Non-monotone Fixpoint Iterations to Resolve Second Order Effects

Alfons Geser,¹ Jens Knoop,¹ Gerald Lüttgen,² Oliver Rüthing,³ Bernhard Steffen¹

Technical Report TR-96-01
Department of Computer Science
North Carolina State University
March 1996

¹Fakultät für Mathematik und Informatik, Universität Passau, Innstraße 33, D-94032 Passau, Germany.
²Department of Computer Science, N.C. State University, Raleigh, NC 27695-8206, USA. The author is supported by the German Academic Exchange Service under grant D/95/09026 (Doktorandenstipendium HSP II / AUFE).
³Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität, Freiestraße 1-9, D-24105 Kiel, Germany.

This paper appears in Proceedings of the 6th International Symposium on Compiler Construction (CC’96), Linköping, Sweden, April 24 - 26, 1996.
ABSTRACT We present a new fixpoint theorem which guarantees the existence and the finite computability of the least common solution of a countable system of recursive equations over a wellfounded domain. The functions are only required to be increasing and delay-monotone, the latter being a property much weaker than monotonicity. We hold that wellfoundedness is a natural condition as it guarantees termination of every fixpoint computation algorithm. Our fixpoint theorem covers, under the wellfoundedness condition, all the known 'synchronous' versions of fixpoint theorems. To demonstrate its power and versatility we contrast an application in data flow analysis, where known versions are applicable as well, to a practically relevant application in program optimization, which due to its second order effects, requires the full strength of our new theorem. In fact, the new theorem is central for establishing the optimality of the partial dead code elimination algorithm considered, which is implemented in the new release of the Sun SPARC Compiler language systems.

Keywords: Data Flow Analysis, Program Optimization, Fixpoint Computation, Second Order Effect, Non-monotone Function, Chaotic Iteration, Partial Dead Code Elimination, Code Motion, Workset Algorithm.

---

4SPARCompiler is a registered trademark of SPARC International, Inc., and is licensed exclusively to Sun Microsystems, Inc.
1 Motivation and Related Work

Many practically relevant problems in computer science can be characterized by means of the least common solution of a system of recursive equations

\[ x = f_1(x) \]
\[ \vdots \]
\[ x = f_n(x) \]

where \( F = \{ f_k : D \rightarrow D \mid 1 \leq k \leq n \} \) is a family of monotone functions on a wellfounded partial order \((D; \sqsubseteq)\). Solving this system of equations is equivalent to the computation of a fixpoint of \( F \), i.e., a common fixpoint \( x = f_k(x) \) of all \( f_k \). A typical iteration algorithm starts with the initial value \( x \), the smallest element of \( D \), and successively updates the value of \( x \) applying the functions \( f_k \) in an arbitrary order, so as to approximate the least fixpoint of \( F \). People speak of a chaotic iteration.

The origin of fixpoint theorems in computer science dates back to the fundamental work of Tarski [20]. Tarski’s theorem considers a monotone function and guarantees the existence of its least fixpoint 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 [14] for a survey of the history of fixpoint theory.

Vector iteration [18] provides such a generalization, where one computes the least fixpoint \( \bar{x} = (x^1, \ldots, x^m) \in D^m \) of a monotone vector function \( \bar{f} = (f^1, \ldots, f^m) \). Liberalizing Tarski’s iteration \( \bar{x}_0 = \bar{1}, \bar{x}_1 = \bar{f}(\bar{x}_0), \bar{x}_2 = \bar{f}(\bar{x}_1), \ldots \), where \( \bar{x}_i \) denotes the value of \( \bar{x} \) after the \( i \)-th iteration, one may choose \( \bar{x}_0 = \bar{1}, \bar{x}_1 = \bar{f}_J(\bar{x}_0), \bar{x}_2 = \bar{f}_J(\bar{x}_1), \ldots \), where \( J_i \subseteq \{1, \ldots, n\} \) and the \( k \)-th component \( f^k_J(\bar{x}_i) \) of \( \bar{f}_J(\bar{x}_i) \) is \( f^k(\bar{x}_i) \) if \( k \in J_i \) and \( \bar{x}_i^k \) otherwise. Intuitively, at each step \( i \) the set \( J_i \) denotes the indices \( k \) of the components which are updated. It is known that a fairness condition for the \( J_i \), which guarantees that every function \( f^k \) is considered sufficiently often, is mandatory. Considering the vectors \( \bar{x} \) as objects and the update operations \( f_J \) as functions, we have a clear instance of the chaotic iteration above (see also Section 3). Recent contributions to fixpoint theory provide efficient strategies for vector iteration, e.g., by using demand driven evaluation strategies (cf. [22, 8]).

The vector approach has been further generalized towards asynchronous iterations [1, 3, 21, 23], where \( f_J \) may use components of a choice of earlier vectors \( \bar{x}_j \), where \( j \leq i \), of the iteration.

Despite its power the vector iteration approach turns out to be too restrictive in two aspects. First, the functions involved in the fixpoint iteration may be such that they cannot be regarded as components of a single func-
tion \( f \). To our knowledge, the only serious attack to this problem has been made by P. and R. Cousot [5]. The common fixpoints of a family \( \mathcal{F} = \{ f_k \}_{k \in \mathbb{N}} \) of monotonic functions are described by iterations, given that each pair \( f_k, f_k^0 \) commutes: \( f_k(f_k^0(x)) = f_k^0(f_k(x)) \) for all \( x \in D \).

