
ELSEVIER Science of Computer Programming 22 (1994) 283–306

Science of Computer Programming

Approximate fixed points in abstract interpretation

Chris Hankin*, Sebastian Hunt

Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, UK

Revised December 1993

Abstract

Much of the earlier development of abstract interpretation, and its application to imperative programming languages, has concerned techniques for finding fixed points in large (often infinite) lattices. The standard approach in the abstract interpretation of functional languages has been to work with small, finite lattices and this supposedly circumvents the need for such techniques. However, practical experience has shown that, in the presence of higher-order functions, the lattices soon become too large (although still finite) for the fixed point finding problem to be tractable. This paper develops some approximation techniques which were first proposed by Hunt and shows how these techniques relate to the earlier use of widening and narrowing operations by the Cousots.

1. Introduction

Abstract interpretation is a semantics-based technique for the static analysis of programs. The results of such an analysis may be used as part of program verification, as a basis for program transformation or for automated debugging. In essence, an abstract interpretation is a non-standard semantics for the language, in which the denotation of the program is (efficiently) computable at compile-time, together with a correctness relation which relates the results of the abstract interpretation to some base semantics. Correctness (which we will not discuss further) is normally expressed in terms of a *safety* requirement; most interesting questions about program behaviour are undecidable and so the best we can hope is that the results of our analysis will include the actual program behaviour (rather than exactly predict it). Throughout this paper, our primary concern will be functional languages.

For the sake of concreteness we will consider a simply typed functional language; however our techniques are more widely applicable and the restriction to

* Corresponding author.

SSDI 0167-6423(94)00007-2

monomorphic types is not as severe as might at first appear [1]. The language of types is:

$$\tau ::= int \mid bool \mid List(\tau) \mid \tau \to \tau \mid \tau \times \tau$$

where A is a ground type, $List(\tau)$ represents the domain of finite and infinite lists with elements of type τ, $\to$ is the function type constructor and $\times$ is a product constructor. The language of expressions is:

$$e ::= x_\sigma \mid c_\sigma \mid e_1 e_2 \mid (e_1, e_2) \mid \lambda(x_{\sigma 1}, \ldots, x_{\sigma n}).e$$

where c_σ is a constant of type σ; we will leave the set of constants at each type unspecified and we will introduce specific constants as we need them in the text.

An *interpretation* is a triple $(\{D_A\}, \{c_\sigma\}, \{List, \times\})$ which specifies a domain for each base type, the interpretation for each of the constants and an interpretation for the list and product type constructors. We will assume that the $\to$ constructor is always interpreted as the (continuous) function space constructor; thus the interpretation of types involving $\to$ is:

$$D_{\sigma \to \tau} = [D_\sigma \to D_\tau] \quad \text{(the cpo of continuous functions).}$$

For example, the *standard* interpretation for our simple functional language might specify:

$$D_{int} = \mathbb{N}_\perp,$$

$$D_{bool} = \{tt, ff\}_\perp,$$

$$\ldots$$

$$\mathbf{if}_{bool \to \sigma \to \sigma \to \sigma} \perp x\, y = \perp_\sigma,$$

$$\mathbf{if}_{bool \to \sigma \to \sigma \to \sigma} tt\, x\, y = x,$$

$$\mathbf{if}_{bool \to \sigma \to \sigma \to \sigma} ff\, x\, y = y,$$

$$\ldots$$

$$\mathbf{fix}_{(\sigma \to \sigma) \to \sigma} f = \bigsqcup f^i(\perp),$$

$$D_{List(\tau)} = \mathbf{1} + (D_\tau \times D_{List(\tau)}) \quad \text{where } \mathbf{1} \text{ is a one element set, } + \text{ is separated sum and } \times \text{ is product,}$$

$$D_{\tau 1 \times \tau 2} = D_{\tau 1} \times D_{\tau 2} \quad \text{(cartesian product).}$$

Given an interpretation, this induces the valuation function for expressions (with respect to some environment, ρ).

$$[\![x_\sigma]\!]\rho = \rho(x),$$

$$[\![c_\sigma]\!]\rho = c_\sigma,$$

$$[\![e_1 e_2]\!]\rho = [\![e_1]\!]\rho\, [\![e_2]\!]\rho,$$

$$[\![(e_1, e_2)]\!]\rho = ([\![e_1]\!]\rho, [\![e_2]\!]\rho),$$

$$[\![\lambda(x_{\sigma 1}, \ldots, x_{\sigma n}).e]\!]\rho = \lambda(\zeta_1 \in D_{\sigma 1}, \ldots, \zeta_n \in D_{\sigma n}).[\![e]\!]\rho[x_i \mapsto \zeta_i \mid 1 \leqslant i \leqslant n].$$

An *abstract* interpretation is an interpretation, in the above sense, which captures some program property; it has been usual practice within the functional programming community to restrict abstract interpretations so that the interpretations of types are all finite lattices—this is not a restriction in the classical framework [4] and we will return to this point later. Notice that in the finite case, continuity is equivalent to monotonicity and thus $\rightarrow$ constructs the lattice of monotonic functions of appropriate type.

An example of abstract interpretation is *strictness analysis*. A unary function is strict if, when applied to an undefined argument, it returns an undefined result:

$$f\bot = \bot$$

the generalisation to n-ary functions is straightforward. We follow [2] and [12] in specifying the following abstract interpretation for strictness analysis:

$D_{int} = D_{bool} = \mathbf{2}$ (the two point domain $\{0,1\}$ with $0 \sqsubseteq 1$),

...

$op_{A \times A \rightarrow A}\, x\, y = x \sqcap y$ for any strict binary operator on base types,

$\mathbf{if}_{bool \rightarrow \sigma \rightarrow \sigma \rightarrow \sigma}\, 0\, x\, y = 0$,

$\mathbf{if}_{bool \rightarrow \sigma \rightarrow \sigma \rightarrow \sigma}\, 1\, x\, y = x \sqcup y$,

...

$\mathbf{fix}_{(\sigma \rightarrow \sigma) \rightarrow \sigma} f = \bigsqcup f^i(\bot)$,

$D_{List(\tau)} = ((D_\tau)_\bot)_\bot$,

$D_{\tau 1 \times \tau 2} = D_{\tau 1} \times D_{\tau 2}$ (cartesian product).

The intuition behind the interpretation of base types is that 0 represents undefined and 1 represents any value. The interpretation of lists is due to [12]. Lists of base type elements are abstracted by the four point domain, **4**:

$\top_\in$

|

$\bot_\in$

|

∞

|

$\bot$

The intuition behind these four values is that $\bot$ represents the undefined list; ∞ represents the undefined list, all of the infinite lists and any list which ends in $\bot$ (i.e.

the finite approximations of infinite lists); $\bot_\epsilon$ represents the same as ∞ plus the finite lists with at least one $\bot$ element; and $\top_\epsilon$ represents all lists.

Our running example throughout this paper is the function

$$append: List(int) \times List(int) \to List(int)$$

with definition:

$$append(nil, y) = y,$$

$$append(h{:}t, y) = h{:}append(t, y)$$

where : is an infix *cons* operation. Note that the strictness analysis interpretation of *append* will be an element of the lattice $\mathbf{4} \times \mathbf{4} \to \mathbf{4}$.

