Finiteness Conditions for Strictness Analysis*

Flemming Nielson, Hanne Riis Nielson
Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark
e-mail:{fnielson,hrnielson}@daimi.aau.dk

July 1993

Abstract

We give upper bounds on the number of times the fixed point operator needs to be unfolded for strictness analysis of functional languages with lists. This extends previous work both in the syntax-directed nature of the approach and in the ability to deal with Wadler's method for analysing lists. Limitations of the method are indicated.

1 Introduction

Strictness analysis for functional programs by means of abstract interpretation is a very powerful technique: both in terms of the accuracy of the results produced and in the applicability to various language constructs. The main disadvantage of the method is that the computational cost may be too high for many applications and as a result the method is not usually incorporated in a compiler.

*Excluding the appendices, this is a preprint of a paper to appear in Proceedings of the *Workshop on Static Analysis 1993* to be published by Springer Lecture Notes in Computer Science.

Rather than resorting to cruder methods, e.g. based on variations of type analysis, we believe that it is possible to identify certain programs where the cost may be analysed in advance and determined not to be excessive. This would allow the compiler to perform the abstract interpretation in those instances where the cost is not prohibitive. The notion of cost we will be taking throughout this paper is the number of iterations needed to reach the fixed point.

In [6] we developed first results along this line. Section 2 contains a brief review of the main results of [6] but with a change of emphasis that is more suited to a structural approach (for functional programs). Section 3 then develops our main results for simple strictness analysis and in Section 4 we add the analysis of lists using Wadler's "inverse cons" method. Finally, Section 5 contains the conclusion and the appendices some of the proofs.

2 Boundedness

The abstract interpretation of a recursive program gives rise to a functional

$$H : (A \to B) \to (A \to B)$$

Typically, and as we shall assume throughout, A and B are finite complete lattices: this means that all subsets Y of A (resp. B) have least upper bounds denoted $\bigsqcup Y$ (or $y_1 \sqcup \cdots \sqcup y_n$ if $Y = \{y_1, \cdots, y_n\}$). Furthermore all functions will be monotone: for H this means that if $h_1 \sqsubseteq h_2$ then $H(h_1) \sqsubseteq H(h_2)$ for all $h_1, h_2 \in A \to B$. The least fixed point of H is given by

$$\text{FIX}$$

$$H = \bigsqcup \{H^i \perp \mid i \geq 0\}$$

where $\perp$ is the least element of $A \to B$. Clearly if $H^{k+1} \perp = H^k \perp$ then FIX $H = H^k \perp$ because of the monotonicity of H. By the finiteness of A and B there will always be some (perhaps large) k such that this holds.

The notion of k-boundedness is of interest when the functional H is additionally additive: this is the case when $H(h_1 \sqcup h_2) = H(h_1) \sqcup H(H(h_2)$ for all $h_1, h_2 \in A \to B$. It is helpful to write

$$H^{[k]}h = \bigsqcup\{H^i h \mid 0 \geq i < k\}$$

and motivated by [4] we say that H is *k-bounded* if

$$H^k h \sqsubseteq H^{[k]}h \text{ for all } h \in A \to B$$

We shall write

$$H \in \mathcal{A}(k)$$

to mean that H is additive and k-bounded.

Proposition 2.1 When $H \in \mathcal{A}(k)$, i.e. H is k-bounded and additive, we have FIX $H = H^{[k]} \perp = H^{k-1} \perp = H^k \perp$.

Proof: This is a revised version of [6, Lemma 11]; some key facts (necessary for the subsequent proofs) are presented in Appendix A. $\square$

3 Strictness Analysis

To motivate the form of the functionals considered we begin with a brief review of strictness analysis. To this end consider a simply typed λ-calculus with constants, a conditional and a fixed point construct. The types are

$$t ::= \texttt{num} \mid \texttt{bool} \mid t_1 \times t_2 \mid t_1 \to t_2$$

Types:	$\lVert \texttt{num} \rVert$	$=$	$\mathbf{2}$
	$\lVert \texttt{bool} \rVert$	$=$	$\mathbf{2}$
	$\lVert t_1 \times t_2 \rVert$	$=$	$\lVert t_1 \rVert \times \lVert t_2 \rVert$
	$\lVert t_1 \to t_2 \rVert$	$=$	$\lVert t_1 \rVert \to \lVert t_2 \rVert$
Expressions:	$\lVert c \rVert \rho$	$=$	$\hat{c}$
	$\lVert x \rVert \rho$	$=$	ρx
	$\lVert \texttt{fst}\ e \rVert \rho$	$=$	$fst(\lVert e \rVert \rho)$
	$\lVert \texttt{snd}\ e \rVert \rho$	$=$	$snd(\lVert e \rVert \rho)$
	$\lVert (e_1, e_2 \rVert \rho$	$=$	$(\lVert e_1 \rVert \rho, \lVert e_2 \rVert \rho)$
	$\lVert \texttt{lam}\ x.e \rVert \rho$	$=$	$\lambda a. \lVert e \rVert (\rho[x \mapsto a])$
	$\lVert e_1 e_2 \rVert \rho$	$=$	$\lVert e_1 \rVert \rho (\lVert e_2 \rVert \rho)$
	$\lVert \texttt{if}\ e\ \texttt{then}\ e_2\ \texttt{else}\ e_2 \rVert$	$=$	$\lVert e \rVert \rho \triangleright (\lVert e_1 \rVert \rho \sqcup \lVert e_2 \rVert \rho)$
	$\lVert \texttt{fix}\ e \rVert \rho$	$=$	$\text{FIX}\ (\lVert e \rVert \rho)$

Table 1: Strictness Analysis

and the expressions are

$$\begin{aligned} e ::= {} & c \mid x \mid \texttt{fst}\ e \mid \texttt{snd}\ e \mid (e_1, e_2) \mid \texttt{lam}\ x.e \mid e_1\ e_2 \mid \\ & \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \text{FIX}\ e \end{aligned}$$

The expressions are assumed to be well-typed but it is outside the scope of this paper to present the formal machinery for enforcing this.

