Distrib Comput (1993) 7: 43–53

DISTRIBUTED COMPUTING

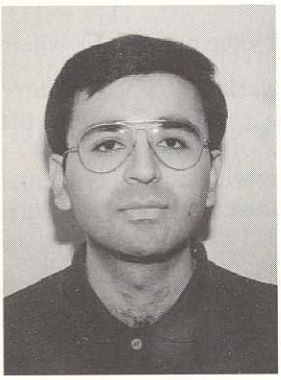


# Convergence of iteration systems

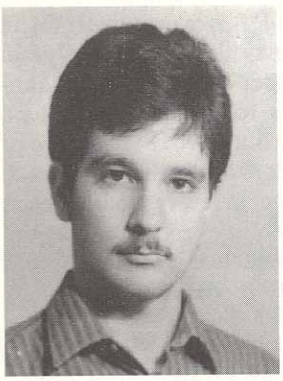
**Anish Arora**[1,2]*, **Paul Attie**[1,2], **Michael Evangelist**[1], **Mohamed Gouda**[1,2]

[1] Microelectronics and Computer Technology Corporation, 3500 West Balcones Center Drive, Austin, Texas, USA
[2] Department of Computer Sciences, The University of Texas at Austin, USA

Received February 1990 / Accepted April 1991

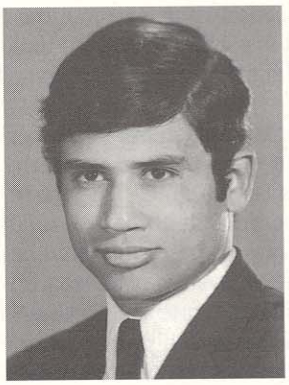


**Anish Arora** is currently completing his Ph.D. degree at the University of Texas at Austin, and has been working in the Software Technology Program at MCC since December 1988. Anish received a B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, New Delhi in 1986, and an M.S. degree in Computer Sciences from the University of Texas at Austin in 1988. His research interests include fault-tolerance, distributed computing, program correctness, and semantics of concurrency.



**Paul Attie** is currently completing his Ph.D. degree of the University of Texas at Austin, and has been a member of technical staff in the Software Technology Program of the MCC since January 1990. Paul received a B.A. degree in Engineering Science from Oxford University in 1981, and an M.Sc. degree from the University of London in 1982. His research interests include distributed computing, temporal logic, and algebraic process theory.

**Michael Evangelist** received his Ph.D. in 1978 from Northwestern, where he did research in formal language theory, graph theory, logic, and computational complexity theory. He taught computer science at Colgate University and, in 1982, became a member of technical staff at Bell Labs and did research in software engineering. Three years later, he joined the Software Technology Program at MCC, where he spent five years working on theoretical and practical issues in the design of distributed systems. Evangelist now heads the Software Engineering Laboratory in the Chicago research center of Andersen Consulting.



**Mohamed G. Gouda** currently works on and teaches the stabilization of computing systems at the University of Texas at Austin. He designs "cute" communication protocols as a hobby, and proves them correct for fun.

**Summary.** An iteration system is a set of assignment statements whose computation proceeds in steps: at each step, an arbitrary subset of the statements is executed in parallel. The set of statements thus executed may differ at each step; however, it is required that each statement is executed infinitely often along the computation. The convergence of such systems (to a fixed point) is typically verified by showing that the value of a given variant function is decreased by each step that causes a state change. Such a proof requires an exponential number of cases (in the number of assignment statements) to be considered. In this paper, we present alternative methods for verifying the convergence of iteration systems. In most of these methods, upto a linear number of cases need to be considered.

**Key words:** Convergence – Stabilization – Iteration systems – Dependency graph

*Correspondence to:* A. Arora at his present address

* *Present address:* Department of Computer Science, The Ohio State University, Columbus OH 43210, USA

## 1 Introduction

Iteration systems are a useful abstraction for computational, physical and biological systems that involve "truly concurrent" events. In computing science, special cases of iteration systems are frequent; for instance, self-stabilizing programs, term-rewriting systems, bottom-up logic program interpreters, compiler attribute-grammars, constraint-satisfaction solvers, and array processors can be represented as iteration systems. This wide applicability derives from the simplicity and generality of the formalism.

Informally, an iteration system is defined by a finite set of variables, $V$. Associated with each variable is a function called its *update function*. The computation of the system proceeds in steps. At each step, the variables in an arbitrary subset of $V$ are updated by assigning each variable the value obtained from applying its update function to the current system state. The set of variables thus updated may differ at each step; however, it is required that each variable in $V$ be updated infinitely often.

