UNIVERSITÄT PASSAU

Fakultät für Mathematik und Informatik

Chaotic Fixed Point Iterations

Alfons Geser, Jens Knoop, Gerald Lüttgen, Bernhard Steffen
Fakultät für Mathematik und Informatik
Universität Passau
Innstraße 33
D–94032 Passau
Germany

Oliver Rüthing
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität
Preußerstraße 1–9
D–24105 Kiel
Germany

MIP-9403
Oktober 1994

Abstract

In this paper we present a new *fixed point theorem* applicable for a countable system of recursive equations over a *wellfounded* domain. Wellfoundedness is an essential feature of many computer science applications as it guarantees termination of the corresponding fixed point computation algorithms. Besides being a natural restriction, it marks a new area of application, where not even monotonicity is required. We demonstrate the power and versatility of our fixed point theorem, which under the wellfoundedness condition covers all the known 'synchronous' versions of fixed point theorems, by means of applications in *data flow analysis* and *program optimization.*

Keywords

Fixed point, chaotic iteration, vector iteration, data flow analysis, program optimization, workset algorithm, partial dead code elimination.

Contents

1 Introduction

Many practically relevant problems in computer science can be formalized as follows. Given some *family of functions* $\mathcal{F} =_{df} \{f_k \mid k \in I\!N\}$ on a *wellfounded partial order* $\langle D; \sqsubseteq \rangle$, find the least solution of the system of *recursive equations* $x = f_1(x), x = f_2(x), \ldots$, i.e. the *least common fixed point*, $\mu\mathcal{F}$, of the functions f_k. A typical algorithm solving this task would start with the initial value $x_0 = \perp$ for x, where $\perp$ denotes the least element of D, and successively apply the functions $f_k \in \mathcal{F}$ in an arbitrary order producing a so called *chaotic iteration*, e.g.

$$x_1 = f_1(x_0),\ x_2 = f_5(x_1),\ x_3 = f_{27}(x_2),\ x_4 = f_1(x_3),\ \ldots$$

Our main theorem guarantees the existence of a common fixed point of $\mathcal{F}$, if all functions in $\mathcal{F}$ are increasing with respect to $\sqsubseteq$, and it guarantees the success of the procedure above, if the functions are additonally *locally monotonic* and applied sufficiently often. Whereas local monotonicity is a generalization of the usual monotonicity property, the second requirement is essentially a *fairness property.*

The remainder of the paper is structured as follows. In Section 2 we present a new fixed point theorem. In particular, we show that *vector iterations* are a special case of chaotic iterations. In Section 3 we demonstrate the power and versatility of our theorem by an application each in *data flow analysis* and *program optimization.* In data flow analysis it is the key for proving correctness of some common iterative *workset algorithms*, i.e. they compute the *extreme fixed point* solution of a data flow analysis problem. In program optimization it is central for proving the optimality of an algorithm for *partial dead code elimination.* Section 4 contains our conclusions and directions to future work.

Related Work

The origin of fixed point theorems in computer science dates back to the fundamental work of Tarski [Tar55]. Tarski's theorem considers a monotonic function and guarantees the existence of its least fixed point with respect to a complete partial order. This setup, however, turned out to be too restrictive for a lot of practically relevant applications which led to a number of generalizations. See [LNS82] for a survey of the history of fixed point theory.

In numerical analysis one is interested in computing the least fixed point of $x = f(x)$ where x is a vector $(x^0, \ldots, x^{n-1}) \in D^n$ and $f : D^n \to D^n$ is a monotonic vector function $(f^0, \ldots, f^{n-1})$ where $f^k : D^n \to D$ (cf. [Rob76]). Liberalizing Tarski's iteration $x_0 = \perp, x_1 = f(x_0), x_2 = f(x_1), \ldots$, one may choose $x_0 = \perp, x_1 = f_{J_0}(x_0), x_2 = f_{J_1}(x_1), \ldots$, where the k-th component $f_{J_i}(x)^k$ of $f_{J_i}(x)$ is $f^k(x)$ if $k \in J_i$ and x^k otherwise. Intuitively, at each step i the set J_i denotes the indices k of the components which are updated. We call such an iteration a *(synchronous) vector iteration.* As we will show, vector iterations are special chaotic iterations. The vector approach has been further generalized towards *asynchronous iterations* [Cou77, ÜD89, Wei93], where f_{J_i} may use components of a choice of earlier vectors x_j, with $j \leq i$, of the iteration.

It is worth noting that the vector iteration approach is still concerned with the fixed point of a *single* function f which is expressed in terms of the common fixed point of the vector functions $f^0, f^1, \ldots, f^{n-1}$. Moreover, as in Tarski's setup, it requires monotonicity of f.

In contrast, we treat common fixed points of a *family of functions* $\mathcal{F} =_{df} \{f_k \mid k \in I\!N\}$, $f_k : D \to D$, which satisfies a weaker notion of monotonicity, called local monotonicity, since monotonicity is often too restrictive in practice.

Basic Notions

A *partial order* $\langle D; \sqsubseteq \rangle$ is a set D together with a reflexive, antisymmetric, and transitive binary relation $\sqsubseteq \subseteq D \times D$. A sequence $(d_i)_{i \in I\!N}$ of elements $d_i \in D$ is called an *(ascending) chain* if $\forall i \in I\!N.\ d_i \sqsubseteq d_{i+1}$. A chain $T =_{df} (d_i)_{i \in I\!N}$ is *stationary* if $\{d_i \mid i \in I\!N\}$ is finite. The partial order relation $\sqsubseteq$ is called *wellfounded* if every chain is stationary. A function $f : D \to D$ on D is *increasing* if $d \sqsubseteq f(d)$ for all $d \in D$, and *monotonic* if $\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$. If $\mathcal{F} =_{df} (f_k)_{k \in I\!N}$ is a family of functions and $s = (s_1, \ldots, s_n) \in I\!N^*$ then f_s is defined by the composition $f_s =_{df} f_{s_n} \circ \cdots \circ f_{s_1}$.

