Bounded fixed point iteration*

Hanne Riis Nielson Flemming Nielson

Computer Science Department
Aarhus University
Denmark

July 1991

Abstract

In the context of abstract interpretation for languages without higher-order features we study the number of times a functional need to be unfolded in order to give the least fixed point. For the cases of total or monotone functions we obtain an exponential bound and in the case of strict and additive (or distributive) functions we obtain a quadratic bound. These bounds are shown to be tight in that sufficiently long chains of functions can be shown to exist. Specialising the case of strict and additive functions to functionals of a form that would correspond to iterative programs we show that a linear bound is tight. This is related to the analyses studied in the literature (including strictness analysis).

1 Introduction

We consider the problem of computing fixed points in static program analysis. The whole purpose of static analysis is to get information about programs

*A very preliminary version of this paper was entitled "the Complexity of Static Program Analysis".

without actually running them and it is important that the analyses always terminate. In general, the analysis of a recursive (or iterative) program will itself be recursively defined and it is therefore important to "solve" this recursion such that termination is ensured.

In the denotational approach to static analysis the problem is addressed as follows. To each program the analysis associates an element d of a complete lattice $(D, \sqsubseteq)$ of abstract values. In the case of an iterative or recursive programming construct the value d is determined as the least fixed point, $\mathit{FIX}\ H$, of a continuous functions $H : D \rightarrow D$. Formally, the fixed point of H is defined by

$$\mathit{FIX}\ H = \sqcup\{H^i\bot \mid i \geq 0\}$$

where $\bot$ is the least element of D and $\sqcup$ is the least upper bound operation on D. It is well-known that the iterands $H^i\bot$ form an increasing chain in D and that

$$\text{if } H^k\bot = H^{k+1}\bot \text{ for some } k \geq 0 \text{ then } \mathit{FIX}\ H = H^k\bot$$

So the obvious algorithm for computing $\mathit{FIX}\ H$ will be to determine the iterands $H^0\bot, H^1\bot, \cdots$ one after the other while testing for stabilisation, i.e. equality with the predecessor. The cost of this algorithm depends on

- the number k of iterations needed before stabilisation,
- the cost of comparing two iterands, and
- the cost of computing a new iterand.

We shall study how to minimise the cost of the above algorithm for a *first-order framework* of static program analysis. We shall assume that the lattice D has the form

$$A^p \rightarrow B^q$$

where A and B are *finite complete lattices* and p and q are positive numbers. A number of interesting analyses for first-order functions languages fall within this framework, for example forward and backward strictness analyses [M81, WH87], constant propagation [NN89], liveness analysis [NN89] and demand analysis [BH89]. Also denotations formulations of many traditional analyses for imperative languages [ASU86, MR90] can be formulated within the framework. The running example of this paper is a variant of the definition-use analysis and it is presented in a denotational style in Appendix A with correctness considerations in Appendix B. (The motivation behind the analysis and its correctness are described at length in [NN92].)

We shall consider three versions of the framework:

- the *general framework* where functions of $A^p \to B^q$ only are required to be total; this is written $A^p \to_t B^q$.
- the *monotone framework* where functions of $A^p \to B^q$ must be monotone; this is written $A^p \to_m B^q$.
- the *completely additive framework* where functions of $A^p \to B^q$ must be strict and additive (or distributive); this is written $A^p \to_{sa} B^q$.

We show that the number k of iterations needed to compute the fixed point of an arbitrary continuous functional H is at most

- *exponential* in the general and the monotone frameworks, and
- *quadratic* in the completely additive framework.

In each of the three cases the bounds are shown to be tight in a certain sense.

The above results hold for arbitrary continuous functionals H. In the case where H is in iterative form we get a further improvement of the bounds:

> H is in *iterative form* if $H\ h = f \sqcup h \circ g$ for strict and additive functions f and g.

We then show that the number k of iterations needed to compute the fixed point is at most

- *linear*, and furthermore

- the fixed point can be computed *pointwise*.

Again the complexity result is tight in a certain sense.

The immediate applicability of these approaches are illustrated on the example analysis of Appendix A. We also discuss the more general applicability of the results by referring to various analyses presented in the literature.

2 The general and monotone frameworks

We shall first introduce some notation. Let $(L, \sqsubseteq)$ be a *finite complete lattice*, that is

- $\sqsubseteq$ is a partial order on L, and
- each subset Y of L has a least upper bound in L denoted $\sqcup Y$.

We write

$\mathbf{C}\ L$	:	for the cardinality of L
$\mathbf{RC}\ L$	:	for the number of non-bottom elements of L, i.e. $\mathbf{RC}\ L + 1 = \mathbf{C}\ L$
$\mathbf{H}\ L$	:	for the maximal length of chains in L

where a chain $d_0 \sqsubset d_1 \sqsubset \cdots \sqsubset d_k$ has length k (and contains $k+1$ distinct elements); here $d_i \sqsubset d_{i+1}$ denotes $d_i \sqsubseteq d_{i+1} \wedge d_i \neq d_{i+1}$.

Fact 1: $\mathbf{C}(L^n) = (\mathbf{C}\ L)^n$ for $n \geq 1$. □

Fact 2: $\mathbf{H}(L^n) = n \cdot\ \mathbf{H}\ L$ for $n \geq 1$. □

We write $\mathbf{2}$ for the complete lattice with two elements $\mathsf{0}$ and $\mathsf{1}$ ordered by $\mathsf{0} \sqsubseteq \mathsf{1}$. Thus $\mathbf{C}\ \mathbf{2} = 2$, $\mathbf{RC}\ \mathbf{2} = 1$ and $\mathbf{H}\ \mathbf{2} = 1$.

2.1 An upper bound

In the general framework we have

Proposition 3: $\mathbf{H}(A^p \rightarrow_t B^q) \leq (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. $\square$

Proof: Let $h_i : A^p \rightarrow_t B^q$ and assume that

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_k$$

We shall then show that $k \leq (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$.

From $h_i \sqsubset h_{i+1}$ we get that there exists $w \in A^p$ such that $h_i\ w \sqsubset h_{i+1}\ w$ because the ordering on $A^p \rightarrow_t B^q$ is defined componentwise. But then there exists $j \in \{1, \ldots, q\}$ such that $h_i\ w \downarrow j \sqsubset h_{i+1}\ w \downarrow j$ because the ordering on B^q is defined componentwise.

Now for each $h_i \sqsubset h_{i+1}$ we get a pair (w, j) and each pair can occur at most $\mathbf{H}\ B$ times. There are at most $(\mathbf{C}\ A)^p$ distinct values for w and q distinct values for j. So for the chain

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_k$$

the value k will be at most $(\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$. $\square$

Any monotone function is a total function so Proposition 3 yields:

Corollary 4: $\mathbf{H}(A^p \rightarrow_m B^q) \leq (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. $\square$

It is straightforward to strengthen Proposition 3 to show that if we restrict the functions of $A^p \rightarrow_t B^q$ to be strict, written $A^p \rightarrow_s B^q$, then

$$\mathbf{H}(A^p \rightarrow_s B^q) \leq ((\mathbf{C}\ A)^p - 1) \cdot q \cdot \mathbf{H}\ B$$

The reason is that the element w in the pairs (w, j) cannot be equal to $\perp$, the least element of A^p. Thus there are $\mathbf{RC}(A^p)$ choices for w and $\mathbf{RC}(A^p) = (\mathbf{C}\ A)^p - 1$.

We shall now apply Proposition 3 to the special chains obtained when computing fixed points:

Theorem 5: In the general framework any continuous factional

$$H : (A^p \rightarrow_t B^q) \rightarrow (A^p \rightarrow_t B^q)$$

satisfies $\mathit{FIX}\ H = H^k\bot$ for

$$k = (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$$

This result carries over to the monotone framework as well. □

Proof: Consider the chain

$$H^0\bot \sqsubseteq H^1\bot \sqsubseteq \cdots$$

Since $A^p \rightarrow_t B^q$ is a finite complete lattice it cannot be the case that all $H^i\bot$ are distinct. Let k' be the minimal index for which $H^{k'}\bot = H^{k'+1}\bot$. Then

$$H^0\bot \sqsubset H^1\bot \sqsubset \cdots \sqsubset H^{k'}\bot$$

Using Proposition 3 we then get that $k' \leq (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$, i.e. $k' \leq k$. Since $H^{k'}\bot = \mathit{FIX}\ H$ and $H^{k'}\bot \sqsubseteq H^k\bot \sqsubseteq \mathit{FIX}\ H$ we get $\mathit{FIX}\ H = H^k\bot$ as required. □

Example 6: The analysis of Append A considers the specie case where A and B are the two-point domain **2**. In this case Theorem 5 specialises to

> In the general framework any continuous functional
>
> $$H : (\mathbf{2}^p \rightarrow_t \mathbf{2}^q) \rightarrow (\mathbf{2}^p \rightarrow_t \mathbf{2}^q)$$
>
> satisfies $\mathit{FIX}\ H = H^k\bot$ for
>
> $$k = 2^p \cdot q$$
>
> This result carries over to the monotone framework as well.

