Reversing Abstract Interpretations

John Hughes and John Launchbury*
Department of Computing Science,
University of Glasgow,
{rjmh, jl}@dcs.glasgow.ac.uk

1 Introduction

Many semantic analyses of functional languages have been developed using the Cousots' *abstract interpretation* framework [CC77]. Some, such as Mycroft's pioneering strictness analysis [Myc81] and Burn, Hankin and Abramsky's extension of it to higher-order [BHA86], operate on *abstract values* representing the past history of the computation, and are therefore called *forwards analyses.* Others, such as Wadler and Hughes' projection-based strictness analysis [WH87], or Hall's analysis of strictness patterns [Hal87] propagate *abstract contexts* representing the future of the computation, and are called *backwards analyses.* However, although the type of abstract information may suggest a "natural" direction, it is in fact possible to perform any analysis in either direction. The goal of this paper is to show how to reverse any given analysis.

Why might one prefer one direction of analysis over another? We shall draw an analogy with solving a differential equation on an interval. Solutions may be found by iterating from one end of the interval to the other, with the two possible directions corresponding to backwards and forwards analysis. But the purpose of an analysis is to answer a question, and such questions correspond to giving the boundary conditions at one end of the interval and asking for the function's value at the other. In such a case it's clearly preferable to start solving the equation at the end where the boundary conditions are known. Note that it's not impossible to work in the other direction—one can always use trial and error to find boundary conditions at the beginning that produce the right values at the end—but in general working in the "wrong" direction will require many solutions to be calculated where one would suffice in the other direction. We will see exactly this effect arising in the case of strictness analysis.

Every analysis associates with each function in the source program a corresponding abstract function. To reverse an analysis we have to "invert" these abstract functions. We begin by considering the conditions under which one function can be said to safely approximate the inverse of another. We show that there is a best re-

*Work supported by ESPRIT BRA 3124 - Semantique

versal of each abstract function; and how best reversals interact with the combining forms of a programming language.

Sometimes the best reversal of an abstract function carries less information than the original. This raises the question "How much less?" One way to compare abstract functions with their reversals is to reverse the reversal again, but this may lose still more information. However, there is a class of functions whose reversal carries exactly the same information, and which may therefore be reversed any number of times with no loss. These turn out to be *Galois connections*. There is a best approximating Galois connection to each abstract function, which provides an upper bound on the information lost by reversal.

The application we consider in this paper is the reversal of Burn, Hankin and Abramsky's strictness analysis. The analysis of the conditional proves hard to reverse; we therefore derive a rule for backwards analysis directly from the concrete semantics. The power of this backwards rule is incomparable with that of the forwards rule, disproving the old chestnut that conditionals are better analysed forwards. The analysis derived is a previously known backwards analysis, but it's relation to BHA and the corresponding proof of correctness were previously unknown.

We go on to consider Wadler's 4-point abstract domain for lists [Wad87]. The reversal of his analysis turns out to be simpler than the original. In fact, Wadler's forwards analysis of case expressions contains a complication which can be seen as necessary to obtain a good reversal of a backwards form!

Finally, we derive a backwards analysis of higher-order programs from the BHA forwards analysis. Perhaps not surprisingly, we fail to obtain a particularly accurate analysis.

2 Background

2.1 The Object Language

We will discuss analyses in the context of a simple typed functional language based on categorical notation. Types are base types (such as `Int`), and types built from them using $\times$, `List` (in section 5) and $\rightarrow$ (in section 6). Terms denote continuous functions, and are built from an unspecified collection of primitive functions using combining forms. The basic term syntax is

$$term \quad ::= \quad ide \mid term \circ term \mid < term, term > \mid \mu ide.\ term$$

Here $\circ$ denotes composition, $<\mathtt{f},\mathtt{g}>$ denotes the function $<\mathtt{f},\mathtt{g}>\ x = (\mathtt{f}\ x, \mathtt{g}\ x)$ and $\mu\mathtt{f}.\mathtt{H}(\mathtt{f})$ is the recursive function satisfying $\mu\mathtt{f}.\mathtt{H}(\mathtt{f}) = \mathtt{H}(\mu\mathtt{f}.\mathtt{H}(\mathtt{f}))$. The primitives include at least the projection functions π_1, π_2, and the constant functions $K_c\ x = c$. Although our language is monomorphic, for notational convenience we will allow polymorphic *primitives* such as $\sqcup : \forall X.\ X \times X \rightarrow X$. Occurrences of such primitives should be read as the appropriate member of a family of monomorphic functions.

We will consider extensions of the language with other combining forms such as the conditional $(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})$ (see section 3.4). In particular, we can extend this first-order language to a higher-order one by adding the combining form Λ (curry) and the polymorphic primitive ap, where $(\Lambda\mathtt{f})\ x = \lambda y.\mathtt{f}(x, y)$ and $\mathtt{ap}\ (f, x) = f\ x$.

2.2 Abstract Interpretation

We'll use the same language to express *abstract functions*, which we distinguish notationally using italics. We restrict the types over which abstract functions are defined to be *finite lattices*. This is consistent with [BHA86] and [WH87] for example, where the restriction is used to guarantee termination of analysis[1]. Finiteness is important here for a different reason: it means continuity reduces to monotonicity in the proofs which follow, and indeed some of the functions we construct need not be continuous in the infinite case.

Abstract functions come with a notion of *safe approximation*: we say it is safe to approximate *upwards* if an abstract function f can be replaced by any $f' \sqsupseteq f$ without compromising the correctness of conclusions drawn from the analysis. Less commonly, it may be safe to approximate downwards. When an analyser cannot predict which of two abstract functions f or g applies (for example, in the analysis of a conditional) it may safely approximate by $f \sqcup g$ if the direction of safe approximation is upwards, or by $f \sqcap g$ if it is downwards.

To define an analysis we associate each type A with a corresponding abstract type $A^\sharp$, and give a safety condition relating concrete functions $\mathtt{f} : A \to B$ to abstract functions $f : A^\sharp \to B^\sharp$ for a forwards analysis, or $f : B^\sharp \to A^\sharp$ for a backwards one. The safety condition tells us when an abstract function faithfully reflects the behaviour of the concrete one, and must be consistent with the notion of safe approximation for abstract functions. Since this condition relates the *semantics* of a concrete function to an abstract function it is not immediately useful in a compiler, but we can compute safe abstract functions for any term given abstract functions for the primitives and ways of deriving abstract functions for compound terms from those for their subterms. This process is called abstract interpretation.

