Finding Fixpoints in Finite Function Spaces Using Neededness Analysis and Chaotic Iteration

Niels Jørgensen
Roskilde University

This paper (except for the proofs contained in Appendix B) will be presented at SAS '94, *1st International Static Analysis Symposium*, September 28–30, Namur, Belgium. The conference proceedings will be published by Springer Verlag in the series *Lecture Notes in Computer Science*.

Finding Fixpoints in Finite Function Spaces Using Neededness Analysis and Chaotic Iteration

Niels Jørgensen

Computer Science Dept., Roskilde University
P.O. Box 260, DK-4000 Roskilde, Denmark
nielsj@dat.ruc.dk

Abstract. A new and efficient algorithm for computing the least fixpoint of a functional on a finite function space is defined. The algorithm applies to the computation of the least fixpoint (global or local) induced by an arbitrary system of functional equations in a certain formalism. The algorithm employs a variant of Cousot and Cousot's chaotic iteration [2], and uses neededness (or dependency) information to guide the fixpoint iteration, as for instance in Kildall's early algorithm [11]. The neededness analysis is *dynamic* as in the algorithms proposed by Muthukumar and Hermenegildo [6] and Le Charlier *et al.* [9], with the main difference being that our neededness analysis is of a more "shallow" nature, and that our approach is more iterative and less recursive. The complexity result implies that the worst-case number of iterations per equation is independent of the total number of equations in the equation system, where an iteration corresponds to evaluating an equation once with respect to given values of the functional and primitive parameters.

1 Introduction

In dataflow analysis of imperative languages, the program properties of interest are frequently expressed in terms of an equation system

$$x_j = E_j(x_1, \ldots, x_m) \qquad (1 \leq j \leq m) \tag{1}$$

with the result of the analysis being the least solution $\Delta = \langle d_1, \ldots, d_m \rangle$ with respect to $\langle x_1, \ldots, x_m \rangle$. The least solution is equal to the least fixpoint of the function that maps $\langle x_1, \ldots, x_m \rangle$ to $\langle E_1, \ldots, E_m \rangle$. An iterative fixpoint algorithm computes a series of approximations $\ldots, \Delta^n, \Delta^{n+1}, \ldots$ that stabilises at the least fixpoint. In each step, Δ^{n+1} is obtained from Δ^n essentially by feeding Δ^n into the R.H.S. of, say equation j.

A neededness (or dependency) analysis relates variable x_j to those x_i that occur non-trivially in E_j, and provides the basis for the following optimisation of the fixpoint computation: The equation for x_j is only considered when there is a change in the approximating value for some variable x_i that x_j needs. The solution value d_j is typically a property associated with a node in the program's flow graph, and the j'th equation in principle relates the node-value to all the node-values. The optimisation exploits that the computation of d_j^{n+1} only needs the values d_i^n associated with the *successor or predecessor nodes* of node j. An early algorithm based on this idea was proposed by Kildall

[11]. A more general treatment of the idea can be found in O'Keefe [8]. For an overview of dataflow analyses that can be formalised as an equation system (1) see Marlowe and Ryder [12].

In this paper we investigate how a neededness analysis can be used to optimise the fixpoint computation when the fixpoint is a tuple of functions rather than primitive values. Thus we consider a *functional* equation system

$$f_j(x_1, \ldots, x_k) = E_j(f_1, \ldots, f_m, x_1, \ldots, x_k) \qquad (1 \leq j \leq m) \tag{2}$$

(with the x_i's being implicitly universally quantified, and all functions being fully applied). We seek the least fixpoint $\langle \phi_1, \ldots, \phi_m \rangle$ of the functional that maps $\langle f_1, \ldots, f_m \rangle$ to $\langle \lambda\langle x_1, \ldots, x_k \rangle.E_1, \ldots, \lambda\langle x_1, \ldots, x_k \rangle.E_m \rangle$. Many program analysis problems require the more expressive system (2) for their formalisation. This includes so-called polyvariant analyses of functional programs (for instance strictness analysis as in [13]). Also in this category are analyses of logic programs where, for each predicate in the program, the result is a function that maps descriptions of the predicate's input to descriptions of its output. (See [15] for a classification of logic programming analyses).

In practice, when faced with a fixpoint problem as given by (2), a common approach has been to exploit that it may suffice to compute a local fixpoint. This may be referred to as a minimal function graph type of approach, cf. Jones and Mycroft's semantic notion [7]. Cousot and Cousot already in [2] studied the properties of a class of "functional" chaotic iteration sequences that converge to a local fixpoint. In our opinion, the most promising approach is to combine the idea of restricting attention to a local fixpoint (with the aim of being able to *disregard* a large part of the domain) with a neededness analysis (with the aim of reducing the number of times that each element in the relevant sub-domain is *re*-evaluated). In the context of analysis of logic programs, algorithms along these lines have been proposed by Muthukumar and Hermenegildo [6] and Le Charlier, Musumbu, and Van Hentenryck [9], the latter algorithm having been shown to be optimal for certain classes of programs and perform very well in practice [10].

The algorithm presented in this paper originates from an algorithm described in [5] for analysis of Prolog programs. The algorithm is similar to the one proposed in [9] in the sense that they both rely on a *dynamic* neededness analysis; its worst-case complexity bound improves by two orders of magnitude over [9]. The main difference is that our neededness analysis is of a more "shallow" nature; our algorithm only cares about direct needs, whereas [9] in a sense takes direct as well as indirect needs into account, via performing a transitive closure. We generalise the Prolog-algorithm of [5] so that it applies to the general fixpoint problem induced by (2), and discuss modifications for computing a local fixpoint. Formal and simple proofs of the algorithm's correctness and of the bounds on the number of iterations are provided. An iteration involves the computation of $E_j(f_1, \ldots, f_m, x_1, \ldots, x_k)$ for given j and parameters $\langle f_1, \ldots, f_m \rangle$ and $\langle x_1, \ldots, x_k \rangle$. We show that the number of iterations *per equation* is independent of the total number m of equations.

Dynamic neededness information. As an informal example of the relevance of recording dynamic neededness information during the fixpoint computation, consider a simple groundness analysis of the Prolog program for list reversal in Figure 1.

The groundness analysis proves that if the first argument to predicate rev is ground at call, then upon success the second argument is also ground, as one would expect. On the other hand, the analysis cannot infer that groundness of the second argument implies groundness of the first, so it is a weak analysis. The groundness analysis is described in more detail in Appendix A and formalised in [4]. The fixpoint is a function $\Phi = \langle rev^{\#}, app^{\#} \rangle$ where $rev^{\#} : \{\mathsf{G}, \mathsf{A}\}^2 \to \{\mathsf{G}, \mathsf{A}\}^2$ $app^{\#} : \{\mathsf{G}, \mathsf{A}\}^3 \to \{\mathsf{G}, \mathsf{A}\}^3$ and the modes in $\{\mathsf{G}, \mathsf{A}\}$ describe *ground* terms and *all* terms, respectively, with $\mathsf{G} \sqsubseteq \mathsf{A}$.

```
rev([], []).
rev([H|T], Rev) :-
  rev(T, RevT), app(RevT, [H], Rev).
app([], L, L).
app([H|L1], L2, [H|L3]) :-
  app(L1, L2, L3).
```

$rev^{\#} : \{\mathsf{G}, \mathsf{A}\}^2 \to \{\mathsf{G}, \mathsf{A}\}^2$

$$
\begin{array}{lcl}
rev^{\#}\langle \mathsf{G}, \mathsf{G} \rangle & = & \langle \mathsf{G}, \mathsf{G} \rangle \\
rev^{\#}\langle \mathsf{G}, \mathsf{A} \rangle & = & \langle \mathsf{G}, \mathsf{G} \rangle \\
rev^{\#}\langle \mathsf{A}, \mathsf{G} \rangle & = & \langle \mathsf{A}, \mathsf{G} \rangle \\
rev^{\#}\langle \mathsf{A}, \mathsf{A} \rangle & = & \langle \mathsf{A}, \mathsf{A} \rangle
\end{array}
$$

Fig. 1. A prolog program for list reversal, and the function $rev^{\#}$ obtained by groundness analysis.

The neededness set for the call pattern $\langle \mathsf{A}, \mathsf{G} \rangle$ for rev is $\{f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle, f_{app}\langle \mathsf{G}, \mathsf{A}, \mathsf{G} \rangle\}$ when the fixpoint approximation $\langle f_{rev}, f_{app} \rangle$ satisfies $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{G}, \mathsf{G} \rangle$.

However, the neededness set becomes $\{f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle, f_{app}\langle \mathsf{A}, \mathsf{A}, \mathsf{G} \rangle\}$ when the fixpoint approximation satisfies $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{A}, \mathsf{A} \rangle$, as it will eventually.

The difference in the neededness sets (and thus the dynamic nature of the neededness information) comes about as follows. When considering the second clause for rev, we first *match* the call pattern $\langle \mathsf{A}, \mathsf{G} \rangle$ and the clause head $rev([H|T], Rev)$. This yields an abstract substitution $\{Rev\}$ which describes all substitions that ground variable Rev. Then we *project* the abstract substitution on the first body atom $rev(T, RevT)$ yielding the call pattern $\langle \mathsf{A}, \mathsf{A} \rangle$ for rev. If $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{G}, \mathsf{G} \rangle$ it follows that the variables T and $RevT$ are now also ground, and eventually we obtain the call pattern $\langle \mathsf{G}, \mathsf{A}, \mathsf{G} \rangle$ for app. By contrast, if $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{A}, \mathsf{A} \rangle$ it is only the variable Rev which is known to be ground before the call to app, yielding the new call pattern $\langle \mathsf{A}, \mathsf{A}, \mathsf{G} \rangle$.