In Appendix A it is shown (Fact A.4) that the factorial program gives rise to a continuous functions $H : (\mathbf{2}^3 \rightarrow_t \mathbf{2}^3) \rightarrow (\mathbf{2}^3 \rightarrow_t \mathbf{2}^3)$ so, according to the theorem, at most $2^3 \cdot 3 = 24$ iterations are needed to determine the fixed point. However, a simple calculation in Appendix A shows that the fixed point is obtained ready after the *first* iteration! □

2.2 The bound is tight

Motivated by this example one may wonder whether the bound of Theorem 5 is too pessimistic. The following result shows that this need not be the case:

Proposition 7: $\mathbf{H}(A^p \rightarrow_m B^q) \geq (\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. $\square$

The proof is in two stages where the first is expressed as a lemma:

Lemma 8: $\mathbf{H}(A^p \rightarrow_m \mathbf{2}) \geq (\mathbf{C}\ A)^p$ for $p \geq 1$. $\square$

Proof: It is sufficient to construct a monotone factions $h_i : A^p \rightarrow_m \mathbf{2}$ such that

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_k$$

for $k = (\mathbf{C}\ A)^p$.

The first step is to enumerate the elements of A^p. Let $w_1, w_2, \ldots, w_k$ be chosen such that

$$w_j \text{ is one of the maximal elements of the set } A^p \setminus \{w_1, \ldots, w_{j-1}\}$$

For $0 \leq i \leq k$ define the function h_i by

$$h_i\ w_i \begin{cases} \mathsf{0} & \text{if } i < j \\ \mathsf{1} & \text{if } j \leq i \end{cases}$$

Clearly each h_i is a well-defined total function. To see that it is monotone assume that $w \sqsubset w'$. Then $w = w_j$ and $w' = w_l$ for some j and l satisfying $l < j$. There are three cases:

$$\begin{array}{lcllcllcl} i < l < j & : & h_i\ w_j & = & \mathsf{0} & \text{and} & h_i\ w_l & = & \mathsf{0} \\ l \leq i < j & : & h_i\ w_j & = & \mathsf{0} & \text{and} & h_i\ w_l & = & \mathsf{1} \\ l < j \leq i & : & h_i\ w_j & = & \mathsf{1} & \text{and} & h_i\ w_l & = & \mathsf{1} \end{array}$$

In all cases $h_i\ w \sqsubseteq h_i\ w'$ so h_i is monotone. Clearly $h_i \sqsubset h_{i+1}$ for all i and it is also easy to see that there are the required number of functions. This proves the lemma. $\square$

Proof of Proposition 7: It is sufficient to construct a sequence of functions $h_i : A^p \to_m B^q$ satisfying

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_{k \cdot r}$$

for $k = (\mathbf{C}\ A)^p$ and $r = q \cdot \mathbf{H}\ B$.

Let $v_0 \sqsubset v_1 \sqsubset \cdots \sqsubset v_r$ be a chain in B^q (Since $\mathbf{H}(B^q) = q \cdot \mathbf{H}\ B$ such a chain will exist.) The construction of Lemma 8 can be applied to the function spaces $A^p \to_m \{v_{j-1}, v_j\}$ for $1 \leq j \leq r$ and gives functions

$$h_i^j : A^p \to_m B^q$$

for $0 \leq i \leq k$. Furthermore

$$h_0^j \sqsubset h_1^j \sqsubset \cdots \sqsubset h_k^j$$

Now $h_k^j = h_0^{j+1}$ must be the case so we have a chain

$$h_0^1 \sqsubset h_1^1 \sqsubset \cdots \sqsubset h_k^1 \sqsubset h_1^2 \sqsubset \cdots \sqsubset h_k^2 \sqsubset \cdots \sqsubset h_k^r$$

This chain has the required number of elements and we have completed the proof. □

All monotone functions are total functions so Proposition 7 yields:

Corollary 9: $\mathbf{H}(A^p \to_t B^q) \geq (\mathbf{C}\ A)^p \cdot\ q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. □

Combining these results we get

$$\mathbf{H}(A^p \to_t B^q) = \mathbf{H}(A^p \to_m B^q)$$

meaning that the maximal length of chains of total functions and monotone functions are the same. This may be slightly surprising since it is well-known that there are more total functions than monotone functions.

2.3 Applicability of results

We believe that all interesting analyses will give rise to monotone functions so the real limitations of the results of this section are due to the following:

- only first-order languages are handled, and
- the lattices of properties must be finite.

Therefore it is *not* surprising that some of the well-known upper bounds on the required number of iterands are direct corollaries of Theorem 5. As an example [PC87] states that strictness analysis of a first-order language bound. They consider functions in domains of has an exponential upper the form

$$\mathbf{2}^p \rightarrow \mathbf{2}$$

and using Theorem 5 we immediately get that at most 2^p iterands we be needed. It is not known whether this is a tight bound but it has been shown that the exponential upper bound of strictness anaysis of the untyped (higher-order) λ-calculus is indeed tight [HY86].

3 The completely additive framework

We shall now assume that the functions of interest are strict and additive; by strictness of a function h we mean that $h\bot = \bot$ and by additivity that $h(d_1 \sqcup d_2) = (h\ d_1) \sqcup (h\ d_2)$. Since the complete lattices considered are all finite it follows that a strict and additive function h is so completely additive, that is $h(\sqcup Y) = \sqcup\{h\ d \mid d \in Y\}$ for all subsets Y.

Following [G71] an element d of a complete lattice $(L, \sqsubseteq)$ is *join-irreducible* if for all $d_1, d_2 \in L$:

$$d = d_1 \sqcup d_2 \text{ implies } d = d_1 \text{ or } d = d_2$$

and a complete lattice $(L, \sqsubseteq)$ is *distributive* if for all $d, d_1, d_2 \in L$:

$$d \sqcap (d_1 \sqcup d_2) = (d \sqcap d_1) \sqcup (d \sqcap d_2)$$

Clearly $\perp$ is always join-irreducible but we shall be more interested in the non-trivial join-irreducible elements, i.e. those that are not $\perp$. To this end we shall write

> **RJC** L : for the number of non-bottom join-irreducible elements of L.

We thus have **RJC 2** $= 1$.

Fact 10: $\mathbf{RJC}(L^n) = n \cdot \mathbf{RJC}\ L$ and if L is distributive so is L^n for $n \geq 1$.

Lemma 11: If $(L, \sqsubseteq)$ is a finite complete lattice we have

$$w = \sqcup\{x \mid x \sqsubseteq w, x \text{ is join-irreducible and } x \neq \perp\}$$

for all $w \in L$. $\square$

Proof: Assume by way of contradiction that the claim of the Lemma is false. Let $W \subseteq L$ be the set of $w \in L$ for which the condition fails. Since W is finite and non-empty it has a minimal element w. From $w \in W$ it follows that w is not join-irreducible. Hence there exist w_1 and w_2 such that

$$w = w_1 \sqcup w_2,\ w \neq w_1,\ w \neq w_2$$

It follows that $w_1 \sqsubset w$, $w_2 \sqsubset w$ and by choice of w that $w_1 \notin W$ and $w_2 \notin W$. We may then calculate

$$\begin{array}{rcl}
w & = & w_1 \sqcup w_2 \\
& = & \bigsqcup\{x \mid x \sqsubseteq w_1,\ x \text{ is join-irreducible and } x \neq \perp\} \\
& & \sqcup \bigsqcup\{x \mid x \sqsubseteq w_2,\ x \text{ is join-irreducible and } x \neq \perp\} \\
& = & \bigsqcup\{x \mid (x \sqsubseteq w_1 \text{ or } x \sqsubseteq w_2),\ x \text{ is join-irreducible and } x \neq \perp\} \\
& \sqsubseteq & \bigsqcup\{x \mid x \sqsubseteq w,\ x \text{ is join-irreducible and } x \neq \perp\} \\
& \sqsubseteq & w
\end{array}$$

shows that

$$w = \bigsqcup\{x \mid x \sqsubseteq w,\ x \text{ is join-irreducible and } x \neq \bot\}$$

and contradicts $w \in W$. Hence $W = \emptyset$ and the claim of the Lemma holds. $\square$

Lemma 12: If $(L, \sqsubseteq)$ is a distributive complete lattice the conditions

(i) x is join-irreducible
(ii) $x \sqsubseteq w_1 \sqcup w_2$ implies $x \sqsubseteq w_1$ or $x \sqsubseteq w_2$

are equivalent.

