is *ω–algebraic* if the set of compact elements in D, written as $K(D)$, is countable, and if for every element $d \in D$, the set $\{e \mid e \in K(D), e \sqsubseteq_D d\}$ is directed and $d = \bigsqcup\{e \mid e \in K(D), e \sqsubseteq_D d\}$.

An element e in a cpo D is *compact* if for every directed subset $X \subseteq D$ such that $e \sqsubseteq_D \bigsqcup X$, we have $e \sqsubseteq_D x$ for some $x \in X$.

We list here some technical definitions and basic properties of bound–complete ω–algebraic cpos, as well as the properties of Scott–open and Scott–closed powerdomains. Some of their proofs can be found in Barendregt [3], Gierz, Hofmann, Keimel, Lawson, Mislove, and Scott [10], Schmidt [14], and Stoy [15].

Fact A.36 Let D be a domain. Then,

1. if $A \in \mathcal{P}_C(D)$, then $\overline{A} \in \mathcal{P}_O(D)$;
2. if $A \in \mathcal{P}_O(D)$, then $\overline{A} \in \mathcal{P}_C(D)$.

□

Fact A.37 Let D be a domain. Then,

1. $\mathcal{P}_C(D)$ is a lattice with $X \sqcap Y = X \cap Y$ and $X \sqcup Y = X \cup Y$ for every $X, Y \in \mathcal{P}_C(D)$;
2. $\mathcal{P}_O(D)$ is a lattice with $X \sqcap Y = X \cup Y$ and $X \sqcup Y = X \cap Y$ for every $X, Y \in \mathcal{P}_O(D)$.

□

Furthermore, by the following standard result in lattice theories,

Fact A.38 Let D be a poset. If for every subset $X \subseteq D$, there exists $\sqcap X \in D$, then D is a complete lattice with $\bigsqcup X = \sqcap\{a \mid a \in D, x \sqsubseteq_D a \text{ for all } x \in X\}$. □

we are able to show the following.

Corollary A.39 Let D be a domain. Then,

1. $\mathcal{P}_C(D)$ is a complete lattice with $\sqcap \mathcal{X} = \bigcap \mathcal{X}$ and $\bigsqcup \mathcal{X} = \bigcap\{Y \mid Y \in \mathcal{P}_C(D), Y \supseteq \bigcup \mathcal{X}\}$ for every subset $\mathcal{X} \subseteq \mathcal{P}_C(D)$;
2. $\mathcal{P}_O(D)$ is a complete lattice with $\sqcap \mathcal{X} = \bigcup \mathcal{X}$ and $\bigsqcup \mathcal{X} = \bigcup\{Y \mid Y \in \mathcal{P}_O(D), Y \subseteq \bigcap \mathcal{X}\}$ for every subset $\mathcal{X} \subseteq \mathcal{P}_O(D)$.

□

Remark A.40 Let D be a domain. Then, it is not necessary true that $\bigsqcup \mathcal{X} = \bigcup \mathcal{X}$ for every subset $\mathcal{X} \subseteq \mathcal{P}_C(D)$; since $\bigcup \mathcal{X}$ may not be in $\mathcal{P}_C(D)$. It is not necessary true either that $\bigsqcup \mathcal{X} = \bigcap \mathcal{X}$ for every subset $\mathcal{X} \subseteq \mathcal{P}_O(D)$; since $\bigcap \mathcal{X}$ may not be in $\mathcal{P}_O(D)$.