Recording dynamic neededness information allows for omitting evaluation of the call pattern $\langle \mathsf{A}, \mathsf{G} \rangle$ for rev, as long as the fixpoint approximation does not map the needs to values that are larger than at the time of the most recent evaluation.

The paper is organised as follows:

Sections 2–4 consider the fixpoint problem $f(x_1, \ldots, x_x) = E(f, x_1, \ldots, x_k)$. That is, the single equation case ($m = 1$) of the functional equation system (2). Section 2 provides a formalism for defining such fixpoint problems; Section 3 defines the basic method of using neededness information in that context; and Section 4 defines and analyses a neededness-based, chaotic algorithm for solving the fixpoint problem.

Section 5 discusses the fully general variant of the functional fixpoint problem (2). We indicate by an informal argument that the more formal discussion of correctness and complexity presented in Section 4 extend to the general case. Section 6 sketches how a neededness analysis may be used in combination with a minimal function graph type of approach, for the purpose of restricting efforts to the computation of a local fixpoint. Section 7 concludes. Appendix A describes the above Prolog-example in more detail, and Appendix B contains some proofs.

2 A single equation fixpoint problem

For simplicity, we begin with a single equation fixpoint problem. Definition 1 sets up a formalism where an equation $f(x_1, \ldots, x_k) = E$ induces a certain functional and its least fixpoint. The functional is viewed as a finite computational procedure involving the application of certain base functions and the functional parameter, with all functions being first-order. The R.H.S. expression E may contain arbitrary nested occurrences of f, corresponding to calls to the functional parameter. Such nested calls are a major source of difficulty in designing a fixpoint algorithm based on neededness analysis, because they are the reason for the dynamic nature of the neededness information. Once we can deal with this in the single equation context, the generalisation to multiple equations is straightforward.

D is a finite lattice. An equation in the formalism induces the least fixpoint of a functional in $(D^k \to D) \to D^k \to D$. It is assumed that the function space formed by $\to$ contains monotonic as well as nonmonotonic functions. Cartesian products and function spaces are ordered "pointwise". We write $\sqsubseteq$ for the partial ordering on D, and $\preceq$ for the one on $D^k \to D$, and fix for the least fixpoint operator.

Definition 1 Suppose that for each j, b_j is a base function symbol, which is interpreted as a monotonic base function $base_j : D^k \to D$, where k is a fixed arity. Let the language L of expressions be defined by

$$\begin{array}{lll} e ::= x_i & & \text{(primitive parameter symbols, with } 0 \leq i \leq k\text{)} \\ \quad | \quad b_j(e_1, \ldots, e_k) & & \text{(base functions symbols)} \\ \quad | \quad f(e_1, \ldots, e_k) & & \text{(the functional parameter symbol)} \end{array}$$

and let the evaluation function $\mathbf{E} : L \to (D^k \to D) \to D^k \to D$ be defined by:

$$\begin{array}{lll} \mathbf{E}\,[\![x_i]\!]\,\phi\,\Delta & = d_i & \text{(the } i\text{'th element of } \Delta\text{)} \\ \mathbf{E}\,[\![b_j(\ldots, e_i, \ldots)]\!]\,\phi\,\Delta & = base_j(\ldots, \mathbf{E}[\![e_i]\!]\phi\Delta, \ldots) & \\ \mathbf{E}\,[\![f(\ldots, e_i, \ldots)]\!]\,\phi\,\Delta & = \phi(\ldots, \mathbf{E}[\![e_i]\!]\phi\Delta, \ldots) & \end{array}$$

Then the equation $f(x_1, \ldots, x_k) = E$ induces the functional $\mathbf{E}[\![E]\!]$ and its least fixpoint $fix(\mathbf{E}[\![E]\!])$. □

A unique least fixpoint $fix\ (\mathbf{E}\,[\![E]\!])$ exists for any E, since $\mathbf{E}\,[\![E]\!]$ is monotonic when restricted to the monotonic functions in $D^k \to D$. See Figure 2 for an example L-expression and its interpretation under $\mathbf{E}$.

The distinction between a syntactic level (L, f, etc.) and a semantic level ($\mathbf{E}$, ϕ, etc.) facilitates reasoning about the algorithm, saying for instance "for all subexpressions E_{sub} of E, so-and-so holds". In saying so, we want to maintain a distinction between the symbol f (as occurring at various places in E) and the functions $\phi, \phi', \ldots$ that appear as the functional parameter to $\mathbf{E}[\![E]\!]$. The requirement that the fixpoint and the base functions all have functionality $D^k \to D$ is made for simplicity. The crucial property of base functions is monotonicity.

The complexity analysis makes use of the following notation. We write $\bot_D$ for $\sqcap D$ and $\top_D$ for $\sqcup D$. $\mathbf{S}D$ is the number of elements in D, and $\mathbf{H}D$ is the height of D, that is,

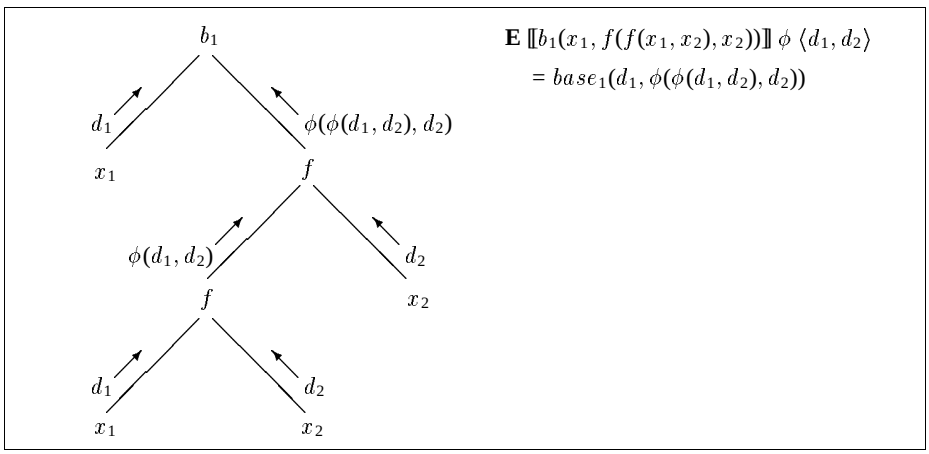

Fig. 2. The L-expression $b_1(x_1, f(f(x_1, x_2), x_2))$ and its interpretation under **E**.

the maximum length of chains from $\perp_D$ to $\top_D$. (The length of a chain is considered to be one less than the number of elements in it). For $d \in D$ we write $|\, d \,|$ for the maximum length of chains from d to $\top_D$. By these definitions we have $|\perp_D| = \mathbf{H}D$, and also

$$d \sqsubseteq d' \Rightarrow |\, d \,| \geq |\, d' \,| \text{ and } d \sqsubset d' \Rightarrow |\, d \,| \geq 1 + |\, d' \,| \tag{3}$$

The power set of D is written $\wp D$. We write

$\phi[\Delta \mapsto \varepsilon]$ for $\lambda x.$if $x = \Delta$ then ε else $\phi(x)$
$\phi{\uparrow_{mon}}$ for $\sqcap\{\phi' \mid \phi \preceq \phi'$ and ϕ' is monotonic$\}$.

It may be the case that the function $\phi[\Delta \mapsto \varepsilon]$ is nonmonotonic even when ϕ is monotonic, and we use $(\cdot{\uparrow_{mon}})$ to produce the least monotonic function lying above some possibly nonmonotonic function.

3 Method: neededness analysis

This section defines the neededness function **N**. In our terminology, information about **N** is *neededness information*. A fixpoint algorithm that relies on *neededness analysis* uses neededness information to infer that certain applications of the induced functional $\mathbf{E}[\![E]\!]$ may safely be omitted, since the applications will not change the current fixpoint approximation. The method or basic idea described in the section is very simple. In particular, Lemma 4 is a straightforward extension to functionals of ideas underlying algorithms proposed by Kildall [11], O'Keefe [8], and others.

The neededness function is defined in terms of the set R_E of subexpressions of E having f as the outermost function symbol, *i.e.*, the "recursive calls" to f:

$$R_E = \{f(\ldots, e_i, \ldots) \mid f(\ldots, e_i, \ldots) \text{ is a subexpression of } E\}$$

Observe that $\mathbf{S}R_E$ is at most the number of occurrences of f in E. (The size of the set may be strictly smaller if there are multiple occurrences having identical arguments).

The neededness set $\mathbf{N}[\![E]\!]\phi\Delta$ consists of those Δ_{need} for which the value of $\phi\Delta_{need}$ is needed in the computation of $\mathbf{E}[\![E]\!]\phi\Delta$.

Definition 2 The neededness function $\mathbf{N} : L \rightarrow (D^k \rightarrow D) \rightarrow D^k \rightarrow \wp(D^k)$ is defined as follows:

$$\mathbf{N}\,[\![E]\!]\,\phi\,\Delta = \bigcup_{f(\ldots,e_i,\ldots)\in R_E} \{\langle \ldots, \mathbf{E}[\![e_i]\!]\phi\Delta, \ldots\rangle\}$$

□

Example 3 When $E = b_1(x_1, f(f(x_1, x_2), x_2))$ (as in Figure 2) we have

$$R_E = \{f(x_1, x_2), f(f(x_1, x_2), x_2)\}$$

Thus for given ϕ and $\Delta = \langle d_1, d_2\rangle$ we obtain the neededness set

$$\mathbf{N}\,[\![E]\!]\,\phi\,\langle d_1, d_2\rangle = \{\langle d_1, d_2\rangle, \langle\phi(d_1, d_2), d_2)\rangle\}$$

□

The following lemma provides a basis for designing an algorithm that relies on neededness analysis. The lemma establishes the safety of omitting computation of $\mathbf{E}[\![E]\!]\phi\Delta$ when a condition in terms of **N** is satisfied.