Second, and even worse, program transformations may have second order effects (cf. Rosen, Wegman and Zadeck [19]). Typically, program transformations are not idempotent; one transformation may have a strong impact on the profitability of another transformation; often the transformation functions involved are no longer monotone. Then none of the known fixpoint theorems apply.

In this paper we offer a new fixpoint theorem which does not require monotonicity. Given \( d \subseteq d' \), monotonicity amounts to show \( f_k(d) \subseteq f_k(d') \). Instead, we allow that the expression \( f_k(d') \) may be replaced by \( f_j(d') \) for any \( j \), and even by some arbitrary composition of functions applied to \( d' \). If the functions are increasing, i.e. \( x \subseteq f_k(x) \) holds for all \( k \in \mathbb{N} \), this task becomes the easier to solve, the longer the compositions are.

We require only two very weak conditions for technical convenience. First, we require that the underlying domain is wellfounded, a condition which is reasonable in practice, because it means termination of the iteration. In fact we are confident that wellfoundedness is not essential if one can afford nontermination. Second, we require that all functions in \( \mathcal{F} \) are increasing. This requirement is not really restrictive, as we show in Sections 2 and 3.

Our fixpoint theorem is applicable to arbitrary countable families of functions \( \mathcal{F} = \{ f_k : D \rightarrow D \mid k \in \mathbb{N} \} \) on wellfounded partial orders \( \langle D; \sqsubseteq \rangle \): Under the above mentioned premises our theorem guarantees the existence of a least common fixpoint of \( \mathcal{F} \), which is reached eventually by any fair chaotic iteration (cf. Definition 2.1).

The remainder of the paper is structured as follows. We present the new fixpoint theorem in Section 2. In Section 3 we show that vector iterations are a special case of chaotic iterations. Section 4 demonstrates the power of our theorem by applying it to a classical data flow analysis algorithm, and by treating a problem beyond the scope of classical fixpoint theorems: the proof of the optimality of a program optimization for partial dead code elimination [12], which is composed of program transformations having second order effects. This algorithm is implemented in Version 4.0 of the Sun SPARCompiler language systems, which underlines the practical relevance of the new fixpoint theorem. Section 5 contains our conclusions and directions to future work. Finally, the appendix contains all technical proofs of the paper.
2 The Fixpoint Theorem

In this section, we present our new fixpoint theorem, which guarantees under certain premises that a family of functions, \( \mathcal{F} = \{f_k\}_{k \in \mathbb{N}} \), has a least common fixpoint \( \mu \mathcal{F} \), together with a corresponding 'generic' terminating algorithm. This requires the following 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 \( \langle d_i \rangle_{i \in \mathbb{N}} \) of elements \( d_i \in D \) is called an (ascending) chain if \( \forall i \in \mathbb{N} \). \( d_i \sqsubseteq d_{i+1} \). A chain \( T = \{d_i \}_{i \in \mathbb{N}} \) is stationary if \( \{d_i \mid i \in \mathbb{N}\} \) is finite. The partial order relation \( \sqsubseteq \) is called wellfounded if every chain is stationary. A function \( f : D \rightarrow D \) on \( D \) is increasing if \( d \sqsubseteq f(d) \) for all \( d \in D \), and monotone if \( \forall d, d' \in D \). \( d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d') \). If \( \mathcal{F} = \{f_k\}_{k \in \mathbb{N}} \) is a family of functions and \( s = (s_1, \ldots, s_n) \in \mathbb{N}^* \), then \( f_s \) is defined by the composition \( f_s = f_{s_n} \circ \cdots \circ f_{s_1} \).

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

**Definition 2.1 (Strategy, Chaotic Iteration Sequence, Fairness)**

Let \( \langle D; \sqsubseteq \rangle \) be a partial order and \( \mathcal{F} = \{f_k\}_{k \in \mathbb{N}} \) be a family of increasing functions \( f_k : D \rightarrow D \). A strategy is any function \( \gamma : \mathbb{N} \rightarrow \mathbb{N} \). A strategy \( \gamma \) and an element \( d \in D \) induce a chaotic iteration \( f_{\gamma}(d) = (d_i)_{i \in \mathbb{N}} \) of elements \( d_i \in D \) inductively defined by \( d_0 = d \) and \( d_{i+1} = f_{\gamma(i)}(d_i) \). A strategy \( \gamma \) is called fair iff

\[
\forall i, k \in \mathbb{N}, (f_k(d_i) \neq d_i) \; \text{implies} \; \exists j > i, d_j \neq d_i
\]

Fixpoint theorems usually require that the considered functions are monotone. In practice, however, functions are often not monotone, but satisfy the following weaker notion.