Also, we have $\perp_{\mathcal{P}_C(D)} = \emptyset$ and $\top_{\mathcal{P}_C(D)} = D$. It follows that $\perp_{\mathcal{P}_O(D)} = D$ and $\top_{\mathcal{P}_O(D)} = \emptyset$. □

Definition A.41 Let D be a domain. Let $d \in D$ and $X \subseteq D$. Define

1. $\downarrow d = \{x \mid x \in D, x \sqsubseteq_D d\}$;
2. $\uparrow d = \{x \mid x \in D, d \sqsubseteq_D x\}$;
3. $\downarrow X = \{y \mid y \in \downarrow x, x \in X\}$;
4. $\uparrow X = \{y \mid y \in \uparrow x, x \in X\}$;
5. $X^c = \bigcap\{Y \mid Y \in \mathcal{P}_C(D), Y \supseteq X\}$

□

$\downarrow d$ is usually called the *lower set* of d; $\uparrow d$ is called the *upper set* of d. Note that $\downarrow d$ is Scott–closed in D for each $d \in D$. However, $\uparrow d$ is Scott–open in D iff d is a compact element in D. Also, by definition, X^c is the least Scott–closed subset of D which contains X.

Some properties of lower/upper sets are stated in the following facts.

Fact A.42 If O is open in D, then $O = \bigcup \mathcal{O}$ for some subset $\mathcal{O} \subseteq \{\uparrow e \mid e \in K(D)\}$. □

Fact A.43 Let D be a domain. Let $x \in D$ and $X, Y \subseteq D$. Let $\mathcal{X}$ be a directed subset of $\mathcal{P}_C(D)$. Then,

1. if $x \in X^c$ but $x \notin \downarrow X$, then x cannot be compact;
2. if $X \subseteq Y$, then $X^c \sqsubseteq_{\mathcal{P}_C(D)} Y^c$;
3. $\bigsqcup \mathcal{X} = (\bigcup \mathcal{X})^c$;
4. if $X \subseteq \bigcup \mathcal{X}$ and X is directed, then $\bigsqcup X \in \bigsqcup \mathcal{X}$.

□

We now show that both $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are domains if D is a domain.

Lemma A.44 Let D be a domain. Then

1. $\mathcal{P}_C(D)$ is a domain with $K(\mathcal{P}_C(D)) = \{\bigcup \mathcal{X} \mid \mathcal{X} \text{ is a finite subset of } \{\downarrow e \mid e \in K(D)\}\}$;
2. $\mathcal{P}_O(D)$ is a domain with $K(\mathcal{P}_O(D)) = \{\overline{E} \mid E \in K(\mathcal{P}_C(D))\}$.

□

PROOF. We need to show that both $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are bounded–complete and ω–albebraic. Since both $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are complete lattices, they are bounded–complete. It remains to show that both $\mathcal{P}_C(D)$ and $\mathcal{P}_O(D)$ are ω–algebraic.

Let $\mathcal{E} = \{\bigcup \mathcal{X} \mid \mathcal{X}$ is a finite subset of $\{\downarrow e \mid e \in K(D)\}\}$.

1. We will show that $\mathcal{E} \subseteq K(\mathcal{P}_C(D))$, $\mathcal{E}$ is countable, and for every $X \in \mathcal{P}_C(D)$, $X = \bigsqcup\{E \mid E \in \mathcal{E},\ E \sqsubseteq_{\mathcal{P}_C(D)} X\}$. Then, it follows that $\mathcal{P}_C(D)$ is ω–algebraic with $K(\mathcal{P}_C(D)) = \mathcal{E}$.

   Suppose $\mathcal{E} \not\subseteq K(\mathcal{P}_C(D))$. Then there exists a $E \in \mathcal{E}$ such that E is not a compact element of $\mathcal{P}_C(D)$. Therefore, there exists a directed subset $\mathcal{X} \subseteq \mathcal{P}_C(D)$ such that $E \sqsubseteq_{\mathcal{P}_C(D)} \bigsqcup \mathcal{X}$ but $E \not\sqsubseteq_{\mathcal{P}_C(D)} X$ for every $X \in \mathcal{X}$. That is, there exists a compact $e \in D$ such that $e \in \bigsqcup \mathcal{X}$ but $e \notin \bigcup \mathcal{X}$. But, by Fact A.43, this cannot be true. Therefore, $\mathcal{E} \subseteq K(\mathcal{P}_C(D))$. It is easy to show that $\mathcal{E}$ is countable, given that $K(D)$ is countable.