The strictness analysis is specified in Table 1. In the *type part* we write **2** for the complete lattice $(\{0,1\}, \sqsubseteq)$ where $0 \sqsubseteq 1$. We write $D_1 \times D_2$ for the cartesian product of D_1 and D_2, and we write $D_1 \to D_2$ for the complete lattice of monotone functions from D_1 to D_2 ordered pointwise.

The *expression part* of the analysis associates a property $\hat{c}$ with each constant c. To specify the analysis of expressions with free variables we use an environment ρ mapping variables to properties. The analysis of the conditional uses the operator $\triangleright$ defined by

$$d_0 \triangleright d = \begin{cases} \bot & \text{if } d_0 = 0 \\ d & \text{if } d_0 = 1 \end{cases}$$

where $\bot$ is the least element of the lattice that d belongs to and where d_0 belongs to **2**. This is then lifted pointwise to functions

$$(h_0 \rhd h) = \lambda d.(h_0\ d) \triangleright (h\ d).$$

A Structural Approach to Boundedness

Given a functional H as might arise from the above strictness analysis the aim now is to find sufficient conditions for H to be additive and k-bounded for some hopefully low value of k. We begin with a simple fact and a brief review of the main results from [6]; then we move on to a more general treatment of the operators $\sqcup$ and $\rhd$.

Fact 3.1 $Id = \lambda h.h \in \mathcal{A}(1), \lambda h.g \in \mathcal{A}(2)$ and $\lambda h. \perp \in \mathcal{A}(1)$.

The monotone length $len_m(h)$ of a function $h \in A \to B$ is given by

$$len_m(h) = max\{l_m(h, a) \mid a \in A\}$$

where $l_m(h, a) = min\{i \mid h^i(a) \in \{a, h(a), \ldots, h^{i-1}(a)\}\downarrow, i > O\}$. Here we write $Y\downarrow$ for the down-closure of Y, i.e. the set $\{d \mid \exists y \in Y : d \sqsubseteq y\}$.

Lemma 3.2 $\lambda h.\ g_1 \circ h \circ g_2 \in \mathcal{A}(len_m(g_1)) \cdot len_m(g_2))$ if g_1 is additive.

Proof: This is essentially [6, Lemma 25]. $\square$

Corollary 3.3 $\lambda h.\ h \circ g \in \mathcal{A}(len_m(g))$ and if g is additive then $\lambda h.\ g \circ h \in \mathcal{A}(len_m(g))$.

Proof: When id is the identity function we have $len_m(id) = 1$. $\square$

We now extend the development of [6] by considering the least upper bound operator.

Lemma 3.4 $H_1 \sqcup H_2 \in \mathcal{A}(k_1 + k_2 - 1)$ if $H_1 \in \mathcal{A}(k_1), H_2 \in \mathcal{A}(k_2)$ and if H_1 and H_2 commute (i.e. $H_1 \circ H_2 = H_2 \circ H_1$) and B is not trivial.

Proof: See Appendix B. $\square$

Corollary 3.5 $H \sqcup Id \in \mathcal{A}(k)$ if $H \in \mathcal{A}(k)$ and B is not trivial.

Lemma 3.6 $H_1 \sqcup H_2 \in \mathcal{A}(k_1 + 1)$ if $H_1 \in \mathcal{A}(k)$ and $H_2 = \lambda h.g$ (for some $g \in A \to B$).

Proof: See Appendix B. $\square$

Remark This shows that if $H = \lambda h.g \sqcup (G\ h)$ and $G \in \mathcal{A}(k)$ then $H \in \mathcal{A}(k{+}1)$ so that FIX $H = H^k \perp$ (as opposed to $H^{k-1} \perp$). Since functionals of the form H typically arise for iterative programs this explains the naturality of the definition of k-boundedness in the setting of [4]; in our setting it might have been more natural to redefine k-boundedness of H to mean $H^{k+1} \sqsubseteq H^{[k+1]}$.

We next turn to the $\rhd$ operator.

Fact 3.7 We have the following properties of $\rhd$:

- $h_0 \rhd (h_1 \sqcup h_2) = (h_0 \rhd h_1) \sqcup (h_0 \rhd h_2)$.
- $(h_1 \rhd h_2) \circ h_3 = (h_1 \circ h_3) \rhd (h_2 \circ h_3)$.
- $h_1 \rhd (h_2 \rhd h_3) = (h_1 \sqcap h_2) \rhd h_3$.

Lemma 3.8 $\lambda h.g \rhd (H\ h) \in \mathcal{A}(k)$ and if there exises a (monotone) functional $\delta H \in (A \to \mathbf{2}) \to (\mathbf{A} \to \mathbf{2})$ such that $H(h_1 \rhd h_2) = (\delta H(h_1)) \rhd H(h_2)$ for $h_1, h_2 \in A \to B$.

Proof: See Appendix B. □

Fact 3.9 $\delta(\lambda h.\ h \circ g_2)$ and if g_1, is strict then $\delta(\lambda h.\ g_1 \circ h \circ g_2) = \lambda h.\ h \circ g_2$.