**Definition 2.2 (Delay-Monotonicity)**

Let \( \langle D; \sqsubseteq \rangle \) be a partial order and \( \mathcal{F} = \{f_k\}_{k \in \mathbb{N}} \) be a family of functions \( f_k : D \rightarrow D \). Then \( \mathcal{F} \) is called delay-monotone, if for all \( k \in \mathbb{N} \):

\[
d \sqsubseteq d' \; \text{implies} \; \exists s \in \mathbb{N}^*. f_k(d) \sqsubseteq f_k(d')
\]

If every \( f_k \) is a monotone function in the usual sense, then \( \mathcal{F} \) is delay-monotone. Note that delay-monotonicity does in general not carry over to proper subsets of \( \mathcal{F} \).

Now, we are prepared for our main result, which, in particular, yields that \( \bigsqcup \mathcal{F}(\bot) \) is independent of the choice of \( \gamma \).

**Theorem 2.3 (Chaotic Fixpoint Iterations)**

Let \( \langle D; \sqsubseteq \rangle \) be a wellfounded partial order with least element \( \bot \), let \( \mathcal{F} \) be a delay-monotone family \( \{f_k\}_{k \in \mathbb{N}} \) of increasing functions, and let \( \gamma : \mathbb{N} \rightarrow \mathbb{N} \).
be a fair strategy. Then the least common fixpoint $\mu F$ of $F$ exists and is given by $\bigcup f_i(\bot)$. In particular, $\mu F$ is always reached within a finite number of iteration steps.

Note that the following counterexample shows that “increasing” is essential. Let $\bot \sqsubset a$, and $f_1(\bot) = f_1(a) = \bot$, $f_2(\bot) = f_2(a) = a$, i.e. both functions are monotone, but $f_1$ is not increasing. Indeed, $f_1$ and $f_2$ have no common fixpoints.

Theorem 2.3 suggests an iterative strategy for computing the least fixpoint of $F$. One defines $\gamma(i)$ at step $i$ during the run of the algorithm. Whenever $d_i$ is not yet a fixpoint of $F$, i.e. there is some $k \in \mathbb{N}$ 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.

3 Special Case: Vector Iterations

Let $\langle C; \sqsubseteq C \rangle$ be a wellfounded partial order and $D = C^n$ for some $n \in \mathbb{N}$, ordered by the pointwise extension $\sqsubseteq$ of $\sqsubseteq_C$. Now let $f : D \rightarrow D$ be a monotone 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 $f$ to its components, $f^i$, i.e. $f(d) = (f^1(d), \ldots, f^n(d))$ and perform selective updates. Here and in the sequel we use an upper index $i$ at a vector of length $n$ to select its $i$-th component. A vector iteration is an iteration of the form $d_1 = f_{J_1}(\bot), d_2 = f_{J_2}(d_1), \ldots$, where $J_i \subseteq \{1, \ldots, n\}$ and

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

performs a selective update of the components specified by $J$. The set of common fixpoints of the function family $F =_{def} \{ f_J \mid J \subseteq \{1, \ldots, n\} \}$ is equal to the set of fixpoints of $f$. Note that each $f_J$ is monotone since $f$ is monotone.
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. A set strategy is any function \( \gamma : \mathbb{N} \to \mathcal{P}\{1, \ldots, n\} \).

The intended meaning being that \( \gamma(i) \) yields a set \( J_i \) of indices in \( \{1, \ldots, n\} \) of components to be updated at step \( i \). A set strategy is called fair, iff

\[
\forall i \in \mathbb{N}, J \subseteq \mathbb{N}. (f_J(d_i) \neq d_i \implies \exists j > i. d_j \neq d_i)
\]

The following result shows that for a monotone vector function \( f \), every chaotic iteration sequence is a chain, i.e. a totally ordered subset of \( D \).

**Lemma 3.1 (Vector Iterations)**

Let \( (C; \sqsubseteq_C) \) be a wellfounded partial order with least element \( \bot_C \), let \( n \in \mathbb{N} \), and let \( D = C^n \) be ordered by the pointwise extension \( \sqsubseteq \) of \( \sqsubseteq_C \). Let \( f = (f_1, \ldots, f^n) \) be a monotone function on \( D \), and let \( \mathcal{F} = \{f_J : J \subseteq \{1, \ldots, n\}\} \), where the functions \( f_J : D \to D \) are defined as above, and let \( \gamma : \mathbb{N} \to \mathcal{P}\{1, \ldots, n\} \) be a set strategy. Then every chaotic iteration \( f_\gamma(\bot) \) is a chain.

Without loss of generality we may assume that \( D \) is the smallest set that contains \( \bot \) and is closed under \( \mathcal{F} \) and \( \sqsubseteq \). Then increasingness means exactly that every iteration yields a chain. In other words, for vector iterations the increasingness property is no real restriction.

The following corollary is a special case of Theorem 2.3 for vector iterations and a consequence of Lemma 3.1. In particular, if \( |\mathcal{F}| = 1 \) our corollary reduces to Tarski’s theorem [20] in the case of wellfounded partial orders.