More formally, *append* is represented by the expression

fix *Append*

where *Append* is the expression:

$$\lambda f.\lambda(x, y).\mathbf{case}\ y(\lambda(h, t).h{:}f(t, y))x$$

and **case** has the standard interpretation:

$$\mathbf{case}\ s\, f \bot = \bot$$

$$\mathbf{case}\ s\, f\, nil = s$$

$$\mathbf{case}\ s\, f(a{:}x) = f(a, x)$$

see [12] for the benefits of treating pattern matching in this way. In the strictness analysis we use the following interpretations for : and **case**:

$$0{:}\bot = \infty, \quad 1{:}\bot = \infty,$$

$$0{:}\infty = \infty, \quad 1{:}\infty = \infty,$$

$$0{:}\bot_\epsilon = \bot_\epsilon, \quad 1{:}\bot_\epsilon = \bot_\epsilon,$$

$$0{:}\top_\epsilon = \bot_\epsilon, \quad 1{:}\top_\epsilon = \top_\epsilon,$$

$$\mathbf{case}\ s\, f \bot = \bot,$$

$$\mathbf{case}\ s\, f \infty = f(1, \infty),$$

$$\mathbf{case}\ s\, f \bot_\epsilon = f(0, \top_\epsilon) \sqcup f(1, \bot_\epsilon),$$

$$\mathbf{case}\ s\, f \top_\epsilon = s \sqcup f(1, \mathbf{T}_\epsilon).$$

But what is the strictness analysis denotation of *append*?

Any account of abstract interpretation of functional languages must address the problem of defining a suitable abstraction of functional values. There are a number of alternatives, ranging from the relational approach espoused in the Cousots' work and instantiated in the minimal function graph approach of [9] to the approach of [2] and

Table 1
The abstract *append* function

x \ y	$\bot$	∞	$\bot_\epsilon$	$\top_\epsilon$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
∞	∞	∞	∞	∞
$\bot_\epsilon$	∞	∞	$\bot_\epsilon$	$\bot_\epsilon$
$\top_\epsilon$	∞	∞	$\bot_\epsilon$	$\top_\epsilon$

our current approach where functions are abstracted by functions. In such a setting it is well known that the problem of finding fixed points is of n-iterated exponential complexity [11]. There was a naive expectation that the development of clever algorithms such as the frontiers algorithm [3, 8, 10] would ameliorate this situation but practical experience has shown that this was misplaced optimism.

The computation of a fixed point, **fix** f, requires the construction of the Kleene sequence $\{f^i(\bot) \mid i \geqslant 0\}$ which terminates when two successive iterates are equal. The reader may confirm that the sequence for the strictness analysis of *append* converges after three iterations ($f^2(\bot) = f^3(\bot)$) with the result given in Table 1.

The test for convergence involves ensuring that two successive iterates are pointwise equal; for this example the test therefore involves up to 16 equality tests, the whole process involving a maximum of 48 tests.

In practice, we have found that the problem of computing fixed points for many higher-order functions involving list-type arguments is intractable. In [7] and [8], we developed a formal approach to allow the evaluation of approximate fixed points, in fact generating upper and lower bounds for the true fixed point. In the classical approach to abstract interpretation pioneered by Patrick and Radhia Cousot it is common to work with lattices that do not satisfy the ascending chain condition, i.e. the lattices may contain infinitely ascending chains, and, in this context, it is essential to work with approximate fixed points; they have developed a general theory of *widening* and *narrowing* operations to support this work. Their approach relies on the observation that the Kleene sequence can be expressed as a recurrence:

$$x_0 = \bot$$

$$x_i = \begin{cases} x_{i-1} \sqcup f(x_{i-1}), & \text{if } x_{i-1} \sqsubseteq f(x_{i-1}), \\ x_{i-1}, & \text{otherwise} \end{cases}$$

and that the iteration can be speeded up by replacing $\sqcup$ by a coarser operation ∇ which will force convergence to a safe approximation of the fixed point; this approximation can then be refined using a decreasing iteration involving a narrowing operation Δ. It is possible to relate our approach to the widening/narrowing approach of the Cousots and this is our programme in this paper.

In the next section, we review the main results from [8]. Section 3 develops the theory somewhat further and presents a scheme for using the approach in fixed point computation. Section 4 defines widening and narrowing operations and demonstrates the correspondence between the two approaches. Section 5 presents a modified scheme for finding approximate fixed points and we conclude in Section 6.

2. The abstraction ordering

In this section we introduce a family of finite lattices which has proved useful in formulating a range of different analyses. Members of the family are related by an ordering, $\leqslant$; our aim is to provide a formal way of moving between lattices related by $\leqslant$. Eventually we will use this mechanism as a basis for our technique of evaluating approximate fixed points; the approximations will be true fixed points in some smaller lattice.

We work with the least family of finite lattices $\mathscr{L}$ such that:

$$\mathbf{2} \in \mathscr{L}$$

$$D_\perp \in \mathscr{L} \quad \text{if } D \in \mathscr{L}$$

$$D^\top \in \mathscr{L} \quad \text{if } D \in \mathscr{L}$$

$$D_1 \times D_2 \in \mathscr{L} \quad \text{if } D_1, D_2 \in \mathscr{L}$$

$$[D \to D'] \in \mathscr{L} \quad \text{if } D, D' \in \mathscr{L}$$

where $[D \to D']$ is the lattice of monotonic functions from D to D'. Such a family of finite lattices has proved to be useful in a number of analyses, including strictness analysis (as demonstrated above), parallel sharing analysis, and binding time analysis.

The abstraction ordering on $\mathscr{L}$ is defined by:

$$\mathbf{2} \leqslant D \quad \text{for all } D \in \mathscr{L}$$

$$D_\perp \leqslant D'_\perp \quad \text{if } D \leqslant D'$$

$$D^\top \leqslant D'^\top \quad \text{if } D \leqslant D'$$

$$D_1 \times D_2 \leqslant D'_1 \times D'_2 \quad \text{if } D_1 \leqslant D'_1 \text{ and } D_2 \leqslant D'_2$$

$$[A \to B] \leqslant [A' \to B'] \quad \text{if } A \leqslant A' \text{ and } B \leqslant B'$$

Note that $\leqslant$ does not capture the usual notion of approximation which has $\to$ contravariant in its first argument. We will see that $\leqslant$ means that there is a Galois connection between the two lattices—this amounts to the standard domain theoretic practice of using embedding–projection pairs to avoid the contravariance of $\to$. Notice that $\leqslant$ is a partial order. With respect to the type of the *append* function, we have for example:

$$\mathbf{2} \times \mathbf{4} \to \mathbf{4} \leqslant \mathbf{4} \times \mathbf{4} \to \mathbf{4}$$

and

$$\mathbf{2} \to \mathbf{4} \leqslant \mathbf{2} \times \mathbf{4} \to \mathbf{4}$$

The mechanism for moving between related lattices can be formulated most elegantly by employing some elementary category theory. We introduce the categories **FL**, $\mathbf{FL}^{ec}$ and $\mathbf{FL}^{ep}$.

- **FL**: finite lattices, monotone maps,
- $\mathbf{FL}^{ec}$: finite lattices, embedding–closure pairs,
- $\mathbf{FL}^{ep}$: finite lattices, embedding–projection pairs.

An embedding–closure pair is a pair of continuous functions $(e: A \to B, c: B \to A)$ such that:

$$e \circ c \geqslant id \quad \text{and} \quad c \circ e = id$$

and an embedding–projection pair is a pair of continuous functions $(e: A \to B$, $p: B \to A)$ such that:

$$e \circ p \leqslant id \quad \text{and} \quad p \circ e = id$$

We write the left and right components of an embedding–closure (embedding–projection) pair ϕ as ϕ^e and ϕ^c (ϕ^e and ϕ^p). Given a embedding–closure pair, ϕ, ϕ^e and ϕ^c determine each other uniquely. Similarly for embedding–projection pairs. Thus $\mathbf{FL}^{ec}$ and $\mathbf{FL}^{ep}$ may both be viewed as sub-categories of **FL**.