2.3 The Burn, Hankin and Abramsky Framework

In the BHA approach, concrete and abstract types are related by a family of *abstraction functions* $abs_A : A \to A^\sharp$ and the safety condition relating $\mathtt{f} : A \to B$ to $f : A^\sharp \to B^\sharp$ is $abs_B \circ \mathtt{f} \sqsubseteq f \circ abs_A$.

Abstract values are associated with Scott-closed[2] sets of concrete values via *concretisation* functions, $conc_A\ a = \{x \mid abs_A\ x \sqsubseteq a\}$. The safety condition can be reformulated as $\forall x, a\ .\ x \in conc\ a \Rightarrow \mathtt{f}\ x \in conc(f\ a)$.

There is a best abstract function for each concrete function $\mathtt{f} : A \to B$ given by $\bigsqcup \circ \wp_H(abs_B \circ \mathtt{f}) \circ conc_A$ where $(\wp_H f)\ X = \{f\ x \mid x \in X\}^*$ and X^* denotes the Scott-closure of X. Products are abstracted as products, so $(A \times B)^\sharp = A^\sharp \times B^\sharp$ and $abs_{A\times B}\ (x, y) = (abs_A\ x, abs_B\ y)$.

The following theorems justify a very simple abstract interpretation of terms:

Theorem 1
If f and g are safe for $\mathtt{f}$ and $\mathtt{g}$ respectively, then
(i) $f \circ g$ is safe for $\mathtt{f} \circ \mathtt{g}$
(ii) $< f, g >$ is safe for $< \mathtt{f}, \mathtt{g} >$

[1] Although finiteness is commonly required there are other ways of ensuring termination — see [CC77].

[2] A set S is Scott-closed if it is downwards closed, and whenever all the elements of a chain lie in S, so does the limit.

Theorem 2
Let H and H be functionals such that whenever f is safe for f, then $H\ f$ is safe for H f. Then $\mu f.H\ f$ is safe for μf.H f.

Similar theorems must be proved for each proposed analysis. This approach extends very naturally to the higher-order case. We define $(A \to B)^\sharp = A^\sharp \to B^\sharp$ and $abs_{A\to B}\ \mathtt{f} = \bigsqcup \circ \wp_H(abs_B \circ \mathtt{f}) \circ conc_A$, with abstract interpretation justified by:

Theorem 3
If f is safe for f, then (i) Λf is safe for Λf, and (ii) *ap* is safe for ap.

Strictness analysis is cast in this framework by abstracting base types as the 2-point domain $\mathbf{2} = \{0 \sqsubseteq 1\}$, with abstraction defined by

$$abs_{Base}\ x = \begin{cases} 0 & \text{if } x = \bot \\ 1 & \text{otherwise} \end{cases}$$

It follows that all abstraction functions are strict and $\bot$-reflecting: in other words $abs\ x = \bot \Leftrightarrow x = \bot$. From this and the safety condition $abs \circ \mathtt{f} \sqsubseteq f \circ abs$ we see that if f is strict, f must be too. We can test for strictness of f by testing whether $f\ 0 = 0$.

2.4 Galois Connections

Definition
A *Galois connection* between lattices A and B is a pair of monotonic functions $f : A \to B$ and $g : B \to A$ such that $f \circ g \sqsupseteq id$ and $g \circ f \sqsubseteq id$, or equivalently $\forall x, y.\ g\ y \sqsubseteq x \Leftrightarrow y \sqsubseteq f\ x$. f is called the *upper component* and g is called the *lower component.* □

Theorem 4
Let (f, g) be a Galois connection. Then (i) $f\ \top = \top$ and $g\ \bot = \bot$, (ii) f distributes over $\sqcap$, g distributes over $\sqcup$, and (iii) $g\ y$ is the least x such that $y \sqsubseteq f\ x$ and $f\ x$ is the greatest y such that $g\ y \sqsubseteq x$.

Corollary 5
Each component of a Galois connection uniquely determines the other.

In view of the Corollary we will sometimes be sloppy and say "the Galois connection f" instead of "the upper-component of a Galois connection f".

Galois connections were used by the Cousots to relate abstraction and concretisation, and consequently often appear in papers on abstract interpretation. The use we are making of them is quite different: we use Galois connections as *abstract* functions, the Cousots used them as *abstraction* functions.

3 Reversing an Analysis

3.1 Safe Reversals

Suppose we're given an abstract function $f : A \to B$ to reverse. In general, f will not have an exact inverse and we will need to approximate. We therefore need to know

in which direction approximation is safe: suppose the safe direction is upwards so that any $f' \sqsupseteq f$ may safely be used instead of f. Furthermore, suppose the questions we want to answer are of the form

Does $y \sqsubseteq f\ x$?

Since it's safe to approximate f upwards, such questions can safely be answered 'yes' when the correct answer is 'no', but must never be answered 'no' when the correct answer is 'yes'. Thus, 'no' means 'no', whereas 'yes' means 'maybe'.

When can a reversed function $f^r : B \to A$ be used to answer such questions? Since f^r is a kind of inverse, we'll ask instead

Does $f^r\ y \sqsubseteq x$?

We can use the answer to this question as an answer to the previous one provided $y \sqsubseteq f\ x \Rightarrow f^r\ y \sqsubseteq x$, since then we can never answer 'no' by mistake (negate both sides to obtain the more intuitive implication).

Definition
f^r is a *safe reversal* of f if $\forall x, y\ .\ y \sqsubseteq f\ x \Rightarrow f^r\ y \sqsubseteq x$, or equivalently, if $f^r \circ f \sqsubseteq id$
□

Note that safe reversals are always strict, and that any $f^{r'} \sqsubseteq f^r$ is also a safe reversal of f. In other words, safe reversals can be safely approximated in the opposite direction from abstract functions.

3.1.1 Example

In the case of BHA strictness analysis the test for strictness is usually phrased slightly differently. For example, if f is an integer-valued function of three integer parameters and f is its abstract function, then strictness is tested in each argument separately by asking the questions

Does $f\ (0,1,1) = 0$?
Does $f\ (1,0,1) = 0$?
Does $f\ (1,1,0) = 0$?

Of course, we can instead ask

Does $1 \sqsubseteq f\ (0,1,1)$?
Does $1 \sqsubseteq f\ (1,0,1)$?
Does $1 \sqsubseteq f\ (1,1,0)$?

whose answers are the negations of those above. But suppose f^r is a safe reversal of f, and $f^r 1 = (1,1,0)$. Now we can answer the three questions by answering

Does $f^r 1 \sqsubseteq (0,1,1)$?
Does $f^r 1 \sqsubseteq (1,0,1)$?
Does $f^r 1 \sqsubseteq (1,1,0)$?