**Corollary 3.2 (Chaotic Vector Iterations)**

Let \( (C; \sqsubseteq_C) \) be a wellfounded partial order with least element \( \bot_C \), let \( n \in \mathbb{N} \), and let \( D = C^n \) be ordered by the pointwise extension \( \sqsubseteq \) of \( \sqsubseteq_C \). Let \( f = (f_1, \ldots, f^n) \) be a monotone function on \( D \), let \( \mathcal{F} = \{f_J : J \subseteq \{1, \ldots, n\}\} \), and let \( \gamma \) be a fair set strategy. Then \( \bigsqcup f_\gamma(\bot) \) is the least fixpoint \( \mu \mathcal{F} \) of \( \mathcal{F} \).

In particular, \( \mu \mathcal{F} = \mu f \), and \( \mu \mathcal{F} \) is always reached within a finite number of iteration steps.

4 Applications

In this section, we demonstrate our Fixpoint Theorem 2.3 by proving the correctness and termination of a workset algorithm for data flow analysis, and by establishing terminating optimal program optimization on the basis of program transformations with second order effects. Whereas the first application can already be handled by Corollary 3.2, which reflects the scope of classical vector iteration approaches as they are common in practice, the second application requires the full strength of our main Theorem 2.3.
since the component transformations of the optimization, the algorithm for *partial dead code elimination* of [12], are not even monotone on the relevant domain. Here, the new theorem is central for establishing the optimality of this algorithm, which is implemented in Version 4.0 of the Sun SPARCompiler language systems.

4.1 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. [7, 16]). 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 4.2) it is common to represent programs as directed flow graphs \( G = (N, E, s, e) \) 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 \( s \) and \( e \) the unique start node and end node of \( G \), which are assumed to possess no predecessors and successors, respectively. Moreover, \( \text{pred}_G(n) = \{ m \mid (m, n) \in E \} \) and \( \text{succ}_G(n) = \{ m \mid (n, m) \in E \} \) denote the set of all immediate predecessors and successors of a node \( n \), respectively. Finally, every node \( n \in N \) is assumed to lie on a path from \( s \) to \( e \), i.e. every node \( n \in N \) is reachable from \( s \), and \( e \) is reachable from every node \( n \in N \).

Theoretically wellfounded are DFAs that are based on abstract interpretation (cf. [4, 15]). 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 monotone (or even continuous) transformation function on a wellfounded partial order \( \langle \mathcal{C}, \sqsubseteq \rangle \) with least element \( \bot \), 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 4.1 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 fixpoint (MFP)* approach in the sense of Kam and Ullman [9].
\[\text{pre}[s] := c_0;\]
\[\text{forall } n \in N \setminus \{s\} \text{ do } \text{pre}[n] := \bot \text{ od;}\]
\[\text{workset} := N;\]
\[\text{while workset} \neq \emptyset \text{ do}\]
\[\text{choose } n \in \text{workset in}\]
\[\text{workset} := \text{workset} \setminus \{n\};\]
\[\text{new} := \text{pre}[n] \cup \bigcup \{[m]([\text{pre}[m]]) \mid m \in \text{pred}_G(n)\};\]
\[\text{if } \text{new} \supseteq \text{pre}[n] \text{ then}\]
\[\text{pre}[n] := \text{new};\]
\[\text{workset} := \text{workset} \cup \text{succ}_G(n)\]
\[\text{fi}\]
\[\text{fi}\]
\[\text{od}\]

**FIGURE 2. A Workset Algorithm**

**Equation System 4.1**

\[
\text{pre}(n) = \begin{cases} 
  c_0 & \text{if } n = s \\
  \bigcup \{[m]([\text{pre}[m]]) \mid m \in \text{pred}_G(n)\} & \text{otherwise}
\end{cases}
\]

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

We show that termination and correctness in this approach are a consequence of Corollary 3.2. To begin with, let \(G = (N, E, s, e)\) be the flow graph under consideration, and let \([\ ] : N \rightarrow (C \rightarrow C)\) be a local abstract semantics, such that all semantic functions are monotone. Without loss of generality we identify in the following the set of nodes of \(N\) with the set of natural numbers \(\{1, \ldots, n\}\), where \(n\) denotes the number of nodes of \(N\).

Now let us define \(D=\cup C^n\) equipped with the pointwise extension of \(\subseteq\). One easily verifies that \(D\) is a wellfounded partial order. A value \(d = (d^1, \ldots, d^n)\) represents an annotation of the flow graph where the value \(d^k\) is assigned to node \(k\).

For every node \(k\) of the flow graph we define a function \(f^k : D \rightarrow C\) by

\[
f^k(d^1, \ldots, d^n) = \biguplus d^k
\]

where

\[
d^k = d^k \uplus \bigcup \{[m]([d^m]) \mid m \in \text{pred}_G(k)\}
\]

Intuitively, \(f^k\) describes the effect of a computation of the local semantics at node \(k\). The following lemma states that the DFA problem is modelled correctly.
Lemma 4.2 For all $d \in D$ we have: $d$ is a solution of Equation System 4.1 if and only if $d$ is a fixpoint of $f = c(f_1, \ldots, f_n)$.

The workset algorithm of Figure 2 follows the general pattern of the non-deterministic skeleton algorithm of Figure 1 with $F = \{ f_{(k)} \mid 1 \leq k \leq n \}$. It profits from a set workset of indices which satisfies the invariant: workset $\supseteq \{ k \mid f_{(k)}(d) \neq d \}$. One easily verifies that $f$ is monotone. Hence, the premises of Corollary 3.2 are satisfied and we obtain the following theorem.