We introduce the functors $\__\perp$, $\_^\top$, $\_ \times \_$ and $\_ \to \_$ on $\mathbf{FL}^{ec}$ and $\mathbf{FL}^{ep}$. The definitions of these functors are the expected ones, in particular the functors on $\mathbf{FL}^{ec}$ are given in Table 2. (Note that *lift* (*colift*) is the usual injection function into a lifted

Table 2
The functors on $\mathbf{FL}^{ec}$

Functor	Objects	Morphisms
$\__\perp$	$A_\perp$	for $: A \to B$, $_\perp : A_\perp \to B_\perp$ is defined by: $_\perp\perp = \perp$ $\quad$ $_\perp(lift\ a) = lift(a)$ $_\perp\perp = \perp$ $\quad$ $_\perp(lift\ b) = lift(b)$
$\_^\top$	$A^\top$	dual to the above
$\_ \times \_$	AB	for $: A \to B$ and $: C \to D$, $: AC \to BD$ is defined by: $()(a, c) = (a, c)$ $()(b, d) = (b, d)$
$\_ \to \_$	$[A \to B]$	for $: A \to B$ and $: C \to D$, $\to : [A \to C] \to [B \to D]$ is defined by: $(\to)f = \circ f \circ$ $(\to)f = \circ g \circ$

(topped) domain.) The definitions of the functors on $\mathbf{FL}^{ep}$ are analogous to those on $\mathbf{FL}^{ec}$. It is straightforward to verify that these are indeed functors on the appropriate category ($\mathbf{FL}^{ep}$ and $\mathbf{FL}^{ec}$). We next define two pairs of maps between pairs of lattices related by the abstraction ordering. The safe maps give overestimates of values and the live maps give underestimates.

Definition 2.1. For each $A \leqslant B \in \mathscr{L}$, we define an $\mathbf{FL}^{ec}$-morphism $Safe_{A,B}: A \to B$. We write $UpS_{A,B}$ for $Safe^{e}_{A,B}$ and $DownS_{B,A}$ for $Safe^{c}_{A,B}$:

$$UpS_{\mathbf{2},B}\,a = \begin{cases} \bot_B, & \text{if } a = 0, \\ \top_B, & \text{if } a = 1, \end{cases}$$

$$DownS_{B,\mathbf{2}}\,b = \begin{cases} 0, & \text{if } b = \bot_B, \\ 1, & \text{otherwise}, \end{cases}$$

$$Safe_{A_\bot,B_\bot} = (Safe_{A,B})_\bot,$$

$$Safe_{A^\top,B^\top} = (Safe_{A,B})^\top,$$

$$Safe_{A1\times B1,\,A2\times B2} = Safe_{A1,A2} \times Safe_{B1,B2},$$

$$Safe_{[A1\to B1],[A2\to B2]} = Safe_{A1,A2} \to Safe_{B1,B2}.$$

We must verify that:

(i) $UpS_{\mathbf{2},B} \circ DownS_{B,\mathbf{2}} \geqslant id_B$,

(ii) $DownS_{B,\mathbf{2}} \circ UpS_{\mathbf{2},B} = id_{\mathbf{2}}$.

These verifications are routine.

Definition 2.2. For each $A \leqslant B \in \mathscr{L}$, we define an $\mathbf{FL}^{ep}$-morphism $Live_{A,B}: A \to B$. We write $UpL_{A,B}$ for $Live^{e}_{A,B}$ and $DownL_{B,A}$ for $Live^{p}_{A,B}$. The definitions of the *Live* maps are as for the *Safe* maps except for $DownL_{B,\mathbf{2}}$ which is:

$$DownL_{B,\mathbf{2}}\,b = \begin{cases} 1, & \text{if } b = \top_B, \\ 0, & \text{otherwise}. \end{cases}$$

Returning to our example, the strictness analysis interpretation of *append* belongs to the lattice $\mathbf{4} \times \mathbf{4} \to \mathbf{4}$ and so we may consider approximations to append in $\mathbf{2} \to \mathbf{4}$ and $\mathbf{2} \times \mathbf{4} \to \mathbf{4}$. Examples of the relevant *Safe* and *Live* maps are given by:

$$UpS_{\mathbf{2}\times\mathbf{4}\to\mathbf{4},\mathbf{4}\times\mathbf{4}\to\mathbf{4}} f = UpS_{\mathbf{4},\mathbf{4}} \circ f \circ (DownS_{\mathbf{4},\mathbf{2}} \times DownS_{\mathbf{4},\mathbf{4}}),$$

$$DownS_{\mathbf{4}\times\mathbf{4}\to\mathbf{4},\mathbf{2}\times\mathbf{4}\to\mathbf{4}} f = DownS_{\mathbf{4},\mathbf{4}} \circ f \circ (UpS_{\mathbf{2},\mathbf{4}} \times UpS_{\mathbf{4},\mathbf{4}}),$$

$$UpL_{\mathbf{2}\to\mathbf{4},\mathbf{4}\times\mathbf{4}\to\mathbf{4}} f = UpL_{\mathbf{4},\mathbf{4}} \circ f \circ (DownL_{\mathbf{4}\times\mathbf{4},\mathbf{2}}),$$

$$DownL_{\mathbf{4}\times\mathbf{4}\to\mathbf{4},\mathbf{2}\to\mathbf{4}} f = DownL_{\mathbf{4}\cdot\mathbf{4}} \circ f \circ (UpL_{\mathbf{2},\mathbf{4}\times\mathbf{4}}).$$

It is tempting to treat the *Live* maps as dual to the *Safe* maps, i.e.:

$$Live_{A,B} = Safe_{A^{OP},B^{OP}}.$$

Unfortunately, this is not possible since the family $\mathscr{L}$ is not closed under the operation of forming opposites; in particular, the opposite of **2** is $\{0,1\}$ with $1 \sqsubseteq 0$ which is not a member of $\mathscr{L}$. However, the family is closed under the operation of forming opposites *up to isomorphism* (for example, **2** is isomorphic to its opposite and thus, in some senses, self-dual). Formally, we can establish an order ($\leqslant$) isomorphism $O: \mathscr{L} \to \mathscr{L}$ such that for each $A \in \mathscr{L}$ we have:

$$O(A) \cong A^{OP}$$

the isomorphism being established by a (contravariant) map $R_A: A \to O(A)$. Details of O and R are given in the appendix. Using this approach, *Safe* and *Live* are dual up to isomorphism (see Fact A.2).

The following properties of the *Safe* and *Live* maps are standard for embedding–closure pairs and embedding–projection pairs [6]:

- *UpS* and *UpL* are injective,
- *DownS* and *DownL* are onto and strict,
- *UpS* is $\top$-preserving and *UpL* is strict.

In addition we have the following lemma and the dual result (which is proved in the appendix).

Lemma 2.3. *For all* $A \leqslant B \in \mathscr{L}$, $UpS_{A,B}$ *is strict.*

Proof. By induction on the height of the proof that $A \in \mathscr{L}$. □

Corollary 2.4. *For all* $A \leqslant B \in \mathscr{L}$, $UpL_{A,B}$ *is* $\top$*-preserving.*

In what follows we will assume "dual" results such as this corollary to be clear and will not spell out the details of their proof.

The following lemma establishes that *Safe* is a functor from the (poset) category $\mathscr{L}$ to $\mathbf{FL}^{ec}$.

Lemma 2.5. *For all* $A \leqslant B \leqslant C \in \mathscr{L}$,
(i) $Safe_{A,C} = Safe_{B,C} \circ Safe_{A,B}$,
(ii) $Safe_{A,A} = id_A$.