2 Theory

In this section we present our new fixed point theorem which gives sufficient conditions that a family of functions, $\mathcal{F} =_{df} (f_k)_{k \in I\!N}$, has a least common fixed point $\mu\mathcal{F}$. Here each f_k, $k \in I\!N$, is a function $f_k : D \to D$ on a wellfounded partial order $\langle D; \sqsubseteq \rangle$ with least element $\perp$. We present a skeleton algorithm which computes $\mu\mathcal{F}$ if the conditions hold.

2.1 The Main Theorem

The following notions are central for dealing with fixed point iterations of a family of functions.

Definition 2.1 (Strategy, Chaotic Iteration Sequence and Fairness)
Let $\langle D; \sqsubseteq \rangle$ be a partial order and $\mathcal{F} =_{df} (f_k)_{k \in I\!N}$ be a family of increasing functions $f_k : D \to D$. A strategy *is any function $\gamma : I\!N \to I\!N$. A strategy γ and an element $d \in D$ induce a* chaotic iteration *$f_\gamma(d) = (d_i)_{i \in I\!N}$ of elements $d_i \in D$ inductively defined by $d_0 = d$ and $d_{i+1} = f_{\gamma(i)}(d_i)$. A strategy γ is called* fair *iff*

$$\forall i \in I\!N.\, (\exists k \in I\!N.\, f_k(d_i) \neq d_i \textit{ implies } \exists j > i.\, d_j \neq d_i)$$

Fixed point theorems usually require that the considered functions are monotonic. In practice, however, functions are often *not* monotonic, but satisfy the following weaker notion.

Definition 2.2 (Local Monotonicity)
Let $\langle D; \sqsubseteq \rangle$ be a partial order and $\mathcal{F} =_{df} (f_k)_{k \in I\!N}$ be a family of functions $f_k : D \to D$. Then $\mathcal{F}$ is called locally monotonic, *if for all $k \in I\!N$:*

$$d \sqsubseteq d' \textit{ implies } \exists s \in I\!N^*.\, f_k(d) \sqsubseteq f_s(d')$$

```
$d := \bot$;
while $\exists\, k \in I\!N.\ d \neq f_k(d)$ do
    choose $k \in I\!N$ where $d \neq f_k(d)$ in
        $d := f_k(d)$
    ni
od
```

Figure 1: The Nondeterministic Skeleton Algorithm

If every f_k is a *monotonic* function in the usual sense, then $\mathcal{F}$ is locally monotonic. But note that local monotonicity in general does *not* carry over to proper subsets of $\mathcal{F}$.

The following theorem states our main result.

Theorem 2.3 (Chaotic Fixed Point Iterations)
Let $\langle D; \sqsubseteq\rangle$ be a wellfounded partial order with least element $\bot$, $\mathcal{F} =_{df} (f_k)_{k \in I\!N}$ a locally monotonic family of increasing functions, and $\gamma : I\!N \to I\!N$ a fair strategy. Then $\bigsqcup f_\gamma(\bot)$ is the least fixed point $\mu\mathcal{F}$ of $\mathcal{F}$.

As a consequence of the uniqueness of the *least* fixed point, $\bigsqcup f_\gamma(\bot)$ is independent of the choice of γ.

Proof *The* wellfoundedness *of* $\sqsubseteq$, *the* increasingness *of the functions of* $\mathcal{F}$, *and the* fairness *of the strategy directly imply that $\bigsqcup f_\gamma(\bot)$ is a fixed point. Thus, we are left with showing that $\bigsqcup f_\gamma(\bot)$ is a lower bound for every fixed point of $\mathcal{F}$. Let d be an arbitrary fixed point of $\mathcal{F}$, i.e. $f_k(d) = d$ for all $f_k \in \mathcal{F}$ and as a consequence $f_s(d) = d$ for all $s \in I\!N^*$. We show $d_i \sqsubseteq d$ for all $i \in I\!N$ by induction on i. For $i = 0$, we have $d_0 = \bot \sqsubseteq d$. The induction step, $d_{i+1} = f_{\gamma(i)}(d_i) \sqsubseteq f_s(d) = d$ for some $s \in I\!N^*$, then follows from the induction hypothesis, the local monotonicity, and the fixed point property of d. Hence $d_i \sqsubseteq d$ for all i and so $\bigsqcup f_\gamma(\bot) \sqsubseteq d$ by definition of $\bigsqcup$.* □

Theorem 2.3 induces an iterative strategy for computing the least fixed point of $\mathcal{F}$. One defines $\gamma(i)$ at step i during the run of the algorithm. Whenever d_i is not yet a fixed point of $\mathcal{F}$, i.e. there is some $k \in I\!N$ where $f_k(d_i)$ is strictly greater than d_i, one chooses $\gamma(i) = k$ for an arbitrary such k. This idea is illustrated in the nondeterministic skeleton algorithm presented in Figure 1.

2.2 Vector Iterations

Let $\langle C; \sqsubseteq_C\rangle$ be a wellfounded partial order and $D = C^n$ for some $n \in I\!N$, ordered by the pointwise extension $\sqsubseteq$ of $\sqsubseteq_C$. Now let $f : D \to D$ be a monotonic function. Instead of iterating $d_1 = f(\bot), d_2 = f(d_1), \ldots$ according to Tarski's theorem, one may pass over to a dissection of f into its components, f^k, i.e. $f(d) = (f^0(d), f^1(d), \ldots, f^{n-1}(d))$ and perform selective updates.[1] Note that the set of fixed points of the function f and the set

[1]Upper indices select a component of a vector of length n.

of common fixed points of the function family $\{f^0, \ldots, f^{n-1}\}$ coincide. A *vector iteration* is an iteration of the form $d_1 = f_{J_0}(\bot), d_2 = f_{J_1}(d_1), \ldots$, where $J_i \subseteq \{0, \ldots, n-1\}$ and

$$f_J(d)^i =_{df} \begin{cases} f^i(d) & \text{if } i \in J \\ d^i & \text{otherwise} \end{cases}$$

performs a selective update of the components specified by J. The set of common fixed points of the function family $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{0, \ldots, n-1\}\}$ is equal to the set of fixed points of f. Note that each f_J is monotonic since f is monotonic.