So with a *single* call of f^r, we discover that f is strict in its first and second arguments, but not necessarily in its third.

Forwards analysis may require many abstract evaluations to find all the strictness of a function, especially if its arguments are of complex types. The reversed analysis finds all the strictness in one abstract evaluation. Recalling our discussion of differential equations, this suggests that the boundary conditions for strictness analysis make it "naturally" a backwards analysis.

3.1.2 Computing Safe Reversals

Safe reversals of abstract functions can be computed efficiently if we know safe reversals for the primitives, and if we can derive safe reversals of compound terms from safe reversals of their subterms. We'll discuss primitives in the next subsection; the following theorem helps us do the latter.

Theorem 6
If f^r and g^r are safe reversals of f and g respectively, then (i) $g^r \circ f^r$ is a safe reversal of $f \circ g$, and (ii) $f^r \circ \pi_1 \sqcup g^r \circ \pi_2$ is a safe reversal of $< f, g >$.

Proof
(i) $(g^r \circ f^r) \circ (f \circ g) = g^r \circ f^r \circ f \circ g \sqsubseteq g^r \circ g \sqsubseteq id$
(ii) $(f^r \circ \pi_1 \sqcup g^r \circ \pi_2) \circ < f, g > = f^r \circ f \sqcup g^r \circ g \sqsubseteq id$ □

To find a safe reversal of recursive functions we need a safe reversal of $K_\perp$, the constant undefined function. One such is

$$K_\perp^r\, y = \begin{cases} \perp & \text{if } y = \perp \\ \top & \text{otherwise} \end{cases}$$

Now we can reverse recursive functions using the following theorem.

Theorem 7
Let H and H^r be functionals such that for all f, H^r maps safe reversals of f to safe reversals of $H(f)$. Then $\bigsqcap_{n=0}^{\infty} (H^r)^n(K_\perp^r)$ is a safe reversal of $\mu f.H(f)$.

Proof
Since $K_\perp^r \circ K_\perp \sqsubseteq id$, we can show by induction that $\forall n \;.\; (H^r)^n(K_\perp^r) \circ H^n(K_\perp) \sqsubseteq id$. But since we are working in finite lattices all ascending and descending chains are eventually stationary[3], so there is an N such that

$$\bigsqcap_{n=0}^{\infty} (H^r)^n(K_\perp^r) \circ \bigsqcup_{n=0}^{\infty} H^n(K_\perp) = (H^r)^N(K_\perp^r) \circ H^N(K_\perp) \sqsubseteq id$$

□

In applications the functional H will be built up using composition, tupling, and so on, and a suitable H^r will be constructed using the rules above.

[3]This theorem could be proved for infinite lattices using continuity, but its dual cannot.

3.2 Best Reversals

Since safe reversals can safely be approximated downwards, it's natural to ask whether there are best, or greatest safe reversals. The following definition and theorem assure us that there are.

Definition
Given any function f, we define $f^-\ y = \bigsqcap\{x \mid y \sqsubseteq f\ x\}$ □

Theorem 8
f^- is the greatest safe reversal of f.

Proof
It is clear from the definition that $y \sqsubseteq f\ x \Rightarrow f^-\ y \sqsubseteq x$, so f^- is a safe reversal of f. Moreover, it is the greatest safe reversal, for if f^r is any other safe reversal of f, then $y \sqsubseteq f\ x \Rightarrow f^r\ y \sqsubseteq x$, and so $f^r\ y \sqsubseteq \bigsqcap\{x \mid y \sqsubseteq f\ x\} = f^-\ y$ □

As a corollary to this theorem, we can now show that any function f^r is a safe reversal of f merely by showing that $f^r \sqsubseteq f^-$.

Best reversals of primitives are now easily calculated. For example,

$$
\begin{array}{lcl}
K_\perp^-\ y & = & \begin{cases} \perp & \text{if } y = \perp \\ \top & \text{otherwise} \end{cases} \\
id^-\ y & = & y \\
\pi_1^-\ y & = & (y, \perp) \\
\pi_2^-\ y & = & (\perp, y) \\
\sqcap^-\ y & = & (y, y) \\
\sqcup^-\ y & = & (\perp, \perp)
\end{array}
$$

Clearly the last of these loses all information; abstract functions involving $\sqcup$ are therefore hard to reverse accurately.

One may ask whether the methods above for reversing compound terms produce best reversals from best reversals. Unfortunately they do not. For example, $\sqcup \circ <f,f> = f$ and so $(\sqcup \circ <f,f>)^- = f^-$ but applying the methods developed yields

$$
\begin{array}{lcl}
(\sqcup \circ <f,f>)^- & \sqsupseteq & <f,f>^- \circ \sqcup^- \\
& \sqsupseteq & (f^- \circ \pi_1 \sqcup f^- \circ \pi_2) \circ \sqcup^- \\
& = & f^- \circ K_\perp \\
& = & K_\perp \qquad \text{[since all reversals are strict]}
\end{array}
$$

so all information is lost. It can therefore be worthwhile deriving special reversal rules for constructs defined as combinations of the primitives.

3.3 Reversible Analyses are Galois Connections

Suppose we are given a safe reversal f^r of f. Can we reconstruct (a safe approximation to) f from it? Reversals are just like abstract functions, except that it's safe

to approximate them downwards rather than upwards. Clearly we can construct a dual theory by inverting the ordering: f^{rr} will be a safe reversal of f^r if

$$\forall x, y \,.\, f^r\ y \sqsubseteq x \ \Rightarrow\ y \sqsubseteq f^{rr}\ x$$

or equivalently, $f^{rr} \circ f^r \sqsupseteq id$. We'll write the best reversal of f^r in this dual theory as $(f^r)^+$, and where there's no risk of confusion we'll be sloppy and write $(f^-)^+$ also as f^+.

It's easy to show that if f is an abstract function, then $f \sqsubseteq f^+$. In other words, the safe reversal of a safe reversal safely approximates the original abstract function. But what if f^+ is actually equal to f? In that case the two safety conditions can be combined to give

$$\forall x, y \,.\, f^-\ y \sqsubseteq x \ \Leftrightarrow\ y \sqsubseteq f^+\ x$$

This tells us that the two directions of analysis have exactly the same power. Any question of the form $y \sqsubseteq f^+\ x$ can be exactly answered by a question of the form $f^-\ y \sqsubseteq x$, and vice versa. Interestingly, it is also the condition under which f^+ and f^- form a Galois connection. Hence the slogan: *reversible analyses are Galois connections.* We can now strengthen Theorem 6 in a pleasing way.