Proof. First we prove (i) by induction over the type of A.
$A = \mathbf{2}$:

$$UpS_{B,C}(UpS_{2,B}\ 0) = UpS_{B,C}\ \bot_B \quad \text{(by definition)}$$
$$= \bot_C \quad \text{(by Lemma 2.3),}$$

$$UpS_{B,C}(UpS_{2,B}\ 1) = UpS_{B,C}\ \top_B \quad \text{(by definition)}$$
$$= \top_D \quad \text{(since } UpS \text{ is top-preserving),}$$
$$DownS_{B,2}(DownS_{C,B}\ x) = 0 \Rightarrow DownS_{C,B}\ x = \bot_B$$
$$\Rightarrow x = \bot_C.$$

Both implications hold because *DownS* is bottom-reflecting which follows by a simple argument using the properties of embedding–projection pairs. In this case $DownS_{C,2}\ x = 0$ as well.

$$DownS_{B,2}(DownS_{C,B}\ x) = 1$$
$$\Rightarrow DownS_{C,B}\ x \neq \bot_B \quad \text{(by definition)}$$
$$\Rightarrow x \neq \bot_C \quad \text{(since } DownS \text{ is strict)}$$

and thus $DownS_{C,2}\ x = 1$.

The result follows by extensionality.

The inductive cases are all the same and follow from the functorial properties of the constructors that we use; we will illustrate two cases—the unary functor $\__\perp$ and the binary functor $\_ \times \_$:

$A = A'_\perp$: Then $B = B'_\perp$ and $C = C'_\perp$ and then:

$$Safe_{A'_\perp, C'_\perp} = (Safe_{A',C'})_\perp$$
$$= (Safe_{B',C'} \circ Safe_{A',B'})_\perp \quad \text{(by induction hypothesis (IH))}$$
$$= (Safe_{B',C'})_\perp \circ (Safe_{A',B'})_\perp \quad \text{(since } \__\perp \text{ is a functor)}$$
$$= Safe_{B',C'} \circ Safe_{A,B}.$$

$A \equiv A1 \times A2$: Then $B \equiv B1 \times B2$ and $C \equiv C1 \times C2$ and:

$$Safe_{A,C} = Safe_{A1,C1} \times Safe_{A2,C2}$$
$$= (Safe_{B1,C1} \circ Safe_{A1,B1}) \times (Safe_{B2,C2} \circ Safe_{A2,B2}) \quad \text{(by IH)}$$
$$= (Safe_{B1,C1} \times Safe_{B2,C2}) \circ (Safe_{A1,B1} \times Safe_{A2,B2})$$
$$= Safe_{B,C} \circ Safe_{A,B}.$$

(ii) follows from $Safe_{2,2} = id_2$ since functors ($\_ \times \_$, $\_ \to \_$, etc.) preserve identities.

By the "dual" of the lemma, *Live* is a functor from $\mathscr{L}$ to $\mathbf{FL}^{ep}$. □

The main result concerning these maps and their interaction with the least fixed point operator, **fix**, in [8] is the following.

Fact 2.6. *For all lattices* $D, D' \in \mathscr{L}$ *such that* $D' \leqslant D$:

$$UpS\ \mathbf{fix}_{D'} \sqsupseteq \mathbf{fix}_D$$

and

$$UpL\,\mathbf{fix}_{D'} \sqsubseteq \mathbf{fix}_{D}.$$

Fact 2.6 gives a formal basis for the method of finding upper and lower bounds for a true fixed point by iterating in a smaller lattice using safe and live approximations, respectively. In this paper we develop this technique and relate it to Cousot's widening and narrowing operations.

We close this section by returning to the *append* example, we will consider the evaluation of the fixed point in $\mathbf{2}\to\mathbf{4}$. We first consider the safe approximation $\mathbf{fix}\,F_0$ where

$$F_0 = DownS_{((4\times 4)\to 4)\to((4\times 4)\to 4),(2\to 4)\to(2\to 4)}\,F$$

and F is the abstract interpretation of *Append*, i.e.:

$$F = \lambda f.\lambda(x,y).\,\mathbf{case}\,y(\lambda(h,t).h{:}f(t,y))\,x.$$

We simplify F_0 in the following way:

$$\begin{aligned}
&DownS_{((4\times 4)\to 4)\to((4\times 4)\to 4),(2\to 4)\to(2\to 4)}\,F\\
&\quad = DownS_{(4\times 4)\to 4,2\to 4}\circ F\circ UpS_{2\to 4,(4\times 4)\to 4} \quad \text{(by definition)}\\
&\quad = \lambda g.DownS_{(4\times 4)\to 4,2\to 4}(F(UpS_{2\to 4,(4\times 4)\to 4}\,g)) \quad \text{(by extensionality)}\\
&\quad = \lambda g.DownS_{4,4}\circ(F(UpS_{2\to 4,(4\times 4)\to 4}\,g))\circ UpS_{2,4\times 4}\\
&\quad = \lambda g.(F(UpS_{2\to 4,(4\times 4)\to 4}\,g))\circ UpS_{2,4\times 4} \quad \text{(by Lemma 2.5)}\\
&\quad = \lambda g.(\lambda f.(Ff)\circ UpS_{2,4\times 4})\,(UpS_{2\to 4,(4\times 4)\to 4}\,g)\\
&\quad = (\lambda f.(Ff)\circ UpS_{2,4\times 4})\circ UpS_{2\to 4,(4\times 4)\to 4}
\end{aligned}$$

and now we can rewrite $\mathbf{fix}\,F_0$ as:

$$\mathbf{fix}(\lambda f.((\lambda(x,y).\mathbf{case}\,y(\lambda(h,t).h{:}f(t,y))\,x)\circ UpS_{2,4\times 4})\circ UpS_{2\to 4,(4\times 4)\to 4}).$$

The reader should verify that this iteration converges after two iterations and that each test involves only 2 equality tests (thus a total of 4 tests—compare with the 48 tests needed earlier), leading to the fixed point, $append_0$:

$$append_0\,0 = \bot,$$

$$append_0\,1 = \top_{\in}.$$

Notice that

$$g = UpS_{2\to 4,4\times 4\to 4}\,append_0 = UpS_{4,4}\circ append_0\circ DownS_{4\times 4,2}$$

is defined by

$$g(\bot, \bot) = \bot$$

$$g(x, y) = \top_{\epsilon}, \quad \text{if } x \neq \bot \text{ or } y \neq \bot$$

which, as predicted by Fact 2.6 is an upper (safe) approximation of the least fixed point presented earlier. The live approximation in $\mathbf{2} \to \mathbf{4}$ also converges to $append_0$, but

$$h = UpL_{\mathbf{2}\to\mathbf{4},\mathbf{4}\times\mathbf{4}\to\mathbf{4}}\, append_0 = UpL_{\mathbf{4},\mathbf{4}} \circ append_0 \circ DownL_{\mathbf{4}\times\mathbf{4},\mathbf{2}}$$

is defined by

$$h(\top_{\epsilon}, \top_{\epsilon}) = \top_{\epsilon}$$

$$h(x, y) = \bot, \quad \text{if } x \neq \top_{\epsilon} \text{ or } y \neq \top_{\epsilon}.$$

3. Refining approximations

In this section we show how the *Safe* and *Live* maps may be used to obtain progressively more accurate approximations to the true fixed point of a function.

We start by noting that x is *pre-fixed point* of f if

$$x \sqsubseteq f(x)$$

and a *post-fixed point* of f if

$$f(x) \sqsubseteq x$$

We can now state and prove one of the main results of the paper which is a generalisation of the first part of Fact 2.6.