Now let us demonstrate that the vector approach is modelled conveniently in our setting. To this end, we generalize the notion of a *strategy* to that of a *set strategy* whose range is a subset of $\{0, \ldots, n-1\}$.

The following result shows that for a *monotonic* vector function f, every chaotic iteration sequence is a chain.

Lemma 2.4 (Vector Iterations)
Let $\langle C; \sqsubseteq_C\rangle$ be a wellfounded partial order with least element $\bot$, let $n \in I\!N$, and let $D = C^n$ be ordered by the pointwise extension $\sqsubseteq$ of $\sqsubseteq_C$. Let $f = (f^0, f^1, \ldots, f^{n-1})$ be a monotonic function on D, and let $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{0, \ldots, n-1\}\}$ with functions $f_J : D \to D$ as defined above and γ be a set strategy (on $\{0, \ldots, n-1\}$). Then every chaotic iteration $f_\gamma(\bot)$ is a chain.

Proof *Let γ be an arbitrary set strategy on $\{0, \ldots, n-1\}$, and $(d_i)_{i \in I\!N}$ be its induced chaotic iteration. We have to show*

$$\forall i \in I\!N\, \forall J \subseteq I\!N.\ d_i \sqsubseteq f_J(d_i)$$

The proof is by induction on i. By definition of f_J the property

$$\forall k \notin J \subseteq I\!N\, \forall d \in D.\ f_J(d)^k = d^k$$

holds. Therefore, it suffices to show $d_i^k \sqsubseteq_C f_k(d_i)^k$ for $k \in J$. The case where d_i^k is the least element of C, is trivial. Otherwise, d_i^k must have been updated in an earlier step. More precisely, $d_i^k = f_{\gamma(j)}(d_j)^k$ where j is the greatest index $j < i$ such that $k \in \gamma(j)$. By induction hypothesis for each $j' = j, \ldots, i-1$ and the choice of j, we obtain $d_{j'} \sqsubseteq f_{\gamma(j')}(d_{j'}) = d_{j'+1}$, from which $d_j \sqsubseteq d_i$ follows by the transitivity of $\sqsubseteq$. By monotonicity, $d_i^k = f_{\gamma(j)}(d_j)^k \sqsubseteq f_{\gamma(j)}(d_i)^k = f_k(d_i) = f_J(d_i)^k = d_{i+1}^k$, as required. $\Box$

The following corollary is a special case of Theorem 2.3 for vector iterations and follows by Lemma 2.4. In particular, if $|\mathcal{F}| = 1$ our corollary reduces to the wellfounded case of Tarski's theorem. W.l.o.g. we may assume that $D = \{d \in D \mid \exists s \in I\!N^*.\ f_s(\bot) = d\}$.

Corollary 2.5 (Chaotic Vector Iterations)
Let $\langle C; \sqsubseteq_C\rangle$ be a wellfounded partial order with least element $\bot$, let $n \in I\!N$, and let $D = C^n$ be ordered by the pointwise extension $\sqsubseteq$ of $\sqsubseteq_C$. Let $f = (f^0, f^1, \ldots, f^{n-1})$ be a monotonic function on D, and let $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{0, \ldots, n-1\}\}$, and γ be a fair set strategy. Then $\bigsqcup f_\gamma(\bot)$ is the least fixed point $\mu\mathcal{F}$ of $\mathcal{F}$.

3 Applications

In this section we consider applications of the Fixed Point Theorem 2.3 in *data flow analysis* and *program optimization.* In data flow analysis we use Corollary 2.5 as the key for proving the correctness of some iterative *workset algorithms* as they are common in practice. More precisely, we prove that they compute the *extreme*[2] *fixed point* solution of a data flow analysis problem. In program optimization we show that Theorem 2.3 is central for proving the optimality of an algorithm for *partial dead code elimination.*

3.1 Data Flow Analysis: Workset Algorithms

Data flow analysis (DFA) is concerned with the static analysis of programs in order to support the generation of efficient object code by "optimizing" compilers (cf. [Hec77, MJ81]). For imperative languages, it provides information about the program states that may occur at a given program point during execution. Usually, this information is computed by means of some iterative workset algorithm, which can elegantly be modelled by the vector iteration approach.

In DFA and program optimization (cf. Section 3.2) it is common to represent programs as *directed flow graphs* $G = (N, E, \mathbf{s}, \mathbf{e})$ with node set N and edge set E. Nodes $n \in N$ represent the statements, edges $(n, m) \in E$ the nondeterministic branching structure of the program under consideration, and $\mathbf{s}$ and $\mathbf{e}$ the unique *start node* and *end node* of G, which are assumed to possess no predecessors and successors, respectively. Moreover, $pred_G(n) =_{df} \{\, m \mid (m, n) \in E \,\}$ denotes the set of all immediate predecessors, whereas $succ_G(n) =_{df} \{\, m \mid (n, m) \in E \,\}$ denotes the set of all immediate successors of a node n. Finally, every node $n \in N$ is assumed to lie on a path from $\mathbf{s}$ to $\mathbf{e}$, i.e. every node $n \in N$ is reachable from $\mathbf{s}$, and $\mathbf{e}$ is reachable from every node $n \in N$.