Lemma 4 *For all $\Delta \in D^k$ and all $\phi, \psi \in D^k \rightarrow D$ we have:*

$$(\forall\Delta_{need} \in \mathbf{N}[\![E]\!]\phi\Delta : \phi\Delta_{need} = \psi\Delta_{need}) \Rightarrow \mathbf{E}[\![E]\!]\psi\Delta = \mathbf{E}[\![E]\!]\phi\Delta$$

Proof We consider arbitrary Δ, ϕ, and ψ; we assume

$$\forall\Delta_{need} \in \mathbf{N}[\![E]\!]\phi\Delta : \phi\Delta_{need} = \psi\Delta_{need} \tag{4}$$

and show by structural induction that

$$\text{For all subexpressions } E_{sub} \text{ of } E : \mathbf{E}[\![E_{sub}]\!]\psi\Delta = \mathbf{E}[\![E_{sub}]\!]\phi\Delta \tag{5}$$

The base case $E_{sub} = x_i$ is trivial. For the inductive case $E_{sub} = f(\ldots, e_i, \ldots)$ we have the following (we omit the inductive case $E_{sub} = b_i(\ldots, e_i, \ldots)$):

$$\begin{aligned}
\mathbf{E}\,[\![E_{sub}]\!]\psi\Delta &= \psi(\ldots, \mathbf{E}[\![e_i]\!]\psi\Delta, \ldots) \text{ (by definition of } \mathbf{E}) \\
&= \psi(\ldots, \mathbf{E}[\![e_i]\!]\phi\Delta, \ldots) \text{ (by hypothesis for structural induction)} \\
&= \phi(\ldots, \mathbf{E}[\![e_i]\!]\phi\Delta, \ldots) \text{ (by (4))} \\
&= \mathbf{E}\,[\![E_{sub}]\!]\phi\Delta \qquad \text{(by definition of } \mathbf{E})
\end{aligned}$$

□

Example 5 Suppose that ϕ and ψ are iterands of the ascending Kleene sequence for $\mathbf{E}[\![E]\!]$ with $\psi = \mathbf{E}[\![E]\!]\phi$, and that we want to compute the next iterand $\mathbf{E}[\![E]\!]\psi$. Suppose that $A \subseteq D^k$ is the set

$$A = \{\Delta \mid \forall\Delta_{need} \in \mathbf{N}[\![E]\!]\phi\Delta : \phi\Delta_{need} = \psi\Delta_{need}\}$$

Then Lemma 4 yields that $\mathbf{E}[\![E]\!]\psi\Delta = \psi\Delta$ for $\Delta \in A$. Therefore, in computing $\mathbf{E}[\![E]\!]\psi$ we need only compute $\mathbf{E}[\![E]\!]\psi\Delta$ for $\Delta \notin A$. □

While our main interest in neededness information is in avoiding *re*-evaluating certain $\Delta \in D^k$ (as in the above example), we note that neededness information is also of interest in avoiding the evaluation of certain tuples *in the first place*. This is the case when we merely want the fixpoint values for a subset of D^k, and exploit this using a minimal function graph type of algorithm. We make use of neededness information for that purpose in Section 6.

There is a computational cost associated with obtaining and manipulating the neededness information. The main problem that must be dealt with is that the information is of a *dynamic* nature, implying that the information cannot, at a reasonably low cost, be computed in a preprocessing phase, prior to the fixpoint iteration. The reason for the dynamic nature of the neededness information is that nested occurrances of f are allowed in L-expressions. Thus, in Example 3 the neededness set contains the tuple $\langle \phi(d_1, d_2), d_2) \rangle$, where the first argument is relative to the current approximation ϕ.

Two things contribute to reducing the cost of repeatedly updating the neededness information.

First, the neededness set $\mathbf{N}[\![E]\!]\psi\Delta$ can be collected *on the fly*. That is, during evaluation of $\mathbf{E}[\![E]\!]\psi\Delta$ we already have to compute $\mathbf{E}[\![E_{sub}]\!]\psi\Delta$ for all subexpressions E_{sub} of E, so no further application of **E** is required in order to compute the neededness set.

Second, whenever we must update the neededness information for some Δ, we are in a situation where we want to apply $\mathbf{E}[\![E]\!]$, so the first observation always applies. This is expressed in the following lemma (which follows from (5)):

Lemma 6 *For all $\Delta \in D^k$ and all $\phi, \psi \in D^k \to D$ we have:*

$$(\forall \Delta_{need} \in \mathbf{N}[\![E]\!]\phi\Delta : \phi\Delta_{need} = \psi\Delta_{need}) \Rightarrow \mathbf{N}[\![E]\!]\psi\Delta = \mathbf{N}[\![E]\!]\phi\Delta$$

Viewed in conjunction with Lemma 4, we may interpret Lemma 6 as saying *"when for some Δ we are free to omit re-computing the fixpoint approximation, we don't have to update the neededness information either"*.

4 A neededness-based chaotic algorithm

This section defines and analyses a neededness-based algorithm for computing the least fixpoint induced by a given equation $f(\ldots, x_i, \ldots) = E$. The algorithm is chaotic (cf. Cousot and Cousot [1, 2]) in the sense that each iteration contains only a single, pointwise application of the induced functional, using the most recent approximation of the fixpoint (as opposed to performing a number of applications using the same approximation). It computes the global fixpoint.

The algorithm is defined in Figure 3. The functions ϕ and N are thought of as tables, mapping each $\Delta \in D^k$ to its fixpoint value as currently approximated, and to its neededness set, respectively. The work set $W \subseteq D^k$ contains all those tuples Δ that must be processed in some subsequent iteration, according to the neededness analysis.

Initialisation. The initial fixpoint approximation ϕ^0 maps all tuples to $\bot_D$. The function N^0 maps all tuples to the empty (neededness) set. The work set W^0 is the entire set D^k; as a consequence all tuples are processed at least once.

Input: an equation $f(x_1, \ldots, x_k) = E$
Output: ϕ when $W = \emptyset$

Variables:
$\Delta \in D^k$
$\phi \in D^k \to D$
$N \in D^k \to \wp(D^k)$
$A, W \in \wp(D^k)$

$\phi^0 = \lambda\Delta.\bot_D$
$W^0 = D^k$
$N^0 = \lambda\Delta.\emptyset$

Δ^{n+1} = any element in W^n
$\phi^{n+1} = \phi^n[\Delta^{n+1} \mapsto \mathbf{E}[\![E]\!]\phi^n\Delta^{n+1}]\uparrow_{mon}$
$N^{n+1} = N^n[\Delta^{n+1} \mapsto \mathbf{N}[\![E]\!]\phi^n\Delta^{n+1}]$
$W^{n+1} = (W^n \setminus \{\Delta^{n+1}\}) \cup A^{n+1}$
where $A^{n+1} = \{\Delta \in D^k \mid \exists \Delta_{need} \in (N^{n+1}\Delta) : \phi^n\Delta_{need} \sqsubset \phi^{n+1}\Delta_{need}\}$

Fig. 3. Fixpoint algorithm for the single equation fixpoint problem.

Iteration. Iteration $n + 1$ uses $\langle \phi^n, N^n, W^n \rangle$ to compute $\langle \phi^{n+1}, N^{n+1}, W^{n+1} \rangle$, which are then used in iteration $n + 2$, *etc.*

Δ^{n+1} is the tuple selected for the iteration's application of the induced functional $\mathbf{E}[\![E]\!]$. The tuple may be any element in W^n.

ϕ^{n+1} is the new approximation. Taking the monotonic function $\phi^n[\ldots]\uparrow_{mon}$ speeds up convergence, because some tuples may be assigned larger approximating values, without being actually processed. For the safety of this "monotonic closure" operation we observe that (for all ϕ) we have

$$\phi \preceq fix(\mathbf{E}[\![E]\!]) \;\Rightarrow\; \phi\uparrow_{mon} \preceq fix(\mathbf{E}[\![E]\!]) \tag{6}$$

N^{n+1} is the new neededness information. The new neededness set for Δ^{n+1} — the set of tuples $\mathbf{N}[\![E]\!]\phi^n\Delta^{n+1}$ — is collected during the computation of $\mathbf{E}[\![E]\!]\phi^n\Delta^{n+1}$ and does not require additional application of $\mathbf{E}$. No other neededness sets are modified.

W^{n+1} is the work set to be used in the following iteration. It is the union of W^n (with Δ^{n+1} deleted from it) and the set A^{n+1}. A tuple belongs to A^{n+1} if there is a tuple Δ_{need} in the tuple's neededness set, such that Δ_{need} is mapped to a strictly larger value under the new approximation ϕ^{n+1}. In the complexity analysis we focus on the the situation where a tuple Δ is *inserted* into the work set, in the sense that the work set doesn't already contain it. In that situation we write $\Delta \in Put^{n+1}$ where

$$Put^{n+1} = A^{n+1} \setminus (W^n \setminus \{\Delta^{n+1}\}) \tag{7}$$

The algorithm is iterative in the following sense: Each iteration corresponds to the processing of a single element $\Delta \in D^k$, during which all needs Δ_{need} are treated by look-up's in the table holding the previous fixpoint approximation. Alternatively, an algorithm following a more recursive or "depth-first" approach (as in [9]) would in some cases suspend the processing of an element, for the purpose of improving on the approximating value for a need.

A *chaotic iteration sequence* for the given equation $f(\ldots, x_i, \ldots) = E$ is a sequence $\phi^0, \ldots, \phi^n, \ldots, \phi^s$ produced by the algorithm in Figure 3, for some instantiation of the procedure for selecting tuples from the work set.