Proposition 3.1. *Let* $A' \leqslant A \in \mathscr{L}$, *and let*

$$D \equiv [A \to A], \qquad D' \equiv [A' \to A']$$

Then for all $f \in D$ *and for all* $x \in A'$:

$$(DownS_{D,D'} f)x = x \;\Rightarrow\; f(UpS_{A',A}\, x) \sqsubseteq UpS_{A',A}\, x$$

i.e. fixed points in smaller lattices, by the $\leqslant$ *ordering, become post-fixed points in the larger lattice.*

Proof.

$$\begin{aligned} UpS_{A',A}\, x &= UpS_{A',A}((DownS_{D,D'} f)x) \quad \text{(by assumption)} \\ &= (UpS_{A',A} \circ DownS_{A,A'} \circ f \circ UpS_{A',A})x \quad \text{(by definition)} \\ &\sqsupseteq f(UpS_{A',A}\, x) \quad \square \end{aligned}$$

Notice that the dual result does not provide a generalisation of the second part of Fact 2.6 since pre-fixed points are generally related to the greatest fixed point not the least fixed point (the greatest fixed point is the largest pre-fixed point). Thus a possible specialisation of the dual of Proposition 3.1 would be

$$UpL\,\mathbf{gfp}_{D'} \sqsubseteq \mathbf{gfp}_{D}$$

where **gfp** is the greatest fixed point operator; in general, greatest fixed points are less interesting for the purposes of abstract interpretation since they are too conservative.

The process that we use to find approximate fixed points involves a first step which finds coarse upper and lower bounds to the true fixed point by iterating in a small lattice followed by successive steps which refine the bounds by iterating in larger and larger lattices. The following corollary provides a formal basis for this technique:

Corollary 3.2. *For any* $[A \to A], [A' \to A'], [A'' \to A''] \in \mathscr{L}$, *with* $A'' \leqslant A' \leqslant A$ *then for all* $g \in [A \to A]$ *and any fixed point* x *of* $(DownS_{[A\to A],[A''\to A'']}g)$, *we have*:

$$(DownS_{[A\to A],[A'\to A']}g)\,(UpS_{A'',A'}x) \sqsubseteq UpS_{A'',A'}x.$$

Proof. Use Proposition 3.1 with $f = DownS_{[A\to A],[A'\to A']}g$ and use Lemma 2.5 □

This result says that when any fixed point of the approximation of g in a smaller lattice is embedded into any larger lattice it becomes a post-fixed point of the approximation of g in that larger lattice. The dual result for the live maps is that any fixed point becomes a pre-fixed point in the larger lattice. To illustrate this point, consider approximating *append* in $\mathbf{2} \times \mathbf{4} \to \mathbf{4}$. The relevant safe approximation of F is the function:

$$F_1 = \lambda f.((\lambda(x,y).\mathbf{case}\ y(\lambda(h,t).h : f(t,y))\,x) \circ UpS_{\mathbf{2}\times\mathbf{4},\mathbf{4}\times\mathbf{4}}) \circ UpS_{\mathbf{2}\times\mathbf{4}\to\mathbf{4},\mathbf{4}\times\mathbf{4}\to\mathbf{4}},$$

and the post-fixed point of F_1 obtained from the approximation $append_0$ is

$$t = UpS_{\mathbf{2}\to\mathbf{4},\mathbf{2}\times\mathbf{4}\to\mathbf{4}}append_0 = append_0 \circ DownS_{\mathbf{2}\times\mathbf{4},\mathbf{2}}.$$

Thus t is defined by:

$$t(0,\bot) = \bot,$$

$$t(x,y) = \top_{\epsilon}, \quad \text{if } x \neq 0 \text{ or } y \neq \bot.$$

Now, since

$$UpS_{\mathbf{2}\times\mathbf{4},\mathbf{4}\times\mathbf{4}}(0,y) = (\bot, y)$$

we have:

$$F_1(t)\,(0,y) = \bot,$$

$$F_1(t)\,(1,y) = \top_{\epsilon},$$

which is actually a fixed point of F_1.

Specifically considering least fixed points, we have the following proposition.

Proposition 3.3. *For any* $D, D', D'' \in \mathscr{L}$, *such that*:

$$D \equiv [A \to A],$$

$$D' \equiv [A' \to A'],$$

$$D'' \equiv [A'' \to A'']$$

with $A'' \leqslant A' \leqslant A$ *and for any* $f \in D$:

(i) $UpS_{A'',A'}(\mathbf{fix}(DownS_{D,D''}f)) \sqsupseteq \mathbf{fix}(DownS_{D,D'}f)$,

(ii) $UpL_{A'',A'}(\mathbf{fix}(DownL_{D,D''}f)) \sqsubseteq \mathbf{fix}(DownL_{D,D'}f)$.

Proof. These follow immediately from Fact 2.6 and Lemma 2.5(i). □

An illustration of this result is given by the fact that:

$$(\mathbf{fix}\, F_1)(0, y) = \bot,$$

$$(\mathbf{fix}\, F_1)(1, \bot) = (\mathbf{fix}\, F_1)(1, \infty) = \infty,$$

$$(\mathbf{fix}\, F_1)(1, \bot_\epsilon) = \bot_\epsilon,$$

$$(\mathbf{fix}\, F_1)(1, \top_\epsilon) = \top_\epsilon,$$

which should be contrasted with the function t above.

Finally, in this section, we restate some obvious properties of pre- and post-fixed points.

Fact 3.4. *For all* $f \in [A \to A]$ *and* $x \in A$:

(i) *If* x *is a pre-fixed point of* f *and* x *is less than* $\mathbf{fix}\, f$ *then* $\{f^n(x) \mid n \geqslant 0\}$ *is an ascending chain and for all* n:

$$f^n(x) \sqsubseteq \mathbf{fix}\, f.$$

Moreover, since we are working with finite lattices, the chain will eventually stabilise and the limit will be $\mathbf{fix}\, f$.

(ii) *If* x *is a post-fixed point of* f *then* $\{f^n(x) \mid n \geqslant 0\}$ *is a descending chain and for all* n:

$$f^n(x) \sqsupseteq \mathbf{fix}\, f.$$

However, notice that the limit of such a descending chain may not be $\mathbf{fix}\, f$ *but some other fixed point. (Thus the need for the generality of Corollary* 3.2.)

We can now present a scheme for finding approximate fixed points (to any desired accuracy):

Step 1: Choose some small lattice in which the problem of finding fixed points is tractable and iterate from bottom to find the least fixed point of both the safe and live

abstractions of the function. For the *append* example, the least fixed point of both the safe and live approximations in $\mathbf{2} \to \mathbf{4}$ is the function f in Section 2.

Step 2: Apply *UpS* to the safe approximation and apply *UpL* to the live approximation to give upper and lower bounds on the true fixed point in the full lattice. If these agree on the interesting arguments, or if a safe answer is sufficient, use the upper bound. In our example the upper and lower bounds only agree on $(\bot, \bot)$ and $(\top_\epsilon, \top_\epsilon)$ and the upper bound is rather too safe.

Step 3: Apply *UpS* to the safe approximation and *UpL* to the live approximation to move to an intermediate lattice and iterate down from the resultant post-fixed point and up from the resultant pre-fixed point; in our example, the first of these iterations converges to $F_1(t)$ and the second converges to a value which happens to be equal to **fix** F_1. Repeat Step 2.