   It remains to show that for every $X \in \mathcal{P}_C(D)$, the set $\{E \mid E \in \mathcal{E}, E \sqsubseteq_{\mathcal{P}_C(D)} X\}$ is directed and $X = \bigsqcup\{E \mid E \in \mathcal{E}, E \sqsubseteq_{\mathcal{P}_C(D)} X\}$. Let $\mathcal{E}_0 = \{E \mid E \in \mathcal{E},\ E \sqsubseteq_{\mathcal{P}_C(D)} X\}$. It is clear that $\mathcal{E}_0$ is directed. It suffices to show that $X \sqsubseteq_{\mathcal{P}_C(D)} \bigsqcup \mathcal{E}_0$ to complete the proof. Suppose that $X \not\sqsubseteq_{\mathcal{P}_C(D)} \bigsqcup \mathcal{E}_0$. Then, there exists a non–compact $x \in X$ but $x \notin \bigsqcup \mathcal{E}_0$. Since D is algebraic, we have $x = \bigsqcup E$, where $E = \{e \mid e \in K(D), e \sqsubseteq_D x\}$ is directed. But $E \subseteq \bigcup \mathcal{E}_0$. Hence, $x = \bigsqcup E \in \bigsqcup \mathcal{E}_0$, a contradiction.

   We conclude that $\mathcal{P}_C(D)$ is ω–algebraic.

2. Let $\mathcal{X} = \{X_i \mid 1 \leq i \leq n\}$ be a directed subset of $\mathcal{P}_O(D)$. Let $\widetilde{\mathcal{X}}$ denote the directed subset $\{\overline{X_i} \mid 1 \leq i \leq n\}$ of $\mathcal{P}_C(D)$. We first show that $\overline{\bigsqcup X} = \bigsqcup \widetilde{\mathcal{X}}$.

$$\begin{array}{rl} & \overline{\bigsqcup \mathcal{X}} \\ = & \overline{\bigcup\{Y \mid Y \in \mathcal{P}_O(D), Y \subseteq \bigcap \mathcal{X}\}} \\ = & \bigcap\{\overline{Y} \mid Y \in \mathcal{P}_O(D), Y \subseteq \bigcap \mathcal{X}\} \\ = & \bigcap\{Y' \mid Y' \in \mathcal{P}_C(D), \overline{Y'} \subseteq \bigcap \mathcal{X}\} \\ = & \bigcap\{Y' \mid Y' \in \mathcal{P}_C(D), \overline{Y'} \subseteq \bigcup \widetilde{\mathcal{X}}\} \\ = & \bigcap\{Y' \mid Y' \in \mathcal{P}_C(D), Y' \supseteq \bigcup \widetilde{\mathcal{X}}\} \\ = & \bigsqcup \widetilde{\mathcal{X}} \end{array}$$

   It remains to show that $K(\mathcal{P}_O(D)) = \{\overline{E} \mid E \in K(\mathcal{P}_C(D))\} = \widetilde{\mathcal{E}}$. It suffices to show that $\widetilde{\mathcal{E}} \subseteq K(\mathcal{P}_O(D))$ and for every $X \in \mathcal{P}_O(D)$, we have $X = \bigsqcup\{E \mid E \in \widetilde{\mathcal{E}}, E \sqsubseteq_{\mathcal{P}_O(D)} X\}$.