Correctness. By correctness of the single-equation chaotic algorithm we mean that the following must hold for all chaotic iteration sequences: When the algorithm terminates

(*i.e.*, when $W^s = \emptyset$) then $\phi^s = fix(\mathbf{E}[\![E]\!])$ holds. To prove correctness is suffices to establish the following two inequalities.

The inequality $\phi^s \preceq fix(\mathbf{E}[\![E]\!])$ holds due to (6) and since we cannot shoot above the fixpoint by applying the functional $\mathbf{E}[\![E]\!]$ (when starting from the bottom).

The inequality $fix(\mathbf{E}[\![E]\!]) \preceq \phi^s$ can be inferred from the following proposition, using that the work set W is empty at termination, and the fact that $\mathbf{E}[\![E]\!]\phi^s \preceq \phi^s$ implies $fix(\mathbf{E}[\![E]\!]) \preceq \phi^s$. Informally the proposition says that we may safely omit evaluation of any tuple which is not in the current work set.

Proposition 7 *The implication $\Delta \notin W^n \Rightarrow \mathbf{E}[\![E]\!]\phi^n\Delta \sqsubseteq \phi^n\Delta$ holds for all n and Δ.*

The Proposition is proved in Appendix B. The proof begins by establishing the following lemma, which says that the table N records correctly the neededness information for those tuples that are not in the work set. (Proposition 7 is established using Lemma 4 while the proof of Lemma 8 uses Lemma 6).

Lemma 8 *The implication $\Delta \notin W^n \Rightarrow \mathbf{N}[\![E]\!]\phi^n\Delta = N^n\Delta$ holds for all n and Δ.*

Worst case complexity. We measure the complexity of the single-equation chaotic algorithm in terms of the number of times that an expression of the form $\mathbf{E}[\![E]\!]\phi\Delta$ is evaluated.

Consider an arbitrary tuple Δ. The tuple is processed at most once (it is placed in W^0 at initialisation) plus the number of times it is *inserted* ("put") into the work set during iteration (cf. 7). We employ a simple amortised complexity analysis to establish an upper bound on $\mathbf{S}\{n \mid 1 \leq n \leq s \wedge \Delta \in Put^n\}$ for the Δ in question.

The basis for the analysis is the observation that $\Delta \in Put^{n+1}$ requires that one of the needs Δ_{need} of Δ is mapped to a strictly larger value than before. This observation is stated in the following lemma. Part (i) of the lemma says that any subexpression E_{sub} of E is mapped to a sequence of monotonically increasing values. (See Figure 4). Part (ii) says that if $\Delta \in Put^{n+1}$, then it additionally holds that there is *some* subexpression E_{sub} which is now mapped to a *strictly* larger value; we view this subexpression as the *source* of the insertion of Δ into the work set.

Lemma 9 *For all n and Δ we have:*
(i) For all subexpressions E_{sub} of E: $\mathbf{E}\,[\![E_{sub}]\!]\,\phi^n\,\Delta \sqsubseteq \mathbf{E}\,[\![E_{sub}]\!]\,\phi^{n+1}\,\Delta$.
(ii) If $\Delta \in Put^{n+1}$ there exists $E_{sub} \in R_E$ such that: $\mathbf{E}[\![E_{sub}]\!]\phi^n\Delta \sqsubset \mathbf{E}[\![E_{sub}]\!]\phi^{n+1}\Delta$.

Proof (i) Follows from the monotonicity of $\mathbf{E}\,[\![e]\!]$ (for all e) using that $\phi^n \preceq \phi^{n+1}$ always holds.
(ii) Let $\Delta \in Put^{n+1}$. We first establish the equality $\mathbf{N}[\![E]\!]\phi^n\Delta = N^{n+1}\Delta$: When $\Delta = \Delta^{n+1}$ the equality holds by definition of N^{n+1}. When $\Delta \neq \Delta^{n+1}$ we infer $\Delta \notin W^n$ (using (7)), and the equality follows from Lemma 8 using $N^{n+1}(\Delta) = N^n(\Delta)$. Since $\Delta \in A^{n+1}$ and by the equality just shown, there must exist $\Delta_{need} \in \mathbf{N}[\![E]\!]\phi^n\Delta$ such that

$$\phi^n\,\Delta_{need} \sqsubset \phi^{n+1}\,\Delta_{need} \tag{8}$$

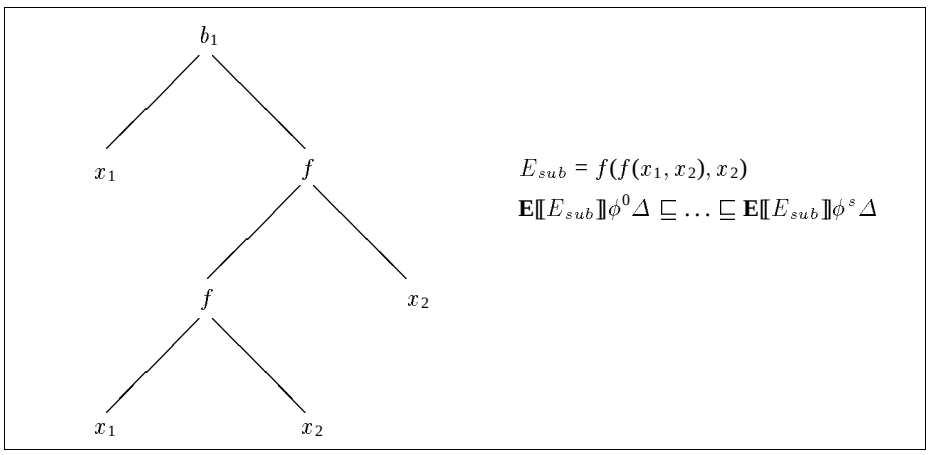

Fig. 4. The subexpression E_{sub} is mapped to a monotonically increasing sequence of values when evaluated against the iterands in a chaotic iteration sequence $\phi^0, \ldots, \phi^s$ (cf. Lemma 9, part (i)). Note that the sequence forms a chain of length $\leq \mathbf{H}D$.

By definition of **N**, the set R_E must contain an expression $E_{sub} = f(\ldots, e_i, \ldots)$ such that $\Delta_{need} = \langle \ldots, \mathbf{E}[\![e_i]\!]\phi^n\Delta, \ldots \rangle$. It follows that

$$\begin{array}{lll} & \phi^n\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^n\Delta, \ldots\rangle \sqsubset \phi^{n+1}\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^n\Delta, \ldots\rangle & \text{(by insertion into (8))} \\ \Rightarrow & \phi^n\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^n\Delta, \ldots\rangle \sqsubset \phi^{n+1}\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^{n+1}\Delta, \ldots\rangle & \text{(since } \phi^n \preceq \phi^{n+1}\text{)} \\ \Rightarrow & \mathbf{E}[\![E_{sub}]\!]\,\phi^n\,\Delta \quad \sqsubset \quad \mathbf{E}[\![E_{sub}]\!]\,\phi^{n+1}\,\Delta & \text{(by definition of } \mathbf{E}\text{)} \end{array}$$

□

By the properties (3) of $|\cdot|$ (see Section 2) we obtain from Lemma 9:

Lemma 10 *For all n and Δ we have:*
(i) For all subexpressions E_{sub} of E: $|\mathbf{E}[\![E_{sub}]\!]\,\phi^n\,\Delta| \geq |\mathbf{E}[\![E_{sub}]\!]\,\phi^{n+1}\,\Delta|$.
(ii) If $\Delta \in Put^{n+1}$ there exists $E_{sub} \in R_E$ s.t. : $|\mathbf{E}[\![E_{sub}]\!]\phi^n\Delta| \geq 1 + |\mathbf{E}[\![E_{sub}]\!]\phi^{n+1}\Delta|$.

The above lemma motivates the following definition of the "potential" function **Pot** which sums up over the numbers $|\mathbf{E}[\![E_{sub}]\!]\phi\Delta|$.

Definition 11 The function **Pot** : $L \to (D^k \to D) \to D^k \to \mathcal{N}$ is defined as follows:

$$\mathbf{Pot}\,[\![E]\!]\,\phi\,\Delta = \sum_{e \in R_E} |\mathbf{E}[\![e]\!]\phi\Delta|$$

□

Example 12 Let $E = b_1(x_1, f(f(x_1, x_2), x_2))$ (cf. Figures 2 and 4). Then

$$\begin{array}{rl} \mathbf{Pot}\,[\![E]\!]\,\phi^0\,\langle d_1, d_2\rangle = & |\mathbf{E}[\![f(x_1, x_2)]\!]\phi^0\langle d_1, d_2\rangle| + |\mathbf{E}[\![f(f(x_1, x_2), x_2)]\!]\phi^0\langle d_1, d_2\rangle| \\ = & |\phi^0(d_1, d_2)| + |\phi^0(\phi^0(d_1, d_2), d_2)| \leq \mathbf{H}D * 2 \end{array}$$

□

The following theorem gives an upper bound on the cost of the neededness-based algorithm. The proof (see Appendix B) establishes that **Pot**$[\![E]\!]\phi^n\Delta$ is a bound on the number of times that Δ is inserted into the work set, from iteration $n+1$ to the last iteration s. The proof also uses that $\mathbf{Pot}\,[\![E]\!]\,\phi^0\,\Delta \leq \mathbf{H}D * \mathbf{S}R_E$, as can be verified directly from the definitions of **Pot** and R_E.