Proof: Condition (ii) always implies condition (i). To see that condition (i) implies condition (ii) we calculate

$$\begin{array}{rcl} x \sqsubseteq w_1 \sqcup w_2 & \Leftrightarrow & x = x \sqcap (w_1 \sqcup w_2) \\ & \Leftrightarrow & x = (x \sqcap w_1) \sqcup (x \sqcap w_2) \\ & \Leftrightarrow & \exists i : x = x \sqcap w_i \\ & \Leftrightarrow & \exists i : x \sqsubseteq w_i \end{array}$$

where we used the connection between $\sqsubseteq$ and $\sqcap$, distributivity, join-irreducibility of x, and the connection between $\sqsubseteq$ and $\sqcap$. $\square$

Corollary 13 [G71, p73] If L is a finite and distributive complete lattice then we have $\mathbf{RJC}\ L = \mathbf{H}\ L$. $\square$

3.1 An upper bound

In the completely additive framework we have

Proposition 14: $\mathbf{H}(A^p \rightarrow_{sa} B^q) \leq p \cdot \mathbf{RJC}\ A \cdot q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. $\square$

Proof: The proof is a refinement of that of Proposition 3 so we begin by assuming that $h_i \in A_p \rightarrow_{sa} B^q$ and that

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_k$$

We shall show that $k \leq p \cdot \mathbf{RJC}\ A \cdot q \cdot \mathbf{H}\ B$.

As in the proof of Proposition 3 we get a pair (w, j) such that $h_i\ w \downarrow j \sqsubset h_{i+1}\ w \downarrow j$ for each $h_i \sqsubset h_{i+1}$. The element w is an arbitrary element of A^p so in the proof of Proposition 3 there was $\mathbf{C}(A^p)$ choices for w. We shall now show that w can be chosen as a non-trivia join-irreducible element of A^p thereby reducing the number of choices to $\mathbf{RJC}(A^p)$. Calculations similar to those in the proof of Proposition 3 we then give the required upper bound on k.

The element w satisfies $h_i\ w \sqsubset h_{i+1}\ w$. By Lemma 11 we have

$$w = \bigsqcup\{x \mid x \sqsubseteq w,\ x \text{ is a join-irreducible and } x \neq \perp\}$$

From the strictness and additivity of h_i and h_{i+1} we get

$$\begin{array}{lcl} h_i\ w & = & \bigsqcup\{h_i\ x \mid x \sqsubseteq w,\ x \text{ is join-irreducible and } x \neq \perp\} \\ h_{i+1}\ w & = & \bigsqcup\{h_{i+1}\ x \mid x \sqsubseteq w,\ x \text{ is join-irreducible and } x \neq \perp\} \end{array}$$

It cannot be the case that $h_i\ x = h_{i+1}\ x$ for all non-bottom join-irreducible elements x of A^p since then $h_i\ w = h_{i+1}\ w$. So let x be a non-bottom join-irreducible element where $h_i\ x \sqsubset h_{i+1}\ x$. Then there we only be $\mathbf{RJC}(A^p)$ choices for x and this completes the proof. □

We can now apply Proposition 14 to the special chains obtained when computing fixed points:

Theorem 15: In the completely additive framework any continuous functional

$$H : (A^p \rightarrow_{sa} B^q) \rightarrow (A^p \rightarrow_{sa} B^q)$$

satisfies $\mathit{FIX}\ H = H^k \perp$ for

$$k = p \cdot \mathbf{RJC}\ A \cdot q \cdot \mathbf{H}\ B.$$

□

Proof: Analoguous to the proof of Theorem 5. □

The equality test between the iterands $H^0\perp, H^1\perp, \cdots$ can be simplified in this framework. To see this consider two functions $h_1, h_2 \in A^p \rightarrow_{sa} B^q$. Then

$h_1 = h_2$

if and only if

$h_1\ x = h_2\ x$ for all non-trivial join-irreducible elements x of A^p, i.e. all elements $(\perp, \cdots, a, \cdots, \perp)$ where a is a non-bottom join-irreducible element of A.

Example 16: In the case where A and B are the two-point domain $\mathbf{2}$, Theorem 15 specialises to

In the completely additive framework any continuous functional

$$H : (\mathbf{2}^p \rightarrow_{sa} \mathbf{2}^q) \rightarrow (\mathbf{2}^p \rightarrow_{sa} \mathbf{2}^q)$$

satisfies $\mathit{FIX}\ H = H^k \perp$ for

$$k = p \cdot q$$

The analysis of Appendix A turns out to be in the completely additive framework (Fact A.5). For the factorial program where $p = q = 3$ we therefore get that at most $3 \cdot 3 = 9$ iterands need to be computed. This is a substantial improvement of the bound (24) determined in Example 6 but still the first iterand is equal to the fixed point! □

3.2 The bound is tight

Motivated by this example we shall show that the bound of Proposition 14 is tight when A is distributive.

Proposition 17: $\mathbf{H}(A^p \rightarrow_{sa} B^q) \geq p \cdot \mathbf{H}\ A \cdot q \cdot \mathbf{H}\ B$ for $p, q \geq 1$. □

The proof follows the same pattern as that of Proposition 7 so we shall first establish

Lemma 18: $\mathbf{H}(A^p \rightarrow_{sa} \mathbf{2}) \geq p \cdot \mathbf{H}\ A$ for $p \geq 1$. □

Proof: It is sufficient to construct functions $h_i : A^p \rightarrow_{sa} \mathbf{2}$ such that

$$h_0 \sqsubset h_1 \sqsubset \cdots \sqsubset h_k$$

for $k = p \cdot \mathbf{H}\ A$.

Let $a_0 \sqsubset \cdots \sqsubset a_k$ be a chain of A^p and note that $a_0 = \bot$ and $a_k = \top$ must be the case. Define h_i by

$$h_i\ w = \begin{cases} \mathsf{0} & \text{if } w \sqsubseteq a_{k-i} \\ \mathsf{1} & \text{otherwise} \end{cases}$$

for $0 \leq i \leq k$. Clearly each h_i is strict and monotone. For additivity we calculate

$$\begin{aligned} h_i(w_1 \sqcup w_2) &= \begin{cases} \mathsf{0} & \text{if } w_1 \sqcup w_2 \sqsubseteq a_{k-i} \\ \mathsf{1} & \text{otherwise} \end{cases} \\ &= \begin{cases} \mathsf{0} & \text{if } w_1 \sqsubseteq a_{k-i} \text{ and } w_2 \sqsubseteq a_{k-i} \\ \mathsf{1} & \text{otherwise} \end{cases} \\ &= \left(\begin{cases} \mathsf{0} & \text{if } w_1 \sqsubseteq a_{k-i} \\ \mathsf{1} & \text{otherwise} \end{cases}\right) \sqcup \left(\begin{cases} \mathsf{0} & \text{if } w_2 \sqsubseteq a_{k-i} \\ \mathsf{1} & \text{otherwise} \end{cases}\right) \\ &= (h_i\ w_1) \sqcup (h_i\ w_2) \end{aligned}$$

Finally, $h_i \sqsubset h_{i+1}$ since $a_{k-i} \sqsupset a_{k-(i+1)}$. □

Proof of Proposition 17: The proof is similar to that of Proposition 7 except that it uses the functions constructed in the proof of Lemma 18 rather than those from Lemma 8. We omit the details. □

Combining Proposition 14 and 17 and using Corollary 13 we get

$$\mathbf{H}(A^p \rightarrow_{sa} B^q) = p \cdot \mathbf{H}\ A \cdot q \cdot \mathbf{H}\ B$$

provided that A is distributive.

3.3 Applicability of results

Compared with the development of Section 2 we have considered the additional requirement that

- the functions of concern must be strict and additive.

Furthermore we have seen that if the domain of the functions is distributive then the bound on the number of iterations is tight.

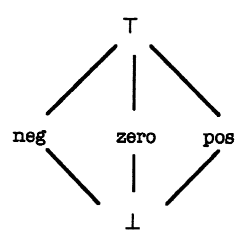
It turns out that there are a number of interesting analyses that do *not* satisfy these conditions, but, fortunately there are also a large class of analyses that do, as e.g. the example analysis of Appendix A.