Example 3.10 As an example of a tail-recursive program we consider the *factorial program with an accumulator*. It can be written as

$$\begin{array}{ll}\text{FIX } (\texttt{lam}\ fac.\ \texttt{lam}\ xa. & \texttt{if}\ (=0)(\texttt{fst}\ xa\ \texttt{then}\ \texttt{snd}\ xa \\ & \texttt{else}\ fac((-1)(\texttt{fst}\ xa), *\ xa))\end{array}$$

Here $(= 0)$ tests for equality with 0, $*$ is the multiplication operator and (-1) subtracts one from its argument. The strictness analysis will therefore give rise to a functional H of the form

$$H\ h = g_0 \rhd (g_1 \sqcup h \circ g_2)$$

which may be rewritten to

$$H\ h = (g_0 \rhd g_1) \sqcup (g_0 \rhd (h \circ g_2))$$

using Fact 3.7. The functions g_0, g_1 and g_2 are given by

$$\begin{array}{rcl} g_0 & = & \mathit{fst} \\ g_1 & = & \mathit{snd} \\ g_2 & = & \mathit{tuple}(\mathtt{fst}, \hat{*}) \end{array}$$

where $\mathit{tuple}(h_1, h_2)x = (h_1(x), h_2(x))$ and $\hat{*}(x_1, x_2) = x_1 \sqcap x_2$. Since g_2 is reductive (i.e. $g_2 \sqsubseteq \mathit{id}$) it follows that $\mathit{len}_m(g_2) = 1$. By Corollary 3.3, Lemma 3.8, Fact 3.9, and Lemma 3.6 the functional H is 2-bounded and by Proposition 2.1 the first unfolding will give the fixed point.

Lemma 3.11 Let $H : (A \to B) \to (A \to B)$ be defined by

$$H\ h = g \circ \mathit{tuple}(h \circ g_1, g_2)$$

where $g : B \times B \to B,\ g_1 : A \to A$ and $g_2 : A \to B$. Assume that

- g is *associative* i.e. $g(g(b_1, b_2), b_3) = g(b_1, g(b_2, b_3))$ for all $b_1, b_2, b_3 \in B$,
- g is strict and additive *in its left argument*, i.e. $g(\bot, b) = \bot$ and $g(b_1 \sqcup b_2) = g(b_1, b) \sqcup g(b_2, b)$ for all $b, b_1, b_2 \in B$, and
- g has a *right identity* b_0 i.e. $g(b, b_0) = b$ for all $b \in B$, and
- $k = \mathit{len}_m(\mathit{tuple}(g_1 \circ \mathit{fst},\ g \circ \mathit{tuple}(g_2 \circ \mathit{fst},\ \mathit{snd})))$.

Then $H \in \mathcal{A}(k)$ and $\delta H = \lambda h.\ h \circ g_1$.

Proof: See Appendix B. This result was stated but not proved in [6]. □

One undesirable feature of the above lemma is that we need to take the length of a composite function. However, the lemma suffices for treating a non-accumulator version of factorial.

Example 3.12 The usual *factorial program* can be written as

$$\text{FIX } (\mathtt{lam}\ \mathit{fac}.\ \mathtt{lam}\ x.\ \mathtt{if}\ (=0)(x)\ \mathtt{then}\ 1\ \mathtt{else}\ *(fac((-1)x), x))$$

The strictness analysis will therefore give rise to a functional H of the form

$$H\ h = g_0 \rhd (g_1 \sqcup g \circ tuple(h \circ g_2, g_3))$$

which may be rewritten to

$$H\ h = (g_0 \rhd g_1) \sqcup (g_0 \rhd g \circ tuple(h \circ g_2, g_3))$$

using Fact 3.7. The functions are $g_0 = \lambda x.x,\ g_1 = \lambda x.1,\ g_2 = \lambda x.x,\ g_3 = \lambda x.x$ and $g = \lambda(x_1, x_2).x_1 \sqcap x_2$. The function $tuple(g_2 \circ fst,\ g \circ tuple(g_3 \circ fst,\ snd))$ then amounts to the function called g_2 in Example 3.10.

4 Strictness Analysis for Lists

We shall now extend the typed λ-calculus with lists:

$$t ::= \cdots \mid t\ \texttt{list}$$

The syntax of expressions is extended with constructs for building lists and for taking them apart:

$$e ::= \cdots \mid \texttt{nil} \mid \texttt{cons}\ e_1\ e_2 \mid \texttt{case}\ e\ \texttt{of}\ \texttt{nil} : e_1 \parallel \texttt{cons}\ x_1, x_2 : e_2$$

We shall follow [9] and construct the lattice of properties for lists by a double lifting of the lattice of the element type: if D is the lattice of properties for the elements of the list then $(D_\perp)_\perp$ will be the lattice of properties of the lists. The least element of $(D_\perp)_\perp$ is denoted 0, the second least element 1 and the remaining elements are denoted $d\epsilon$ where d is an element of D. We write $\top$ for the largest element of D. The idea then is that

- 0: denotes the undefined list,
- 1: denotes additionally all infinite lists and all partial lists ending in the undefined list,
- $d\epsilon$: denotes additionally all finite lists where the meet of the elements satisfies property d (for d not being $\top$)[1], and
- $\top\epsilon$: denotes all lists.