Theorem 13 *Suppose we compute the least fixpoint* fix (**E**$[\![E]\!]$) *by providing the equation* $f(x_1, \ldots, x_k) = E$ *as input to the single-equation algorithm (Figure 3). Then the total number of times that the algorithm evaluates an expression of the form* **E**$[\![E]\!]\phi\Delta$ *is bounded by* $\mathbf{S}D^k * (1 + \mathbf{H}D * \mathbf{S}R_E)$.

In the above complexity bound, the constant 1 in the factor $(1 + \mathbf{H}D * \mathbf{S}R_E)$ is due to the placing of Δ in the work set at initialisation; the product $\mathbf{H}D * \mathbf{S}R_E$ reflects that during iteration, each subexpression $f(\ldots, e_i, \ldots)$ is the source of inserting Δ into the work set at most $\mathbf{H}D$ times. The factor $\mathbf{S}D^k$ is by consideration of all tuples $\Delta \in D^k$.

The above, relatively coarse-grained complexity analysis does not account for the costs of the following operations:

The test for stabilisation is merely a test of whether the work set is empty, so this cost can obviously be neglected.

The "monotonic closure" operation in iteration $n + 1$ is the computation of $\phi^{n+1} = \phi^n[\ldots]\uparrow_{mon}$ when $\phi^n[\ldots]$ is at hand. (This is similar to an application of a function called *adjust* in Le Charlier *et al.*'s [9]). The operation can be implemented as follows. Suppose d is obtained as the value of **E**$[\![E]\!]\phi^n\Delta^{n+1}$. We first test whether $d = \phi^n(\Delta^{n+1})$. When this is true, $\phi^{n+1} = \phi^n$ is already monotonic. When false, we consider the fathers of Δ^{n+1}, *i.e.*, the set of tuples Δ_f lying immediately above Δ^{n+1} wrt. $\sqsubseteq$, and test whether $d \sqsubseteq \phi^n(\Delta_f)$. When false, we let $\phi^{n+1}(\Delta_f) = d \sqcup \phi^n(\Delta_f)$, and proceed recursively by considering the fathers of Δ_f. Thus, there is a cost in terms of some number of the primitive operations: testing $d \sqsubseteq d'$? and computing $d \sqcup d'$. The cost seems acceptable in view of the observation that either we only have to perform few of the primitive operations; or the closure operation leads to many non-trivial updates of the fixpoint approximation, so that the number of future iterations is reduced.

Collecting and managing the neededness information. Collection of the neededness set **N**$[\![E]\!]\phi^n\Delta^{n+1}$ involves no applications of **E** other than those required for the computation of **E**$[\![E]\!]\phi^n\Delta^{n+1}$, as discussed in Section 3. A data structure for managing the neededness information should support respondings to queries of the form: For which Δ does it holds that a given Δ_{need} belongs to the neededness set **N**$[\![E]\!]\phi\Delta$? (This is when the approximating value $\phi\Delta_{need}$ has been changed). Roughly speaking, it is convenient to use a data structure which not only links any tuple Δ to the tuples it needs, but also links any tuple Δ_{need} to those that need it.

5 A multiple equation fixpoint problem

In this section we consider a functional equation system of the form

$$\begin{aligned} f_1(x_1, \ldots, x_k) &= E_1 \\ &\vdots \\ f_m(x_1, \ldots, x_k) &= E_m \end{aligned} \tag{9}$$

We first re-define the language L, the evaluation function **E**, and the neededness function **N** (in Section 2 they were defined for the single equation problem). Then we define an algorithm which computes the least fixpoint $\Phi = \langle \phi_1, \ldots, \phi_m \rangle$ induced by (9), with $\phi_j : D^k \to D$. Finally, we indicate how the proof of correctness and the complexity analysis of the single equation fixpoint algorithm extend to the multiple case.

The re-defined language L has m distinct function symbols $Fct = \{f_1, \ldots, f_m\}$:

$$e ::= \ldots \mid f_r(e_1, \ldots, e_k) \qquad (f_r \in Fct)$$

The evaluation function is re-defined as follows:

$$\begin{array}{ll} \mathbf{E}\,[\![e]\!] : (\{1,\ldots,m\} \to D^k \to D) \to (D^k \to D) & \\ \mathbf{E}\,[\![x_i]\!]\,\Phi\,\Delta & = d_i \\ \mathbf{E}\,[\![b_j(\ldots,e_i,\ldots)]\!]\,\Phi\,\Delta & = base_j(\ldots,\mathbf{E}[\![e_i]\!]\Phi\Delta,\ldots) \\ \mathbf{E}\,[\![f_r(\ldots,e_i,\ldots)]\!]\,\Phi\,\Delta & = \phi_r(\ldots,\mathbf{E}[\![e_i]\!]\Phi\Delta,\ldots) \end{array}$$

The equation system (9) is interpreted as inducing the least fixpoint of the functional

$$\lambda\Phi.\langle \mathbf{E}[\![E_1]\!]\Phi, \ldots, \mathbf{E}[\![E_m]\!]\Phi\rangle$$

The basis for the neededness analysis is the re-defined neededness function:

$$\begin{array}{rl} \mathbf{N}\,[\![e]\!] & : (\{1,\ldots,m\} \to D^k \to D) \to D^k \to \wp(Fct \times D^k) \\ \mathbf{N}\,[\![E]\!]\,\Phi\,\Delta & = \bigcup_{f_r(\ldots,e_i,\ldots)\in R_E} \{\langle f_r, \langle \ldots, \mathbf{E}[\![e_i]\!]\Phi\Delta, \ldots\rangle\rangle\} \end{array}$$

A neededness set contains pairs $\langle f_r, \Delta_{need}\rangle$, where the first argument is the function symbol of the subexpression that yields the need. (In Section 1 and Appendix A we compress the notation and write $f_{rev}(\mathsf{A},\mathsf{A})$ instead of $\langle f_{rev}, \langle \mathsf{A},\mathsf{A}\rangle\rangle$, *etc.*).

Example 14 Consider the equation system

$$\begin{array}{l} f_1(x_1,x_2) = E_1 \text{ where } E_1 = b_1(x_1, f_2(f_2(x_1,x_2),x_2)) \\ f_2(x_1,x_2) = E_2 \text{ where } E_2 = b_1(x_1, f_1(f_1(x_1,x_2),x_2)) \end{array}$$

The neededness sets associated with the two R.H.S. expressions are:

$$\begin{array}{l} \mathbf{N}\,[\![E_1]\!]\,\langle\phi_1,\phi_2\rangle\,\langle d_1,d_2\rangle = \{\langle f_2, \langle d_1,d_2\rangle\rangle, \langle f_2, \langle \phi_2(d_1,d_2), d_2\rangle\rangle\} \\ \mathbf{N}\,[\![E_2]\!]\,\langle\phi_1,\phi_2\rangle\,\langle d_1,d_2\rangle = \{\langle f_1, \langle d_1,d_2\rangle\rangle, \langle f_1, \langle \phi_1(d_1,d_2), d_2\rangle\rangle\} \end{array}$$

□

The chaotic, neededness-based algorithm for the multiple equation fixpoint problem (Figure 5) uses two two-dimensional tables. The table $\Phi = \langle\phi_1,\ldots,\phi_m\rangle$ holds the fixpoint approximation, and $N = \langle N_1,\ldots,N_m\rangle$ is for the neededness information. In iteration $n+1$, a pair $\langle j, \Delta^{n+1}\rangle$ is selected from the work set. Then Δ^{n+1} is provided as input to equation j, possibly leading to a modification of the j'th argument ϕ_j of the fixpoint approximation. If the R.H.S. of equation j

$$E_j = \ldots f_r(\ldots)\ldots$$

contains a call to the r'th function, then a pair $\langle f_r, \Delta_{need}\rangle$ is one of the needs that will be placed in the neededness set $N_j^{n+1}(\Delta)$.

Correctness. The multiple equation algorithm is correct if any chaotic iteration sequence (obtained by instantiation of the procedure for selecting pairs $\langle j,\Delta\rangle$ from the work set) stabilises at an interand Φ^s which is equal to the least fixpoint induced by the equation system. The main point in the correctness proof is to establish that the implication $\langle j,\Delta\rangle \notin W^n \Rightarrow \mathbf{E}\,[\![E_j]\!]\,\Phi^n\,\Delta \sqsubseteq \phi_j^n\,\Delta$ holds for all j, n, and Δ. The proof is a straightforward extension of the proof of Proposition 7.

Input: m equations $f_j(x_1,\ldots,x_k) = E_j$
Output: $\Phi = \langle \phi_1,\ldots,\phi_m\rangle$ when $W = \emptyset$

Variables:
$\Delta \in D^k$
$\phi_j \in D^k \to D$
$N_j \in D^k \to \wp(Fct \times D^k)$
$A, W \in \wp(\{1,\ldots,m\} \times D^k)$

$\Phi^0 = \lambda j.\lambda\Delta.\bot_D$
$W^0 = \lambda j.D^k$
$N^0 = \lambda j.\lambda\Delta.\emptyset$

$\langle j, \Delta^{n+1}\rangle$ = any pair in W^n
$\Phi^{n+1} = \langle \phi_1^n,\ldots,\phi_j^n[\Delta^{n+1} \mapsto \mathbf{E}[\![E_j]\!]\Phi^n\Delta^{n+1}]\uparrow_{mon},\ldots,\phi_m^n\rangle$
$N^{n+1} = \langle N_1^n,\ldots,N_j^n[\Delta^{n+1} \mapsto \mathbf{N}[\![E_j]\!]\Phi^n\Delta^{n+1}],\ldots,N_m^n\rangle$
$W^{n+1} = (W^n \setminus \{\langle j,\Delta^{n+1}\rangle\}) \cup A^{n+1}$

where $A^{n+1} = \{\langle i,\Delta\rangle \in \{1,\ldots,m\}\times D^k \mid \exists \Delta_{need} : \begin{array}{l}\langle f_j, \Delta_{need}\rangle \in (N_i^{n+1}\Delta) \\ \wedge\ \phi_j^n\Delta_{need} \sqsubset \phi_j^{n+1}\Delta_{need}\end{array}\}$

Fig. 5. Fixpoint algorithm for a system of m equations. (In bottom five lines we write j for j^{n+1}).