In allowing an arbitrary subset of variables to be updated at each step, our formalism admits a large number of widely varying computations. These range from the sequential computations in which exactly one variable is updated at every step to the parallel computation in which each variable is updated at every step. By comparison, traditional semantics admit computations of lesser variety. For example, interleaving requires one enabled event to be executed at each step (see, for instance, the work on CSP [12], UNITY [5] and I/O Automata [15]), whereas maximal parallelism requires that all enabled events are executed at every step (see, for instance, the work on systolic arrays [14] and cellular automata [17]).

The variety in the computations admitted by our formalism complicates the task of verifying iteration systems. This fact and the fact that iteration systems have wide applicability suggest that it is useful to develop techniques which simplify the verification of properties of iteration systems.

A property of interest in iteration systems is convergence. This property is useful in studying the self-stabilization of distributed programs (cf. [2, 3, 6, 7, 9]), convergence of chaotic relaxation methods in numerical analysis (cf. [4, 16]), and self-organization in neural networks (cf. [1, 13]). Informally, an iteration system is called convergent if on starting in an arbitrary state the system is guaranteed to reach a fixed point; that is, a state in which no update can cause a state change. The standard method for verifying that an iteration system is convergent is to exhibit a variant function (cf. [8]) whose value is bounded from below and is decreased by at least some constant at each step that causes a state change. Since any subset of the variables can be updated in a step, the number of cases that need to be considered are $2^n - 1$, where $n$ is the number of variables in the system. In this paper, we discuss new methods for verifying the convergence of iteration systems. Nearly all these methods require upto $n$ cases to be considered.

The rest of this paper is organized as follows. In Sect. 2, we formally define iteration systems and their dependency graphs. (The dependency graph of an iteration system captures the "depends on" relation between the variables in the system.) In Sect. 3, we identify two classes of iteration systems, namely those that have acyclic dependency graphs and those that have self-looping dependency graphs. We then present a theorem that establishes efficient proof obligations for verifying the convergence of these two classes. This theorem is extended to general iteration systems in Sect. 4. In Sect. 5, we redefine iteration systems to allow nondeterministic updates of variables and we show that, with minor modifications, our previous results continue to hold in nondeterministic iteration systems. In Sect. 6, we exhibit efficient proof obligations for verifying convergence that are based on a finer grain notion of dependency graphs. Concluding remarks are in Sect. 7.

## 2 Iteration systems

An *iteration system, I,* is defined by the pair $(V, F)$, where

- $V$ is a finite, nonempty set of variables. Each variable $v$ in $V$ has a predefined domain $Q_v$. Let $Q$ denote the cartesian product of the domains of all variables in $V$.
- $F$ is a set of "update" functions with exactly one function $f_v$ associated with each variable $v$ in $V$, where $f_v$ is a mapping from $Q$ to $Q_v$.

A *state* $q$ of $I$ is an element of $Q$. We adopt the notation $q_v$ to denote the value of variable $v$ in state $q$. A state $q$ is called a *fixed point* of $I$ iff for each variable $v$ in $V$, $f_v(q) = q_v$.

A *step* of $I$ is defined to be a subset of $V$; informally, a step identifies those variables that are updated when the step is executed. A *round* of $I$ is a minimal, finite sequence of steps with the property that each variable in $V$ is an element of at least one step in the round. Minimality of a sequence denoting a round implies that no prefix of the sequence is itself a round. A *computation* of $I$ is an infinite sequence of rounds. Notice that since each variable is updated at least once in every round, each variable is updated infinitely often in every computation.

The *application* of a finite sequence of steps $S$ to a state $q$, denoted $S \circ q$, is the state $q'$ defined inductively as follows:

- if $S$ is the empty, sequence, then $q' = q$
- if $S$ is a single step, then for every variable $v$ in $V$,

$$q'_v = \begin{cases} f_v(q), & \text{if } v \in S \\ q_v, & \text{otherwise} \end{cases}$$

- if $S$ is the sequence $S''$ appended to sequence $S'$ (denoted $S = S'; S''$), then $q' = S'' \circ (S' \circ q)$.

A computation $C$ is called *convergent* iff for every state $q$, there exists a finite prefix, $S$, of $C$ such that $S \circ q$ is a fixed point of $I$. An iteration system is called *convergent* iff all of its computations are convergent.