Theorem 9
If f and g are (the upper components of) Galois connections, then

(i) $f \circ g$ is a Galois connection

(ii) $< f, g >$ is a Galois connection

Proof
The lower components of these Galois connections are given in Theorem 6, and the proof is very similar to the proof given there. □

Of the primitives discussed so far, id, π_1, π_2 and $\sqcap$ are all (the upper components of) Galois connections, and so can be analysed equally well in either direction. However, $K_\perp$ and $\sqcup$ are not. Their double reversals are

$$\begin{aligned} K_\perp^+\ x &= \begin{cases} \top & \text{if } x = \top \\ \perp & \text{otherwise} \end{cases} \\ \sqcup^+\ x &= \top \end{aligned}$$

It turns out that the triple reversals of these primitives are the same as their single reversals, so that $K_\perp^+$ and $\sqcup^+$ *are* Galois connections. Such cases are very important because it means that the double reversal of an abstract function is of exactly the same power as the single reversal. Thus the power of the single reversal may be directly compared with the original. In the case just above, for example, we can see that a backwards analysis using $\sqcup^-$ will have the same power as a forwards analysis that approximates $x \sqcup y$ by $\top$. It is clear that this is a very poor approximation.

We can extend the same idea to show that every abstract function has a best approximating Galois connection.

Theorem 10
For every abstract function f, there is a least $g \sqsupseteq f$ such that g is the upper component of a Galois connection.

Proof
We construct g as follows. We know that the double reversal of f satisfies $f \sqsubseteq f^+$, and therefore $f \sqsubseteq f^+ \sqsubseteq (f^+)^+ \sqsubseteq \ldots$ Because we are working in finite lattices, this increasing chain must eventually be stationary: call the limit g. Clearly $f \sqsubseteq g$. Moreover, since $g^+ = g$, g is a Galois connection.

It remains to show that if h is a Galois connection with $f \sqsubseteq h$, then $g \sqsubseteq h$ also. But, we know that best reversal is anti-monotonic, and so double reversal is monotonic. From $f \sqsubseteq h$ we may therefore conclude $f^+ \sqsubseteq h^+ = h$. By induction $f^{+^n} \sqsubseteq h$ for all n, and so $g \sqsubseteq h$. □

The only combining form we have not yet discussed is recursion. Since $K_\perp$ is not a Galois connection, it's hardly surprising that recursive functions are not necessarily Galois connections either. However, we can prove an analogue of Theorem 7.

Theorem 11
Let H be a functional which maps Galois connections to Galois connections. Then $\bigsqcup_{n=0}^{\infty} H^n(K_\perp^+)$ is a Galois connection, with lower component $\bigsqcap_{n=0}^{\infty} (H^-)^n(K_\perp^-)$ where $H^-(g^-) = (H(g^+))^-$

Proof
Similar to Theorem 7. □

Thus backwards analysis of a recursive function has the same power as forwards analysis using a variant of recursion which starts from $K_\perp^+$ rather than $K_\perp$. By inspection, $K_\perp^+$ is the hyper-strict function, and so at least for the purpose of strictness analysis it seems that little useful information will be lost.

3.4 Example: Reversing Conditionals

In this section we apply the theory developed so far to the analysis of conditionals. The conditional construct we analyse works at the function level:

$$(\mathtt{p} \rightarrow \mathtt{f};\mathtt{g})\ \mathtt{x} = \begin{cases} \mathtt{f\ x} & \text{if } \mathtt{p\ x} = \mathtt{true} \\ \mathtt{g\ x} & \text{if } \mathtt{p\ x} = \mathtt{false} \\ \perp & \text{otherwise} \end{cases}$$

To give the BHA abstract interpretation we need a new operator:

$$x \rhd y = \begin{cases} \perp & \text{if } x = 0 \\ y & \text{otherwise} \end{cases}$$

Promoting $\rhd$ to operate on functions, we can write the abstract interpretation of a conditional as $p \rhd (f \sqcup g)$.

How can we reverse this abstract function? It turns out that $\rhd$ is a Galois connection with lower component

$$\rhd^- y = \begin{cases} (0, \perp) & \text{if } y = \perp \\ (1, y) & \text{otherwise} \end{cases}$$

from which we can infer

$$(p \rhd h)^- y \sqsupseteq \begin{cases} \bot & \text{if } y = \bot \\ p^- \ 1 \sqcup h^- y & \text{otherwise} \end{cases}$$

Intuitively, for the result of a conditional to be defined, the condition must be defined and the branches must be sufficiently defined.

We still need to reverse $(f \sqcup g)$, which we can do as follows:

$$\begin{array}{lcl} (f \sqcup g)^- & = & (\sqcup \circ < f, g >)^- \\ & \sqsupseteq & < f, g >^- \circ \sqcup^- \\ & = & < f, g >^- \circ K_\bot \\ & = & K_\bot \end{array}$$

Using this reversal we find

$$\begin{array}{lcl} (p \rhd (f \sqcup g))^- & \sqsupseteq & < p, f \sqcup g >^- \circ \rhd^- \\ & \sqsupseteq & (p^- \circ \pi_1 \sqcup (f \sqcup g)^- \circ \pi_2) \circ \rhd^- \\ & = & p^- \circ \pi_1 \circ \rhd^- \end{array}$$

That is,

$$(p \rhd (f \sqcup g))^- y \sqsupseteq \begin{cases} \bot & \text{if } y = \bot \\ p^- \ 1 & \text{otherwise} \end{cases}$$

If p is a Galois connection, then this is the lower component of a Galois connection whose upper component is $p \rhd \top$. Thus backwards analysis of a conditional is equivalent to forwards analysis where we ignore the branches and simply use the strictness in the condition.