Theorem 4.3 (Correctness and Termination)

Every run of the workset algorithm terminates with the MFP-solution $\text{pre}_{ce}$.

4.2 Program Optimization: Partial Dead Code Elimination

In this section, we demonstrate an application of the Chaotic Fixpoint Iteration Theorem 2.3 in program optimization by proving the optimality of the partial dead code elimination algorithm of [12]. 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 of the program the first use of this 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 of the program.

Conceptually, the elimination of partially dead occurrences of an assignment pattern $\alpha$ (for short: partially dead $\alpha$-occurrences) can be decomposed into two steps. First, moving them as far as possible in the direction of the control flow, and second, removing all dead $\alpha$-occurrences. In order to preserve the program semantics, both the sinking and the elimination steps must be admissible. This is defined in full detail in [12]. Therefore, we restrict the presentation here to those parts that are essential for 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 some assignment pattern is often the premise that occurrences of other assignment patterns can be eliminated at all. In [12] this is taken care of by repeatedly applying admissible assignment sinking and elimination steps to the assignment patterns of the argument program until the program stabilizes, i.e. until a fixpoint is reached. The correctness of this iterative approach is a consequence of Theorem 2.3, as we show in the remainder of this section, where we consider an arbitrary, but fixed program $G$.

For a program $G'$, we write $G' \vdash_{ce} G''$ if the flow graph $G''$ results from
G' by applying an admissible assignment sinking or elimination transformation. We denote the set of all admissible assignment sinking and dead code elimination functions by \( S \) and \( E \), respectively. Additionally, we abbreviate \( S \cup E \) by \( T \). It consists of all functions \( f_{G_1, G_2} : G \to G' \) defined by
\[
\forall G' \in G. f_{G_1, G_2}(G') = \begin{cases} 
G_2 & \text{if } G' = G_1 \\
G' & \text{otherwise}
\end{cases}
\]
where \( G_1, G_2 \in G \) and \( G_1 \vdash_G G_2 \). We also write \( G' \vdash_T G'' \) for \( f(G') = G'' \).

Then, \( G = \{ G' \mid G \vdash_G G' \} \) denotes the universe of programs resulting from \( G \) by partial dead code elimination.

In order to compare the quality of different programs in \( G \), we introduce the relation "better" between programs of \( G \). Note that this relation is reflexive. In fact, at least as good would be the more precise but uglier notion.

**Definition 4.4 (Optimality)**

1. Let \( G', G'' \in G \). Then \( G' \) is better than \( G'' \), in signs \( G'' \preceq_G G' \), if and only if for every assignment pattern \( \alpha \) and every program path \( p \) leading from the start node to the end node of the argument program there are at most as many occurrences of \( \alpha \) in \( G' \) as in \( G'' \).

2. \( G^* \in G \) is optimal if and only if \( G^* \) is better than any other program in \( G \).

It is easy to check that the relation \( \preceq_G \) is reflexive, transitive, and well-founded. Unfortunately, it is not antisymmetric. Hence, there may be several programs being optimal in the sense of Definition 4.4. In order to apply Theorem 2.3, we thus consider the partial order \( \vdash_G \) instead of \( \preceq_G \), but we reconsider the relation \( \preceq_G \), subsequently (cf. Theorem 4.13).

In addition to \( S \) and \( E \), we define the set of maximal assignment sinkings and eliminations, which are the functions involved in the partial dead code elimination algorithm of [12]. A function \( f_{G_1, G_2} \in S (E) \) is called maximal, if for all functions \( f_{G_3, G_4} \in S (E) \) there is a function \( f_{G_5, G_6} \in T \) satisfying \( f_{G_5, G_6} = f_{G_3, G_4} \circ f_{G_1, G_2} \). The set of all maximal sinking and elimination functions are denoted by \( S^{max} \) and \( E^{max} \), respectively, and \( T^{max} \subseteq T \) denotes the union of \( S^{max} \) and \( E^{max} \). Finally, we denote the set of (maximal) admissible assignment sinkings and eliminations with respect to an assignment pattern \( \alpha \) by \( T_\alpha \) and \( T^{max}_\alpha \). As a first result we obtain the

\[1\]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 \( G'' \).
Dominance Lemma 4.5, which follows immediately from the definitions of $T_{\alpha}^{\text{max}}$ and $T_{\alpha}$.

**Lemma 4.5 (Dominance)**

Let $G_1 \in \mathcal{G}$, let $f \in T_{\alpha}^{\text{max}}$ and $g \in T_{\alpha}$ be corresponding functions, i.e., both sinking or both elimination functions; let $G_1 \vdash_{se}^f G_2$, and let $G_1 \vdash_{se}^g G_3$. 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 proven 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 $T$ is at least as good as its argument with respect to $\equiv$. It is in the same equivalence class after sinking and trivial elimination steps, i.e., elimination steps where no assignment occurrence has been eliminated; and it is better otherwise. This follows immediately from the constraints that are satisfied by admissible assignment sinkings and eliminations.