As shown in the following examples of iteration systems, it is convenient to represent an iteration system by a set of assignment statements, one for each variable.

Each statement has the form ⟨*variable*⟩ := ⟨*corresponding update function*⟩. We will later prove each of these iteration systems to be convergent.

*Example 1* (Greatest Common Divisor). Let $x$, $y$ and $z$ be variables that range over the natural numbers. Then, the three assignment statements

$x := \textbf{if } x > y \textbf{ then } x - y \textbf{ else } x$

$y := \textbf{if } x < y \textbf{ then } y - x \textbf{ else } y$

$z := \textbf{if } x = y \textbf{ then } 0 \textbf{ else } z + 1$

define a convergent iteration system. At fixed point, the value of $x$ is the greatest common divisor of the initial values of $x$ and $y$, and $y = x$ and $z = 0$.

*Example 2* (Minimum of a Bag). Let $x$ be an integer array of size $n$. Then, the following assignment statements:

$x[1] := x[1]$

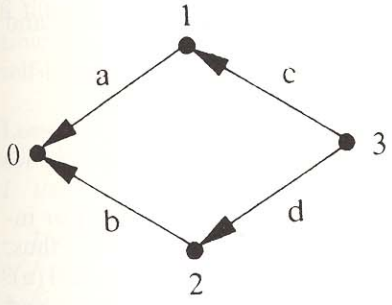
$x[2] := \min(x[2], x[1])$

$\vdots$

$x[n] := \min(x[n], x[n-1])$

define a convergent iteration system. At fixed point, the value of each $x[i]$ is the minimum of the initial values in the sub-array $x[1], x[2], \ldots, x[i]$.

*Example 3* (Shortest Path). Consider the directed graph




It has four nodes 0–3, and four edges. Each edge is labeled with a non-negative integer constant denoting its length. Associated with each node $i$ is a variable $v[i]$, of type *record*, that has an integer component *first*$[i]$ and a component *second*$[i]$ that ranges over the nodes that are adjacent from $i$. The assignment statements

$v[0] := (0, 0)$

$v[1] := (a, 0)$

$v[2] := (b, 0)$

$v[3] := \textbf{if } first[1] + c \leq first[2] + d$
$\textbf{then } (first[1] + c,\ 1) \textbf{ else } (first[2] + d,\ 2)$

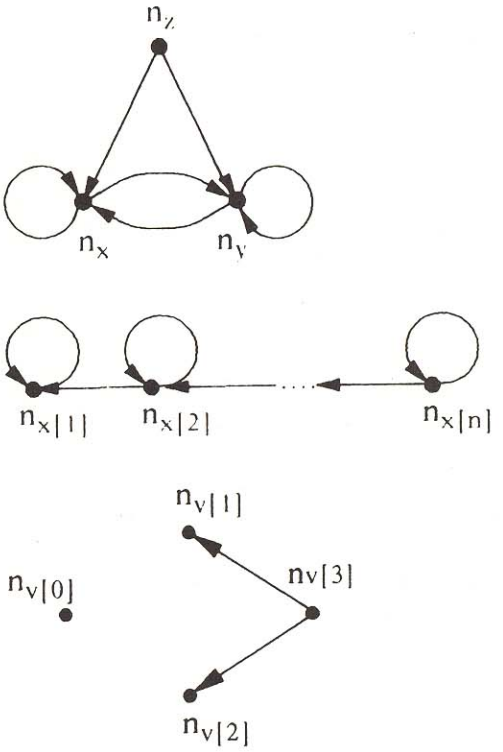
define a convergent iteration system. At fixed point, each *first*$[i]$ is the length of the shortest path from node $i$ to node 0, and *second*$[i]$ is the nearest neighbor to node $i$ along this path. Extending the above system for an arbitrary directed graph is straightforward.

The objective of this paper is to identify proof obligations that are sufficient to establish the convergence of iteration systems. Towards this end, the following two definitions will prove useful shortly.

Let $v$ and $w$ be variables in $V$. We say $v$ *depends on* $w$ iff there exist two states $q$ and $q'$ of $I$ such that $q$ and $q'$ differ only in their value of $w$ and $f_v(q) \neq f_v(q')$. Informally, $v$ *depends on* $w$ iff a change in the value of $w$ can cause a change in the value assigned to $v$ by its update function $f_v$.