Repetition of Step 2 for the example leads to upper and lower bounds which agree on the points $(\bot, \bot)$, $(\bot, \infty)$, $(\bot, \bot_\epsilon)$, $(\bot, \top_\epsilon)$ and $(\top_\epsilon, \top_\epsilon)$. From this we can infer that the *append* function is strict in its first argument (since the first four pairs are all mapped to $\bot$). Unfortunately if we want further information, the only improvement that we can get is by iterating upwards in the original lattice starting from the pre-fixed point generated from **fix** F_1. In more realistic examples, too large for inclusion in this paper, further refinement is possible.

One word of caution: if the post- (pre-)fixed point of the safe (live) image of the function in some intermediate lattice happens to be a fixed point then no further improvement of the upper (lower) bound is possible in that lattice. On the other hand, when the pre-fixed point is a fixed point, it must be the least fixed point.

4. Widening and narrowing operations

We now relate the foregoing to the Cousots' approach which is based on the use of widening and narrowing operations. We start by recalling some definitions and results from [5].

Definition 4.1. For any complete lattice L, an operation $\nabla \in \mathbb{N} \to ((L \times L) \to L)$ is a *widening operation* iff it satisfies the following conditions.

(i) $\forall j > 0, \forall x, y \in L, x \sqcup y \sqsubseteq x \nabla(j) y$.

(ii) For all ascending chains $x_0 \sqsubseteq x_1 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots$ in L, the chain $y_0 = x_0$, $y_1 = y_0 \nabla(1) x_1, \ldots, y_n = y_{n-1} \nabla(n) x_n$ is eventually stable; i.e. there exists a $k \geqslant 0$ such for all $i \geqslant k$, $y_i = y_k$.

A widening operator may be used to generate an "accelerated" fixed point iteration (which in general will overshoot the least fixed point) as shown by the following proposition.

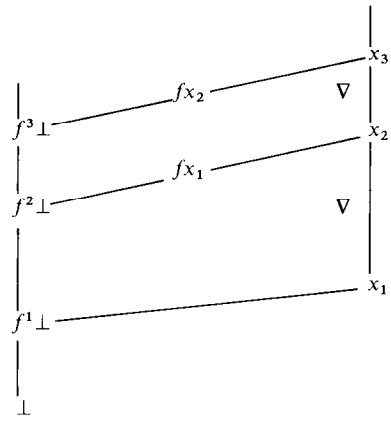

Fig. 1.

Proposition 4.2. *Let f be a monotone operator on L and ∇ a widening operator. The limit u of the sequence*

$$x_0 = \perp,$$

$$x_{n+1} = \begin{cases} x_n, & \text{if } f(x_n) \sqsubseteq x_n, \\ x_n \nabla (n+1) f(x_n), & \text{otherwise,} \end{cases}$$

can be computed in a finite number of steps. Moreover $\mathbf{fix}(f) \sqsubseteq u$ *and* $f(u) \sqsubseteq u$.

The iteration process described by the proposition and its relationship to the Ascending Kleene Chain is illustrated in Fig. 1.

Let $f: B \to B$ with $A \leqslant B \in \mathscr{L}$ and consider the sequence $(DownSf)^n \perp$. For the purposes of comparison with widening, we embed this sequence into B using UpS. It is easily shown that the resulting sequence $UpS((DownSf)^n \perp)$ is just $(UpS \circ DownS \circ f)^n \perp$, giving the modified diagram shown in Fig. 2. Consequently we have the following result.

Lemma 4.3. *For any lattices $D, D' \in \mathscr{L}$, such that $D' \leqslant D$,*

$$\lambda j.\lambda(x, y).x \sqcup UpS_{D',D}(DownS_{D,D'}\, y)$$

is a widening operation.

Proof. Observe that $UpS_{D',D}(DownS_{D,D'}\, y) \sqsupseteq y$ by the definition of embedding–closure pairs and thus,

$$x \sqcup UpS_{D',D}(DownS_{D,D'}\, y) \sqsupseteq x \sqcup y.$$

Any ascending chain must be eventually stable since the lattices are all finite. □

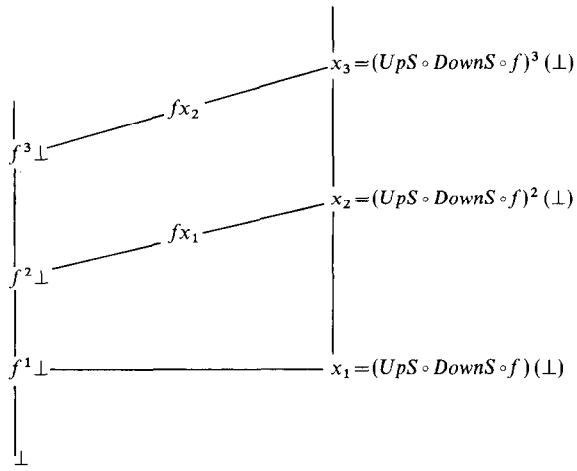

Fig. 2.

In our earlier discussion, we presented three steps for computing approximate fixed points. We have now shown that Step 1 of the process is an instance of the Cousot's notion of widening. However, it may be preferable to use the approach of the last section for efficiency reasons since the explicit use of the widening operator requires us to work in a larger lattice.

Still considering the safe maps, we now turn to the process of refining the approximation and start by defining the concept of narrowing.

Definition 4.4. For any complete lattice L, an operation $\Delta \in \mathbb{N} \to ((L \times L) \to L)$ is a *narrowing operation* iff it satisfies the following conditions:

(i) $\forall j > 0$, $(\forall x, y \in L: y \sqsubseteq x)$, $y \sqsubseteq x\Delta(j)y \sqsubseteq x$.

(ii) For all descending chains $x_0 \sqsupseteq x_1 \sqsupseteq \cdots \sqsupseteq x_n \sqsupseteq \cdots$ in L, the chain $y_0 = x_0$, $y_1 = y_0\Delta(1)x_1, \ldots, y_n = y_{n-1}\Delta(n)x_n$ is eventually stable; i.e. there exists a $k \geqslant 0$ such for all $i \geqslant k$, $y_i = y_k$.

Proposition 4.5. *Let f be a monotone operator on L and Δ a narrowing operator. Let $u \in L$ be such that $\mathbf{fix} f \sqsubseteq u$ and $f(u) \sqsubseteq u$. The decreasing chain*

$$x_0 = u$$

$$x_{n+1} = x_n\Delta(n+1)f(x_n)$$

is eventually stable. Moreover $\forall k \geqslant 0$, $\mathbf{fix}(f) \sqsubseteq x_k$.

Step 3 of the procedure outlined in the last section proposed the use of a decreasing iteration which we might reasonably expect to correspond to a narrowing. We now present a very general process, which corresponds to Step 3, in which each iterate may

be from a different intermediate lattice. We consider a sequence (with possible repetitions) of lattices $A_0, \ldots, A_n$ such that $A_i \leqslant A_{i+1}$ and $A_n \equiv A$, and we let $D_i \equiv [A_i \to A_i]$. We construct the sequence

$$z_0, z_1, \ldots$$

where

z_0 = the fixed point found in Step 1,

$z_{i+1} = (DownS_{D,Di+1} f)\,(UpS_{Ai,Ai+1}\, z_i)$.

The embedding of $\{z_n\}$ into A via the maps $UpS_{Ai,A}$ results in the decreasing sequence associated with the narrowing operation defined in the following lemma.

Lemma 4.6. *For any sequence of lattices $A_0, A_1, \ldots, A_n = A \in \mathscr{L}$, such that $A_i \leqslant A_{i+1}$ (the A_i need not be distinct), $D_i \equiv [A_i \to A_i]$, $D \equiv D_n$:*

$$\Delta \equiv \lambda j.\lambda(x, y).x \sqcap UpS_{Aj,A}(DownS_{A,Aj}\, y)$$

is a narrowing operation.