In some cases this is the best we can do. For example, consider the function cond defined by $\mathsf{cond} = \pi_1 \rightarrow \pi_2; \pi_3$. The best reversal of $\pi_2 \sqcup \pi_3$ really is $K_\bot$, and so all we can say about cond is that it is strict in its first argument. However, if f and g have some strictness in common then we may be able to find a much better reversal of $f \sqcup g$:

$$\begin{array}{lcl} (f \sqcup g)^- y & = & \bigsqcap \{x \mid y \sqsubseteq f \ x \sqcup g \ x\} \\ & = & \bigsqcap \{x \mid y \sqsubseteq y_1 \sqcup y_2 \ \wedge \ y_1 \sqsubseteq f \ x \ \wedge \ y_2 \sqsubseteq g \ x\} \\ & \sqsupseteq & \bigsqcap \{x \mid y \sqsubseteq y_1 \sqcup y_2 \ \wedge \ f^- \ y_1 \sqsubseteq x \ \wedge \ g^- \ y_2 \sqsubseteq x\} \\ & = & \bigsqcap \{f^- \ y_1 \sqcup g^- \ y_2 \mid y \sqsubseteq y_1 \sqcup y_2\} \end{array}$$

Although we've now expressed a safe reversal of $f \sqcup g$ in terms of f^- and g^- the need to consider all $\sqcup$-factorisations of y makes this formula unsuitable for use in practice: in general there are too many of them. In the particular case when y is an element of the two-point domain, however, it can be simplified to

$$(f \sqcup g)^- y \sqsupseteq f^- y \sqcap g^- y$$

In the next section we'll show that, in fact, this form can always be used.

4 Relating Backwards Analysis to the Concrete Semantics

So far we have studied reversal of *abstract functions*, using only the notion of safe approximation of one abstract function by another and, except for examples, have made no reference to the concrete semantics. The theory is therefore applicable to any analysis, including those such as Wadler and Hughes' projection analysis which do not fit the BHA framework. Now we restrict ourselves to this framework: not surprisingly, we can derive better results in this special case.

BHA abstract functions satisfy the following safety condition: an abstract function f is *safe* for a concrete function $\mathtt{f}$ if $abs \circ \mathtt{f} \sqsubseteq f \circ abs$. If f^r is a safe reversal of f, then we have $f^r \circ abs \circ \mathtt{f} \sqsubseteq f^r \circ f \circ abs \sqsubseteq abs$. We can take this relationship between f^r and $\mathtt{f}$ as a *definition* of safety for backwards abstract functions.

Definition
An abstract function f^r is *safe backwards* for a concrete function $\mathtt{f}$ if $f^r \circ abs \circ \mathtt{f} \sqsubseteq abs$, or equivalently $\forall x, a \;.\; a \sqsubseteq abs\ (\mathtt{f}\ x) \Rightarrow f^r\ a \sqsubseteq abs\ x$. □

That is, if $\mathtt{f}$'s result is at least as defined as a, then $\mathtt{f}$'s argument must be at least as defined as $f^r\ a$. Clearly, this safety condition justifies the test for strictness developed in section 3.1.

We can now construct a theory of backwards abstract interpretation dual to BHA. We associate abstract values with Scott-open sets[4] via a concretisation function $conc\ a = \{x \mid a \sqsubseteq abs\ x\}$. The safety condition can then be re-expressed as

$$\forall x, a \;.\; \mathtt{f}\ x \in conc\ a \Rightarrow x \in conc\ (f^r\ a)$$

Scott-open sets form a complete lattice ordered by superset (isomorphic to the Hoare power domain including $\{\}$); $conc$ and $\bigsqcap$ are monotonic.

There is a best backwards abstract function for each concrete function $\mathtt{f}$ given by

$$\mathtt{f}^\sharp = \bigsqcap \cdot \circ\ \wp_O abs\ \circ\ \mathtt{f}^{-1} \circ conc$$

where $(\wp_O\ f)\ X = \{f\ x \mid x \in X\}^\circ$ and $\mathtt{f}^{-1}\ Y = \{x \mid \mathtt{f}\ x \in Y\}$. Here X° denotes the *interior* of the upward closure of X.

The following theorems, analogous to Theorems 6 and 7, enable us to compute backwards abstract functions by abstract interpretation.

Theorem 12
If f^r and g^r are safe backwards for $\mathtt{f}$ and $\mathtt{g}$ respectively, then

(i) $g^r \circ f^r$ is safe backwards for $\mathtt{f} \circ \mathtt{g}$

(ii) $f^r \circ \pi_1 \sqcup g^r \circ \pi_2$ is safe backwards for $< \mathtt{f}, \mathtt{g} >$.

Proof
omitted for space reasons. □

[4] A set S is *Scott-open* if it is upwards-closed, and whenever the limit of a chain $\bigsqcup_i x_i \in S$, there is some n such that $x_n \in S$. Equivalently, a set is Scott-open if its complement is Scott-closed.

Theorem 13
Let H and H^r be functionals such that whenever f^r is safe backwards for f then $H^r\ f^r$ is safe backwards for H f. Then $\bigsqcap_{n=0}^{\infty} (H^r)^n(K_\perp^-)$ is safe backwards for μf.H f.

Proof

$$
\begin{array}{lll}
(\mu \mathtt{f}.\mathtt{H}\ \mathtt{f})\ x \in conc\ a & & \\
\Rightarrow \bigsqcup_{n=0}^{\infty} \mathtt{H}^n(K_\perp)\ x \in conc\ a & & \\
\Rightarrow \exists n.\ \mathtt{H}^n(K_\perp)\ x \in conc\ a & & \text{[since } conc\ a \text{ is Scott-open]} \\
\Rightarrow \exists n.\ x \in conc((H^r)^n(K_\perp^-)\ a) & & \text{[by safety of } H^r\text{]} \\
\Rightarrow x \in conc(\bigsqcap_{n=0}^{\infty} (H^r)^n(K_\perp^-)\ a) & &
\end{array}
$$

□

Clearly a safe reversal of a safe forwards abstract function is safe backwards, but our interest is in safe backwards abstract functions which are *not* safe reversals of forwards ones. In particular consider the conditional $(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})$. The best safe backwards abstract function is $(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})^\sharp = \bigsqcap \circ \wp_O abs \circ (\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})^{-1} \circ conc$. But

$$
(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})^{-1}\ S = \begin{cases} \{\perp\}^\uparrow & \text{if } \perp \in S \\ (\mathtt{p}^{-1}\{\mathtt{true}\} \cap \mathtt{f}^{-1}\ S) \cup & \\ \quad (\mathtt{p}^{-1}\{\mathtt{false}\} \cap \mathtt{g}^{-1}\ S) & \text{otherwise} \end{cases}
$$

where $\{\perp\}^\uparrow$ is the upwards closure of $\{\perp\}$. Using this, and the fact that *abs* (and therefore *conc*) are strict and $\perp$-reflecting we obtain,

$$
(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})^\sharp\ y \sqsupseteq \begin{cases} \perp & \text{if } y = \perp \\ \mathtt{p}^\sharp\ 1 \sqcup (\mathtt{f}^\sharp\ y \sqcap \mathtt{g}^\sharp\ y) & \text{otherwise} \end{cases}
$$

and so $(\mathtt{p} \rightarrow \mathtt{f}; \mathtt{g})^\sharp \sqsupseteq \sqcup \circ (\mathtt{p}^\sharp \times (\mathtt{f}^\sharp \sqcap \mathtt{g}^\sharp)) \circ \rhd^-$.