Theoretically wellfounded are DFAs that are based on *abstract interpretation* (cf. [CC77, Mar93]). The point of this approach is to replace the "full" semantics of a program by a simpler more abstract version, which is tailored to deal with a specific problem. Usually, the abstract semantics is specified by means of a *local semantic functional*

$$[\![\ ]\!] : N \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$$

which gives abstract meaning to every program statement in terms of a monotonic (or even continuous) transformation function on a wellfounded partial order $\langle \mathcal{C}; \sqsubseteq \rangle$ with least element $\perp$, whose elements express the DFA-information of interest.

Given a program G and a local abstract semantics $[\![\ ]\!]$, the goal of DFA is to annotate the program points of G with DFA-information that properly reflect the run-time behaviour of G with respect to the problem under consideration. Formally, this annotation is defined by the least solution of Equation System 3.1 which specifies the consistency between pre-conditions of the statements of G expressed in terms of $\mathcal{C}$ with respect to some start information $c_0 \in \mathcal{C}$. This annotation is known as the solution of the *minimal fixed point (MFP)* strategy in the sense of Kam and Ullman [KU77].

[2]I.e., either minimal or maximal.

```
$pre[\mathbf{s}] := c_0$;
forall $n \in N \backslash \{\mathbf{s}\}$ do $pre[n] := \bot$ od;
$workset := N$;
while $workset \neq \emptyset$ do
    choose $n \in workset$ in
        $workset := workset \backslash \{ n \}$;
        $new := pre[n] \sqcup \bigsqcup \{ [\![ m ]\!](\mathbf{pre}[m]) \mid m \in pred_G(n) \}$;
        if $new \sqsupset pre[n]$ then
            $pre[n] := new$;
            $workset := workset \cup succ_G(n)$
        fi
    ni
od
```

Figure 2: A Workset Algorithm

Equation System 3.1

$$\boldsymbol{pre}(n) \quad = \quad \begin{cases} c_0 & \text{if } n = \mathbf{s} \\ \bigsqcup \{ [\![m]\!](\boldsymbol{pre}(m)) \mid m \in pred_G(n) \} & \text{otherwise} \end{cases}$$

In practice the *MFP*-solution, which we denote by $\mathbf{pre}_{c_0}$, is computed by means of some iterative workset algorithm (see Figure 2).

The correctness of this approach is a consequence of Corollary 2.5 as we are going to show. To begin with, let $G = (N, E, \mathbf{s}, \mathbf{e})$ be the flow graph under consideration, and let $[\![\;]\!] : N \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$ be a local abstract semantics, such that all semantic functions are monotonic. Next let $\mathtt{n}$ denote the number of nodes of N, i.e. $\mathtt{n} =_{df} |N|$, and let $\nu : \{0, \ldots, \mathtt{n}-1\} \rightarrow N$ be a bijective mapping between the set $\{0, \ldots, \mathtt{n}-1\}$ of natural numbers and the set N of nodes of G.

Now let us define $D =_{df} \mathcal{C}^{\mathtt{n}}$ equipped with the pointwise extension of $\sqsubseteq$. One easily verifies that D is a wellfounded partial order. A value $d = (d^0, \ldots, d^{\mathtt{n}-1})$ represents a configuration of the flow graph where the value d^k is assigned to node $\nu(k)$.

For every node $\nu(k)$ of the flow graph we define a function $f_k : D \rightarrow D$ by

$$f_k(d^0, \ldots, d^{\mathtt{n}-1}) =_{df} (d'^0, \ldots, d'^{\mathtt{n}-1})$$

where

$$d'^i = \begin{cases} d^i & \text{if } i \neq k \\ d^i \sqcup \bigsqcup \{ [\![\nu(m)]\!](d^m) \mid \nu(m) \in pred_G(\nu(i)) \} & \text{otherwise} \end{cases}$$

Intuitively, f_k describes the effect of a computation of the local semantics at node $\nu(k)$ for the whole configuration. One easily verifies that all f_k are monotonic, and, as a consequence of Lemma 2.4, also increasing. Hence, they are also locally monotonic, where $\mathcal{F}$ is given by $(f_k)_{k<\mathtt{n}}$. The following lemma states that the task is modelled correctly.

Lemma 3.2 *For all $d \in D$ we have: d is a fixed point of Equation System 3.1 if and only if d is a fixed point of $\mathcal{F}$.*

The workset algorithm of Figure 2 follows the general pattern of the nondeterministic skeleton algorithm of Figure 1. It profits from a set *workset* of indices which satisfies the invariant: $workset \supseteq \{k | \ f_k(d) \neq d\}$. As shown above, the premises of Theorem 2.3 are satisfied. Hence we obtain the following correctness theorem.

Theorem 3.3 (Correctness)
Every run of the workset algorithm terminates and computes the MFP-solution $\boldsymbol{pre}_{c_0}$.

3.2 Program Optimization: Partial Dead Code Elimination

In this section we demonstrate an application of the Chaotic Fixed Point Iteration Theorem 2.3 in program optimization by proving the optimality of the *partial dead code elimination* algorithm of [KRS94b]. Intuitively, an assignment in a program is *dead* if its left hand side variable is dead immediately after its execution, i.e., if on every program continuation reaching the end node of the program the first use of the left hand side variable is preceded by a redefinition of it. Correspondingly, an assignment is *partially dead*, if it is dead along some program paths reaching the end node of the program.

Conceptually, the elimination of partially dead occurrences of an assignment pattern α (for short: partially dead α-occurrences) can be decomposed into two steps. First, by moving them as far as possible in the direction of the control flow, and second, by removing all α-occurrences, whose left hand side variable is dead after the execution of the assignment it occurs in. In order to preserve the program semantics, both the sinking and the elimination steps must be *admissible*. The corresponding definitions are given in full detail in [KRS94b], and, therefore, we do not recall them here, but restrict the presentation of the framework of [KRS94b] to those parts that are essential for the application of Theorem 2.3.