   Suppose that $\widetilde{\mathcal{E}} \not\subseteq K(\mathcal{P}_O(D))$. Then

$$\begin{array}{rl} & \text{there exists a } E \in \widetilde{\mathcal{E}} \text{ such that } E \notin K(\mathcal{P}_O(D)) \\ \Rightarrow & \text{there exists a } E \in \widetilde{\mathcal{E}} \text{ and a directed subset } \mathcal{X} \subseteq \mathcal{P}_O(D) \text{ such that} \\ & E \sqsubseteq_{\mathcal{P}_O(D)} \bigsqcup \mathcal{X}, \text{ but } E \not\sqsubseteq_{\mathcal{P}_O(D)} X \text{ for every } X \in \mathcal{X} \\ \Rightarrow & \text{there exists a } E \in \widetilde{\mathcal{E}} \text{ and a directed subset } \mathcal{X} \subseteq \mathcal{P}_O(D) \text{ such that} \\ & \overline{E} \sqsubseteq_{\mathcal{P}_C(D)} \bigsqcup \widetilde{\mathcal{X}}, \text{ but } \overline{E} \not\sqsubseteq_{\mathcal{P}_C(D)} \overline{X} \text{ for every } X \in \mathcal{X} \\ \Rightarrow & \text{there exists a } E \in \mathcal{E} \text{ and a directed subset } \mathcal{X}' \subseteq \mathcal{P}_C(D) \text{ such that} \\ & E \sqsubseteq_{\mathcal{P}_C(D)} \bigsqcup \mathcal{X}', \text{ but } E \not\sqsubseteq_{\mathcal{P}_C(D)} X' \text{ for every } X' \in \mathcal{X}'. \end{array}$$

   But this cannot be true because each element in $\mathcal{E}$ is a compact element of $\mathcal{P}_C(D)$.

   It remains to show that for every $X \in \mathcal{P}_O(D)$, we have $X = \bigsqcup\{E \mid E \in \widetilde{\mathcal{E}}, E \sqsubseteq_{\mathcal{P}_O(D)} X\}$. Let $\mathcal{E}_0 = \{E \mid E \in \widetilde{\mathcal{E}},\ E \sqsubseteq_{\mathcal{P}_O(D)} X\}$. It suffices to show that $X \sqsubseteq_{\mathcal{P}_O(D)} \bigsqcup \mathcal{E}_0$ to complete the proof.

Since $\overline{X} \in \mathcal{P}_C(D)$ and $\mathcal{P}_C(D)$ is ω–algebraic with $K(\mathcal{P}_C(D)) = \mathcal{E}$, we have

$$\begin{array}{lll}
& \overline{X} \sqsubseteq_{\mathcal{P}_C(D)} & \bigsqcup\{E \mid E \in \mathcal{E}, E \sqsubseteq_{\mathcal{P}_C(D)} \overline{X}\} \\
\Rightarrow & \overline{X} \sqsubseteq_{\mathcal{P}_C(D)} & \bigsqcup\{E \mid E \in \mathcal{E}, \overline{E} \sqsubseteq_{\mathcal{P}_O(D)} X\} \\
\Rightarrow & \overline{X} \sqsubseteq_{\mathcal{P}_C(D)} & \bigsqcup\{\overline{E} \mid E \in \widetilde{\mathcal{E}}, E \sqsubseteq_{\mathcal{P}_O(D)} X\} \\
\Rightarrow & \overline{X} \sqsubseteq_{\mathcal{P}_C(D)} & \bigsqcup\widetilde{\mathcal{E}}_0 \\
\Rightarrow & \overline{X} \sqsubseteq_{\mathcal{P}_C(D)} & \overline{\bigsqcup\mathcal{E}_0} \\
\Rightarrow & X \sqsubseteq_{\mathcal{P}_O(D)} & \bigsqcup\mathcal{E}_0.
\end{array}$$

◇

The following two facts deal with continuous functions between domains.

Fact A.45 Let D_α and D_β be domains. Then f is a continuous function from D_α to D_β iff

$$f\ x = \bigsqcup\{f\ e \mid e \in K(D_\alpha), e \sqsubseteq_{D_\alpha} x\},$$

where $x \in D_\alpha$. □

Fact A.46 $D_\alpha \to D_\beta$, the continuous function space from D_α to D_β, is a domain if both D_α and D_β are domain. □

B Image Functions and Inverse Image Functions

Lemma B.47 let D_α and D_β be domains. Then both $\mathcal{P}_C(f)$ and $\mathcal{P}_O(f^{-1})$ are well–defined and continuous for every continuous function $f \in D_\alpha \to D_\beta$. □

Proof. By Definition A.41, $\mathcal{P}_C(f)$ is well–defined. By Fact 2.5, $\mathcal{P}_O(f^{-1})$ is well–defined.
It remains to show both $\mathcal{P}_C(f)$ and $\mathcal{P}_O(f^{-1})$ are continuous.