The *dependency graph* of $I$ is a directed graph whose nodes correspond to the variables in $V$ and whose directed edges correspond to the *depends on* relation; that is, the set of nodes of the dependency graph is $\{n_v \mid v \in V\}$ and its set of directed edges is $\{(n_v, n_w) \mid v \in V, w \in V,$ and $v$ *depends on* $w\}$.

The dependency graphs of the iteration systems in Examples 1, 2 and 3 are as follows:




Henceforth, we shall use 'variable' and 'node' interchangeably when referring to the dependency graph of an iteration system.

## 3 Convergence of non-cyclic systems

An iteration system is called *acyclic* iff its dependency graph is acyclic. It is called *self-looping* iff its dependency graph has one or more cycles, and all its cycles are self-loops.

In this section we state a fundamental theorem concerning the convergence of acyclic and self-looping iteration systems. The implications of this theorem are discussed subsequently.

**Theorem 1.** *If an iteration system is*

(a) *acyclic, then it is convergent, and if it is*
(b) *self-looping and has one convergent computation, then it is convergent.*

To prove the theorem we need to introduce the following new concepts: "*rank*", "*stable*", "$\#steps$", "$\uparrow$", and "$\downarrow$".

Consider an iteration system $I$. For an acyclic or self-looping $I$, define *rank* to be the function that assigns to each variable $v$ in $V$ a positive integer as follows:

$$rank(v) = 1 + \max\{rank(w) \mid (w \in V \wedge v \neq w \wedge v \text{ depends on } w)\}$$

By convention, the value of "max" applied to the empty set is 0; thus, the rank of a variable that does not depend on any other variable is 1. Notice that this definition is recursive and requires, in order to be well-defined, that the dependency graph of $I$ has no cycle of length two or greater. Therefore, this definition applies only to acyclic and self-looping systems.

A variable $v$ in $V$ is *stable* at state $q$ iff for every finite sequence of steps, $S$, the value of $v$ in $q$ is the same as its value in state $S \circ q$.

Let $\#steps(C, q, k)$ denote the partial function that returns the number of steps in the minimal prefix $S$ of computation $C$ such that every variable of rank at most $k$ is stable at the state $S \circ q$. The value of $\#steps(C, q, k)$ is undefined when no such prefix exists.

Let $k$ be any positive integer, then $C \uparrow k$ denotes the unique prefix of computation $C$ that consists of exactly $k$ rounds, and $C \downarrow k$ denotes the computation that results after removing the prefix $C \uparrow k$ from computation $C$.

*Proof of part (a).* The proof is by a straightforward induction on the rank of variables. We argue that after the application of the first $k$ rounds ($k \leq |V|$) of an arbitrary computation $C$ to an arbitrary state $q$, all variables with rank at most $k$ are stable at the resulting state $(C \uparrow k) \circ q$. Since the rank of any variable in $V$ is at most $|V|$, it follows that after $|V|$ rounds the system is at a fixed point. For the base case, note that the update functions of variables with rank $= 1$ are constant functions.

*Proof of part (b).* We need to prove that if some computation $C'$ is convergent then for an arbitrary computation $C$ and an arbitrary state $q$ there exists a positive integer $i$ such that all variables in $V$ are stable at state $(C \uparrow i) \circ q$; that is, $(C \uparrow i) \circ q$ is a fixed point.

The proof proceeds by induction on the rank $k$ of variables. Let the induction hypothesis be:

$$\exists i \, \forall v \; (rank(v) \leq k - 1 \Rightarrow v \text{ is stable at } (C \uparrow i) \circ q)$$

Note that as $C'$ is convergent, $\#steps(C', q, k)$ is well defined for every rank $k$.

*Base Case.* $k = 1$.
A variable whose rank is 1 depends on no other variable. Therefore, in computation $C$, the sequence of values that a rank 1 variable takes starting from its value in state $q$, is a function only of the number of updates to it. Hence, it is necessarily stable after $\#steps(C', q, 1)$ updates to it in any computation starting in state $q$. Since each round contains at least one update to every variable, each variable whose rank is 1 is stable at $(C \uparrow \#steps(C', q, 1)) \circ q$.