An example of an analysis that does *not* give rise to strict and additive functions is strictness analysis [M81]. The abstract meaning of the conditional is often defined by

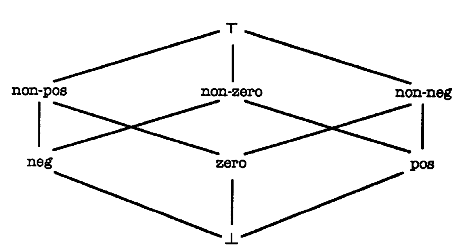
$$\mathtt{if}^{\#}(x, y, z) = x \sqcap (y \sqcup z)$$

and it is easy to show that $\mathtt{if}^{\#}$ cannot be an additive function. However, there are analyses of first-order functional languages that do give rise to strict and additive functions, an example is the liveness analysis of [NN89].

The potential restriction to finite and distributive complete lattices is more severe. Consider for example an anaysis for detection of signs. One possibility we be to base it on a complete lattice of the form



However, distributivity fail. An alternative wood be to use a more refined lattice as



which is distributive (with `neg`, `zero`, `pos` and $\perp$ being the join-irreducible elements).

The finite distributive complete lattices are characterised in the following lemma where we write $\mathcal{P}(D)$ for the Hoare power dome of the cpo D.

Lemma 19: L is a finite and distributive complete lattice if ad only if $L = \mathcal{P}(E)$ for some finite cpo E. $\square$

Proof: This is essentially Theorem 9 of Section 7 in [G71]; in the notation of [G71, p72] we have $L = \mathrm{H}(\mathrm{J}(L))$ and we have $\mathrm{H}(D) = \mathcal{P}(D_\perp)$ for all finite partial orders D. $\square$

4 Iterative program schemes

The upper bounds expressed by Theorems 5 and 15 are obtained without any assumptions about the functional H except that it is a continuous function over the relevant lattices. In this section we shall restrict the form of H.

For iterative programs as e.g. those considered in Appendix A the functional H will typically have the form

$$H\ h = f \sqcup h \circ g$$

where f and g are strict and additive functions. Then the iterands $H^i\perp$ will be strict and additive so we shall restrict our attention to the completely additive framework.

4.1 An upper bound

The first reset is a refinement of Theorem 15:

Theorem 20: In the completely additive framework the fixed point of a functional

$$H : (A^p \to_{sa} B^q) \to (A^p \to_{sa} B^q)$$

defined by

$$H\ h = f \sqcup h \circ g$$

for $f \in A^p \to_{sa} B^q$ and $g \in A^p \to_{sa} A^p$ can be computed pointwise. More precisely, if for some $w \in A^p$ and $k \geq 0$

$$H_0^k \bot\ w = H_0^{k+1} \bot\ w$$

then $\mathit{FIX}\ H\ w = f(H_0^k \bot\ w)$ where $H_0\ h = \mathtt{id} \sqcup h \circ g$. Furthermore, it is possible to take $k = p \cdot \mathbf{H}\ A$. $\square$

Basically this result says that in order to compute $\mathit{FIX}\ H$ on a particular value w it is sufficient to determine the values of the iterands $H_0^i \bot$ at w and then compare these values. So rather than having to test the extensional equality of two functions on a set of arguments we only need to test the equality of two function values. Furthermore, the theorem states that this test has to be performed at most a linear number of times.

To prove the theorem we need a couple of lemmas:

Lemma 21: Let $H\ h = f \sqcup h \circ g$ for $f \in A^p \to_{sa} B^q$ and $g \in A^p \to^{sa} A^p$. Then for $i \geq 0$ we have

$$H^{i+1} \bot = \sqcup \{ f \circ g^j \mid 0 \leq j \leq i \}. \qquad \square$$

Proof: We proceed by numerical induction on i. If $i = 0$ then the result is immediate as $H^1 \bot = f \sqcup \bot \circ g = f = \bigsqcup \{ f \circ g^j \mid 0 \leq j \leq 0 \}$. For the induction step we calculate:

$$\begin{array}{rcl} H^{i+2} \bot & = & f \sqcup (H^{i+1} \bot) \circ g \\ & = & f \sqcup (\sqcup \{ f \circ g^j \mid 0 \leq j \leq i \}) \circ g \\ & = & \sqcup \{ f \circ g^j \mid 0 \leq j \leq i+1 \} \end{array}$$

where the last equality follows from the pointwise definition of $\sqcup$ on $A^p \to_{sa} B^q$. $\square$

Lemma 22: Let $H\ h = f \sqcup h \circ g$ for $f \in A^p \to_{sa} B^q$ and $g \in A^p \to_{sa} A^p$. Then

$$FIX\ H = f \circ FIX\ H_0$$

where $H_0\ h = \mathtt{id} \sqcup h \circ g$. □

Proof: We shall first prove that

$$H^i \bot = f \circ H_0^i \bot \text{ for } i \geq 0.$$

The case $i = 0$ is immediate. So assume that $i > 0$. We shall then apply Lemma 21 to H and H_0 and get

$$\begin{array}{rcl} H^i \bot & = & \bigsqcup\{f \circ g^j \mid 0 \leq j \leq i-1\} \\ H_0^i \bot & = & \bigsqcup\{g^j \mid 0 \leq j \leq i-1\} \end{array}$$

Since f is additive we get

$$H^i \bot = f \circ \sqcup\{g^j \mid 0 \leq j \leq i-1\} = f \circ H_0^i \bot$$

as required.

We now have

$$\begin{array}{rcl} FIX\ H & = & \bigsqcup\{H^i \bot \mid i \geq 0\} \\ & = & \bigsqcup\{f \circ H_0^i \bot \mid i \geq 0\} \\ & = & f \circ \bigsqcup\{H_0^i \bot \mid i \geq 0\} \\ & = & f \circ FIX\ H_0 \end{array}$$

where the third equality uses the continuity of f which is ensured by monotonicity and finiteness of the complete lattices in question. □

Proof of Theorem 20: We shall first prove that if $H_0^k \bot\ w = H_0^{k+1} \bot\ w$ then $FIX\ H\ w = f(H_0^k \bot\ w)$. This is done in two stages.

First assume that $k = 0$. Then $H_0^0 \bot\ w = H_0^1 \bot\ w$ amounts to $\bot = w$. Using Lemma 21 and the strictness of g we get

$$H_0^{i+1} \bot\ \bot = \bigsqcup\{g^j \bot \mid 0 \leq j \leq i\} = \bot$$

for $i \geq 0$ and thereby $FIX\ H_0 \bot = \bot$. But then Lemma 22 gives

$$FIX\ H\bot = f(FIX\ H_0\bot) = f\bot = f(H_0^0\bot\ \bot)$$

as required.

Secondly assume that $k > 0$. From $H_0^k\bot\ w = H_0^{k+1}\ w$ we get, using Lemma 21, that

$$\bigsqcup\{g^j\ w \mid 0 \leq j < k\} = \bigsqcup\{g^j\ w \mid 0 \leq j \leq k\}$$

This means that

$$g^k\ w \sqsubseteq \bigsqcup\{g^j\ w \mid 0 \leq j < k\}$$

We shall now prove that for all $l \geq 0$

$$g^{k+l}\ w \sqsubseteq \bigsqcup\{g^j\ w \mid 0 \leq j < k\} \qquad (*)$$

We have already established the basis $l = 0$. For the induction step we get

$$\begin{array}{rcl} g^{k+l+1}\ w & = & g(g^{k+l}\ w) \\ & \sqsubseteq & g(\bigsqcup\{g^j\ w \mid 0 \leq j < k\}) \\ & = & \bigsqcup\{g^j\ w \mid 1 \leq j \leq k\} \\ & \sqsubseteq & \bigsqcup\{g^j\ w \mid 0 \leq j < k\} \end{array}$$

where we have used the additivity of g. This proves $(*)$. Using Lemma 21 and $(*)$ we get

$$\begin{array}{rcl} H_0^{k+l}\bot\ w & = & \sqcup\{g^j\ w \mid 0 \leq j < k+l\} \\ & = & \sqcup\{g^j\ w \mid 0 \leq j < k\} \\ & = & H_0^k\bot\ w \end{array}$$

for all $l \geq 0$. This means that $FIX\ H_0\ w = H_0^k\bot\ w$ and using Lemma 22 we get

$$FIX\ H\ w = f(H_0^k\bot\ w)$$

as required.