1. In order to prove that $\mathcal{P}_C(f)$ is continuous, it suffices to show that

$$(\mathcal{P}_C(f)) \bigsqcup \mathcal{E} \sqsubseteq_{\mathcal{P}_C(D_\beta)} \bigsqcup (\mathcal{P}_C(f))\{\mathcal{E}\},$$

where $\mathcal{E} = \{E \mid E \in K(\mathcal{P}_C(D_\alpha)), E \sqsubseteq_{\mathcal{P}_C(D_\alpha)} X\}$.
Since $\downarrow(f\{\bigcup\mathcal{E}\}) \subseteq \bigcup((\mathcal{P}_C(f))\ \{\mathcal{E}\})$, by Fact A.43, we have

$$(f\{\bigcup\mathcal{E}\})^c \sqsubseteq_{\mathcal{P}_C(D_\beta)} (\bigcup((\mathcal{P}_C(f))\ \{\mathcal{E}\}))^c = \bigsqcup (\mathcal{P}_C(f))\{\mathcal{E}\}.$$

Also, we have $(\mathcal{P}_C(f)) \bigsqcup\mathcal{E} = (f\{\bigsqcup\mathcal{E}\})^c$. Therefore, it suffices to show that

$$(f\{\bigsqcup\mathcal{E}\})^c \sqsubseteq_{\mathcal{P}_C(D_\beta)} (f\{\bigcup\mathcal{E}\})^c$$

to complete the proof.
Suppose that it is not true. Then, there exists a $b \in (f\{\bigsqcup\mathcal{E}\})^c$ but $b \notin (f\{\bigcup\mathcal{E}\})^c$.
Since D_β is ω–algebraic, we let $b = \bigsqcup B$, where

$$\begin{array}{rcll}
B & = & \{e_i \mid e_i \in K(D_\beta), e_i \in (f\{\bigsqcup\mathcal{E}\})^c, e_i \sqsubseteq_{D_\beta} b\} & \\
& = & \{e_i \mid e_i \in K(D_\beta), e_i \in f\{\bigsqcup\mathcal{E}\}, e_i \sqsubseteq_{D_\beta} b\} & \text{(by Fact A.43).}
\end{array}$$

For each $e_i \in B$, we know that $e_i = f\ a_i$ for some $a_i \in \bigsqcup\mathcal{E}$. Again, since D_α is ω–algebraic, we let $a_i = \bigsqcup A_i$, where

$$\begin{array}{rcll}
A_i & = & \{e_{i,j} \mid e_{i,j} \in K(D_\alpha), e_{i,j} \in \bigsqcup\mathcal{E}, e_{i,j} \sqsubseteq_{D_\alpha} a_i\} & \\
& = & \{e_{i,j} \mid e_{i,j} \in K(D_\alpha), e_{i,j} \in \bigcup\mathcal{E}, e_{i,j} \sqsubseteq_{D_\alpha} a_i\} & \text{(by Fact A.43).}
\end{array}$$

Hence, $f\{A_i\} \subseteq f\{\bigcup \mathcal{E}\}$ for each A_i. Therefore, we get $\bigsqcup f\{A_i\} \in (f\{\bigcup \mathcal{E}\})^c$.

Since f is continuous, we have $e_i = f \bigsqcup A_i = \bigsqcup f\{A_i\} \in (f\{\bigcup \mathcal{E}\})^c$. Hence, $B \subseteq (f\{\bigcup \mathcal{E}\})^c$. And we have $b = \bigsqcup B \in (f\{\bigcup \mathcal{E}\})^c$, a contradiction.

2. We want to show that
$$(\mathcal{P}_O(f^{-1})) \bigsqcup \mathcal{Y} = \bigsqcup (\mathcal{P}_O(f^{-1}))\{\mathcal{Y}\},$$
for every directed subset $\mathcal{Y} \subseteq \mathcal{P}_O(D_\beta)$.