The relevance of Theorem 2.3 for partial dead code elimination stems from the fact that assignment sinking and elimination steps in general have *second order effects*, i.e. they usually enable assignment sinking and elimination steps for other assignment patterns. For example, eliminating the partially dead occurrences of a given assignment pattern is often the premise that occurrences of other assignment patterns can be eliminated at all. In [KRS94b] this is taken care of by repeatedly applying admissible assignment sinking and elimination steps to the assignment patterns of the program under consideration until the program stabilizes, i.e. until a fixed point is reached. The correctness of this iterative approach is a consequence of Theorem 2.3, as we are going to show in the remainder of this section.

Given a program G, we will write $G \vdash_{se} G'$ if the flow graph G' results from G by applying an admissible assignment sinking or elimination transformation. Then,

$$\mathcal{G} =_{df} \{ G' \mid G \vdash_{se}^{*} G' \}$$

denotes the *universe* of programs resulting from partial dead code elimination.

Moreover, we denote the set of all admissible assignment sinking and dead code elimination functions by $\mathcal{S}$ and $\mathcal{E}$, respectively, and introduce the abbreviation $\mathcal{T} =_{df} \mathcal{S} \cup \mathcal{E}$. $\mathcal{T}$ consists of all functions $f_{G_1,G_2} : \mathcal{G} \to \mathcal{G}$ defined by

$$\forall\, G' \in \mathcal{G}.\ f_{G_1,G_2}(G') =_{df} \begin{cases} G_2 & \text{if } G' = G_1 \\ G' & \text{otherwise} \end{cases}$$

where $G_1, G_2 \in \mathcal{G}$ with $G_1 \vdash_{se} G_2$.

Additionally, we introduce the set of maximal assignment sinkings and eliminations. A function $f_{G_1,G_2} \in \mathcal{S}$ ($\mathcal{E}$) is called *maximal*, if for all functions $f_{G_1,G_3} \in \mathcal{S}$ ($\mathcal{E}$) there is a function $f_{G_3,G_2} \in \mathcal{T}$ with $f_{G_1,G_2} = f_{G_3,G_2} \circ f_{G_1,G_3}$. We denote the set of all maximal sinking and elimination functions by $\mathcal{S}^{max}$ and $\mathcal{E}^{max}$, respectively, and denote the union of $\mathcal{S}^{max}$ and $\mathcal{E}^{max}$ by $\mathcal{F} \subseteq \mathcal{T}$. The functions of $\mathcal{F}$ are important because they are precisely those functions that are involved in the partial dead code elimination algorithm of [KRS94b].

Finally, given a function $f \in \mathcal{T}$ and a program $G' \in \mathcal{G}$, we introduce the following conventions: The notations $f(G') = G''$ and $G' \vdash^{f}_{se} G''$ are used alternatively. If α is an assignment pattern, then $\mathcal{T}_\alpha$ ($\mathcal{F}_\alpha$) denotes the set of (maximal) admissible assignment sinkings and eliminations with respect to α. If $f \in \mathcal{E}_\alpha$, then $\alpha_{f,G'}$ denotes the set of α-occurrences in G' that are eliminated by f.

In order to compare the quality of different programs in $\mathcal{G}$, we introduce the relation "better" between programs of $\mathcal{G}$.

Definition 3.4 (Optimality)

1. *Let $G', G'' \in \mathcal{G}$. Then G' is* better[3] *than G'', in signs $G'' \precsim G'$, if and only if*

$$\forall\, p \in \mathbf{P}[\mathbf{s},\mathbf{e}]\ \forall \alpha \in \mathcal{AP}.\ \alpha\#(p_{G'}) \leq \alpha\#(p_{G''})$$

   where $\mathcal{AP}$ denotes the set of all assignment patterns occurring in G, and $\alpha\#(p_{G'})$ and $\alpha\#(p_{G''})$ denote the number of occurrences of the assignment pattern α on p in G' and G'', respectively.[4]

2. *$G^* \in \mathcal{G}$ is* optimal *if and only if G^* is better than any other program in $\mathcal{G}$.*

It is easy to check that the relation $\precsim$ is wellfounded. Unfortunately, it is not a partial order on $\mathcal{G}$, but a pre-order only, i.e., it is reflexive and transitive (but not antisymmetric). Hence, there may be several programs being optimal in the sense of Definition 3.4. In order to apply Theorem 2.3, we therefore consider the partial order $\vdash^{*}_{se}$ instead of $\precsim$.

As a first result we have the Dominance Lemma 3.5, which immediately follows from the definitions of $\mathcal{F}_\alpha$ and $\mathcal{T}_\alpha$.

[3]Note that this relation is reflexive. In fact, *at least as good* would be the more precise but uglier notion.

[4]Partial dead code elimination preserves the branching structure of the argument program. Hence, starting from a path in G, we can easily identify corresponding paths in G' and G''.

Lemma 3.5 (Dominance)
Let $G_1 \in \mathcal{G}$, let $f \in \mathcal{F}_\alpha$ and $g \in \mathcal{T}_\alpha$ be corresponding functions, i.e. both sinking or both elimination functions, let $G_1 \vdash_{se}^{g} G_2$, and $G_1 \vdash_{se}^{f} G_3$. Then we have: $G_2 \vdash_{se} G_3$. In particular: $G_3 \neq G_1$ if $G_2 \neq G_1$.

The next lemma can be proved by a straightforward induction on the length of a derivation sequence. The point for proving the induction step is that a program resulting from a transformation of $\mathcal{T}$ is at least as good as its argument with respect to $\precsim$. More detailed, it is in the same equivalence class after a sinking (elimination) step, if no assignment occurrence has been moved out of a loop (eliminated), and it is better otherwise. This follow immediately from the constraints that are satisfied by admissible assignment sinkings and eliminations.