To complete the proof of the theorem we have to show that one may take $k = p \cdot \mathbf{H}\ A$. For this it suffices to show that one cannot have a chain

$$H_0^0 \bot\ w \sqsubset H_0^1 \bot\ w \sqsubset \cdots \sqsubset H_0^k \bot\ w \sqsubset H_0^{k+1} \bot\ w$$

in A^p. But this is immediate since $k+1 > \mathbf{H}(A^p)$. $\square$

Example 23: In the case where A and B are the two-point domain $\mathbf{2}$, Theorem 20 specialises to

> In the completely additive framework the fixed point of a functional
>
> $$H\ :\ (\mathbf{2}^p \to_{sa} \mathbf{2}^q) \to (\mathbf{2}^p \to_{sa} \mathbf{2}^q)$$
>
> defined by
>
> $$H\ h = f \sqcup h \circ g$$
>
> for $f \in \mathbf{2}^p \to_{sa} \mathbf{2}^q$ and $g \in \mathbf{2}^p \to_{sa} \mathbf{2}^p$ can be computed pointwise. More precisely, if for some $w \in \mathbf{2}^p$ and $k \geq 0$
>
> $$H_0^k \bot\ w = H_0^{k+1} \bot\ w$$
>
> then $\mathit{FIX}\ H\ w = f(H_0^k\ w)$ where $H_0\ h = \mathtt{id} \sqcup h \circ g$. Furthermore, it is possible to take $k = p$.

The functionals considered in the analysis of Appendix A turn out to be of the required form. For the factorial program where $p = q = 3$ we get that at most 3 iterands need to be computed. Again we have obtained a substantial improvement compared with Example 16 (and Example 6). $\square$

The one shortcoming of Theorem 20 is that one has to test for stabilisation of the iterands of H_0 in order to determine when the fixed point of H has been reached. To overcome this say that a function f is *smash order reflecting* if it satisfies the following property:

> if $f\ x \sqsubseteq f\ y$
> then $x \sqsubseteq y$ or $f\ y = f\ \top$

Then one can strengthen the statement of Theorem 20 to

if f is smash order reflecting
and $H^k\bot\ w = H^{k+1}\bot\ w$
then $\mathit{FIX}\ H\ w = H^k\bot\ w$

For the proof note that $H^k\bot\ w = H^{k+1}\bot\ w$ implies that $f(H_0^k\bot\ w) = f(H_0^{k+1}\ w)$. We then have two possibilities. In one of these $H_0^k\bot\ w = H_0^{k+1}\ w$ so that the result follows thorn the proof of Theorem 20 as stated. In the other case we have

$$f(\top) = f(H_0^k\bot\ w) \sqsubseteq f(\mathit{FIX}\ H_0) \sqsubseteq f(\top)$$

showing

$$\mathit{FIX}\ H\ w = f(\mathit{FIX}\ H_0\ w) = f(H_0^k\bot\ w) = H^k\bot\ w$$

Thus we have the desired result in both cases. – In the analysis of Appendix A the function f corresponds to the function CHECK which is smash order reflecting.

4.2 The bound is tight

The above example shows that the upper bound given by Theorem 20 is quite close to the number of iterations needed. We shall now show that the bound is indeed tight.

Proposition 24: In the completely additive framework there exists a continuous functional

$$H\ :\ (A^p \rightarrow_{sa} B^q) \rightarrow (A^p \rightarrow_{sa} B^q)$$

of the form

$$H\ h = f \sqcup h \circ g$$

for $f \in A^p \to_{sa} B^q$ and $g \in A^p \to_{sa} A^p$ and there exists $w \in A^p$ such that at least $p \cdot \mathbf{H}\ A$ iterations are needed. More precisely $H^k \bot\ w \neq \mathit{FIX}\ H\ w$ for $k = (p \cdot \mathbf{H}\ A) - 1$ provided that $\mathbf{H}\ B > 0$ and $\mathbf{H}\ A > 0$. $\square$

The proof is in two stages where the first takes the form of a lemma:

Lemma 25: There exists a function $g : A^p \to_{sa} A^p$ and an element $w \in A^p$ such that $\sqcup\{g^i\ w \mid 0 \leq i < n\} \neq \sqcup\{g_i\ w \mid 0 \leq i \leq n\}$ whenever $0 \leq n \leq (p \cdot \mathbf{H}\ A) - 1$. $\square$

Proof: Let $m = \mathbf{H}\ A$ and let $a_0 \sqsubset \cdots \sqsubset a_m$ be a chain in A of maximal length. Define the function $\sigma : A \to A$ by

$$\sigma\ x = \begin{cases} a_0 & \text{if } x = a_0 \\ a_m & \text{if } \{m\} = \{i \mid x \sqsubseteq a_i\} \\ a_{1+\min\{i \ \mid\ x\ \sqsubseteq\ a_i\}} & \text{otherwise} \end{cases}$$

noting that well-definedness follows from $a_m = \top$ so that $\{i \mid x \sqsubseteq a_i\}$ cannot be empty. The function is strict since $a_0 = \bot$. It is monotone since $x \sqsubseteq x'$ implies that $\{i \mid x \sqsubseteq a_i\} \supseteq \{i \mid x' \sqsupseteq a_i\}$ and it follows that $\min\{i \mid x \sqsubseteq a_i\} \leq \min\{i \mid x' \sqsubseteq a_i\}$. It is additive because

$$\begin{array}{l} \{i \mid x \sqcup x' \sqsubseteq a_i\} = \\ \{i \mid x \sqsubseteq a_i \wedge x' \sqsubseteq a_i\} = \\ \{i \mid x \sqsubseteq a_i\} \cap \{i \mid x' \sqsubseteq a_i\} \end{array}$$

so that $\min\{i \mid x \sqcup x' \sqsubseteq a_i\}$ is the greater one of $\min\{i \mid x \sqsubseteq a_i\}$ and $\min\{i \mid x' \sqsubseteq a_i\}$. Finally σ satisfies that $\sigma(a_0) = a_0$, $\sigma(a_i) = a_{i+1}$ for $i \in \{1, \ldots, m-1\}$ and $\sigma(a_m) = a_m$.

Define the element $w \in A^p$ by $w = (a_1, a_0, \ldots, a_0)$ and define the function g by

$$g(x_1, \ldots, x_p) = (\sigma(x_p), x_1, \ldots, x_{p-1})$$

Clearly g is well-defined, strict, monotone and additive (as σ is). For $0 \leq i < m$ and $0 \leq j < p$ we claim that

$$g^{i\cdot p+j}w = (x'_1, \ldots, x'_p) \text{ where } x'_l = \begin{cases} a_{i+1} & \text{if } l = j+1 \\ a_0 & \text{otherwise} \end{cases}$$

The proof is by complete induction on the value of $i \cdot p + j$. So consider $i \in \{0, \cdots, m-1\}$ and $j \in \{0, \cdots, p-1\}$ and assume that the result holds for all i' and j' with $i' \cdot p + j' < i \cdot p + j$. We now have three cases.

If $i = j = 0$ the result follows from the definition of w.

If $j = 0$ and $i > 0$ we set $i' = i - 1$ and $j' = p - 1$ and note that $0 \leq i' < m$, $0 \leq j' < p$ and $i' \cdot p + j' < i \cdot p + j$. The induction hypothesis gives

$$g^{i' \cdot\, p+j'}w = (a_0, \cdots, a_0, a_{i'+1})$$

so that

$$g^{i' \cdot\, p+j'+1}w = (a_{i'+2}, a_0, \cdots, a_0)$$

Since $i \cdot p + j = i' \cdot p + j' + 1$ and $a_{i+1} = a_{i'+2}$ this proves the result.

If $j > 0$ we set $j' = j - 1$ and note that $0 \leq i < m$, $0 \leq j < p$ and $i \cdot p + j' < i \cdot p + j$. The induction hypothesis gives

$$g^{i\cdot p+j'}w = (x'_1, \ldots, x'_p) \text{ where } x'_l = \begin{cases} a_{i+1} & \text{if } l = j'+1 \\ a_0 & \text{otherwise} \end{cases}$$

so that

$$g^{i\cdot p+j'+1}w = (x''_1, \ldots, x''_p) \text{ where } x''_l = \begin{cases} a_{i+1} & \text{if } l = j'+2 \\ a_0 & \text{otherwise} \end{cases}$$

Since $i \cdot p + j = i \cdot p + j' + 1$ and $j + 1 = j' + 2$ this proves the result.