Types:	$\lVert t\ \texttt{list} \rVert$	$=$	$(\lVert t \rVert_\perp)_\perp$
Expressions:	$\lVert \texttt{nil} \rVert\ \rho$	$=$	$\top_\epsilon$
	$\lVert \texttt{cons}\ e_1 e_2 \rVert \rho$	$=$	$(\lVert e_1 \rVert \rho)\epsilon \sqcap (\lVert e_2 \rVert_\rho)$
	$\lVert \texttt{case}\ e\ \texttt{of nil} : e_1 \vert$		
	$\texttt{cons}\ x_1 x_2 : e_2 \rVert \rho$	$=$	$(isnil(\lVert e \rVert \rho) \rhd \lVert e_1 \rVert \rho) \sqcup$
			$\bigsqcup(\mathcal{P}(\lambda(a_1, a_2).\lVert e_2 \rVert \rho[x_1 \mapsto a_1[x_2 \mapsto a_2])$
			$(Split(\lVert e \rVert \rho)))$

Table 2: Strictness analysis for lists

The strictness analysis of Table 1 is now extended with the clauses of Table 2. For `nil` we observe that the only property describing the empty list is T_ϵ. For `cons` $e_1 e_2$ we combine the property of the head with the property of the tail using a greatest lower bound operation. For the `case` construct we want to "reverse" this construction. To this end we use two auxiliary operations

$$isnil\text{:}\ (D\perp)\perp \rightarrow \mathbf{2}$$
$$split\text{:}\ (D\perp)\perp \rightarrow \mathcal{P}(D \times (D\perp)\perp$$

Here $\mathcal{P}(D)$ is the *lower powerdomain* of D. When D is a finite complete lattice one may take $\mathcal{P}(D)$ to have as elements those non-empty subsets Y of D that satisfy $Y = Y\downarrow$ (i.e. Y is downward closed); the partial order is subset inclusion. Then $\mathcal{P}(D)$ will also be a finite complete lattice with least element $\{\perp\}$ and greatest element D. We may now define the functions *isnil* and *split* by

$$isnil : d = \begin{cases} 0 & \text{if } d \neq T_\epsilon \\ 1 & \text{if } d = T_\epsilon \end{cases}$$

$$split\ d = \{(d_1, d_2) \mid d_1\epsilon \sqcap d_2 \sqsubseteq d\}$$

Thus *isnil* d will return 1 if d is a property of the empty list and *split* d will return (the downward closed set of) all possible pairs of properties that the head und the tail of the list could have had. In the case where $D = \mathbf{2}$ we can tabulate *isnil* and *split* as follows:

	0	1	0ϵ	1ϵ
isnil	O	O	O	1
split	$\{(1,0)\}\downarrow$	$\{(1,1)\}\downarrow$	$\{(0,1\epsilon),(1,0\epsilon)\}\downarrow$	$\{(1,1\epsilon)\}\downarrow$

In the definition of $\|\texttt{case}\ e\ \texttt{of nil} : e_1|\texttt{cons}\ x_1x_2 : e_2\|\rho$ we first determine the property of the list $\|e\|\rho$. If it could possibly be a property of the empty list we must have a contribution from $\|e_1\|\rho$; this is expresed using the $\rhd$ operator. Whether or not this is the case the property of the list is split into a set of properties of the head and the tail and we must have a contribution from $\|e_2\|\rho$ for each of these possibilities. This is expressed using the operator

$$\mathcal{P} : (D_1 \to D_2) \to (\mathcal{P}(D_1) \to \mathcal{P}(D_2))$$

which extends its first argument pointwise to operate on elements in the power domain: for $Y \in \mathcal{P}(D_1)$ we have

$$\mathcal{P}(h)(Y) = \{h(d) \mid d \in Y\}\downarrow$$

In other words $\mathcal{P}$ is extended to a functor. Finally, all contributions are combined by taking least upper bounds.

Boundedness Results for Lists

To obtain k-boundedness results for functionals arising from the analysis of lists we begin with a characterization of the $\mathcal{P}$ operator. For this it is helpful to write $\{\!|\,|\!\} = \lambda d.(\{d\}\downarrow)$.

Fact 4.1

- $\bigsqcup \circ \mathcal{P}(h_1 \sqcup h_2) = (\bigsqcup \circ \mathcal{P}(h_1)) \sqcup (\bigsqcup \circ \mathcal{P}(h_2))$
- $\mathcal{P}(h) \circ \{\!|\,|\!\} = \{\!|\,|\!\} \circ h$
- $\bigsqcup \circ \{\!|\,|\!\} = id$
- $\mathcal{P}(h_1 \circ h_2) = \mathcal{P}(h_1) \circ \mathcal{P}(h_2)$
- $\bigsqcup \circ \mathcal{P}(\bigsqcup) = \bigsqcup \circ \bigcup$

- $\bigcup \circ \mathcal{P}(\mathcal{P}(h)) = \mathcal{P}(h) \circ \bigcup$
- $\mathcal{P}(\mathcal{P}(h)) \circ \mathcal{P}(\{\!|\,|\!\}) = \mathcal{P}(\{\!|\,|\!\}) \circ \mathcal{P}(h)$
- $\bigcup \circ \mathcal{P}(\bigcup) \circ \mathcal{P}(\mathcal{P}(h)) = \bigcup \circ \mathcal{P}(h) \circ \bigcup$

Proof Most of these results are straightforward. Some of them are treated in greater detail in [2]. □

Instead of using the measure len_m, of Section 3 we shall be able to obtain better results by following [6] and defining

$$len_{sa}(h) = max\{l_{sa}(h,Y) \mid Y \in \mathcal{P}(A)\}$$

where $l_{sa}(h,Y) = min\{i \mid h^i(Y) \sqsubseteq \bigsqcup\{Y, h(Y), \ldots, h^{i-1}(Y)\}, i > 0\}$.

Fact 4.2 $1 \leq len_{sa}(h) \leq len_m(h)$ for all functions h.

Lemma 4.3 $\lambda h. \bigsqcup \circ \mathcal{P}(h \circ g_1) \circ g_0 \in \mathcal{A}(k)$ for $k = len_{sa}(\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))$.