Induction Step. $k > 1$.
Let $q' = (C \uparrow i) \circ q$, where $i$ is the least integer that satisfies the induction hypothesis. Hence,

$$\forall v \; (rank(v) \leq k - 1 \Rightarrow v \text{ is stable at } q') \tag{1}$$

Applying $C'$ to $q'$, we know that every variable with rank $\leq k$ will be stable within $\#steps(C', q', k)$ updates to that variable

$$\forall v \; (rank(v) \leq k \Rightarrow v \text{ is stable at } (C' \uparrow \#steps(C', q', k)) \circ q')$$

From (1), all variables of rank $\leq k - 1$ are stable at state $q'$. Thus, all computations starting in state $q'$ will produce the same sequence of values (as a function of the number of updates) for each variable of rank $k$. As each round contains at least one update per variable, every variable of rank $k$ will be stable after $\#steps(C', q', k)$ rounds of any computation. In particular, for the computation $C \downarrow i$ (that is, $C$ with $(C \uparrow i)$ removed),

$$\forall v \; (rank(v) \leq k \Rightarrow v \text{ is stable at } ((C \downarrow i) \uparrow \#\text{steps}(C', q', k)) \circ q')$$

Let $j = \#steps(C', q', k)$. Now $((C \downarrow i) \uparrow j) \circ q' = ((C \downarrow i) \uparrow j) \circ ((C \uparrow i) \circ q) = (C \uparrow i); ((C \downarrow i) \uparrow j) \circ q = (C \uparrow (i + j)) \circ q$ and so,

$$\forall v \; (rank(v) \leq k \Rightarrow v \text{ is stable at } (C \uparrow (i + j)) \circ q)$$

and the inductive hypothesis is established for rank $k$. □

The examples in Sect. 2 illustrate Theorem 1. For instance, the iteration system of Example 3 is acyclic; thus, each of its computations is convergent by Theorem 1(a). The iteration system of Example 2 is self-looping, and any computation where the first step updates $x[1]$, the second updates $x[2]$, and so on is convergent; thus, each computation of the system is convergent by Theorem 1(b).

As mentioned in the introduction, verifying the convergence of an iteration system is generally accomplished by exhibiting a variant function whose value is bounded from below and is decreased by each step that causes a state change. Such a proof requires $2^n - 1$ cases to be considered, where $n$ is the number of variables in the system. In contrast, Theorem 1 shows that verifying the convergence of acyclic systems requires no such case analysis.

The theorem also states that the convergence of all computations of a self-looping iteration system can be established from the convergence of a computation of choice. One possibility is to choose this computation to be the one in which each variable is updated at every step. The convergence of this computation can then be proved by the variant function method which, in this instance, needs only one case to be considered. Another possibility

is to choose computations in which exactly one variable is updated at each step; in this instance, the variant function method requires $n$ cases.

Theorem 1 cannot be made to apply to all iteration systems. Consider, for example, the iteration system defined by the two assignment statements

$x := y$

$y := x$.

Although this system has many computations that are convergent (for example, all computations where exactly one variable is updated at the first step), it also has a computation that is not convergent (for example, the computation where both $x$ and $y$ are updated at each step). Thus, unlike Theorem 1, one cannot establish that this system is convergent by exhibiting one convergent computation. Similar examples have been presented in [6] and [16].

In fact, it is straightforward to show that neither conclusion of Theorem 1 applies to any class of iteration systems that properly contains the acyclic and self-looping systems. The proof for this follows from a construction that exhibits, for each directed graph $G$ that has a cycle of two or more nodes, an iteration system $I$ such that the dependency graph of $I$ is $G$, and $I$ has both convergent and non-convergent computations.

The following lemma states that for any cyclic iteration system $I$ there is a self-looping system which captures a subset of the computations of $I$ and, thereby, is a possible implementation for $I$. This shows that the class of self-looping systems is rich.

**Lemma 1.** *For each iteration system $I$ that is neither acyclic nor self-looping, there exists a self-looping iteration system $I'$ that satisfies the following two conditions:*

*– There is a one-to-one correspondence between the states of $I$ and the states of $I'$.*
*– Every step of $I'$ is equivalent to some step of $I$ under state correspondence. That is, if $q \in I$ and $q' \in I'$ denote an arbitrary pair of corresponding states then, for an arbitrary step $S'$ of $I'$, there exists a step $S$ of $I$ such that $S \circ q$ corresponds to $S' \circ q'$.*

*Proof.* We construct $I'$ from $I$ by the following operation on each maximally strongly connected component $M$, in the dependency graph of $I$, that has two or more nodes. Replace all variables in $M$ with a new variable of type record; the fields of this record correspond, in a one-to-one manner, to the replaced variables. Further, define the update function of the new variable such that each field is updated using the update function of its corresponding variable in $M$. The net effect of this construction is that each $M$ in the dependency graph of $I$ collapses into a single node with a self-loop in the dependency graph of $I'$, thereby yielding a self-looping system. □