There are functions that can be shown strict using this rule that cannot be shown strict by the forwards analysis. An example is $+ \circ (\pi_1 \rightarrow < K_1, \pi_2 >; < \pi_2, K_1 >)$. Backwards analysis shows

$$
\begin{array}{ll}
(+ \circ (\pi_1 \rightarrow < K_1, \pi_2 >; < \pi_2, K_1 >))^\sharp\ 1 & \\
= (\pi_1 \rightarrow < K_1, \pi_2 >; < \pi_2, K_1 >)^\sharp\ (1,1) & \\
= \pi_1^\sharp\ 1 \sqcup (< K_1, \pi_2 >^\sharp\ (1,1) \sqcap < \pi_2, K_1 >^\sharp\ (1,1)) & \\
= (1,0) \sqcup ((K_1^\sharp\ 1 \sqcup \pi_2^\sharp\ 1) \sqcap (\pi_2^\sharp\ 1 \sqcup K_1^\sharp\ 1)) & \\
= (1,0) \sqcup (((0,0) \sqcup (0,1)) \sqcap ((0,1) \sqcup (0,0))) & \\
= (1,0) \sqcup ((0,1) \sqcap (0,1)) & \\
= (1,1) &
\end{array}
$$

and so the function is strict in both arguments. Forwards analysis cannot discover strictness in the second argument, because when it has abstract value *0* then the values of the two branches of the conditional are $(1,0)$ and $(0,1)$, and taking the least upper bound loses the information that the argument was *0*. This example has also been noticed by Hunt [Hun91].

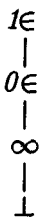
It is not true, therefore, that conditionals are "good" forwards and "bad" backwards. They are bad in both directions, but in different ways! An analyser which

repeatedly worked backwards and forwards, using the results of each stage to improve the next, could discover more information than an analyser working in either direction alone.

The backwards analysis we have derived in this section is essentially the same as Johnsson's [Joh81] or the simplest strictness analysis discussed in [Hug88]. It can also be thought of as an abstraction of Dybjer's inverse image analysis [Dyb91], which also used inverse images of Scott-open sets.

5 Wadler's 4-point Domain

In this section we consider the abstraction of lists of atomic values by elements of Wadler's 4-point domain [Wad87]. The abstract domain is



$\bot$ abstracts just the undefined list; ∞ abstracts lists whose last tail is $\bot$ and their limits, infinite lists; $0\in$ abstracts lists ending in `nil` and containing an undefined element; $1\in$ abstracts lists ending in `nil` all of whose elements are defined. For example,

$$\begin{array}{lcl} \mathit{abs}\ \texttt{(cons 1 (cons 2 }\bot\texttt{))} & = & \infty \\ \mathit{abs}\ \texttt{[1,2,}\bot\texttt{]} & = & 0\in \\ \mathit{abs}\ \texttt{[1,2]} & = & 1\in \end{array}$$

If `f`'s abstract function maps ∞ to $\bot$ we may conclude that `f` is tail-strict; if it maps $0\in$ to $\bot$ we may conclude that `f` is head-and-tail-strict.

Lists are built using `cons` and `nil` and taken apart by pattern matching. Wadler gives a special rule for analysing `case` expressions, but we will instead simulate pattern-matching with the functions `null` and `uncons`:

$$\texttt{uncons}\ xs \;=\; \begin{cases} (x, xs') & \text{if } xs = \texttt{cons}\ x\ xs' \\ \bot & \text{otherwise} \end{cases}$$

The abstract value of `nil` is $1\in$, and the abstraction of `cons` is given below.

cons	$\bot$	∞	$0\in$	$1\in$
0	∞	∞	$0\in$	$0\in$
1	∞	∞	$0\in$	$1\in$

For our analysis of conditionals of the form (`null` $\rightarrow$ `f;g`) to match Wadler's rule for `case` in accuracy, we have to abstract `null`'s boolean result by an element of the four-point domain $\{\bot, \mathit{true}, \mathit{false}, \top\}$. With this abstraction of booleans better

forwards and backwards analyses of conditionals can be derived: the new backwards rule is

$$(\mathtt{p} \to \mathtt{f};\mathtt{g})^\sharp\ y \sqsupseteq \begin{cases} \bot & \text{if } y = \bot \\ (\mathtt{p}^\sharp\ \mathit{true} \sqcup \mathtt{f}^\sharp\ y) \sqcap & \\ \quad (\mathtt{p}^\sharp\ \mathit{false} \sqcup \mathtt{g}^\sharp\ y) & \text{otherwise} \end{cases}$$

The abstractions of `null` and `uncons` are now:

null	
$\bot$	$\bot$
∞	*false*
$0\in$	*false*
$1\in$	$\top$

uncons	
$\bot$	$(0,\bot)$
∞	$(1,\infty)$
$0\in$	$(1,1\in)$
$1\in$	$(1,1\in)$

But there is a problem — *uncons* does not distinguish $0\in$ from $1\in$! The reason is that $0\in$ = *cons 1* $0\in$ = *cons 0* $1\in$, and *uncons* must approximate both possibilities by their least upper bound. The resulting analysis has very little power, which is why Wadler gave a special rule for entire `case` expressions.

But now consider a backwards analysis. K_{nil}, *cons*, and *null* are all Galois connections and so may be reversed at once. Reversing *uncons* is pointless—it would produce an equally uninformative backwards abstract function—but we can instead determine the best backwards abstract function for `uncons`. It is:

uncons^r	$\bot$	∞	$0\in$	$1\in$
0	$\bot$	∞	$0\in$	$0\in$
1	∞	∞	$0\in$	$1\in$

(To interpret this table intuitively, think of the first argument as the demand for the head of a `cons`-cell, and the second argument as the demand for the tail. The result is then the demand for the whole `cons`-cell. $1\in$ should be interpreted as a head-and-tail-strict demand, and $0\in$ as a tail-strict demand.)