Proof: See Appendix C. □

Example 4.4 The *length function* computing the length of a list can be written as

FIX(`lam` *length.* `lam` *xs.* `case` *xs* `of nil:` 0 || `cons` *y us:* (+ 1) (*length ys*))

The overall type of this program is (t_α `list`) $\rightarrow$ `num`. In the analysis we shall follow the approach of [1] and interpret the type t_α, by the domain **2**.

The strictness analysis gives rise to a functional H of the form

$$H\ h = ((isnil \circ g_0) \rhd g_1) \sqcup (\bigsqcup \circ \mathcal{P}(h \circ g_2) \circ split \circ g_0)$$

where

$$\begin{array}{rcl} g_0 & = & id \\ g_1 & = & \lambda xs.1 \\ g_2 & = & snd. \end{array}$$

Now consider

$$k = len_{sa}(g') \text{ where } g' = \mathcal{P}(snd) \circ \bigcup \circ \mathcal{P}(split) \circ \mathcal{P}(id)$$

One can show that g' is idempotent ($g' = g' \circ g'$) and this means that $len_{sa}(g') = 2$. It follows from Lemmas 4.3 and 3.6 that H is 3-bounded and hence by Proposition 2.1 only 2 iterations are needed to compute the fixed point. A simple calculation shows that indeed two iterations are needed.

Example 4.5 As an example of a tail recursive program we shall consider the function `foldl` with type $(t_\alpha \to t_\beta \to t_\alpha) \to t_\alpha \times (t_\beta\ \texttt{list}) \to t_\alpha$. It can be written as

$$\begin{array}{l} \texttt{lam}\ f.\ \mathrm{FIX}(\texttt{lam}\ fld.\ \texttt{lam}\ ax.\ \texttt{case snd}\ ax\ \texttt{of nil}:\ \texttt{fst}\ ax\ \| \\ \qquad\qquad \texttt{cons}\ y\ ys:\ \ fld\ ((f\ (\texttt{fst}\ ax)\ y),\ ys)) \end{array}$$

Interpreting the types t_α and t_β as $\mathbf{2}$ one can show that the strictness analysis gives rise to a functional H_g defined by

$$H_g\ h = ((isnil \circ g_0) \rhd g_1) \sqcup (\bigsqcup \circ \mathcal{P}(h \circ g_2) \circ pack \circ tuple(g_1, split \circ g_0))$$

where

$$\begin{array}{rcl} pack & = & \lambda(x, \{y_1, \ldots, y_n\}).\{(x, y_1), \ldots (x, y_n)\} \downarrow \\ g_0 & = & snd \\ g_1 & = & fst \\ g_2 & = & tuple(g \circ tuple(id, fst \circ snd), snd \circ snd). \end{array}$$

and g is the analysis (in uncurried form) of the parameter f. Thus we have to determine

$$k_g = len_{sa}(g') \text{ where } g' = \mathcal{P}(g_2) \circ \bigcup \circ \mathcal{P}(pack \circ tuple(g_1, split \circ g_0))$$

The value obtained for k_g will depend on the properties of g but one can show that in all cases $k_g \leq 3$. Hence H_g is 4-bounded and at most 3 iterations will be needed.

5 Conclusion

The computation of fixed points plays an important role in abstract interpretation and hence also for strictness analysis by means of abstract interpretation. One major problem is that the number of unfoldings needed for the fixed point operator may be very high. Nothing can be done about this in general, but the results of this paper may be used in a compiler when detecting the situations in which strictness analysis by abstract interpretation will not be prohibitively expensive.

In [3] the concatenation function on lists is defined as `foldr append nil` and is shown to give a function that is particularly bad to analyse. Our results do not directly improve upon this, but it is instructive to note that the results of Example 4.5 may be of use: if by program transformation we are able to translate the definition using `foldr` into one that uses `foldl` then the required number of iterations will be very low. Again one might expect such program transformations to be part of the compiler's repertoire for improving the performance of the program.

As [3] points out the costs involved in tabulating each iteration may also be very high. An idea to overcome this is to note that we need only know the value of FIX H for those arguments that come up in the "recursive calls" for the argument in which we are interested. Thus one might use "minimal function graphs" to keep track of the arguments needed and then it will only be necessary to tabulate the value of $H^k \perp$ on arguments in this set[2]. In general this set will not be a singleton as this is only the case for analysis functions that turn out to be additive [5] and this is not so for strictness analysis.

Acknowledgement

This research was partially supported by the DART-project (funded by the Danish Research Councils).

[2]Similarly, if we instead test for stabilization then it suffices to test for stabilization for elements in this set.

References

[1] S.Abramsky: Strictness Analysis and Polymorphic Invariance, *Programs as Data Objects*, Springer Lecture Notes in Computer Science **217** 1-23, 1986.

[2] G.L.Burn, C.Hankin, S.Abramsky: Strictness Analysis for Higher-Order Functions, *Science of Computer Programming* **7**,1986.

[3] S.Hunt, C.Hankin: Fixed Points and Frontiers: a New Perspective, *Journal of Functional Programming* **1**, 1991.

[4] T.J.Marlowe, B.G.Ryder: Properties of Data Flow Frameworks - A Unified Model, *Acta Informatica* **28**,1990.

[5] H.R.Nielson, F.Nielson: Bounded Fixed Point Iteration, *Proceedings of the ACM Sympoisium on Principles of Programming Languages*, 1992. An expanded version appeared in *Journal of Logic and Computation* **2** *4*, 1992.