Lemma 3.6 *We have: $G' \vdash_{se}^{*} G'' \Rightarrow G' \precsim G''$*

It is worth noting that Lemma 3.6 yields $\vdash_{se}^{*} \subseteq \precsim$. Thus, the wellfoundedness of $\precsim$ carries over to $\vdash_{se}^{*}$. Moreover, the definitions of $\vdash_{se}$ and $\mathcal{F}$ directly yield that the functions of $\mathcal{F}$ are increasing. Thus, we have:

Lemma 3.7 (Wellfoundedness and Increasingness)

1. *The relation $\vdash_{se}^{*}$ is wellfounded.*
2. *All functions $f \in \mathcal{F}$ are increasing, i.e. $\forall\, f \in \mathcal{F}\ \forall\, G' \in \mathcal{G}.\ G' \vdash_{se}^{*} f(G')$.*

Next we are going to show that $\mathcal{T}$ is locally monotonic. We have:

Lemma 3.8 *Let $G_1, G_2, G_3 \in \mathcal{G}$, and $g, h \in \mathcal{T}$ such that $G_1 \vdash_{se}^{g} G_2$ and $G_1 \vdash_{se}^{h} G_3$.*

1. *If $g \in \mathcal{E}_\alpha$, and occ is an α-occurrence in G_1 with $occ \notin \alpha_{g,G_1}$, then we have:*
   If occ is dead in G_1, then it is dead in G_2.
2. *If $g, h \in \mathcal{S}_\alpha$, occ an α-occurrence that has been moved by g into a join-node n of G_2, and occ' an α-occurrence that has been moved by h into a predecessor m of n, then we have:*[5]
   occ is dead in n iff occ' is dead in m.

Obviously, eliminating certain assignment occurrences does not reanimate assignment occurrences that have been dead before. This observation directly implies the first part of Lemma 3.8. The second part of this lemma is a consequence of the admissibility of g and a simple program transformation supposed in [KRS94b] that is typical for code motion transformations (cf. [DRZ92, KRS92, KRS94a, RWZ88]), namely to insert in every edge leading from a node with more than one successor to a node with more than one predecessor a new 'synthetic' node.

[5]A node n is called a *join-node*, if it has more than one predecessor.

$$\begin{tikzcd}
G_1 \arrow[r, mapsto, "g", "se"'] \arrow[d, mapsto, "f", "se"'] & G_2 \arrow[d, mapsto, "f'", "se"'] \\
G_3 \arrow[r, mapsto, "g'", "se"'] & G_4
\end{tikzcd}$$

Figure 3: Commuting Diagram

Lemma 3.9 *Let $G_1, G_2 \in \mathcal{G}$, let $g \in \mathcal{T}$ such that $G_1 \vdash_{se}^{g} G_2$, let $f \in \mathcal{F}$, and let $\alpha, \beta \in \mathcal{AP}$ be two different assignment patterns. Then we have:*

1. *If $f, g \in \mathcal{E}$, then there are transformations $f', g' \in \mathcal{E}$ such that the diagram in Figure 3 commutes.*
2. *If $f, g \in \mathcal{S}$, then there are transformations $f', g' \in \mathcal{S}$ such that the diagram in Figure 3 commutes.*
3. *If $g \in \mathcal{E}_\alpha$ and $f \in \mathcal{S}_\beta$, then there are transformations $g' \in \mathcal{E}_\alpha$ and $f' \in \mathcal{S}_\beta$ such that the diagram in Figure 3 commutes.*
4. *If $g \in \mathcal{S}_\alpha$ and $f \in \mathcal{E}_\beta$, then there are transformations $g' \in \mathcal{S}_\alpha$ and $f' \in \mathcal{E}_\beta$ such that the diagram in Figure 3 commutes.*

Proof *The first part of Lemma 3.9 is proved by investigating two cases: (1) $g, f \in \mathcal{E}_\alpha$, (2) $g \in \mathcal{E}_\alpha$, $f \in \mathcal{E}_\beta$*

In case (1), the maximality of f guarantees: $\alpha_{g,G_1} \subseteq \alpha_{f,G_1}$. Applying Lemma 3.8(1) we obtain that all α-occurrences in $\alpha_{f,G_1} \backslash \alpha_{g,G_1}$ are dead in G_2. Hence, there is a transformation in $\mathcal{E}$, which eliminates all α-occurrences in $\alpha_{f,G_1} \backslash \alpha_{g,G_1}$ in G_2. Choosing this transformation as f', and an arbitrary function of $\mathcal{F}$ leaving G_3 invariant as g', we get:

$$G_2 \vdash_{se}^{f'} G_4 = G_3$$

and therefore as desired:

$$G_3 \vdash_{se}^{g'} G_3 = G_4$$

In case (2) Lemma 3.8(1) yields that α_{g,G_1} and β_{f,G_1} are subsets of the sets of dead α- and β-occurrences in G_3 and G_2, respectively. Hence, there are transformations in $\mathcal{E}$, which eliminate all α-occurrences of α_{g,G_1} in G_3 and all β-occurrences of β_{f,G_1} in G_2. Choosing these transformations as g' and f', respectively, we obtain as desired

$$G_2 \vdash_{se}^{f'} G_4 \quad \text{and} \quad G_3 \vdash_{se}^{g'} G_4$$

Similarly to the proof of the first part of Lemma 3.9 also in the proof of the second part two cases must be considered: (2a) $g, f \in \mathcal{S}_\alpha$, (2b) $g \in \mathcal{S}_\alpha$, $f \in \mathcal{S}_\beta$

In case (2a) the Dominance Lemma 3.5 yields the existence of an admissible assignment sinking $f' \in \mathcal{S}$, which directly transforms G_2 into G_3. Thus, by choosing an arbitrary function of $\mathcal{F}$ leaving G_3 invariant as g', we succeed in this case.