**Lemma 4.6** We have: $G' \vdash_{se}^* G'' \Rightarrow G' \equiv G''$

In other words, Lemma 4.6 says $\vdash_{se}^* \subseteq \equiv$. From the wellfoundedness of $\equiv$ and the definitions of $\vdash_{se}$ and $T_{\alpha}^{\text{max}}$ we immediately conclude:

**Lemma 4.7 (Wellfoundedness and Increasingness)**

1. The relation $\vdash_{se}^*$ is wellfounded.
2. All functions $f \in T_{\alpha}^{\text{max}}$ are increasing.

Next, we show that $T$ is delay-monotone. This proof is supported by the following lemma, whose first part is a consequence of the fact that eliminating dead assignment occurrences does not reanimate other dead assignment occurrences, and whose second part is a consequence of the admissibility of $g$ and a simple program transformation supposed in [12] which is typical for code motion transformations (cf. [6, 10, 11, 19]), 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.

**Lemma 4.8** Let $G_1, G_2, G_3 \in \mathcal{G}$, and $g, h \in T$ with $G_1 \vdash_{se}^g G_2$ and $G_1 \vdash_{se}^h G_3$.

1. If $g \in \mathcal{E}_a$, and $\text{occ}$ an $\alpha$-occurrence occurring both in $G_1$ and $G_2$, then we have: If $\text{occ}$ is dead in $G_1$, then it is dead in $G_2$.
2. If $g, h \in \mathcal{S}_a$, $\text{occ}$ an $\alpha$-occurrence that has been moved by $g$ into a node $n$ of $G_2$ with more than one predecessor, and $\text{occ}'$ an $\alpha$-occurrence that has been moved by $h$ into a predecessor $m$ of $n$, then we have: $\text{occ}$ is dead in $n$ iff $\text{occ}'$ is dead in $m$.

Additionally, we have:
Lemma 4.9 Let $G_1, G_2 \in \mathcal{G}$, let $g \in T$, $f \in T^{max}$, and let $\alpha, \beta \in AP$ 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.

Lemma 4.8 and Lemma 4.9 allow us to establish the following lemma, which is the key for proving the delay-monotonicity of $T$.

Lemma 4.10 (Main Lemma)

\[ \forall g \in T. G_1 \vdash_{\ast_e} G_2 \Rightarrow \forall f \in T \exists f_1, \ldots, f_n \in T. f(G_1) \vdash_{\ast_e} f_n \circ \ldots \circ f_1(G_2) \]

The following theorem states the desired delay-monotonicity result. The reasoning closely resembles the classical Newman Lemma [17], saying that confluence follows from local confluence if the given relation is wellfounded. Note that monotonicity does not hold.

Lemma 4.11 (Delay-Monotonicity)

$T$ is delay-monotone, i.e.

\[ \forall f \in T. G' \vdash_{\ast_e} G'' \Rightarrow \exists f_1, \ldots, f_n \in T. f(G') \vdash_{\ast_e} f_n \circ \ldots \circ f_1(G'') \]

Finally, we have to show that the set of common fixpoints of $T^{max}$ and $T$ coincide. Central for proving this result is the Dominance Lemma 4.5. Moreover, we have to check that the fixpoints of $T$ are maximal in $\mathcal{G}$. 
Theorem 4.12 (Fixpoint Characterization)

1. \( G' \in \mathcal{G} \) is a fixpoint of the functions in \( \mathcal{T} \) if and only if \( G' \) is a fixpoint of the functions in \( \mathcal{T}^{\max} \).

2. \( G' \in \mathcal{G} \) is a fixpoint of the functions in \( \mathcal{T} \) if and only if \( G' \) is maximal in \( \mathcal{G} \).

Collecting our results we have: \( \mathcal{T}^{\ast}_{se} \) is a wellfounded (Lemma 4.7(1)) complete partial order on \( \mathcal{G} \), whose least element is \( G \) itself; all functions \( f \in \mathcal{T}^{\max} \) are increasing (Lemma 4.7(2)) and \( \mathcal{T} \) is delay-monotone with respect to \( \mathcal{T}^{\ast}_{se} \) (Lemma 4.11). Hence, Theorem 2.3 is applicable.

Moreover, the function families \( \mathcal{T}^{\max} \) and \( \mathcal{T} \) have the same common fixpoints (Theorem 4.12(1)), and all of their fixpoints are maximal in \( \mathcal{T}^{\ast}_{se} \) (Theorem 4.12(2)).

Combining these results and applying Lemma 4.6 we obtain that there exists a terminating optimal program transformation \([12]:\)

Theorem 4.13 (Optimal Partial Dead Code Elimination)

\( \mathcal{G} \) has (up to local reorderings in basic blocks) a unique optimal element (with respect to \( \mathcal{T}^{\ast}_{se} \)) which can be computed by any fair sequence of function applications from \( \mathcal{T}^{\max} \).

We remark that the optimality of the partial faint code elimination algorithm which is also introduced in \([12]\) as well as the optimality of the algorithm for the uniform elimination of partially redundant expressions and assignments in \([13]\) can be proven in exactly the same fashion.