Complexity analysis. For each equation $f_j(x_1,\ldots,x_k) = E_j$ in (9), we may argue exactly as in the proof of Theorem 13, and thus infer that the number of computations of $\mathbf{E}[\![E_j]\!]\Phi\Delta$ is bound by $\mathbf{S}D^k * (1 + \mathbf{H}D * R_{E_j})$. By summing up over the m equations we obtain that the total number of pointwise applications of the induced functional is bound by

$$\mathbf{S}D^k * \sum_{1\le j\le m} (1 + \mathbf{H}S * \mathbf{S}R_{E_j}) \le m * \mathbf{S}D^k * (1 + \mathbf{H}S * \max_{1\le j\le m} \mathbf{S}R_{E_j})$$

Note that the number of times that a given equation is considered (the number of computations of $\mathbf{E}[\![E_j]\!]\Phi\Delta$ for given j) is independent of m.

6 Computing a local fixpoint

Suppose the task is to find a *local fixpoint*. In the single equation formalism of Section 2, a local fixpoint is a function ϕ satisfying

$$\forall \Delta \in S^0 : \phi\Delta = fix(\mathbf{E}[\![E]\!])\ \Delta$$

where $S^0 \subseteq D^k$ is a set of initially given tuples for which we seek the fixpoint values.

We consider how to compute a local fixpoint using variants of the neededness-based algorithm which are of a minimal function graph [7] type. We use this notion in a broad sense to characterise an algorithm which restricts attention to the tuples in S^0, plus a (hopefully small) set of tuples whose fixpoint values must also be computed, in order to find the local fixpoint.

Suppose $\phi^0,\ldots,\phi^l,\ldots,\phi^s$ is a chaotic iteration sequence produced by the single equation algorithm of Figure 3, and suppose the sequence $\Delta^1,\ldots,\Delta^s$ of tuples selected from the work set satisfies the following property:

$$\text{if } W^n \cap Seen^n \neq \emptyset \text{ then } \Delta^{n+1} \in W^n \cap Seen^n \quad (\text{for } 1 \le n \le l-1) \qquad (10)$$

where

$$\begin{aligned} Seen^0 &= S^0 \\ Seen^{n+1} &= Seen^n \cup \mathbf{N}[\![E]\!]\phi^n \Delta^{n+1} \end{aligned}$$

and where *l* is such that the intersection $W^l \cap Seen^l$ is empty. Then ϕ^l is a local fixpoint.

This observation suggests that a simple, minimal function graph type of algorithm may be designed as follows: Use the selection rule (10), and stop iteration at the first fixpoint approximation ϕ^l where the intersection $W^l \cap Seen^l$ is empty. The remainder $\phi^{l+1}, \ldots, \phi^s$ of the sequence need not be computed. Upon an iteration, there is no need to compute the entire new work set; it suffices to compute the intersection $W^n \cap Seen^n$ directly. Finally, rather than maintaining a full table for the fixpoint approximation ϕ^n, we only record $\phi^n(\Delta)$ explicitly when $\Delta \in Seen^n$.

The algorithm sketched above is simple because the sets of "seen" tuples satisfy $Seen^0 \subseteq \ldots \subseteq Seen^l$. A disadvantage is that the algorithm is relatively coarse in the sense that the accumulated sets may contain "spurious calls" (see Bruynooghe and Gallagher [3]). In the present context, a spurious call is a tuple $\Delta_{spurious}$ for which we compute $\mathbf{E}[\![E]\!]\phi^n \Delta_{spurious}$ (for some $n \leq l$) and where $\Delta_{spurious}$ is not in the *minimal stabilisation set* S^n. The minimal stabilisation set upon iteration $n + 1$ is the set

$$S^{n+1} = fix(\lambda S.S^0 \cup \bigcup_{\Delta \in S} \mathbf{N}[\![E]\!]\phi^n \Delta) \tag{11}$$

i.e., the smallest set of tuples which includes the union of all its neededness sets. The following proposition says that we have reached a local fixpoint when there are no elements of the minimal stabilisation set in the work set.

Proposition 15 *Suppose for some l we have that $S^l \cap W^l = \emptyset$. Then ϕ^l is a local fixpoint.*

The proposition is proved in Appendix B, using that the sequence $\phi^0, \ldots, \phi^l, \ldots, \phi^s$ is a chaotic iteration sequence in the sence of Section 4, so that by Proposition 7 the implication $\Delta \notin W^l \Rightarrow \mathbf{E}[\![E]\!]\phi^l \Delta \sqsubseteq \phi^l \Delta$ holds.

In practice, although the removal of spurious calls is desirable, the computation in each iteration of the minimal stabilisation set is probably too expensive. A compromise such as the method proposed by Rosendahl [14] may be more useful. Proposition 15 ensures correctness of any stabilisation-test $W^n \cap X^n = \emptyset$? where X^n is a superset of the minimal stabilisation set. In particular, the proposition establishes the correctness of the more coarse method based on the selection rule (10).

7 Comparison and conclusion

Table 1 compares three algorithms for computing the least fixpoint induced by a system (9) of m functional equations (either the global or a local fixpoint).

Computing an iterand Φ^{n+1} of the ascending Kleene sequence naively from Φ^n requires the computation of $\mathbf{E}[\![E_j]\!]\Phi^n \Delta$ for all $j \in \{1, \ldots, m\}$ and $\Delta \in D^k$. This yields $m * \mathbf{S}D^k$ applications. There are at most $m * \mathbf{S}D^k * \mathbf{H}D$ iterands (this bound is tight).

The bound for computing a global fixpoint using the multiple equation neededness-based, chaotic algorithm (Figure 5) was established in Section 5. The worst-case number of iterations is considerably lower than by the naive method, when m and the domain D^k are large compared to the number of f-occurrences in the R.H.S. expressions E_j (*i.e.*, the numbers $\mathbf{S}R_{E_j}$). For program analysis applications of the proposed algorithm, the worst-case bound implies the following: If an equation corresponds to a program module or fragment, say a collection of clauses defining the same Prolog predicate, then the cost associated with a given module is independent of the number and form of the remaining modules.

The bound given for the local fixpoint computation is for the coarse "accumulating" method discussed in Section 6. We write $PartialDomSize$ for the total number of tuples that are considered during the iteration process, summed up over the m arguments $\langle \phi^1, \ldots, \phi^m \rangle$ of the fixpoint approximation. Of course, in the worst case the algorithm must consider *all* tuples and compute the global fixpoint. However, in many cases we will have

$$PartialDomSize \ll m * \mathbf{S}D^k$$

Then we achieve the advantage of disregarding a large number of irrelevant elements (the minimal function graph approach), *as well as* having a low bound on the number of times that each relevant element may possibly be considered (the neededness analysis).

naive computation of ascending Kleene-sequence (global fixpoint)	neededness-based chaotic iteration (global fixpoint)	neededness-based chaotic iteration (local fixpoint)
$m * \mathbf{S}D^k$ $* m * \mathbf{S}D^k * \mathbf{H}D$	$m * \mathbf{S}D^k$ $* (1 + \mathbf{H}D * \max_{1 \le j \le m} \mathbf{S}R_{E_j})$	$PartialDomSize$ $* (1 + \mathbf{H}D * \max_{1 \le j \le m} \mathbf{S}R_{E_j})$

Table 1. Worst-case bounds on the number of iterations (the total number of pointwise applications $\mathbf{E}[\![E_j]\!]\Phi\Delta$ of the functional induced by the functional equation system).

The algorithm presented in this paper originates from an algorithm ([5]) for Prolog analysis, having a worst-case complexity bound that improves by two orders of magnitude over the algorithm proposed by Le Charlier *et al.* [9]. We believe that the main difference is that the neededness analysis in [9] is in a sense transitively closed: if a needs b and b needs c, then a needs c. In comparison, the neededness analysis discussed in this paper is more "shallow". Our algorithm is more iterative and less recursive than [9]; the execution of the latter resembles that of a recursive-descent Prolog interpreter to some extent. The recursive or "depth-first" approach of [9] is particularly advantageous in the (common) cases where there are few *re*-evaluations of the same elements of the domain. An interesting problem for future research is, in our view, the possible combination of a shallow neededness analysis (as in this paper) for low worst-case complexity, with a more recursive approach as in the algorithm proposed by Le Charlier *et al.*, for good performance in average cases.

A Dynamic neededness information — an example

This appendix illustrates how a simple Prolog groundness analysis gives rise to dynamic neededness information. Figure 6 shows groundness analysis of the clause

```
rev([H|T],Rev):- rev(T,RevT), app(RevT,[H],Rev).
```

that is, the second clause for the predicate rev in Figure 1. The call pattern is $\langle \mathsf{A}, \mathsf{G} \rangle$.

In the figure, the fixpoint approximation $\Phi = \langle f_{rev}, f_{app} \rangle$ satisfies $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{G}, \mathsf{G} \rangle$ and $f_{app}\langle \mathsf{G}, \mathsf{A}, \mathsf{G} \rangle = \langle \mathsf{G}, \mathsf{G}, \mathsf{G} \rangle$. Then, the computation of a success pattern for $\langle \mathsf{A}, \mathsf{G} \rangle$ wrt. rev has needs $\{f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle, f_{app}\langle \mathsf{G}, \mathsf{A}, \mathsf{G} \rangle\}$. By contrast, if the fixpoint approximation satisfies $f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle = \langle \mathsf{A}, \mathsf{A} \rangle$, the needs become $\{f_{rev}\langle \mathsf{A}, \mathsf{A} \rangle, f_{app}\langle \mathsf{A}, \mathsf{A}, \mathsf{G} \rangle\}$, so the needs are relative to the fixpoint approximation.