Now let $0 \leq i < m$ and $0 \leq j < p$ and write

$$\begin{array}{rcl} (x'_1, \cdots, x'_p) & = & \sqcup\{g^r\ w \ \mid\ 0 \leq r < p \cdot i + j\} \\ (x''_1, \cdots, x''_p) & = & g^{p\cdot i+j}\ w \end{array}$$

It follows that

$$\begin{array}{rcl} x'_{j+1} & = & a_i \\ x''_{j+1} & = & a_{i+1} \end{array}$$

thereby establishing the claim of the Lemma. $\square$

Proof of Proposition 24: Let g and w be as in the proof of Lemma 25. Let $k = (p \cdot \mathbf{H}\ A) - 1$ and define $w' \in A^p$ by[1]

$$w' = \bigsqcup\{g^i\ w\ \ |\ 0 \leq i < k\}$$

so that

$$w' \sqcup (g^k\ w) = \bigsqcup\{g_i\ w\ \ |\ 0 \leq i \leq k\}$$

and hence $w' \neq w' \sqcup (g^k\ w)$. Define the function $f : A^p \to B^q$ by

$$f(x_1, \cdots, x_p) = \begin{cases} (\bot, \cdots, \bot) & \text{if } (x_1, \cdots, x_p) \sqsubseteq w' \\ (\top, \cdots, \top) & \text{otherwise} \end{cases}$$

It is easy to verify that f is a strict and additive function.

From Lemma 21 and the construction of f we have

$$H^k \bot\ w = \bigsqcup\{f(g^j\ w) \mid 0 \leq j < k\} = f\ w' = (\bot, \cdots, \bot)$$

and

$$H^{k+1} \bot\ w = \bigsqcup\{f(g^j\ w) \mid 0 \leq j \leq k\} = f(w' \sqcup g^k\ w) = (\top, \cdots, \top)$$

Here $g^k\ w$ is defined because $k \geq 0$ as $\mathbf{H}\ A > 0$ and $(\bot, \cdots, \bot)$ is different from $(\top, \cdots, \top)$ as $\mathbf{H}\ B > 0$. Since

$$H^k \bot\ w \neq H^{k+1} \bot\ w \sqsubseteq \mathit{FIX}\ H\ w$$

we have demonstrated the claim of the Proposition. $\square$

[1] In the notation of the proof of Lemma 25 we have $w' = (a_m, \cdots, a_m, a_{m-1})$ and $g^k\ w = (a_m, \cdots, a_m)$.

4.3 Applicability of results

Compared with the development of the previous section we now require that the functionals H have a very specific form. In return Theorem 20 gives a very simple method for determining the fixed point of H.

The anaysis of Appendix A fulfils the conditions of this section but certainly there are many analysis that do not.

It is worth observing that Proposition 24 expresses that there are functionals that require $p \cdot \mathbf{H}\ A$ iterations to determine the fixed point. Of course this needs not hold for a particle analysis for a praticular programming language because it may be impossible to express a functional similar to that considered in the proof of Proposition 24. However, for the analysis of Appendix A we are very close as may be illustrated by considering the program

$$\texttt{while } \texttt{x}_n = \texttt{x}_n \texttt{ do } (\texttt{x}_n \texttt{ := } \texttt{x}_{n+1}\texttt{;} \cdots\texttt{; } \texttt{x}_2 := \texttt{x}_1)$$

(The special nature of the 'flow of control' property discussed in Appendix A means that we may have to subtract 1 from the lower bound.)

5 Conclusion

For functionals $H : (A^p \to B^q) \to (A^p \to B^q)$ we have considered ways of bounding the size of the set X_w in

$$\text{if } H^k\bot\ v = H^{k+1}\bot\ v \text{ for all } v \in X_w \text{ then } \mathit{FIX}\ H\ w = H^k\bot\ w$$

and ways of bounding the number k in

$$\mathit{FIX}\ H\ w = H^k\bot\ w$$

Our resets may by summarised as follows:

framework	X_w	k
total or monotone	A^p	$(\mathbf{C}\ A)^p \cdot q \cdot \mathbf{H}\ B$
completely additive	$\{\ x \in A^p \mid x \text{ join-irred.}\}$	$p \cdot \mathbf{RJC}\ A \cdot q \cdot \mathbf{H}\ B$
iterative scheme	$\{\ w\}$	$p \cdot \mathbf{H}\ A$

(where we have not distinguished between H and H_0). Additionally we have shown that the bounds on k are tight in a certain sense (and in the case of the completely additive framework under the assumption that A is distributive).

5.1 Comparison with other work

The *frontiers approach* of [PC87] describes a method for computing fixed points of functionals on domains of the form $\mathbf{2}^p \to \mathbf{2}$. The aim of this work is similar to that of ours: to minimise the cost of computing fixed points. One of the central ideas is to represent a faction $h : \mathbf{2}^p \to \mathbf{2}$ by the inverse images $h^{-1}\ \mathsf{0}$ and $h^{-1}\ \mathsf{1}$. Using the monotonicity of h these sets can be reduced so that they do not contain redundant elements. The computation of the fixed point then proceeds by approximating the frontier of the fixed point from above as well as below.

We expect that our work can be combined with the frontiers approach. In particular we have limited the number of arguments to be considered when comparing functions and this can be used to bond the size of the frontiers to be constructed. As an example the maximal size of a frontier constructed in the completely additive framework will be p. This may explain the remark "Frontier sets are typically small. Only contrived functions seem to have large frontier sets" found in [PC87]. In our terminology, the contrived functions cannot be the additive ones.

Another approach to computing fixed points is the *minimal function graph approach* of [JM86]. In this work one considers the minimal set Y_w of (argument, result) pairs that have to be considered in order to determine the value of a function h for a given argument w :

$$Y_w = \{(d, h\ d) \mid h\ d \text{ must be computed when computing } h\ w\}.$$

The approach can be used to determine the value of the fixed point of H for a given argument w by determining the sets Y_w for the various iterands. In the worst case all possible arguments may have to be considered but on the average fewer arguments we do.

We expect that our work can be combined with the minimal function graph approach. In the completely additive framework the sets Y_w can be reduced as it will only be necessary to consider join-irreducible arguments and for al other arguments the results can be computed using the additivity of the iterands.

5.2 Further work

In our further work we hope to investigate the relationships between the frontiers approach, the minimal function graph approach and that of the present paper. In particular, it would be interesting to bridge the fairly large gap between the results obtained in the monotone framework and those obtained in the completely additive framework. Furthermore, it word be worthwhile to investigate various recursive forms of functionals in an attempt to generalise the results obtained for the iterative forms.

References

[ASU86] A. V. Aho, R. Sethi, J. D. Ullman: Computers - Principles, Techniques and Tools, Addison - Wesley, 1986.

[BH89] B. Bjerner, S. Holmström: A compositions approach to time analysis of first order lazy functions programs, Functional Programming Languages and Computer Architectures, 1989.

[G71] G. Grätzer: Lattice Theory - First Concepts and Distributive Lattices, W. H. Freeman and Company, 1971.

[HY86] P. Hudak, J. Young: Higher-Order Strictness Analysis in Untyped Lambda Calculus, Principles of Programming Languages, 1986.

[JM86] N. D. Jones, A. Mycroft: Dataflow of applicative programs using minimal function graphs, Principles of Programming Languages, 1986.

[M81] A. Mycroft: Abstract interpretation and optimising transformations for applicative programs, Ph. D. thesis, University of Edinburgh, 1981.

[MR90] T. J. Marlowe, B. G. Ryder: Properties of data flow frameworks - A unified model, Acta Informatica vol. 28, 1990.

[NN89] H. R. Nielson, F. Nielson: Transformations on Higher-Order Functions, Functional Programming Languages and Computer Architectures, 1989.

[NN92] H. R. Nielson, F. Nielson: Semantics with Applications - A Formal Introduction for Computer Science, Wiley (to appear in early 1992).

[PC87] S. Peyton-Jones, C. Clack: Finding fixpoints in abstract interpretations, in: Abstract Interpretations of Declarative Languages (edited by S. Abramsky & C. Hankin), Ellis Herwood, 1987.

[WH87] P. Wager, R. J. M. Hughes: Projections for Strictness Analysis, Functional Programming Languages and Computer Architecture, LNCS 274, 1987.

A An example analysis

To illustrate the practical consequences of the theoretics development performed in this paper we shall consider an analysis of a simple imperative language. The language has five syntactic categories:

$$
\begin{array}{lcl}
c \in \mathbf{Con} & : & \text{constants} \\
x \in \mathbf{Var} & : & \text{variables} \\
 & & \mathbf{Var} = \{\mathtt{x}_1, \mathtt{x}_2, \cdots, \mathtt{x}_n\} \\
a \in \mathbf{Aexp} & : & \text{arithmetic expressions} \\
 & & a ::= c \mid x \mid a_1 * a_2 \mid a_1 - a_2 \mid \cdots \\
b \in \mathbf{Bexp} & : & \text{boolean expressions} \\
 & & b ::= a_1 = a_2 \mid \neg b \mid \cdots \\
S \in \mathbf{Stm} & : & \text{statements} \\
 & & S ::= x := a \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\
 & & \quad\;\; \texttt{while } b \texttt{ do } S
\end{array}
$$

The semantics of this language should be intuitively clear. To describe the analysis we assume that

$\mathcal{I} \subseteq \mathbf{Var}$ is a set of *input variables*, and
$\mathcal{Q} \subseteq \mathbf{Var}$ is a set of *output variables*.