5 Conclusions

We have presented a new fixpoint theorem, which gives a sufficient condition for the existence and computability of the least common fixpoint of a family of functions on a wellfounded partial order. The point of this theorem is that for wellfounded partial orders the usual monotonicity condition can be substantially weakened. This allows us to capture a new and interesting class of practically relevant applications. To characterize this class, we discussed applications in data flow analysis and program optimization. Whereas the first application could still be treated by the known fixpoint theorems, the second application requires the generalization developed in this paper. Our new theorem is the key for proving the optimality of the partial dead code elimination algorithm of \([12]\), which is implemented in the new release of the Sun SPARCompiler language systems. Moreover, as our theorem only requires delay-monotonicity, a property being weaker than monotonicity, algorithm designers gain greater flexibility in the construction process than in the classical setup.
6 References


Appendix

Proof of the New Fixpoint Theorem 2.3

Proof: The wellfoundedness of ⊑, the increasingness of the functions of \( \mathcal{F} \), and the fairness of the strategy directly imply that \( \bigcup f_s(\bot) \) is a fixpoint and that it is reached in a finite number of iteration steps. Thus, we are left with showing that \( \bigcup f_s(\bot) \) is a lower bound for every common fixpoint of \( \mathcal{F} \). Let \( \delta \) be an arbitrary fixpoint of \( \mathcal{F} \), i.e. \( f_s(\delta) = \delta \) for all \( f_s \in \mathcal{F} \) and, as a consequence, \( f_s(\delta) = \delta \) for all \( s \in \mathbb{N}^* \). For \( (d_i)_{i \in \mathbb{N}} = f_s(\bot) \) we show \( d_i \subseteq \delta \) for all \( i \in \mathbb{N} \) by induction on \( i \). For \( i = 0 \), we have \( d_0 = \bot \subseteq \delta \). The induction step, \( d_{i+1} = f_s(i)(d_i) \subseteq f_s(\delta) = \delta \) for some \( s \in \mathbb{N}^* \), then follows from the induction hypothesis \( d_i \subseteq \delta \) and delay-monotonicity. Hence \( d_i \subseteq \delta \) for all \( i \) and so \( \bigcup f_s(\bot) \subseteq \delta \) by definition of \( \sqcup \). □

Proof of Lemma 3.1

Proof: Let \( \gamma \) be an arbitrary set strategy on \( \{1, \ldots, n\} \), and \( (d_i)_{i \in \mathbb{N}} \) be its induced chaotic iteration starting from \( \bot = \gamma(\bot, \ldots, \bot) \). We have to show

\[ \forall i \in \mathbb{N}. d_i \subseteq d_{i+1} \, . \]

The proof is by induction on \( i \). Let \( J = \gamma(i) \). By definition of \( f_J \) the property

\[ \forall k \in \{1, \ldots, n\} \setminus J \forall a \in D. f_J(d)^k = d^k \]

holds. Therefore, it suffices to show \( d_i^k \subseteq f_J(d)^k \) for all \( k \in J \). The case \( d_i^k = \bot \) 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 \( j', j, i, \bot \), we obtain \( d_j \subseteq d_{j+1} \), from which \( d_j \subseteq d_i \) follows by transitivity of \( \subseteq \). By monotonicity of \( f_{\gamma(j)} \), we have \( d_j^k = f_{\gamma(j)}(d_j)^k \subseteq \bot \) for all \( j \), \( k \), \( i \), \( f_J(d_i) = f_J(d_{i+1}) \), and the proof is done. □

Proof of Lemma 4.9

For \( f \in \mathcal{E}_\alpha \), let \( \text{elim}_\alpha(f, G') \) denote the set of \( \alpha \)-occurrences in \( G' \) that are eliminated by \( f \).

Proof: The first part of Lemma 4.9 is proven by investigating two cases:

(1a) \( g, f \in \mathcal{E}_\alpha \), (1b) \( g \in \mathcal{E}_\alpha, f \in \mathcal{E}_\beta \).
In case (1a) the maximality of $f$ guarantees $\text{elim}_\alpha(g, G_1) \subseteq \text{elim}_\alpha(f, G_1)$. We obtain that all $\alpha$-occurrences in $\text{elim}_\alpha(f, G_1) \setminus \text{elim}_\alpha(g, G_1)$ are dead in $G_2$ by Lemma 4.8(1). Hence, there is a transformation in $\mathcal{E}$ which eliminates all $\alpha$-occurrences in $\text{elim}_\alpha(f, G_1) \setminus \text{elim}_\alpha(g, G_1)$ in $G_2$. Choosing this transformation as $f'$, and an arbitrary function of $T^{max}$ leaving $G_3$ invariant as $g'$, we get

$$G_2 \vdash f'_{s_x} G_4$$

and, therefore, as desired

$$G_3 \vdash f'_{s_x} G_3 = G_4.$$ 

In case (1b) Lemma 4.8(1) yields that $\text{elim}_\alpha(g, G_1)$ and $\text{elim}_\alpha(f, G_1)$ are subsets of the sets of dead $\alpha$- and $\beta$-occurrences in $G_3$ and $G_2$, respectively. Hence, there are transformations in $\mathcal{E}$ which eliminate all $\alpha$-occurrences of $\text{elim}_\alpha(g, G_1)$ in $G_3$ and all $\beta$-occurrences of $\text{elim}_\beta(f, G_1)$ in $G_2$. Choosing these transformations as $g'$ and $f'$, respectively, we obtain as desired

$$G_2 \vdash f'_{s_x} G_4$$

and

$$G_3 \vdash f'_{s_x} G_3 = G_4.$$ 

Similarly to the proof of the first part of Lemma 4.9, two cases have also to be considered in the proof of its second part: (2a) $g, f \in S_a$, (2b) $g \in S_b$, $f \in S_\beta$.