System $I'$ in Lemma 1 is self-looping; hence, its convergence can be determined by Theorem 1(b). (The convergence of $I'$, however, does not necessarily imply the convergence of the original system $I$.) Consider, for instance, the iteration system in Example 1. This system is neither acyclic nor self-looping because its dependency graph has a maximally strongly connected component consisting of variables $x$ and $y$. By replacing these two variables by one variable with two components, also called $x$ and $y$ for convenience, we obtain the following implementation of the system:

$(x, y) :=$ (**if** $x > y$ **then** $x - y$ **else** $x$, **if** $x < y$ **then** $y - x$ **else** $y$)

$z :=$ **if** $x = y$ **then** 0 **else** $z + 1$.

As this system is self-looping, its convergence can be established by Theorem 1(b).

## 4 Convergence of cyclic iteration systems

In this section, we generalize our analysis for the convergence of acyclic and self-looping systems to the convergence of general iteration systems. Our starting point is to note the basic characteristic of an iteration system that is neither acyclic nor self-looping, namely the existence of at least one maximally strongly connected component in its dependency graph that consists of two or more nodes. For convenience, we call a maximally strongly connected component that has two or more nodes a *district*.

Let $D$ be a district in the dependency graph of an iteration system, $I$. The *iteration system associated with* $D$ is the iteration system $(V_D, F_D)$ that satisfies the following two conditions:

–The set of variables, $V_D$, is the set of all variables in $D$ together with each variable that some variable in $D$ depends on.
– The set of update functions, $F_D$, is defined as follows. The update function for a variable in $D$ is the same as its update function in $I$, whereas the update function for a variable in $V_D$ but not in $D$ is the identity function for that variable.

For instance, the iteration system in Example 1 has one district whose associated iteration system can be defined by the two assignment statements

$x :=$ **if** $x > y$ **then** $x - y$ **else** $x$

$y :=$ **if** $x < y$ **then** $y - x$ **else** $y$.

An iteration system is called *district-convergent* iff the iteration system associated with each district in the dependency graph of the system is convergent. Since acyclic and self-looping systems do not have any districts in their dependency graphs, they are trivially district-convergent.

An iteration system is called *0-cyclic* iff its dependency graph has no maximally strongly connected component that consists of a single node with a self-loop; otherwise, the iteration system is called *1-cyclic*. Note that each iteration system is either 0-cyclic or 1-cyclic; in particular,

acyclic systems are 0-cyclic whereas self-looping ones are 1-cyclic.

The following theorem generalizes Theorem 1.

**Theorem 2.** *If a district-convergent iteration system is*

(a) 0-*cyclic, it is convergent, and if it is*
(b) 1-*cyclic and has one convergent computation, then it is convergent.*

*Proof.* We extend the definition of *rank* in the proof of Theorem 1 to an arbitrary iteration system, $I$, as follows. Consider the "condensation" of its dependency graph (i.e., collapse each district into a single node; see [11]). It is straightforward to see that each cycle in the condensation is a self-loop. Assign ranks to the nodes in the condensation using the previous definition of rank. Now, the *rank* of a variable $v$ in $I$ is defined to be the rank of its corresponding node in the condensation.

The proof proceeds by induction on the rank $k$ of variables, and is similar to the proof of Theorem 1. The induction hypothesis is: for an arbitrary computation $C$ and an arbitrary state $q$ there exists a prefix $S$ of $C$ such that all variables of rank less than $k$ will be stable at $S \circ q$. For both the base case and the induction step, the following arguments suffice:

- if variable $v$ does not depend on any variable or depends only on variables of lower rank (which are stable as $S \circ q$), then $v$ is clearly stable after the first round that follows $S$ in $C$.
- if $v$ depends on itself but on no other variable of the same rank, then the counting argument in the proof of Theorem 1(b) ensures that $v$ will eventually be stable.
- finally, it may be the case that $v$ is in some district $D$. We argue that, once all the variables of lower rank on which $v$ depends are stable, the convergence of the iteration system associated with $D$ guarantees that $v$ will eventually be stable. □

In the remainder of this section we identify two proof obligations which are sufficient to establish the convergence of an iteration system that is associated with a district. These obligations consists of exhibiting either a variant function for each node in a selected set of nodes in the district (Lemma 2), or a single variant function for the whole district (Lemma 3).