By Fact A.42, each $Y \in \mathcal{Y}$ can be written as $\bigcup\{\uparrow e \mid e \in K(D_\beta), e \in Y\}$. We then have
$$\begin{array}{ll} & \bigsqcup \mathcal{Y} \\ = & \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq \bigcap \mathcal{Y}\} \\ = & \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq \bigcup\{\uparrow e \mid e \in K(D_\beta), e \in \bigcap \mathcal{Y}\}\} \\ = & \bigcup\{\uparrow e \mid e \in K(D_\beta), e \in \bigcap \mathcal{Y}\}. \end{array}$$

On the other hand, we have
$$\begin{array}{ll} & \bigsqcup (\mathcal{P}_O(f^{-1}))\{\mathcal{Y}\} \\ = & \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq \bigcap (\mathcal{P}_O(f^{-1}))\{\mathcal{Y}\}\} \\ = & \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq f^{-1}\{\bigcap \mathcal{Y}\}\}. \end{array}$$

Hence, it suffices to show that
$$(\mathcal{P}_O(f^{-1})) \bigcup\{\uparrow e \mid e \in K(D_\beta), e \in \bigcap \mathcal{Y}\} = \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq f^{-1}\{\bigcap \mathcal{Y}\}\}$$
to complete the proof.

Let e_x be a compact element in D_α. Then,
$$\begin{array}{ll} & e_x \in (\mathcal{P}_O(f^{-1})) \bigcup\{\uparrow e \mid e \in K(D_\beta), e \in \bigcap \mathcal{Y}\} \\ \Leftrightarrow & \text{there exists } e_y \in K(D_\beta) \text{ such that } e_y \sqsubseteq_{D_\beta} f\ e_x \text{ and } e_y \in \bigcap \mathcal{Y} \\ \Leftrightarrow & \text{there exists } e_y \in K(D_\beta) \text{ such that } f\ e_x \in \uparrow e_y \text{ and } \uparrow e_x \subseteq f^{-1}\{\uparrow e_y\} \subseteq f^{-1}\{\bigcap \mathcal{Y}\} \\ \Leftrightarrow & e_x \in \bigcup\{Y \mid Y \in \mathcal{P}_O(D_\beta), Y \subseteq f^{-1}\{\bigcap \mathcal{Y}\}\}. \end{array}$$

Since both of the above two sets are Scott–open, by Fact A.42, we conclude that they are the same.

◇

Lemma B.48 Let D_α and D_β be domain, $f \in D_\alpha \to D_\beta$, and B be closed in D_β. Then
$$\{x \mid x \in D_\alpha, fx \in B\} = \bigsqcup\{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))X \sqsubseteq_{\mathcal{P}_C(D_\beta)} B\}.$$

□

Proof. Let $\mathcal{X} = \{X \mid X \in \mathcal{P}_C(D_\alpha), (\mathcal{P}_C(f))X \sqsubseteq_{\mathcal{P}_C(D_\beta)} B\}$. We will show that $\{x \mid x \in D_\alpha, fx \in B\} \subseteq \bigsqcup \mathcal{X}$ and $\bigsqcup \mathcal{X} \subseteq \{x \mid x \in D_\alpha, fx \in B\}$.

We first note that $\mathcal{X}$ is closed in $\mathcal{P}_C(D_\alpha)$ by Corollary 2.6. We will show that $\mathcal{X}$ is also a directed subset of $\mathcal{P}_C(D_\alpha)$. Therefore, $\bigsqcup \mathcal{X} \in \mathcal{X}$. By the definition of $\mathcal{X}$, we then have $(\mathcal{P}_C(f))(\bigsqcup \mathcal{X}) \sqsubseteq_{\mathcal{P}_C(D_\beta)} B$. That is, $(\mathcal{P}_C(f))(\bigsqcup \mathcal{X}) \subseteq B$.