The question to be asked of a statement then is whether there is a functional dependency between the input and output variables, that is whether the final values of the output variables only depend on the initial values of the input variables and not on the initial values of other variables. As an example assume $\mathcal{I} = \{\mathtt{x}_1\}$ and $\mathcal{Q} = \{\mathtt{x}_2\}$ and consider the programs

$$\begin{array}{lcl} \mathtt{fac} & \equiv & \mathtt{x}_2\mathtt{:=}\ \mathtt{1};\ \mathtt{while}\ \neg(\mathtt{x}_1\mathtt{=}\ \mathtt{0})\ \mathtt{do}\ (\mathtt{x}_2\mathtt{:=}\ \mathtt{x}_2\mathtt{*x}_1;\ \mathtt{x}_1\mathtt{:=}\ \mathtt{x}_1\mathtt{-1}) \\ \mathtt{fac'} & \equiv & \mathtt{while}\ \neg(\mathtt{x}_1\mathtt{=}\ \mathtt{0})\ \mathtt{do}\ (\mathtt{x}_2\mathtt{:=}\ \mathtt{x}_2\mathtt{*x}_1;\ \mathtt{x}_1\mathtt{:=}\ \mathtt{x}_1\mathtt{-1}) \end{array}$$

The final value of $\mathtt{x}_2$ in $\mathtt{fac}$ will only depend on the initial value of $\mathtt{x}_1$ so there is a functional dependency between input and output variables. However, the final value of $\mathtt{x}_2$ in $\mathtt{fac}'$ will depend on the initial value of $\mathtt{x}_1$ but also on the initial value of $\mathtt{x}_2$; since $\mathtt{x}_2$ is not an input variable we do not have the required functional dependency.

The analysis we operate on two properties

$$\begin{array}{lcl} \mathtt{0} & : & \text{meaning that the value } \textit{definitely} \text{ only depends on the} \\ & & \text{values of input variables,} \\ \mathtt{1} & : & \text{meaning that the value } \textit{may} \text{ depend on the values of} \\ & & \text{non-input variables.} \end{array}$$

We shall write $\mathbf{2} = \{\mathtt{0}, \mathtt{1}\}$ for this set of properties and equip it with the partial ordering $\sqsubseteq$ defined by $\mathtt{0} \sqsubseteq \mathtt{1}$. Then $(\mathbf{2}, \sqsubseteq)$ is a complete lattice and the least upper bound operation will be written $\sqcup$.

The analysis will keep track of the properties of the variables. However to get a provably correct analysis we also need to keep track of whether or not the *flow of control* only depends on the initial values of the input variables. Therefore the analysis will associate each statement S with a function

$$\mathcal{P}[\![S]\!] : \mathbf{2}^{n+1} \to \mathbf{2}^{n+1}$$

that given properties $(p_1, \cdots, p_n, p_{n+1})$ of the variables $\mathtt{x}_1, \cdots, \mathtt{x}_n$ and of the flow of control holding *before* S will determine the similar information holing *after* S. The definition of $\mathcal{P}$ uses similar functions

$$\begin{array}{lcl} \mathcal{PA}[\![a]\!] & : & \mathbf{2}^{n+1} \to \mathbf{2} \\ \mathcal{PB}[\![b]\!] & : & \mathbf{2}^{n+1} \to \mathbf{2} \end{array}$$

$$
\begin{array}{lcl}
\mathcal{PA}[\![c]\!](p_1,\cdots,p_{n+1}) & = & p_{n+1} \\
\mathcal{PA}[\![\mathtt{x}_i]\!](p_1,\cdots,p_{n+1}) & = & p_i \sqcup p_{n+1} \\
\mathcal{PA}[\![a_1 * a_2]\!] & = & \mathcal{PA}[\![a_1]\!] \sqcup \mathcal{PA}[\![a_1]\!] \\
\mathcal{PA}[\![a_1 - a_2]\!] & = & \mathcal{PA}[\![a_1]\!] \sqcup \mathcal{PA}[\![a_1]\!] \\
 & \vdots & \\
\mathcal{PB}[\![a_1 = a_2]\!] & = & \mathcal{PA}[\![a_1]\!] \sqcup \mathcal{PA}[\![a_1]\!] \\
\mathcal{PB}[\![\neg\, b]\!] & = & \mathcal{PB}[\![b]\!] \\
 & \vdots &
\end{array}
$$

Table 1: Analysis of expressions.

$$
\begin{array}{l}
\mathcal{P}[\![\mathtt{x}_i := a]\!](p_1,\cdots,p_i,\cdots,p_{n+1}) = \\
\qquad (p_1,\cdots \mathcal{PA}[\![a]\!](p_1,\cdots,p_{n+1}),\cdots,p_{n+1}) \\
\mathcal{P}[\![S_1; S_2]\!] = \mathcal{P}[\![S_2]\!] \circ \mathcal{P}[\![S_1]\!] \\
\mathcal{P}[\![\mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2]\!] = \textsc{check}\ (\mathcal{PB}[\![b]\!]) \sqcup \mathcal{P}[\![S_1]\!] \sqcup \mathcal{P}[\![S_2]\!] \\
\mathcal{P}[\![\mathtt{while}\ b\ \mathtt{do}\ S]\!] = \mathit{FIX}\ H \\
\qquad \text{where } H\ h = \textsc{check}\ (\mathcal{PB}[\![b]\!]) \sqcup h \circ \mathcal{P}[\![S]\!]
\end{array}
$$

Table 2: Analysis of statements.

defined for arithmetic and boolean expressions in Table 1. Also it uses the auxiliary function

$$\textsc{check}\text{:}\ (\mathbf{2}^{n+1} \to \mathbf{2}) \to (\mathbf{2}^{n+1} \to \mathbf{2}^{n+1})$$

defined by

$$\textsc{check}\ h\ ps = \begin{cases} ps & \text{if } h\ ps = \mathtt{0} \\ (\mathtt{1},\cdots,\mathtt{1}) & \text{if } h\ ps = \mathtt{1} \end{cases}$$

The definition of $\mathcal{P}$ is given in Table 2.

The overt algorithm for testing whether or not there is a functions dependency between the set $\mathcal{I}$ of input variables and the set $\mathcal{Q}$ of output variables for a statement S then proceeds as follows:

- construct the tuple $ps = (p_1,\cdots,p_n,\mathtt{0})$ where $p_i = \mathtt{0}$ if and only if $x_i \in \mathcal{I}$

- apply the analysis to get $ps' = \mathcal{P}[\![S]\!]ps$,
- let $ps' = (p'_1, \cdots, p'_n, p'_{n+1})$ and return

$$\begin{array}{lcl} \text{YES} & : & \text{if } p'_{n+1} = \mathsf{0} \text{ and } p'_i = \mathsf{0} \text{ for } x_i \in \mathcal{Q}\text{, and} \\ \text{NO?} & : & \text{otherwise.} \end{array}$$

Example A.3: For the program `fac` we can assume that $\mathbf{Var} = \{\mathtt{x}_1, \mathtt{x}_2\}$ (i.e. $n = 2$) and that $\mathcal{I} = \{\mathtt{x}_1\}$ and $\mathcal{Q} = \{\mathtt{x}_2\}$. We then get

$$\mathcal{P}[\![\mathtt{fac}]\!](\mathsf{0},\mathsf{1},\mathsf{0}) = (\mathit{FIX}\ H)(\mathsf{0},\mathsf{0},\mathsf{0})$$

where

$$\begin{array}{rcl} H\ h & = & \text{CHECK } (\lambda(p_1,p_2,p_3).p_1 \sqcup p_3) \\ & \sqcup & h\ \circ (\lambda(p_1,p_2,p_3).(p_1 \sqcup p_3, p_1 \sqcup p_2 \sqcup p_3, p_3)) \end{array}$$

so that

$$H\ h\ (p_1,p_2,p_3) = \begin{cases} (\mathsf{0},p_2,\mathsf{0}) \sqcup h(\mathsf{0},p_2,\mathsf{0}) & \text{if } p_1 = p_3 = \mathsf{0} \\ (\mathsf{1},\mathsf{1},\mathsf{1}) & \text{otherwise} \end{cases}$$

To determine $\mathit{FIX}\ H$ we calculate the iterands $H^i\bot$:

$$\begin{array}{rcl} H^0\bot(p_1,p_2,p_3) & = & (p_1,p_2,p_3) \\ H^1\bot(p_1,p_2,p_3) & = & \begin{cases} (\mathsf{0},p_2,\mathsf{0}) & \text{if } p_1 = p_3 = \mathsf{0} \\ (\mathsf{1},\mathsf{1},\mathsf{1}) & \text{otherwise} \end{cases} \\ H^2\bot(p_1,p_2,p_3) & = & \begin{cases} (\mathsf{0},p_2,\mathsf{0}) & \text{if } p_1 = p_3 = \mathsf{0} \\ (\mathsf{1},\mathsf{1},\mathsf{1}) & \text{otherwise} \end{cases} \end{array}$$

and we see that $\mathit{FIX}\ H = H^1\bot$. Then

$$\mathcal{P}[\![\mathtt{fac}]\!](\mathsf{0},\mathsf{1},\mathsf{0}) = (\mathsf{0},\mathsf{0},\mathsf{0})$$

and the algorithm we give the answer YES.