In case (2a) the Dominance Lemma 4.5 yields the existence of an admissible assignment sinking $f' \in S$, which directly transforms $G_2$ into $G_3$. Thus, by choosing an arbitrary function of $T^{max}$ leaving $G_3$ invariant as $g'$, we succeed in this case.

In order to prove case (2b) consider the program $G_3'$, which results from $G_2$ by reinserting all $\alpha$-occurrences that have been moved by $g$. Let $G_4'$ be the program which results from the maximal $\beta$-sinking to $G_2$, i.e. $G_2 \vdash f'_{s_x} G_4$, and let $G_4$ result from $G_4'$ by eliminating the reinserted $\alpha$-occurrences. Obviously, there is a transformation $f' \in S_\beta$, which directly transforms $G_2$ into $G_4$. The admissibility of $g$ implies that $G_2$ and $G_4$ are identical except for $\beta$-occurrences. Thus, interchanging the roles of $\alpha$ and $\beta$ and applying the same construction to $G_3$, we get the existence of a transformation $g' \in S_a$, which transforms $G_3$ into $G_4$. This completes the proof of case (2b).

The remaining two parts of Lemma 4.9 can now be proven along the proof lines of part (2) by additionally applying Lemma 4.8(1).

Proof of the Main Lemma 4.10

Proof: Let $\alpha, \beta \in AP$ be different assignment patterns. Then the Main Lemma 4.10 is proven by investigating the following cases:
Applying the Dominance Lemma 4.5 we can assume without loss of generality that $f$ is maximal, i.e. $f \in T^{\text{max}}$. The first four cases are then immediate consequences of the corresponding parts of Lemma 4.9. Thus, we are left with the cases (5) and (6), which both can be proven in the same fashion. Thus, we only present the proof of case (5).

In the situation of case (5) let $G'_2$ be the program, which results from $G_2$ by reinserting a labelled version of all $\alpha$-occurrences that have been eliminated by $g$. Due to the labelling the reinserted $\alpha$-occurrences can be distinguished from the remaining ones. In $G'_2$, we assume that only unlabelled $\alpha$-occurrences can be subject to assignment sinkings; however, all $\alpha$-occurrences, i.e. labelled or not, are considered to block the sinking of $\alpha$-occurrences, i.e. no $\alpha$-occurrence can sink across a labelled or unlabelled $\alpha$-occurrence in $G'_2$. Now we choose the uniquely determined maximal $\alpha$-sinking and $\alpha$-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 $\alpha$-occurrences in $G'_4$ we obtain the program $G_4$. Of course, $f_1$ and $f_2$ have corresponding functions in $T$ 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 $\alpha$-elimination transforms $G_3$ into $G_4$ as well. The point here is that due to the reinsertion of $\alpha$-occurrences eliminated by $g$, $G'_2$ has precisely the same ‘$\alpha$-blockades’ as $G_1$. Hence, on join-free paths, i.e. on paths where no node has more than one predecessor, (unlabelled) $\alpha$-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 $\alpha$-occurrence can be blocked in a predecessor $n$ of a join-node $j$, because there is a brother $m$ of $n$ to which no $\alpha$-occurrence is sinkable; in $G_1$, however, the same $\alpha$-occurrence can successfully be sunk into $j$, because some of the $\alpha$-occurrences eliminated by $g$ in $G_1$ are sinkable to $m$. It is worth noting that the $\alpha$-occurrence is dead in $j$ and never becomes live again. Hence, it is eliminated by the subsequent application of $f_2$. This, however, holds for the $\alpha$-occurrence blocked in the predecessor $n$ of $j$ as well, since it is dead according to Lemma 4.8(2). Combining these results we obtain as desired that maximal $\alpha$-elimination transforms $G_3$ into $G_4$.

\[2\] The set of brothers of a node $n$ is given by $\bigcup \{ \text{pred}(m) \mid m \in \text{succ}(n) \}$.

}\[\square\]
**Proof of Theorem 4.12**

**Proof:** Since (2) holds trivially, we only prove (1). The first implication, "\( \Rightarrow \)", is a simple consequence of \( T_{\text{max}} \subseteq T \). The second implication, "\( \Leftarrow \)", is proven by showing the contrapositive. Without loss of generality, we can assume \( g \in T_{a} \setminus T_{a}^{\text{max}} \) and \( G' \vdash_{p} G'' \) with \( G'' \neq G' \). Now, let \( f \in T_{a}^{\text{max}} \) be the uniquely determined function \( f \) of \( T_{a}^{\text{max}} \) corresponding to \( g \). Then the Dominance Lemma 4.5 yields as desired that the program resulting from the application of \( f \) to \( G' \) is different from \( G' \). \( \square \)