Now all four values of the second argument are properly distinguished, and indeed an accurate backwards analysis can be based on these functions[5]. It corresponds to projection-based strictness analysis with the projections for head-strictness discarded [WH87, Bur90]. But uncons^r is not the lower component of a Galois connection since there is no greatest argument mapped to $0\in$, and hence no equally powerful forwards function exists.

We can compare this to the example in section 4 of a function where backwards analysis is more accurate than forwards: the need for forwards analysis to approximate $(0,1)$ and $(1,0)$ by $(1,1)$ in that example is analogous to the need to approximate *cons 0* $1\in$ and *cons 1* $0\in$ by *cons 1* $1\in$ here.

What if we reverse this backwards analysis to derive a more accurate forwards one? We model `case` constructs by $\mathtt{case}(n,f) = \mathtt{null} \to K_n;\ \mathtt{f} \circ \mathtt{uncons}$. The interesting term here is $\mathtt{f} \circ \mathtt{uncons}$. Given a forwards abstract function f for `f`, a safe backwards abstract function for this term is $\mathit{uncons}^r \circ f^-$. So a safe forwards abstract function for the composition is

$$(\mathit{uncons}^r \circ f^-)^+\ x = \sqcup\{y \mid (\mathit{uncons}^r \circ f^-)\ y \sqsubseteq x\}$$

[5]Choosing `hd` and `tl` as primitives instead of `uncons` does *not* lead to a good analysis. The best backwards abstract function for `tl` is $tl^r\ y = \mathit{uncons}^r\ (0,y)$ corresponding to the first row of the table, which again fails to distinguish $0\in$ from $1\in$.

Taking x to be $0\in$ for example, the right hand side is

$$\begin{array}{l} \sqcup\{y \mid uncons^r\ (f^-\ y) \sqsubseteq 0\in\} \\ \quad = \sqcup\{y \mid f^-\ y \sqsubseteq (0,1\in) \vee f^-\ y \sqsubseteq (1,0\in)\} \\ \quad = \sqcup\{y \mid f^-\ y \sqsubseteq (0,1\in)\} \sqcup \sqcup\{y \mid f^-\ y \sqsubseteq (1,0\in)\} \\ \quad = f^+\ (0,1\in) \sqcup f^+\ (1,0\in) \end{array}$$

The other cases are similar, but simpler since there is a unique largest value mapped below x by $uncons^r$. Using this abstract function and interpreting the other parts of $\mathsf{case}(n,f)$ in the standard way leads us to

$$case(n,f) = \begin{cases} \bot & \text{if } x = \bot \\ f^+\ (1,\infty) & \text{if } x = \infty \\ f^+\ (1,0\in) \sqcup f^+\ (0,1\in) & \text{if } x = 0\in \\ n \sqcup f^+\ (1,1\in) & \text{otherwise} \end{cases}$$

which is almost exactly Wadler's rule. The difference is that Wadler omitted the double reversal of f that appears here. Of course the double reversal is unnecessary, but to derive this via reversal we need theory developed in [HL91].

6 Higher-order Functions

Since one of the strengths of BHA analysis is its ability to handle higher-order functions, it's natural to ask what happens when we reverse the corresponding abstract functions. Unfortunately, the reversals are not very informative. This is not surprising since backwards analyses in general have difficulty with higher-order functions.

Consider first ap, with type $(X \to Y) \times X \to Y$. Its best reversal is

$$\begin{array}{rcl} ap^-\ y & = & \bigsqcap\{(f,x) \mid y \sqsubseteq f\ x\} \\ & = & \bigsqcap\{([x \mapsto y],x) \mid x \in X\} \\ & = & (\bigsqcap\{[x \mapsto y] \mid x \in X\}, \bigsqcap\{x \mid x \in X\}) \\ & = & ([\top \mapsto y], \bot) \end{array}$$

where $[x \mapsto y]$ is the step function that maps any $x' \sqsupseteq x$ to y and all other arguments to $\bot$. This is the lower component of a Galois connection whose upper component is $ap^+\ (f,x) = f\ \top$. Thus all of the information about strictness in the argument is lost: backwards analysis can only discover strictness in the *function*.

In the case of currying,

$$\begin{array}{rcl} (\Lambda f)^-\ g & = & \bigsqcap\ \{a \mid g \sqsubseteq (\Lambda f)\ a\} \\ & = & \bigsqcap\ \{a \mid \forall x\ .\ g\ x \sqsubseteq f(a,x)\} \\ & \sqsupseteq & \bigsqcap\ \{a \mid \forall x\ .\ f^r(g\ x) \sqsubseteq (a,x)\} \\ & = & \bigsqcap\ \{a \mid \forall x\ .\ \pi_1(f^r(g\ x)) \sqsubseteq a \wedge \forall x\ .\ \pi_2(f^r(g\ x)) \sqsubseteq x\} \\ & = & \bigsqcap\ \{a \mid \pi_1(f^r(g\ \top)) \sqsubseteq a \wedge \pi_2 \circ f^r \circ g \sqsubseteq id\} \\ & = & \begin{cases} \pi_1(f^r(g\ \top)) & \text{if } \pi_2 \circ f^r \circ g \sqsubseteq id \\ \top & \text{otherwise} \end{cases} \end{array}$$

where f^r is a safe reversal of f. If f is a Galois connection then this is the lower component of a Galois connection with upper component

$$(\Lambda f)^+\ a = \begin{cases} \top & \text{if } a = \top \\ (\Lambda f)\ a & \text{otherwise} \end{cases}$$

from which we see that backwards analysis cannot discover strictness in the second argument of a curried function, since this is equivalent to testing whether $(\Lambda f)^+ \top \bot = \bot$.

7 Relational Reversal

As we've seen, the reversal of an analysis is usually less accurate than the original. However, by working with *sets* of abstract values it's possible to derive an analysis in the opposite direction with *equal* power. Such an analysis is called *relational.*

The basic idea is to promote each abstract function f to $\wp_O f$, operating on upwards-closed sets of abstract values. Whatever f is, it turns out that $\wp_O f$ is the upper component of a Galois connection, with lower component f^{-1}. So backwards abstract functions of the form f^{-1} carry just as much information as the original functions f. Unfortunately, relational analyses seem to be far too costly to use in practice.

One compromise is to combine a locally relational analysis with either backwards or forwards non-relational analyses; the idea being to use the rather expensive relational analysis just for small parts of a program that would be analysed badly by a non-relational method. Within those parts we can mix backwards and forwards abstract functions. For instance, Wadler's rather tricky analysis of case expressions can be derived as a locally relational combination of the accurate backwards abstract function for uncons with the forwards abstract functions used in BHA strictness analysis.