[6] F.Nielson, H.R.Nielson: Finiteness Conditions for Fixed Point Iteration (Extended Abstract), *Proceedings of the ACM Symposium on LISP and Functional Programming*, 1992. An extended version appeared as [7].

[7] F.Nielson, H.R.Nielson: Finiteness Conditions for Fixed Point Iteration, Technical Report DAIMI PB-384, Aarhus University, Demnark, 1992. An extended abstract appeared as [6].

[8] F.Nielson, H.R.Nielson: *Two-Level Functional Languages*, Cambridge Tracts in Theoretical Computer Science **34**, Cambridge University Press, 1992.

[9] P.Wadler: Strictness Analysis on Non-Flat Domains (by Abstract Interpretation over Finite Domains), *Abstract Interpretation of Declarative Languages*, S.Abramsky and C.Hankin (eds.), Ellis Horwood, 1987

A Proofs from Section 2

In order to facilitate the proofs of Appendices B and C we shall review a few insights from [6].

It is helpful to tabulate the first few values of $H^{[n]}$:

	H^n	$H^{[n]}$
$n = 0$	Id	$\lambda h.\ \perp$
$n = 1$	H	Id
$n = 2$	$H \circ H$	$Id \sqcup H$
$n = 3$	$H \circ H \circ H$	$Id \sqcup H \sqcup (H \circ H)$

where $Id = \lambda h.h$ is the identity functional.

Fact A.1 We have the following results:

- If H is additive then $H^{[n+1]} = (H \sqcup Id)^n$.
- $(H^n \perp)_n$ and $(H^{[n]})_n$ are chains but $(H^n)_n$ need not be.
- $\lambda H.\ H^n$ and $\lambda H.\ H^{[n]}$ are monotone (for all n).

Fact A.2 When H is k-bounded and additive we have the following results:

- $\forall n \geq 0 : H^n \sqsubseteq H^{[k]}, H^{[n]} \sqsubseteq H^{[k]}$, and $H^{[n+k]} = H^{[k]}$
- $H^{[k]} \circ H \sqsubseteq H^{[k]}, H \circ H^{[k]} \sqsubseteq H^{[k]}$ and $H^{[k]} \circ H^{[k]} = H^{[k]}$.
- H is $(k+1)$-bounded.
- $k > 0$ or B is trivial (i.e. a one-point lattice).

Proposition 2.1 then essily follows: FIX $H = \bigsqcup\{H^n \perp \mid n \geq 0\} = H^{[k]} \perp = H^{k-1} \perp$. We refer to [6] for any missing details.

B Proofs from Section 3

Proof of Lemma 3.4: Write $k = k_1 + k_2 - 1$. We may calculate

$$(H_1 \sqcup H_2)^n = \bigsqcup_{i_1 \ldots i_n} H_{i_1} \circ \cdots \circ H_{i_n} = \bigsqcup_{p+q=n} H_1^p \circ H_2^q$$

where $H_{i_1} \circ \cdots \circ H_{i_n} = Id$ for $n = 0$; hence we have

$$(H_1 \sqcup H_2)^{[n]} = \bigsqcup_{p+q<n} H_1^p \circ H_2^q$$

Using the facts from Appendix A we have $k_i > 0$ and $H_i^{[k_i]} = (H_i \sqcup Id)^{k_i - 1}$. We may then calculate'

$$\begin{array}{rcl}
(H_1 \sqcup H_2)^n & = & \bigsqcup_{p+q=n} H_1^p \circ H_2^q \\
 & \sqsubseteq & \bigsqcup_{p+q=n} H_1^{[k_1]} \circ H_2^{[k_2]} \\
 & = & H_1^{[k_1]} \circ H_2^{[k_2]} \\
 & = & (H_1 \sqcup Id)^{k_1 - 1} \circ H_2 \sqcup Id)^{k_2 - 1} \\
 & \sqsubseteq & \bigsqcup_{p<k_1, q<k_2} H_1^p \circ H_2^q \\
 & \sqsubseteq & \bigsqcup_{p+q<k} H_1^p \circ H_2^q \\
 & = & (H_1 \sqcup H_2)^{[k]}
\end{array}$$

and this shows the result (when taking $n = k$).

Proof of Lemma 3.6: We may calculate

$$(H_1 \sqcup H_2)^n = \bigsqcup_{i_1 \cdots i_n} H_{i_1} \circ \cdots \circ H_{i_n} = H_1^n \sqcup (\bigsqcup_{p<n} H_1^p) \circ H_2$$

We then have

$$\begin{array}{rcl}
(H_1 \sqcup H_2)^{k+1} & \sqsubseteq & H_1^{[k]} \sqcup (H_1^{[k]} \circ H_2) \\
 & \sqsubseteq & (H_1 \sqcup H_2)^{[k]} \sqcup ((H_1 \sqcup H_2)^{[k]} \circ (H_1 \sqcup H_2)) \\
 & \sqsubseteq & (H_1 \sqcup H_2)^{[k]}
\end{array}$$

and this shows the result.

Proof of lemma 3.8: Write

$$G = \lambda h.\ g \rhd (H\ h)$$

By Fact 3.7 G is additive because H is. Next define G_0 by

$$G_0 = \lambda h.\ g \sqcap \delta H(h)$$

and note that G_0 is monotonic and hence $(G_0^n(\lambda x.1))_n$ is a decreasing chain.

We show

$$G^n(h) = G_0^n(\lambda x.1) \rhd H^n(h) = h$$

by induction on n. For $n = 0$ we have $G^0(h) = h$, $G_0^0(\lambda x.1) = \lambda x.1$, and $H^0(h) = h$ and this shows the base case.