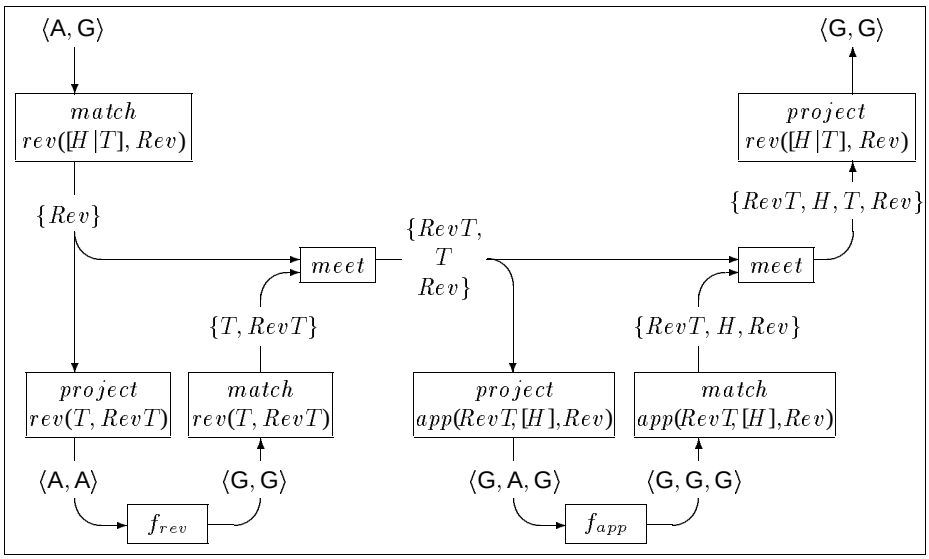

Fig. 6. Groundness analysis of a clause for $rev/2$. The success pattern computed is $\langle \mathsf{G}, \mathsf{G} \rangle$. Boxes represent base functions. Patterns and abstract substitutions are indicated on the arrows.

The abstract operators of the analysis (the base functions, cf. Definition 1) are defined as follows. $p(T_1, \ldots, T_k)$ is a Prolog atom. $\{\mathsf{G}, \mathsf{A}\}$ is a set of modes, where G describes *ground* terms and A describes *all* terms. $ASub$ contains finite sets of variables, where a set of variables describes all substitutions that ground the variables in the set.

$$
\begin{array}{ll}
match\ p(T_1, \ldots, T_k) & : \{\mathsf{G}, \mathsf{A}\}^k \rightarrow ASub \\
project\ p(T_1, \ldots, T_k) & : ASub \rightarrow \{\mathsf{G}, \mathsf{A}\}^k \\
meet & : ASub \rightarrow ASub \rightarrow ASub \\
match\ p(T_1 \ldots T_k)\ \langle m_1 \ldots m_k \rangle & = W_1 \cup \ldots \cup W_k \text{ where } W_i = \begin{cases} vars\ T_i & \text{if } m_i = \mathsf{G} \\ \emptyset & \text{otherwise} \end{cases} \\
project\ p(T_1 \ldots T_k)\ W & = \langle m_1 \ldots m_k \rangle \text{ where } m_i = \begin{cases} \mathsf{G} & \text{if } (vars\ T_i) \subseteq W \\ \mathsf{A} & \text{otherwise} \end{cases} \\
meet\ W\ W' & = W \cup W'
\end{array}
$$

B Proofs

Correctness of the single equation algorithm (Section 4)

Lemma 8 said that the implication $\Delta \notin W^n \Rightarrow \mathbf{N}[\![E]\!]\phi^n\Delta = N^n\Delta$ holds for all n and Δ.

Proof We use induction for $n = 0, 1, \ldots s$ to establish $\forall n : inv_{\mathrm{N}}(n)$ where:

$$inv_{\mathrm{N}}(n) : \Delta \notin W^n \Rightarrow \mathbf{N}[\![E]\!]\phi^n\Delta = N^n\Delta$$

Base case: $inv_{\mathrm{N}}(0)$ holds trivially because $W^0 = D^k$.
Inductive case: We assume $inv_{\mathrm{N}}(n)$. To establish $inv_{\mathrm{N}}(n+1)$ we let $\Delta \notin W^{n+1}$ and show that the equality $\mathbf{N}[\![E]\!]\phi^{n+1}\Delta = N^{n+1}\Delta$ holds. We distinguish between two cases:
$\Delta = \Delta^{n+1}$ where by definition of N^{n+1} we have $N^{n+1} = \mathbf{N}[\![E]\!]\phi^n\Delta$ and then by Lemma 6 the equality follows, using $\Delta \notin A^{n+1}$.
$\Delta \neq \Delta^{n+1}$ where the equality follows directly from the induction hypothesis, using that $N^{n+1}(\Delta) = N^n(\Delta)$. □

Proposition 7 said that the implication $\Delta \notin W^n \Rightarrow \mathbf{E}[\![E]\!]\phi^n\Delta \sqsubseteq \phi^n\Delta$ holds for all n and Δ.

Proof We use induction to establish $\forall n : inv_{\mathrm{E}}(n)$ where:

$$inv_{\mathrm{E}}(n) : \Delta \notin W^n \Rightarrow \mathbf{E}[\![E]\!]\phi^n\Delta \sqsubseteq \phi^n\Delta$$

Base case: $inv_{\mathrm{E}}(0)$ holds trivially because $W^0 = D^k$.
Inductive case. We assume $inv_{\mathrm{E}}(n)$. To establish $inv_{\mathrm{E}}(n+1)$ we let $\Delta \notin W^{n+1}$, and show in two steps that the inequality $\mathbf{E}[\![E]\!]\phi^{n+1}\Delta \sqsubseteq \phi^{n+1}\Delta$ holds:

$$\begin{array}{ll} (1)\ \mathbf{E}[\![E]\!]\phi^{n+1}\Delta = & \mathbf{E}[\![E]\!]\phi^n\Delta \\ (2)\ \mathbf{E}[\![E]\!]\phi^n\Delta \quad \sqsubseteq & \phi^{n+1}\Delta \end{array}$$

(1): Since $\Delta \notin W^{n+1}$ we have $\mathbf{N}[\![E]\!]\phi^{n+1}\Delta = N^{n+1}\Delta$ (by Lemma 8) and then also $\forall \Delta_{need} \in \mathbf{N}[\![E]\!]\phi^{n+1}\Delta : \phi^n(\Delta_{need}) = \phi^{n+1}(\Delta_{need})$ (since $\Delta \notin A^{n+1}$). Thus we can infer (1) using Lemma 4.
(2): We distinguish between two cases:
$\Delta = \Delta^{n+1}$ where (2) follows directly from the definition of ϕ^{n+1}.
$\Delta \neq \Delta^{n+1}$ where we must have $\Delta \notin W^n$ (since $\Delta \notin W^{n+1}$ and Δ was not processed in iteration $n + 1$), so (2) follows from the induction hypothesis using $\phi^n \preceq \phi^{n+1}$. □

Complexity analysis of the single equation algorithm (Section 4)

We first state and prove two lemmas that are prerequisites for the proof of Theorem 13. Recall that $\Delta \in Put^{n+1}$ means that Δ is inserted into W^{n+1} (cf. (7) in Section 4).

Part (ii) of the below lemma says that when $\Delta \in Put^{n+1}$, the potential decreases at least 1.

Lemma 16 *For all n and Δ we have:*
(i) **Pot** $[\![E]\!]\,\phi^n\,\Delta \geq$ **Pot** $[\![E]\!]\,\phi^{n+1}\,\Delta$.
(ii) $\Delta \in Put^{n+1} \Rightarrow$ **Pot** $[\![E]\!]\,\phi^n\,\Delta \geq 1 +$ (**Pot** $[\![E]\!]\,\phi^{n+1}\,\Delta$) .

Proof (i) The inequality is a direct consequence of Lemma 10, part (i).
(ii) By Lemma 10, part (ii), we have that when $\Delta \in Put^{n+1}$ there exists $E_{sub} \in R_E$ such that

$$|\,\mathbf{E}\,[\![E_{sub}]\!]\,\phi^n\,\Delta\,| \geq 1 + |\,\mathbf{E}\,[\![E_{sub}]\!]\,\phi^{n+1}\,\Delta\,| \tag{12}$$

Thus we obtain:

Pot $[\![E]\!]\,\phi^n\,\Delta$
$= \sum_{e \in R_E} |\,\mathbf{E}[\![e]\!]\phi^n\Delta\,|$ (by definition of **Pot**)
$= |\,\mathbf{E}[\![E_{sub}]\!]\phi^n\Delta\,| + \sum_{e \in R_E \setminus \{E_{sub}\}} |\,\mathbf{E}[\![e]\!]\phi^n\Delta\,|$ (by rearranging terms)
$\geq 1 + |\,\mathbf{E}[\![E_{sub}]\!]\phi^{n+1}\Delta\,| + \sum_{e \in R_E \setminus \{E_{sub}\}} |\,\mathbf{E}[\![e]\!]\phi^n\Delta\,|$ (by (12))
$\geq 1 + |\,\mathbf{E}[\![E_{sub}]\!]\phi^{n+1}\Delta\,| + \sum_{e \in R_E \setminus \{E_{sub}\}} |\,\mathbf{E}[\![e]\!]\phi^{n+1}\Delta\,|$ (by Lemma 10 part (ii))
$= 1 +$ (**Pot** $[\![E]\!]\,\phi^{n+1}\,\Delta$) (by definition of **Pot**)

□

Lemma 17 *For all Δ we have* $\mathbf{S}\{n \mid 1 \leq n \leq s \wedge \Delta \in Put^n\} \leq$ **Pot** $[\![E]\!]\,\phi^0\,\Delta$.