In order to prove case (2b) consider the program G_2', which results from G_2 by reinserting all α-occurrences that have been moved by g. Let G_4' be the program which results from the maximal β-sinking to G_2', i.e., $G_2' \vdash_{se}^{f} G_4'$, and let G_4 result from G_4' by eliminating the reinserted α-occurrences. Obviously, there is a transformation $f' \in \mathcal{S}_\beta$, which directly transforms G_2 into G_4. The admissibility of g implies that G_2 and G_4 are identical except for β-occurrences. Thus, interchanging the roles of α and β and applying the same construction to G_2, we get the existence of a transformation $g' \in \mathcal{S}_\alpha$, which transforms G_2 into G_4. This completes the proof of case (2b).

The remaining two parts of Lemma 3.9 can now be proved straightforward along the proof lines of part (2) by additionally applying Lemma 3.8(1). $\square$

Lemma 3.8 and Lemma 3.9 allow us to establish the Main Lemma 3.10, which is the key for proving the local monotonicity of $\mathcal{T}$.

Lemma 3.10 (Main Lemma)
$\forall\, g \in \mathcal{T}.\ G_1 \vdash_{se}^{g} G_2 \Rightarrow \forall\, f \in \mathcal{T}\ \exists\, f_1, \ldots, f_n \in \mathcal{T}.\ f(G_1) \vdash_{se}^{*} f_n \circ \ldots \circ f_1(G_2)$

Proof *Let $\alpha, \beta \in \mathcal{AP}$ be different assignment patterns. Then the Main Lemma 3.10 is proved by investigating the following cases:*

1. $g, f \in \mathcal{E}$
2. $g, f \in \mathcal{S}$
3. $g \in \mathcal{E}_\alpha,\ f \in \mathcal{S}_\beta$
4. $g \in \mathcal{S}_\alpha,\ f \in \mathcal{E}_\beta$
5. $g \in \mathcal{E}_\alpha,\ f \in \mathcal{S}_\alpha$
6. $g \in \mathcal{S}_\alpha,\ f \in \mathcal{E}_\alpha$

Due to the Dominance Lemma 3.5 we can assume without loss of generality that f is maximal, i.e., $f \in \mathcal{F}$. The cases (1), (2), (3), and (4) are then immediate consequences of the corresponding parts of Lemma 3.9. Hence, we are left with the cases (5) and (6), which can be proved in the same fashion. Thus, we only present the proof of case (5) here.

In the situation of case (5) let G_2' be the program, which results from G_2 by reinserting a labelled version of all α-occurrences that have been eliminated by g. Due to the labelling the reinserted α-occurrences can be distinguished from the remaining ones. In G_2', we assume that only unlabelled α-occurrences can be subject to assignment sinkings; however, all α-occurrences, i.e., labelled or not, are considered to block the sinking of α-occurrences, i.e., no α-occurrence can sink across a labelled or unlabelled α-occurrence in G_2'. Now we choose the uniquely determined maximal α-sinking and α-elimination as f_1 and f_2, respectively, and denote the program resulting from the subsequent application of f_1 and f_2 to G_2' by G_4'. By eliminating all labelled α-occurrences in G_4' we obtain the program G_4. Of course, f_1 and f_2 have corresponding functions in $\mathcal{T}$ which directly transform G_2 into G_4. Thus, in order to complete the proof of case (5), it is sufficient to show that a maximal α-elimination transforms G_3 into G_4 as well. The point here is that

$$\begin{tikzcd}
G' \arrow[r, "se"'] \arrow[d, "se"'] \arrow[dr, phantom, "\text{Premise}"] & G''' \arrow[r, "*", "se"'] \arrow[d, "*", "se"'] \arrow[dr, phantom, "\text{Ind. Hyp.}"] & G'' \arrow[d, "*", "se"'] \\
f(G') \arrow[r, "*", "se"'] \arrow[d, "*", "se"'] \arrow[drr, phantom, "\text{Ind. Hyp.}"] & {} \arrow[r, "*", "se"'] & {} \arrow[d, "*", "se"'] \\
{} \arrow[rr, "*", "se"'] & & {}
\end{tikzcd}$$

Figure 4: Commuting Diagram

---

due to the reinsertion of α-occurrences eliminated by g, $G_2^{'}$ has precisely the same 'α-blockades' as G_1. Hence, on join-free paths, i.e., on paths where no node has more than one predecessor, (unlabelled) α-occurrences in $G_2^{'}$ have precisely the same sinking potential as their corresponding occurrences in G_1. Only on paths containing join-nodes the sinking potential can be different: In $G_2^{'}$ an α-occurrence occ can be blocked in a predecessor n of a join-node j, because there is a brother m of n,[6] *to which no α-occurrence is sinkable; in G_1, however, the same α-occurrence can successfully be sunk into j, because some of the α-occurrences of α_{g,G_1} are sinkable to m. It is worth noting that the α-occurrence is dead in j and will never become live again. Hence, it is eliminated by the subsequent application of f_2. This, however, holds for the α-occurrence blocked in the predecessor n of j as well, since it is dead according to Lemma 3.8(2). Combining these results we obtain as desired that maximal α-elimination transforms G_3 into G_4.* $\square$

The following theorem states the desired monotonicity result.

Theorem 3.11 (Local Monotonicity)
$\mathcal{T}$ is locally monotonic, i.e.,

$$\forall\, f \in \mathcal{T}.\; G' \vdash_{se}^{*} G'' \Rightarrow \exists\, f_1, \ldots, f_n \in \mathcal{T}.\; f(G') \vdash_{se}^{*} f_n \circ \ldots \circ f_1(G'')$$

Proof *By König's Lemma, any finitely branching tree where all paths have finite length, is finite. Hence, the length of any path is bounded. The derivation tree with root G with respect to $\vdash_{se}$ has this property. Thus we can prove our claim by induction on the length of the longest path starting from G'. The cases $G' = G''$ and $G' = f(G')$ are trivial. Otherwise, we apply Figure 4 where the inductive hypothesis applies for G''' and $f(G')$, respectively, because the longest path starting from G''' is smaller than the longest path starting from G'. Likewise for $f(G')$, which completes the proof.* $\square$

Finally, we have to show that the set of common fixed points of $\mathcal{F}$ and $\mathcal{T}$ coincide. Central for proving this result is the Dominance Lemma 3.5. Moreover, we have to check that the fixed points of $\mathcal{T}$ are maximal in $\mathcal{G}$.