For the induction step where $n = m + 1$ we have

$$\begin{array}{rcl} G^{m+1}(h) & = & G(G_0^m(\lambda x.1) \rhd H^m(h)) \\ & = & g \rhd H(G_0^m(\lambda x.1) \rhd H^m(h)) \\ & = & g \rhd (\delta H(G_0^m(\lambda x.1)) \rhd (H(H^m(h)))) \\ & = & (g \sqcap \delta H(G_0^m(\lambda x.1))) \rhd (H^{m+1}(h)) \\ & = & G_0^{m+1}(\lambda x.1)) \rhd H^{m+1}(h) \end{array}$$

where we have used Fact 3.7.

To show that G is k-bounded it is sufficient to consider h and x and to show

$$G^k\ h\ x \sqsubseteq G^{|k|}\ h\ x$$

and this amounts to

$$((G_0^k(\lambda x.1)\ x) \rhd (H^k\ h\ x)) \sqsubseteq \bigsqcup_{i<k}(((G_0^i(\lambda x.1)\ x) \rhd (H^i\ h\ x)))$$

If $G_0^k(\lambda x.1)\ x = 0$ this is immediate. Otherwise $(G_0^i(\lambda x.1)\ x) = 1$ for all $i < k$ (by $(G_0^n(\lambda x.1))_n$ being a decreasing chain) and it all amounts to

$$(H^k\ h\ x) \sqsubseteq \bigsqcup_{i<k} (H^i\ h\ x)$$

which follows from the assumption that H is k-bounded.

Proof of Lemma 3.11: This result was stated in [6] but no proof was given and the proof sketched in [7] was somewhat indirect. Hence we give the following direct proof.

Clearly H is additive because of the assumptions on g. Similarly δH is as stated because of the assumptions on g. For the k-boundedness of H we first show that

$$H^n(h) = g \circ \mathit{tuple}(h \circ g_1^n, H^{n-1}(g_2)) \qquad (1)$$

for $n > O$. The proof is by induction on n. The base case $n = 1$ is trivial so consider the induction step $n = m + 1$. We calculate

$$\begin{array}{rcl}
H^{m+1}(h) & = & g \circ \mathit{tuple}(H^m(h) \circ g_1, g_2) \\
& = & g \circ \mathit{tuple}(g \circ \mathit{tuple}(h \circ g_1^m, H^{m-1}(g_2)) \circ g_1, g_2) \\
& = & g \circ \mathit{tuple}(g \circ \mathit{tuple}(h \circ g_1^m \circ g_1, H^{m-1}(g_2) \circ g_1), g_2) \\
& = & g \circ \mathit{tuple}(h \circ g_1^{m+1}, g \circ \mathit{tuple}(H^{m-1}(g_2) \circ g_1, g_2)) \\
& = & g \circ \mathit{tuple}(h \circ g_1^{m+1}, H^m(g_2)).
\end{array}$$

Next we define $H' : (A \times B \to A \times B) \to (A \times B \to A \times B)$ by

$$H'(h') = h' \circ \mathit{tuple}(g_1 \circ \mathit{fst}, g \circ \mathit{tuple}(g_2 \circ \mathit{fst}, \mathit{snd}))$$

and prove that

$$H'^n = h' \circ \mathit{tuple}(g_1^n \circ \mathit{fst}, g \circ \mathit{tuple}(H^{n-1}(g_2) \circ \mathit{fst}, \mathit{snd})) \qquad (2)$$

for $n > O$. The proof is by induction on n. The base case $n = 1$ is trivial so consider the induction step $n = m + 1$. We calculate

$$
\begin{aligned}
H'^{m+1}(h') &= H'^{m}(h') \circ tuple(g_1 \circ fst, g \circ tuple(g_2 \circ fst, snd)) \\
&= h' \circ tuple(g_1^m \circ fst, g \circ tuple(H^{m-1}(g_2) \circ fst, snd)) \\
&\quad \circ tuple(g_1 \circ fst, g \circ tuple(g_2 \circ fst, snd)) \\
&= h' \circ tuple(g_1^m \circ g_1 \circ fst, g \circ tuple(H^{m-1}(g_2) \circ g_1 \circ fst, g \circ tuple(g_2 \circ fst, snd))) \\
&= h' \circ tuple(g_1^{m+1} \circ fst, g \circ tuple(g \circ tuple(H^{m-1}(g_2) \circ g_1 \circ fst, g_2 \circ fst), snd)) \\
&= h' \circ tuple(g_1^{m+1} \circ fst, g \circ tuple(g \circ tuple(H^{m-1}(g_2) \circ g_1, g_2) \circ fst, snd)) \\
&= h' \circ tuple(g_1^{m+1} \circ fst, g \circ tuple(H^{m}(g_2) \circ fst, snd)).
\end{aligned}
$$

Given $h : A \to B$ define $h : A \times B \to A \times B$ by

$$\hat{h}(a, b) = (g(h(a), b), b_0)$$

where b_0 is the rigth identity for g. We shall then show that

$$(H^n(h)(a), b_0) = H'^n(\hat{h})(a, b_0) \qquad (3)$$

for all $a \in A$ and for $n > 0$. The base case $n = 0$ is trivial and when $n > 0$ we use (1) and (2) to calculate

$$
\begin{aligned}
H'^n(\hat{h})(a, b_0) &= \hat{h}(g_1^n(a), g(H^{n-1}(g_2)(a), b_0)) \\
&= \hat{h}(g_1^n(a), H^{n-1}(g_2)(a)) \\
&= (g(h(g_1^n(a), H^{n-1}(g_2)(a)), b_0) \\
&= (H^n(h)(a), b_0).
\end{aligned}
$$

To prove that H is k-bounded it is sufficient to prove for all $h \in A \to B$ that

$$H^k \; h \sqsubseteq \bigsqcup\{H^n \; h \mid 0 \leq n < k$$

and for this it suffices to prove for all $a \in A$ that

$$(H^k \; h \; a, b_0) \sqsubseteq \bigsqcup\{H^n \; h \; a, b_0 \mid 0 \leq n < k\}.$$

Using (3) this may be refomulated to

$$H'^k \; \hat{h} \; (a, b_0) \sqsubseteq \sqcup\{H'^n \; \hat{h} \; (a, b_0) \mid 0 \leq n < k\}.$$

But this follows because the assumptions and Corollary 3.3 show that H' is k-bounded.