Proof.

$$x\Delta(j)\,(u)\,y = x \sqcap UpS_{Aj,A}(DownS_{A,Aj}\, y)$$

$$\sqsubseteq x \quad \text{(by definition of } \sqcap\text{)}.$$

Since $UpS_{Aj,A}(DownS_{A,Aj}\, y) \sqsupseteq y$ (by the defining property of embedding–closure pairs), we also have for $y \sqsubseteq x$ that

$$x\Delta(j)\,(u)\,y \sqsupseteq y.$$

Eventual stability of the sequence follows from finiteness of the lattices. □

To summarise, the upwards iteration in the smaller lattice using a safe approximation of the function is a widening and the refinement of the upper bound by iterating downwards in intermediate lattices is an example of narrowing.

The situation with live maps is somewhat less straightforward. The live approximations approach the true fixed point from below; this is true both of the initial approximation and the successive refinements. This runs counter to the development of [5] and later work. In [4] alternative definitions of widening and narrowing operators are introduced but these do not correspond very closely to our application.

As a closing remark notice that Lemma 4.3 gives the basis for an alternative proof of the post-fixed point property proved in Proposition 3.1 since any widening operation has this property [4].

5. A modified scheme for finding approximate fixed points

A major shortcoming of the scheme described at the end of Section 3 is the fact that the fixed point found for the safe approximation in an intermediate lattice may be other than the least fixed point and so unnecessarily conservative. An illustration of this shortcoming was given by the *append* example, in which the fixed point found for the safe approximation in $\mathbf{2} \times \mathbf{4} \to \mathbf{4}$ was $F_1(t)$ rather than $\mathbf{fix}\, F_1$. In this case it is clear that the least fixed point would be considerably more informative than one found by the suggested scheme.

In this section we describe a simple modification to the original scheme which overcomes the above shortcoming. We need one additional result.

Proposition 5.1. *For all* $A' \leqslant A \in \mathscr{L}$:

$$DownL_{A,A'} \sqsubseteq DownS_{A,A'}.$$

Proof. Routine induction on the height of the proof that $A' \in \mathscr{L}$. □

Corollary 5.2. *Let* $A' \leqslant A \in \mathscr{L}$, *and let*

$$D \equiv [A \to A], \qquad D' \equiv [A' \to A'].$$

Then for all $f \in D$ *and for all* $x \in A'$:

(i) $\mathbf{fix}(DownL_{D,D'}f) \sqsubseteq \mathbf{fix}(DownS_{D,D'}f)$,

(ii) $\mathbf{fix}(DownL_{D,D'}f)$ *is a pre-fixed point of* $DownS_{D,D'}f$.

Proof. The first follows by monotonicity of **fix** and the second since

$$\begin{aligned}\mathbf{fix}(DownL_{D,D'}f) &= (DownL_{D,D'}f)\,(\mathbf{fix}(DownL_{D,D'}f)) \\ &\sqsubseteq (DownS_{D,D'}f)\,(\mathbf{fix}(DownL_{D,D'}f)).\end{aligned}$$ □

The import of this result is that, by Fact 3.4(i), it allows us to iterate *up* to the least fixed point of the safe approximation in an intermediate lattice starting from the least fixed point of the live approximation in that lattice. The modified scheme is thus as follows.

Step 1. As before.

Step 2. As before.

Step 3. Apply *UpL* to the live approximation to move to an intermediate lattice and iterate up to the new live approximation from the resultant pre-fixed point. Then iterate up from the new live approximation to the new safe approximation; in our example both new approximations are equal to $\mathbf{fix}\, F_1$. Repeat Step 2.

With the modified scheme, repetition of Step 2 for the example leads to the bounds given in Tables 3 and 4.

Table 3
A lower bound for *append*

x \ y	$\bot$	∞	$\bot_\epsilon$	$\top_\epsilon$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
∞	$\bot$	$\bot$	$\bot$	$\bot$
$\bot_\epsilon$	$\bot$	$\bot$	$\bot$	$\bot$
$\top_\epsilon$	∞	∞	$\bot_\epsilon$	$\top_\epsilon$

Table 4
An upper bound for *append*

x \ y	$\bot$	∞	$\bot_\epsilon$	$\top_\epsilon$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
∞	∞	∞	$\bot_\epsilon$	$\top_\epsilon$
$\bot_\epsilon$	∞	∞	$\bot_\epsilon$	$\top_\epsilon$
$\top_\epsilon$	∞	∞	$\bot_\epsilon$	$\top_\epsilon$

This upper bound is rather more informative than that found by the previous scheme: in addition to revealing that *append* is strict, it shows that if the result of *append* is to be a finite list (not infinite or ending in bottom), the second argument to *append* must also be finite, and similarly for total lists.

It should be noted that despite the safe approximation found by this scheme being more accurate than that found by the previous scheme, it is no more expensive to compute; in fact one less iteration is required because the starting point (the new live approximation) is already the least fixed point in this example.

To illustrate the behaviour of the above scheme on a more taxing example than *append*, we consider the function

$$foldr : (List(int) \to List(int) \to List(int)) \times List(List(int)) \times List(int) \to List(int)$$

with definition

$$foldr(f, nil, b) = b$$

$$foldr(f, l : ls, b) = f\, l(foldr(f, ls, b)).$$

This can be used, for example, to define the concatenate function for lists of lists of integers:

$$cat\, ls = foldr(append, ls, nil).$$

The strictness analysis interpretation of *foldr* is an element of the lattice

$$\mathbf{(4 \to 4 \to 4)} \times \mathbf{6} \times \mathbf{4 \to 4},$$

Table 5

Lattice	Iterations	Evals	Time
$(\mathbf{4}\to\mathbf{4}\to\mathbf{4})\times\mathbf{6}\times\mathbf{4}\to\mathbf{4}$	7	2854	01:16:17

Table 6

Lattice	Iterations *Live/Safe*	Evals *Live/Safe*	Time *Live/Safe*
$(\mathbf{2}\to\mathbf{2}\to\mathbf{2})\times\mathbf{4}\times\mathbf{2}\to\mathbf{2}$	3/1	31/4	00:04/00:01
$(\mathbf{2}\to\mathbf{3}\to\mathbf{3})\times\mathbf{4}\times\mathbf{3}\to\mathbf{3}$	5/1	54/4	00:15/00:02
$(\mathbf{2}\to\mathbf{4}\to\mathbf{4})\times\mathbf{4}\times\mathbf{4}\to\mathbf{4}$	6/1	112/5	01:55/00:12
$(\mathbf{3}\to\mathbf{4}\to\mathbf{4})\times\mathbf{5}\times\mathbf{4}\to\mathbf{4}$	4/1	148/6	03:51/00:32
$(\mathbf{4}\to\mathbf{4}\to\mathbf{4})\times\mathbf{6}\times\mathbf{4}\to\mathbf{4}$	4/–	163/–	06:20/–
Totals	27	527	13:29

where **6**, the six element chain, is the interpretation used for lists of lists of base type. It can be shown that the graph of the abstract interpretation of *foldr* contains nearly 600 000 elements; thus the need for approximation techniques. A prototype strictness analyser based on the use of frontiers [8] is able to find the fixed point of the functional associated with *foldr* but it is prohibitively expensive, see Table 5. ("Evals" is the number of times an application of the abstract *foldr* was evaluated.)