These results are beyond the scope of this article. They appear in a companion paper [HL91], where we provide generalised backwards and forwards safety conditions relating relational abstract functions to the concrete semantics, and show that a relational analysis may be used as part of a non-relational analysis in the same direction.

8 Related Work and Conclusions

Strictness analysis has given rise to a rich variety of analyses, both forwards and backwards, and the relationship between these has not always been clear. Not only are the directions of analysis often different, but commonly so are the abstract values and their interpretations. Working towards a unified understanding, Burn showed the relationship between BHA strictness analysis and Wadler and Hughes' projection-based strictness analysis through the use of so-called "smash projections' [Bur90]. This allowed the results of each analysis to be related to the results of the other.

Soon afterwards, Hunt presented a forwards strictness analysis based on partial equivalence relations (PERs) [Hun90]. These were particularly interesting as most of the PERs used at the ground types corresponded exactly with projections. In particular, the ever elusive property of head-strictness was captured. However in order for the analysis to be able to derive head-strictness information a double analysis within the case construct was required. Again, this may be viewed as an

instance of obtaining the best reversal of a backwards analysis by considering the case construct as a whole.

Meanwhile, spurred by the discovery of a "naturally forwards" projection-based analysis[6] [Lau89, Lau91], Hughes and Launchbury studied a direction-independent formulation of projection analysis [HL90], in order to assess when a view of the analysis from one direction may equal or be superior to a view from the other. The concept of Galois connections arose here as a means of demonstrating equality. Following this lead, Hunt reformulated much of [HL90] in terms Scott-closed sets, so divesting it of its dependence on projections [Hun91].

The present paper develops the use of Galois connections as abstract functions (i.e. *within* an analysis), and shows that such abstract functions may safely be reversed with no loss of accuracy. Furthermore, *any* abstract function may be safely reversed, though possibly losing information in the process. In the particular case where the reversal is itself a Galois connection, its reduced power may be compared against the original by reversing once more to obtain an abstract function in the original direction having the same power as the reversal.

These ideas and methods were then applied to BHA style abstract interpretation, and provided a link between this and a previously unconnected backwards analysis. In an effort to improve the reversal of the conditional we showed that the best backwards abstraction of the conditional is *incomparable* with the best forwards abstraction. Consequently, neither forwards nor backwards analysis of the conditional may be said to be superior to the other.

Wadler's 4-point abstract domain requires a special interpretation of the case construct to achieve good results. With the experience of reversals, we were able to see exactly where a naive abstract interpretation would lose information: uncons has a good backwards abstraction, but a poor forwards abstraction. Unfortunately the non-relational techniques of this paper are insufficiently powerful to derive Wadler's rule for case directly, but they were able to produce a very similar version.

Finally we applied the techniques to higher order constructs, in order to obtain a backwards analysis of higher order functions. We obtained a simple reversal which may be of some use in practice, but one whose power is significantly less than the forwards version.

Recent work by the Nielsons on complexity measures in abstract interpretation has an interesting connection with the work here [NN92]. They show that finding fixed points over lattices of *completely additive* functions may require at most a quadratic number of unfoldings, whereas general fixpoint finding is exponential. As completely additive functions are lower components of Galois connections, our result that every abstract function has a best approximating Galois connection (obtained by repeated reversal) may be seen as a generic method for deriving cheap approximating analyses.

Although the development of this paper has been with an eye on strictness analysis, many of the results are further reaching: strictness analysis is used mainly as a pedagogic tool, and the techniques may be applied to other analyses.

[6] namely binding-time analysis, as used in partial evaluation

References

[AH87] S.Abramsky and C.L.Hankin eds., *Abstract Interpretation of Declarative Languages.* Ellis Horwood, Chichester, England, 1987.

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

[Bur90] G.L.Burn, *A relationship between abstract interpretation and projection analysis*, POPL, 1990.

[CC77] P.Cousot and R.Cousot, *Abstract Interpretation: A unified lattice model for static analyses of programs by construction of approximation of fixpoints*, POPL, 1977.

[Dyb91] P.Dybjer, *Inverse Image Analysis Generalises Strictness Analysis*, Information and Computation 90, 2, 1991, pp 194-216.

[Hal87] C.V.Hall, *Strictness Analysis Applied to Programs with Lazy List Constructors*, Ph.D. thesis, Indiana University, 1987.

[HL90] R.J.M.Hughes and J.Launchbury, *Towards Relating Forwards and Backwards Analyses*, Glasgow Functional Programming, Ullapool, In *Workshops in Computing*, S-V, 1991.

[HL91] R.J.M.Hughes and J.Launchbury, *Locally Relational Abstract Interpretation*, in preparation, Glasgow University, 1991.

[Hug88] R.J.M.Hughes, *Backwards Analysis of Functional Programs*, In D. Bjørner, A. Ershov and N.D. Jones eds. *Partial Evaluation and Mixed Computation*, Proc. IFIP TC2 Workshop, Denmark, Oct 1987; North-Holland, 1988.

[Hun90] S.Hunt, *PERs generalise projections for strictness analysis*, Glasgow Functional Programming, Ullapool, In *Workshops in Computing*, S-V, 1991.

[Hun91] S.Hunt, *Forwards and Backwards Strictness Analysis: Continuing the Comparison*, unpublished draft, Imperial College, 1991.

[Joh81] T.Johnsson, *Detecting when Call-by-Value can be used instead of Call-by-Need*, Programming Methodology Group, PMG-14, Chalmers, Gothenburg, 1981.

[Lau89] J.Launchbury. *Projection Factorisations in Partial Evaluation.* Ph.D. Thesis, Glasgow University, 1989; *Distinguished Dissertations in Computer Science*, Vol 1, C.U.P. 1991.

[Lau91] J.Launchbury. *Strictness and Binding-Time Analyses: Two for the Price of One*, SIGPLAN PLDI, Toronto, 1991.

[Myc81] A.Mycroft, *Abstract interpretation and optimising transformations for applicative languages*, Ph.D. thesis, University of Edinburgh, 1981.

[NN92] H.R.Nielson and F.Nielson, *Bounded Fixed Point Iteration*, POPL 92.

[WH87] P.Wadler and R.J.M.Hughes, *Projections for Strictness Analysis*, FPCA 87.

[Wad87] P.Wadler, *Strictness analysis on non-flat domains*, In [AH87], 1987.