C Proofs from Section 4

Proof of Lemma 4.3:

It is convenient to abbreviate:

$$G = \lambda h. \bigsqcup \circ \mathcal{P}(h \circ g_1) \circ g_0$$

To see that G is additive we calculate:

$$\begin{array}{rcl} G(h_1 \sqcup h_2) & = & \bigsqcup \circ \mathcal{P}((h_1 \sqcup h_2) \circ g_1) \circ g_0 \\ & = & \bigsqcup \circ \mathcal{P}(h_1 \circ g_1 \sqcup h_2) \circ g_1) \circ g_0 \\ & = & ((\bigsqcup \circ \mathcal{P}(h_1 \circ g_1)) \sqcup (\bigsqcup \circ \mathcal{P}(h_2 \circ g_1))) \circ g_0 \\ & = & (\bigsqcup \circ \mathcal{P}(h_1 \circ g_1) \circ g_0) \sqcup (\bigsqcup \circ \mathcal{P}(h_2 \circ g_1) \circ g_0 \\ & = & G\ h_1 \sqcup G\ h_2 \end{array}$$

where we have used Fact 4.1.

Next we prove that

$$G^i h = \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \{\!|\,|\!\} \qquad (4)$$

for $i \geq 0$. The proof is by induction on i. For $i = O$ we have

$$\bigsqcup \circ \mathcal{P}(h) \circ \{\!|\,|\!\} = \bigsqcup \circ \{\!|\,|\!\} \circ h = h$$

where we have used Fact 4.1. This proves the base case. For the induction step we calculate

$$
\begin{aligned}
G^{i+1}h &= G(\bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \mathcal{P}(g_0))^i \circ \{\!|\,|\!\}) \\
&= \bigsqcup \circ \mathcal{P}(\bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \{\!|\,|\!\} \circ g_1) g_0 \\
&= \bigsqcup \circ \mathcal{P}(\bigsqcup) \circ \mathcal{P}(\mathcal{P}(h)) \circ (\mathcal{P}(\mathcal{P}(g_1)) \circ \mathcal{P} \bigcup \circ \mathcal{P}(\mathcal{P}(g_0)))^i \circ \mathcal{P}\{\!|\,|\!\} \circ \mathcal{P}(g_1) \circ g_0 \\
&= \bigsqcup \circ \bigcup \circ \mathcal{P}(\mathcal{P}(h)) \circ (\mathcal{P}(\mathcal{P}(g_1)) \circ \mathcal{P} \bigcup \circ \mathcal{P}(\mathcal{P}(g_0)))^i \circ \mathcal{P}\{\!|\,|\!\} \circ \mathcal{P}(g_1) \circ g_0 \\
&= \bigsqcup \circ \mathcal{P}(h) \circ \bigcup \circ (\mathcal{P}(\mathcal{P}(g_1)) \circ \mathcal{P} \bigcup \circ \mathcal{P}(\mathcal{P}(g_0)))^i \circ \mathcal{P}\{\!|\,|\!\} \circ \mathcal{P}(g_1) \circ g_0 \\
&= \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \bigcup \circ \mathcal{P}\{||\} \circ \mathcal{P}(g_1) \circ g_0 \\
&= \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \bigcup \circ \mathcal{P}(\mathcal{P}(g_1)) \circ \{\!|\,|\!\} \circ g_0 \\
&= \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0) \circ \{\!|\,|\!\} \\
&= \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \circ \mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0) \circ \{\!|\,|\!\} \\
&= \bigsqcup \circ \mathcal{P}(h) \circ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^{i+1} \circ \{\!|\,|\!\}
\end{aligned}
$$

using Fact 4.1.

To prove that G is k-bounded for $k = len_{sa}(\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))$ we have to show that

$$
G^k h \sqsubseteq \bigsqcup \{ G^i h \mid 0 \leq i \leq k
$$

.

From the definition of len_{sa} we have that

$$
(\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^k Y \sqsubseteq \bigcup \{ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i Y \mid 0 \leq i < k \}
$$

for all $Y \in \mathcal{P}(A)$. Thus for $a \in A$ we have

$$
\begin{aligned}
G^k ha &= \bigsqcup (\mathcal{P}(h)((\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^k \{a\})) \\
&\sqsubseteq \bigsqcup (\mathcal{P}(h)(\bigcup \{ (\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \{a\} \mid 0 \leq i < k \}) \\
&= \bigsqcup (\bigcup \{ (\mathcal{P}(h)((\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \{a\}) \mid 0 \leq i < k \}) \\
&= \bigsqcup (\bigsqcup (\mathcal{P}(h)((\mathcal{P}(g_1) \circ \bigcup \circ \mathcal{P}(g_0))^i \{a\})) \mid 0 \leq i < k \} \\
&= \bigsqcup \{ G^i ha \mid 0 \leq i < k \}
\end{aligned}
$$

Here we have used that $\mathcal{P}(h)$ is additive for all h.