[6] The set of brothers of a node n is given by $\bigcup \{pred(m) \mid m \in succ(n)\}$.

Theorem 3.12 (Fixed Point Characterization)
Let $G' \in \mathcal{G}$.

1. *G' is a fixed point of the functions of $\mathcal{T}$ if and only if G' is a fixed point of the functions of $\mathcal{F}$.*

2. *$G' \in \mathcal{G}$ is a fixed point of the functions of $\mathcal{T}$ if and only if G' is maximal in $\mathcal{G}$.*

Proof *Since (2) holds trivially, we only prove (1). The first implication, "$\Rightarrow$", is a simple consequence of $\mathcal{F} \subseteq \mathcal{T}$. The second implication, " $\Leftarrow$ ", is proved by showing the contrapositive. Without loss of generality, we can assume $g \in \mathcal{T}_\alpha \backslash \mathcal{F}_\alpha$ and $G' \vdash_{se}^{g} G''$ with $G'' \neq G'$. Let now $f \in \mathcal{F}_\alpha$ be the uniquely determined function f of $\mathcal{F}_\alpha$ corresponding to g. Then the Dominance Lemma 3.5 yields as desired that the program resulting from the application of f to G' is different from G'.* $\Box$

Collecting our results we have: $\vdash_{se}^{*}$ is a wellfounded (Lemma 3.7(1)) complete partial order on $\mathcal{G}$, whose least element is G itself; all functions $f \in \mathcal{F}$ are increasing (Lemma 3.7(2)) and $\mathcal{T}$ is locally monotonic with respect to $\vdash_{se}^{*}$ (Theorem 3.11). Hence, Theorem 2.3 is applicable.

Moreover, the function families $\mathcal{F}$ and $\mathcal{T}$ have the same common fixed points (Theorem 3.12(1)), and all of their fixed points are maximal in $\vdash_{se}^{*}$ (Theorem 3.12(2)).

Combining these results and applying Lemma 3.6 we obtain as desired the central theorem of [KRS94b]:

Theorem 3.13 (Existence of Optimal Programs)
$\mathcal{G}$ has an optimal element (with respect to $\underset{\sim}{\sqsubset}$) which can be computed by any sequence of function applications that contains all elements of $\mathcal{F}$ 'sufficiently' often.

It is worth noting that the optimality of the partial faint code elimination algorithm which is also introduced in [KRS94b] can be proved in exactly the same fashion.

4 Conclusions and Future Work

We have presented a new fixed point theorem, which gives a sufficient condition for the existence of the least common fixed point of a family of functions on a wellfounded partial order. The point of this theorem is that the usual monotonicity condition can be weakened for wellfounded partial orders. This allows us to capture a new and interesting class of practically relevant applications. As two representatives of this class, we have discussed applications in *data flow analysis* and *program optimization.*

Currently we are investigating a generalization of our approach to cover asynchronous iterations [Wei93], and we examine its suitability for applications in distant areas of computer science, like e.g. the *Knuth/Bendix procedure* in *term rewriting* and *partitioning algorithms establishing bisimulation* between finite state systems.

References

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conf. Record of the 4th ACM Symp. on Principles of Programming Languages*, pages 238 – 252, Los Angeles, CA, 1977.

[Cou77] P. Cousot. Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice. Technical Report 88, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, September 1977.

[DRZ92] D. M. Dhamdhere, B. K. Rosen, and F. K. Zadeck. How to analyze large programs efficiently and informatively. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation'92*, volume *27*,7 of *ACM SIGPLAN Notices*, pages 212 – 223, San Francisco, CA, June 1992.

[Hec77] M. S. Hecht. *Flow Analysis of Computer Programs.* Elsevier, North-Holland, 1977.

[KRS92] J. Knoop, O. Rüthing, and B. Steffen. Lazy code motion. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation'92*, volume *27*,7 of *ACM SIGPLAN Notices*, pages 224 – 234, San Francisco, CA, June 1992.

[KRS94a] J. Knoop, O. Rüthing, and B. Steffen. Optimal code motion: Theory and practice. *ACM Transactions on Programming Languages and Systems*, 16(4):1117–1155, 1994.

[KRS94b] J. Knoop, O. Rüthing, and B. Steffen. Partial dead code elimination. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation'94*, volume *29*,6 of *ACM SIGPLAN Notices*, pages 147 – 158, Orlando, FL, June 1994.

[KU77] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:309 – 317, 1977.

[LNS82] J.-L. Lassez, V.L. Nguyen, and E.A. Sonnenberg. Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116, 1982.

[Mar93] K. Marriot. Frameworks for abstract interpretation. *Acta Informatica*, 30:103 – 129, 1993.

[MJ81] S. S. Muchnick and N. D. Jones, editors. *Program Flow Analysis: Theory and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Rob76] F. Robert. Convergence locale d'itérations chaotiques non linéaires. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, December 1976.

[RWZ88] B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global value numbers and redundant computations. In *Conf. Record of the 15th ACM Symp. on Principles of Programming Languages*, pages 12 – 27, San Diego, CA, 1988.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[ÜD89] Aydin Üresin and Michel Dubois. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing*, 10:83–92, 1989.

[Wei93] Jiawang Wei. Parallel asynchronous iterations of least fixed points. *Parallel Computing*, 19:887–895, 1993.