Proof It suffices to show the invariant $\forall h \geq 0 : inv_{\text{Pot}}(h)$ where

$$inv_{\text{Pot}}(h) : \mathbf{S}\{n \mid h + 1 \leq n \leq s \wedge \Delta \in Put^n\} \leq \mathbf{Pot}[\![E]\!]\phi^h\Delta$$

The invariant says that $\mathbf{Pot}[\![E]\!]\phi^h\Delta$ is a bound on the number of times that Δ is inserted into the work set, from iteration $h + 1$ onwards. We show the invariant by induction for $h = s, s-1, \ldots, 0$ (recall that ϕ^s is the last iterand).
Base case: $inv_{\text{Pot}}(s)$ holds trivially because $\mathbf{S}\emptyset = 0 \leq \mathbf{Pot}[\![E]\!]\phi^s\Delta$.
Inductive case: We assume $inv_{\text{Pot}}(h + 1)$ and establish $inv_{\text{Pot}}(h)$.
We have:

$$\begin{aligned}
&\mathbf{S}\{n \mid h + 1 \leq n \leq s \wedge \Delta \in Put^n\} \\
&\quad = \begin{cases} \mathbf{S}\{n \mid h + 2 \leq n \leq s \wedge \Delta \in Put^n\} & \text{(when } \Delta \notin Put^{h+1}\text{)} \\ 1 + \mathbf{S}\{n \mid h + 2 \leq n \leq s \wedge \Delta \in Put^n\} & \text{(when } \Delta \in Put^{h+1}\text{)} \end{cases} \\
&\quad \leq \begin{cases} \mathbf{Pot}[\![E]\!]\phi^{h+1}\Delta \\ 1 + \mathbf{Pot}[\![E]\!]\phi^{h+1}\Delta \end{cases} && \text{(by induction hypothesis)} \\
&\quad \leq \mathbf{Pot}[\![E]\!]\phi^h\Delta && \text{(by Lemma 16 part (i) and part (ii), respectively)}
\end{aligned}$$

□

Finally, we can prove Theorem 13, which said that the number of times that the algorithm evaluates an expession of the form $\mathbf{E}\,[\![E]\!]\,\phi\,\Delta$ is bounded by $\mathbf{S}\,D^k * (1 + \mathbf{H}\,S * \mathbf{S}\,R_E)$.

Proof It suffices to show that the number $\mathbf{S}\{n \mid 1 \leq n \leq s \wedge \Delta \in Put^n\}$ is bound by $\mathbf{H}\,S * \mathbf{S}\,R_E$ (for arbitrary $\Delta \in D^k$). We have:

$$\begin{array}{lll} \mathbf{S}\{n \mid 1 \leq n \leq s \wedge \Delta \in Put^n\} = \mathbf{Pot}\,[\![E]\!]\,\phi^0\,\Delta & & \text{(by Lemma 17)} \\ \leq (max_{d \in D}\,|\,d\,|) * \mathbf{S}R_E & & \text{(by definition of \textbf{Pot})} \\ = \mathbf{H}D * \mathbf{S}R_E & & \text{(by definition of } |\cdot|) \end{array}$$

□

The minimal function graph variant of the single-equation algorithm (Section 6)

Proposition 15 said that if for some l we have $S^l \cap W^l = \emptyset$, then ϕ^l is a local fixpoint. Recall that S^{n+1} is the minimal stabilisation set upon iteration $n + 1$ (cf. (11) in Section 6).

Proof We assume $S^l \cap W^l = \emptyset$. Since $\phi^l \preceq \phi^s = fix(\mathbf{E}[\![E]\!])$, to establish that ϕ^l is a local fixpoint it suffices to verify that the inequality

$$fix\ (\mathbf{E}\,[\![E]\!])\ \Delta \sqsubseteq \phi^l\ \Delta \tag{13}$$

holds for all $\Delta \in S^0$. By the above assumption, Proposition 7 yields

$$\forall \Delta \in S^l : \mathbf{E}[\![E]\!]\phi^l\Delta \sqsubseteq \phi^l\Delta \tag{14}$$

so to show (13) it suffices to establish $Q(fix(\mathbf{E}[\![E]\!]))$ where the predicate $Q \subseteq D^k \to D$ is defined by

$$Q(\psi) \Leftrightarrow \forall \Delta \in S^l : \psi\Delta \sqsubseteq \mathbf{E}[\![E]\!]\phi^l\Delta$$

We proceed by fixpoint induction over the monotonic functions in $D^k \to D$. The predicate Q is inclusive since $Q(\lambda\Delta.\bot_D)$ holds and $D^k \to D$ is finite. It remains to establish the implication $Q(\psi) \Rightarrow Q(\mathbf{E}[\![E]\!]\psi)$ for arbitrary monotonic ψ. We assume $Q(\psi)$ and show by structural induction that for all subexpressions E_{sub} of E we have

$$\forall \Delta \in S^l : \mathbf{E}[\![E_{sub}]\!]\psi\Delta \sqsubseteq \mathbf{E}[\![E_{sub}]\!]\phi^l\Delta$$

We show only the case for $E_{sub} = f(\ldots, e_i, \ldots)$. We let Δ be an arbitrary element of S^l, and use that

$$\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^l\Delta, \ldots \rangle \in S^l \tag{15}$$

since $\langle \ldots, \mathbf{E}[\![e_i]\!]\phi^l\Delta, \ldots \rangle \in \mathbf{N}[\![E]\!]\phi^l\Delta = \mathbf{N}[\![E]\!]\phi^{l-1}\Delta$ (by Lemma 6) and since $\mathbf{N}[\![E]\!]\phi^{l-1}\Delta \subseteq S^l$ (the set S^l is "closed wrt. neededness sets", cf. (11)). Thus:

$$\begin{array}{lll} \mathbf{E}[\![E_{sub}]\!]\psi\Delta = \psi(\ldots, \mathbf{E}[\![e_i]\!]\psi\Delta, \ldots) & & \text{(by definition of } \mathbf{E}) \\ \sqsubseteq \psi(\ldots, \mathbf{E}[\![e_i]\!]\phi^l\Delta, \ldots) & & \text{(by hypoth. for struct. induction)} \\ \sqsubseteq \mathbf{E}[\![E]\!]\phi^l(\ldots, \mathbf{E}[\![e_i]\!]\phi^l\Delta, \ldots) & & \text{(by (15) and } Q(\psi)) \\ \sqsubseteq \phi^l(\ldots, \mathbf{E}[\![e_i]\!]\phi^l\Delta, \ldots) & & \text{(by (15) and (14))} \\ = \mathbf{E}[\![E_{sub}]\!]\phi^l\Delta & & \text{(by definition of } \mathbf{E}) \end{array}$$

□

References

1. P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: Mathematical foundations. *Proc. ACM Symposium on Artificial Intelligence and programming languages*, SIGPLAN Notices, **12** (8),1977, pp. 1–12.
2. P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *IFIP Conf. on Formal description of programming concepts*, North-Holland Publ. Co., Amsterdam 1977, pp. 237-277.
3. J. Gallagher and M. Bruynooghe. The derivation of an algorithm for program specialisation. *New Generation Computing*, **9**, 1991, 305–333.
4. N. Jørgensen. *Abstract interpretation of constraint logic programs.* Ph.D. thesis. Computer Science Dept., Roskilde University, 1992.
5. N. Jørgensen. Chaotic fixpoint iteration guided by dynamic dependency. *Proc. Workshop on Static Analysis*, 1993, LNCS 724, Springer-Verlag, pp. 1-14
6. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *J. Logic Programming* 1992, **13**, pp. 315–347.
7. N. D. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs. *Proc. 13th ACM Symposium on Principles of Programming Languages*, 1986, pp. 296–306.
8. R.A. O'Keefe. Finite fixed-point problems. *Proc. 4th International Conference on Logic Programming*, 1987, pp. 729–743.
9. B. Le Charlier, K. Musumbu, P. Van Hentenryck. A generic abstract interpretation algorithm and its complexity analysis. *Proc. 8th International Conference on Logic Programming*, 1991, pp. 64–78.
10. B. Le Charlier and P. Van Hentenryck. *Experimental evaluation of a generic abstract interpretation algorithm for Prolog*. Technical Report CS-91-55, August 1991, Brown University.
11. G.A. Kildall. A unified approach to global program optimization. *Proc. ACM Symposium on Principles of Programming Languages*, 1993, 194–206.
12. T.J. Marlowe and B.G. Ryder. Properties of data flow frameworks. A unified model. *Acta Informatica* 28, 1990, pp. 121–163.
13. S. Peyton-Jones and C. Clack. Finding fixpoints in abstract interpretation, in: S. Abramski and C. Hankin (eds): *Abstract interpretation of declarative languages*. Ellis Horwood, 1987, pp. 246–265.
14. M. Rosendahl. Higher-order chaotic iteration sequences. *Proc. 5th International Symposium on Programming Language and Implementation and Logic Programming*, 1993, LNCS 714, Springer-Verlag, pp. 332–345.
15. P. Van Hentenryck, O. Degimbe, B. Le Charlier, and L. Michel. The impact of granularity in abstract interpretation of Prolog. *Proc. Workshop on Static Analysis*,1993, LNCS 724, Springer-Verlag, pp. 1-14.