Similarly we get

$$\mathcal{P}[\![\texttt{fac}']\!](0,1,0) = (\mathit{FIX}\ H)(0,1,0) = (0,1,0)$$

and the algorithm returns the answer NO? as expected. $\square$

The analysis fulfils certain properties that are referred to in the body of the paper:

Fact A4: The functions $H : (\mathbf{2}^{n+1} \to \mathbf{2}^{n+1}) \to (\mathbf{2}^{n+1} \to \mathbf{2}^{n+1})$ defined by

$$H\ h = \text{CHECK}\ (f) \sqcup h \circ g$$

is a continuous function for all choices of $f : \mathbf{2}^{n+1} \to \mathbf{2}$ and $g : \mathbf{2}^{n+1} \to \mathbf{2}^{n+1}$. $\square$

Fact A.5: $\mathcal{P}[\![S]\!]$ is a strict and additive function. $\square$

B Correctness of the analysis

To be able to demonstrate the correctness of the analysis we need a formal semantics for the imperative language. To this end Tables B.1 and B.2 define semantic functions

$$\begin{array}{lcll} \mathcal{SA} & : & \textbf{Aexp} & \to & (\mathbf{Z}^n \to \mathbf{Z}) \\ \mathcal{SB} & : & \textbf{Bexp} & \to & (\mathbf{Z}^n + \mathbf{T}) \\ \mathcal{S} & : & \textbf{Stm} & \to & (\mathbf{Z}^n \hookrightarrow \mathbf{Z}^n) \end{array}$$

where $\mathbf{Z}$ is the set of integers, $\mathbf{T}$ is the set of truth values ($\texttt{tt}$ and $\texttt{ff}$) and $\mathbf{Z}^n \hookrightarrow \mathbf{Z}^n$ is the set of *partial* functions from $\mathbf{Z}^n$ to $\mathbf{Z}^n$. Alternatively one could have used total functions throughout and made $\mathbf{Z}$ and $\mathbf{T}$ into partially ordered sets themselves. The auxiliary function `cond` used in Table 4 is defined by

$$\texttt{cond}(p, g_1, g_2)(v_1, \cdots, v_n) = \begin{cases} g_1(v_1, \cdots, v_n) & \text{if } p(v_1, \cdots, v_n) = \texttt{tt} \\ g_2(v_1, \cdots, v_n) & \text{if } p(v_1, \cdots, v_n) = \texttt{ff} \end{cases}$$

To express correctness we begin by defining a relation. For tuples $(v_1, \cdots, v_n)$, $(v'_1, \cdots, v'_n) \in \mathbf{Z}^n$ and $(p_1, \cdots, p_{n+1}) \in \mathbf{2}^{n+1}$ we write

$$(v_1, \cdots, v_n) \equiv (v'_1, \cdots, v'_n) \ \underline{\text{rel}} \ (p_1, \cdots, p_{n+1})$$

whenever

$$p_{n+1} = \mathtt{1} \text{ or } (\forall i \leq n \text{ . if } p_i = \mathtt{0} \text{ then } v_i = v'_i)$$

Fact B.3: If $(v_1, \cdots, v_n) \equiv (v'_1, \cdots, v'_n) \ \underline{\text{rel}} \ (p_1, \cdots, p_{n+1})$ and $\mathcal{PA}[\![a]\!](p_1, \cdots, p_{n+1}) = \mathtt{0}$ then

$$\mathcal{SA}[\![a]\!](v_1, \cdots, v_n) = \mathcal{SA}[\![a]\!](v'_1, \cdots, v'_n)$$

□

$$\begin{array}{lcl}
\mathcal{SA}[\![c]\!](v_1, \cdots, v_n) & = & c \\
\mathcal{SA}[\![\mathtt{x}_i]\!](v_1, \cdots, v_n)\hat{E} & = & v_i \\
\mathcal{SA}[\![a_1 * a_2]\!](v_1, \cdots, v_n) & = & \mathcal{SA}[\![a_1]\!](v_1, \cdots, v_n) * \mathcal{SA}[\![a_2]\!](v_1, \cdots, v_n) \\
\mathcal{SA}[\![a_1 - a_2]\!](v_1, \cdots, v_n) & = & \mathcal{SA}[\![a_1]\!](v_1, \cdots, v_n) - \mathcal{SA}[\![a_2]\!](v_1, \cdots, v_n) \\
 & \vdots & \\
\mathcal{SB}[\![a_1 = a_2]\!](v_1, \cdots, v_n) & = & \begin{cases} \mathtt{tt} & \mathcal{SA}[\![a_1]\!](v_1, \cdots, v_n) = \mathcal{SA}[\![a_2]\!](v_1, \cdots, v_n) \\ \mathtt{ff} & \text{otherwise} \end{cases} \\
\mathcal{SB}[\![\neg b]\!](v_1, \cdots, v_n) & = & \begin{cases} \mathtt{tt} & \mathcal{SB}[\![b]\!](v_1, \cdots, v_n) = \mathtt{ff} \\ \mathtt{ff} & \text{otherwise} \end{cases} \\
 & \vdots &
\end{array}$$

Table 3: Semantics of expressions

$$\begin{array}{l}
\mathcal{S}[\![\mathtt{x}_i := a]\!](v_1, \cdots, v_n) = (v_1, \cdots, \mathcal{SA}[\![a]\!](v_1, \cdots, v_n), \cdots, v_n) \\
\mathcal{S}[\![S_1; S_2]\!] = \mathcal{S}[\![S_2]\!] \circ \mathcal{S}[\![S_1]\!] \\
\mathcal{S}[\![\mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2]\!] = \mathtt{cond}(\mathcal{SB}[\![b]\!], \mathcal{S}[\![S_1]\!], \mathcal{S}[\![S_2]\!]) \\
\mathcal{S}[\![\mathtt{while}\ b\ \mathtt{do}\ S]\!] = \mathit{FIX}\ F \\
\qquad \text{where } F\ f = \mathtt{cond}(\mathcal{SB}[\![b]\!], f \circ \mathcal{S}[\![s]\!], \mathtt{id})
\end{array}$$

Table 4: Semantics of statements

Fact B.4: If $(v_1, \cdots, v_n) \equiv (v'_1, \cdots, v'_n) \ \underline{\text{rel}} \ (p_1, \cdots, p_{n+1})$ and $\mathcal{PB}[\![b]\!](p_1, \cdots, p_{n+1}) = \mathtt{0}$ then

$$\mathcal{PB}[\![B]\!](v_1, \cdots, v_n) = \mathcal{PB}[\![b]\!](v'_1, \cdots, v'_n)$$

$\square$

Fact B.5: If $(v_1, \cdots, v_n) \equiv (v'_1, \cdots, v'_n)$ $\underline{\text{rel}}$ $(p_1, \cdots, p_{n+1})$ and $\mathcal{P}[\![S]\!](p_1, \cdots, p_{n+1}) = (p'_1, \cdots, p'_{n+1})$ with $p'_{n+1} = \mathsf{0}$ then

- $\mathcal{S}[\![S]\!](v_1, \cdots, v_n)$ and $\mathcal{S}[\![S]\!](v'_1, \cdots, v'_n)$ are both undefined, or
- $\mathcal{S}[\![S]\!](v_1, \cdots, v_n)$ and $\mathcal{S}[\![S]\!](v'_1, \cdots, v'_n)$ are both defined and
$$\mathcal{S}[\![S]\!](v_1, \cdots, v_n) \equiv \mathcal{S}[\![S]\!](v'_1, \cdots, v'_n) \ \underline{\text{rel}} \ ((p'_1, \cdots, p'_{n+1})$$

$\square$

The correctness of the algorithm of Appendix A now follows.