Starting with an approximation of *foldr* in the lattice $(\mathbf{2}\to\mathbf{2}\to\mathbf{2})\times\mathbf{4}\times\mathbf{2}\to\mathbf{2}$ and refining it in four steps through three intermediate lattices, the behaviour shown in Table 6 is observed. The entries for the safe approximation are omitted for the full lattice since in this case both the live and safe approximation are known to be equal to the true fixed point.

Two features of this behaviour are rather striking. Firstly, note that only one iteration was required for each safe approximation, revealing that in every case the safe and live approximations coincide. Experience with other examples suggests that this behaviour is quite common. Secondly, note that by pursuing the sequence of refinements all the way to the full lattice, the true fixed point was found in less than a fifth of the time taken to compute the fixed point directly in the full lattice. This indicates that, as suggested in [8], there may be some scope for using approximations to reduce the work involved in finding true fixed points by using early members of a refinement sequence (relatively cheap to compute) to 'bypass' a number of the more expensive fixed point iterations in the full lattice.

To assess the accuracy of the approximations we consider the information they give about the function *cat* defined above. The first approximation is sufficient only to reveal that *cat* is strict. The second approximation shows in addition that if the result of *cat* is to be finite then the argument must be a finite list with no undefined elements.

The third approximation gives no additional information. The fourth approximation shows that if the result of *cat* is to be finite then the argument must actually be a finite list with all elements also being finite lists. Finally, the true fixed point shows that for the result of *cat* to be total, the argument must be finite with all elements being total lists.

6. Conclusions

We have developed the work on approximate fixed points first reported in [7] and shown how it connects with the widening/narrowing approach used in traditional abstract interpretation. We have presented a scheme which computes arbitrarily precise upper and lower approximations of the true least fixed point of a function.

An alternative approach is based on the observation that often only a small part of the function graph is actually required. If a suitable superset of the subgraph (which avoids the plateaux problems described in [3]) can be identified then an accurate fixed point in the superset can be used. Since the set of needed elements of the graph may be many orders of magnitude smaller than the cardinality of the graph, this accurate fixed point can be computed very efficiently. These ideas, which are related to Jones and Mycroft's minimal function graphs [9] are currently being developed by a number of researchers and promise a substantial improvement over the techniques developed in this paper.

Acknowledgments

We are indebted to our colleagues on the Semantique project for their willingness to discuss this work and for their constructive criticism; specifically, the categorical approach used in Section 2 was suggested by John Hughes and has led to a considerable simplification of our work. The first author was partially funded by ESPRIT BRA 3124–Semantique and both authors were partially funded by ESPRIT BRA 3074–SemaGraph.

Appendix A. Duality of *Live* and *Safe* maps

The order isomorphism $O: \mathscr{L} \to \mathscr{L}$ is defined by:

$$O(\mathbf{2}) = \mathbf{2},$$

$$O(A_\perp) = (O(A))^\top,$$

$$O(A^\top) = (O(A))_\perp,$$

$$O(A \times B) = O(A) \times O(B),$$

$$O([A \to B]) = [O(A) \to O(B)].$$

Note that $O(O(A)) = A$. For each $A \in \mathscr{L}$, the isomorphism of A^{OP} and $O(A)$ is established via the (contravariant) map $R_A : A \to O(A)$:

$$R_2 0 = 1, \qquad R_2 1 = 0,$$

$$R_{A_\perp} \perp = \top, \qquad R_{A_\perp}(\mathit{lift}\, a) = \mathit{colift}(R_A a),$$

$$R_{A^\top} \top = \perp, \qquad R_{A^\top}(\mathit{colift}\, a) = \mathit{lift}(R_A a),$$

$$R_{A \times B}(a, b) = (R_A a, R_B b),$$

$$R_{[A \to B]} f = R_B \circ f \circ R_{O(A)}.$$

It is easily verified that:

$$R_{O(A)} \circ R_A = id_A.$$

A key property of the R maps is the following:

Lemma A.1. *For all $A, B \in \mathscr{L}$:*

$$R_B(f\, a) = (R_{[A \to B]} f)\,(R_A a).$$

Fact A.2. *For all $A \leqslant B \in \mathscr{L}$:*

(i) $UpL_{O(A),O(B)} = R_{[A \to B]}(UpS_{A,B})$,

(ii) $DownL_{O(B),O(A)} = R_{[B \to A]}(DownS_{B,A})$.

Corollary A.3. *For all $A \leqslant B \in \mathscr{L}$, $UpL_{A,B}$ is $\top$-preserving.*

Proof. *Let $A \leqslant B \in \mathscr{L}$.* Since $O : \mathscr{L} \to \mathscr{L}$ is an order isomorphism, we can write A as $O(A')$ and B as $O(B')$. Then

$$\begin{aligned}
&UpL_{O(A'),O(B')}\, \top_{O(A')} \\
&\quad = (R(UpS_{A',B'}))\, \top_{O(A')} \quad \text{(by Fact A.2)} \\
&\quad = (R(UpS_{A',B'}))\,(R \perp_{A'}) \quad \text{(since } R \text{ an isomorphism of } A'^{\mathrm{OP}} \text{ onto } O(A')) \\
&\quad = R(UpS_{A',B'} \perp_{A'}) \quad \text{(by Lemma A.1)} \\
&\quad = R(\perp_{B'}) \quad \text{(by Lemma 2.3)} \\
&\quad = \top_{O(B')} \quad \text{(since } R \text{ an isomorphism of } B'^{\mathrm{OP}} \text{ onto } O(B')). \qquad \square
\end{aligned}$$

References

[1] S. Abramsky and T.P. Jensen, A relational approach to strictness analysis for higher-order polymorphic functions, in: *Proceedings 18th ACM Symposium on Principles of Programming Languages*, Orlando, FL (1991).

[2] G.L. Burn, C.L. Hankin and S. Abramsky, Strictness analysis for higher-order functions, *Sci. Comput. Programming* **7** (1986) 249–278.

[3] C. Clack and S.L. Peyton Jones, Strictness analysis—a practical approach, in: J.-P. Jouannaud, ed., *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science **201** (Springer-Verlag, Berlin, 1985) 35–49.

[4] P. Cousot, Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treilli, analyse sémantique des programmes, Thèse d'Etat, Université de Grenoble, (1978).

[5] P. Cousot, Semantic foundations of program analysis, in: S.S. Muchnick and N.D. Jones, eds., *Program Flow Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1981) 308–342.

[6] G.K. Gierz, K.H. Hoffmann, K. Keimel, J.D. Lawson, M. Mislove and D.S. Scott, *A Compendium of Continuous Lattices* (Springer-Verlag, Berlin, 1980).

[7] S. Hunt, Frontiers and open sets in abstract interpretation, in: D. MacQueen, ed., *Functional Programming Languages and Computer Architecture* (ACM, New York, 1989) 1–11.

[8] S. Hunt and C.L. Hankin, Fixed points and frontiers: a new perspective, *J. Functional Programming* **1** (1) (1991) 91–120.

[9] N.D. Jones and A. Mycroft, Dataflow analysis of applicative programs using minimal function graphs, Privately circulated manuscript (1985).

[10] C.C. Martin and C.L. Hankin, Finding fixed points in finite lattices, in: G. Kahn, ed., *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science **274** (Springer-Verlag, Berlin, 1987) 426–445.

[11] A.R. Meyer, Complexity of program flow analysis for strictness: application of a fundamental theorem of denotational semantics, Private communication.

[12] P. Wadler, Strictness analysis on non-flat domains (by abstract interpretation over finite domains, in: S. Abramsky and C.L. Hankin, eds., *Abstract Interpretation of Declarative Languages* (Ellis Horwood, Chichester, England, 1987) 266–275.