The intuition underlying Lemma 2 is to "break" each cycle in the district by ensuring that some distinguished variable on the cycle will eventually reach a stable value. This is achieved by exhibiting a variant function for the distinguished variable whose value decreases each time the value of the variable is changed by an update. Once every distinguished variable in the district becomes stable (i.e. has a fixed value), the iteration system associated with the district starts to behave like an acyclic system and, so, eventually reaches a fixed point.

A more general approach to solving the same problem is to exhibit a variant function for all the variables in the district. The value of this function is decreased by each step that causes a state change. See Lemma 3 below.

In what follows, let

$D$ be a district in some iteration system $I$,

$(V_D, F_D)$ be the iteration system associated with $D$,

$Q_D$ be the set of states of $(V_D, F_D)$,

$f_v$ be the update function of a variable $v$ in $(V_D, F_D)$,

$Var_{D'}$ be the set of variables in $D'$, a subgraph of $D$, and

$N$ be an arbitrary set that is well-founded under some relation $<$.

**Lemma 2 (Local Variant).** *If each directed cycle in D has a variable v and a variant function* $\# : Q_v \longrightarrow N$ *such that for each state q in* $Q_D$,

$$\#(f_v(q)) < \#(q_v) \ \vee \ (f_v(q) = q_v)$$

*then the iteration system* $(V_D, F_D)$ *is convergent.*

**Lemma 3 (Global Variant).** *If there is a variant function* $\# : Q_D \longrightarrow N$ *such that for each state q in* $Q_D$, *and for each strongly connected component D′ in D,*

$$\#(Var_{D'} \circ q) < \#(q) \ \vee \ (Var_{D'} \circ q = q)$$

*then the iteration system* $(V_D, F_D)$ *is convergent.*

To prove these lemmas, we augment the set of concepts introduced in the previous proofs by the following: For a variable $v$ in $V$ and subset $W \subseteq V$, define *allpaths*$(v, W)$ to be the subgraph in the dependency graph of $I$ containing (exactly) those paths that begin at $v$, do not contain any variable in $W$ as an intermediate node and end at some variable in $W$.

A variable $v$ is *unaffected* by variable $w$ in state $q$ iff for an arbitrary state $q'$ that differs from $q$ only in its value of $w$, and an arbitrary finite sequence of steps $S$,

$$(S \circ q)_v = (S \circ q')_v .$$

Intuitively, this implies that the value assigned to $v$ in the application of any sequence to the state $q$ is independent of the value of $w$.

*Proof of Lemma 2.* The proof obligation is to show that for an arbitrary computation, $C$, of $(V_D, F_D)$ and an arbitrary state, $q \in Q_D$, there exists a positive integer $i$ such that $(C \uparrow i) \circ q$ is a fixed point of $(V_D, F_D)$.

By the antecedent of the lemma, we can distinguish on each cycle in $D$ some variable that satisfies the property stated in the lemma. Let $W$ be the set of variables thus distinguished, and let $U$ abbreviate the set of variables in $V_D$ but not in $D$.

We show that for an arbitrary variable $v$ in $D$ there exists a positive integer $j$ such that $v$ is stable at $(C \uparrow j) \circ q$. The maximum $j$ for the variables in $D$ is then the appropriate positive integer $i$ for which $(C \uparrow i) \circ q$ is a fixed point. There are 3 cases to be considered.

- $v \in U$: $v$ is already stable at $q$ as it is updated by the identity function; that is, $j = 0$.

- $v \in W$; by the property stated in the lemma and the fact that $N$ is well-founded under $<$, we are guaranteed that $v$ will be stable after a finite number of steps in any computation. Let $k$ be the least positive integer such that all variables in $W$ are stable at $q' = ((C \uparrow k) \circ q)$.
- $v$ is in $V_D$ but not in $W$: let $x$ denote any variable not in *allpaths* $(v, W \cup U)$. We claim that $v$ is *unaffected* by $x$ in $q'$. To prove this, we first note that the construction of *allpaths* $(v, W \cup U)$ ensures that every path from $v$ to $x$ must pass through some variable in $W \cup U$. Since all variables in $W \cup U$ are stable at $q'$, it follows that $v$ is *unaffected* by $x$ in $q'$.