To show that $\mathcal{X}$ is a directed subset of $\mathcal{P}_C(D_\alpha)$, we observe that for every $X_0, X_1 \in \mathcal{X}$,

- $X_0 \sqcup X_1 = X_0 \cup X_1$ is closed in $\mathcal{P}_C(D_\alpha)$ and
- $(\mathcal{P}_C(f))(X_0 \sqcup X_1) = (\mathcal{P}_C(f))X_0 \sqcup (\mathcal{P}_C(f))X_1 \sqsubseteq_{\mathcal{P}_C(D_\beta)} B$.

That is, $X_0 \sqcup X_1 \in \mathcal{X}$; hence, $\mathcal{X}$ is directed.

Suppose that $a \in \bigsqcup\mathcal{X}$. Then $f\ a \in \{f\ x \mid x \in D_\alpha, x \in \bigsqcup\mathcal{X}\}$. But $\{f\ x \mid x \in D_\alpha,\ x \in \bigsqcup\mathcal{X}\} \subseteq (\mathcal{P}_C(f))(\bigsqcup\mathcal{X}) \subseteq B$. We then have $f\ a \in B$; hence, $a \in \{x \mid x \in D_\alpha, f\ x \in B\}$. That is, $\bigsqcup\mathcal{X} \subseteq \{x \mid x \in D_\alpha, fx \in B\}$.

Suppose that $a \in \{x \mid x \in D_\alpha, fx \in B\}$. Then $(\mathcal{P}_C(f)) \downarrow a \sqsubseteq_{\mathcal{P}_C(D_\beta)} B$. By the definition of $\mathcal{X}$, we have $\downarrow a \in \mathcal{X}$. Therefore, $\downarrow a \sqsubseteq_{\mathcal{P}_C(D_\alpha)} \bigsqcup\mathcal{X}$; hence, $a \in \bigsqcup\mathcal{X}$. That is, $\{x \mid x \in D_\alpha, f\ x \in B\} \subseteq \bigsqcup\mathcal{X}$.

This completes the proof. ◇

References

[1] *17th Annual ACM Symposium on Principles of Programming Languages.* A. C. M., January 1990. San Francisco, California, U. S. A.

[2] Samson Abramsky and Chris Hankin, editors. *Abstract Interpretation of Declarative Languages.* Elliss Horwood, 1987.

[3] Hendrik Pieter Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North–Holland, revised edition, 1984.

[4] D. Bjørner, A. P. Ershov, and N. D. Jones, editors. *Partial Evaluation and Mixed Computation.* North–Holland, 1988.

[5] Geoffrey L. Burn. A relationship between abstract interpretation and projection analysis (extended abstract). pages 151–156. In [1].

[6] Geoffrey L. Burn, Chris L. Hankin, and Samson Abramsky. Strictness analysis for higher–order functions. *Science of Computer Programming*, 7(3):249–278, 1986.

[7] Kei Davis and John Hughes, editors. *Functional Programming: Proceedings of the 1989 Glasgow Workshop.* Glasgow, Scotland, U. K., Springer–Verlag, 1990.

[8] Kei Davis and Philip Wadler. Backward strictness analysis: Proved and improved. pages 12–30. In [7].

[9] Peter Dybjer. Inverse image analysis. pages 21–30. In [13].

[10] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove, and Dana S. Scott. *A Compendium of Continuous Lattices.* Springer–Verlag, 1980.

[11] John Hughes. Backwards analysis of functional programs. pages 187–208. In [4].

[12] Gilles Kahn, editor. *Functional Programming Languages and Computer Architecture.* Portland, Oregon, U. S. A., September 1987. Lecture Notes in Computer Science, Volume 274, Springer–Verlag.

[13] Thomas Ottmann, editor. *Automata, Languages and Programming.* Karlsruhe, F. R. G., July 1987. Lecture Notes in Computer Science, Volume 267, Springer–Verlag.

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

[15] Joseph E. Stoy. *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory.* The M. I. T. Press Series in Computer Science. M. I. T. Press, 1977.

[16] Philip Wadler. Strictness analysis on non–flat domains (by abstract interpretation over finite domains). chapter 12, pages 246–265. In [2].

[17] Philip Wadler and R. J. M. Hughes. Projections for strictness analysis. pages 383–407. In [12].