Next, we show that *allpaths* $(v, W \cup U)$ is acyclic. The only variables in *allpaths* $(v, W \cup U)$ that are not in $D$ are in $U$, but these have, by construction, no successors in *allpaths* $(v, W \cup U)$. It follows that all cycles in *allpaths* $(v, W \cup U)$ must be contained in $D$ and, therefore, must pass through some distinguished node in $W$. However, this is impossible in *allpaths* $(v, W \cup U)$ as no distinguished node has a successor. Thus, *allpaths* $(v, W \cup U)$ is acyclic.

Hence, the value of $v$ is affected only by the variables in *allpaths* $(v, W \cup U)$, and since the latter is an acyclic graph with variables of rank 0 (that is, $W \cup U$) stable at state $q'$, we conclude from Theorem 1(a) that on the application of $l = rank(v)$ rounds to $q'$, $v$ will be stable; that is, $v$ is stable at $(C \uparrow j) \circ q$ where $j = k + l$. □

*Proof of Lemma 3.* To prove that the iteration system $(V_D, F_D)$ is convergent, we show that for an arbitrary step $W \subseteq V_D$ and an arbitrary state $q \in Q_D$,

$$\#(W \circ q) < \#(q) \vee (W \circ q = q).$$

Since $N$ is well-founded under $<$, there is no infinitely descending chain of values returned by $\#$ and, hence, after a finite number of steps the iteration system is guaranteed to be at a fixed point.

The proof is organized as follows: first, we show that $W \circ q$ is the same as the state that results from the application of a finite sequence of mutually disjoint steps to $q$, each step of which updates the variables in some strongly connected component of the dependency graph of $(V_D, F_D)$. Then, we argue that by the property stated in the lemma, it must be the case that $\#(W \circ q) < \#(q) \vee (W \circ q = q)$.

Consider the "subgraph induced by" $W$, $G_W$, in the dependency graph of $(V_D, F_D)$ (i.e., its maximal subgraph with node set $W$; see [11]). Take the condensation of $G_W$. As observed in the proof of Theorem 2, the *rank* of the variables in $W$ can be consistently computed via the condensation of $G_W$. Let $k$ be the maximum *rank* thus assigned.

Next, we make the observation that if two variables corresponding to different nodes in the condensation are updated simultaneously, then

- if they are of different rank, the resulting state is the same as the one obtained by updating the higher rank variable first, and then updating the other variable, and
- if they are of the same rank, the resulting state is the same as the one obtained by updating them in any sequential order.

To complete the proof, consider all the variables in $W$ of $rank = i$, $(1 \leq i \leq k)$. These variables can be uniquely partitioned into sets, each of which corresponds to some node in the condensation of $G_W$. Let $W_i$ be an arbitrary sequence of the sets in this partition such that each set appears exactly once. By the observation made in the previous paragraph $(W_k; W_{k-1}; \ldots; W_1) \circ q = W \circ q$. However, by the property stated in the lemma, we know that the application of a step containing exactly the variables in some strongly connected component can only lower the value returned by the variant function if there is a change of state. Hence, if $(W \circ q \neq q)$ then $\#(W \circ q) < \#(q)$. □

As an example, both Lemma 2 and Lemma 3 can be used to show that the iteration system associated with the district in Example 1 is convergent. In using Lemma 2, let the variant functions for both $x$ and $y$ be their respective identity functions. In using Lemma 3, let the global variant function be the sum of $x$ and $y$. In either case, the convergence of the entire system in Example 1 can now be established by Theorem 2.

As compared to the standard method for verifying convergence, which requires $2^n - 1$ cases to be considered (where $n$ is the number of variables in the system), Theorem 2 requires fewer cases. In particular, Theorem 2(a) shows that verifying the convergence of 0-cyclic district-convergent systems requires no case analysis, whereas Theorem 2(b) shows that verifying the convergence of 1-cyclic district-convergent systems requires exhibiting one computation of choice that is convergent. A saving in case analysis is also obtained in establishing district-convergence using Lemma 2: in the worst situation, one case is needed for each district variable $v$ showing that some variant function is decreased by executing the assignment statement of $v$ in an arbitrary state. Thus, upto a linear number of cases (in the number of district variables) are considered. In contrast, each strongly connected component of a district is considered in Lemma 3 and, hence, upto an exponential number of cases may be needed in establishing district-convergence.

## 5 Convergence of nondeterministic systems

So far, the definition of an iteration system associates exactly one (deterministic) update function with each variable. We now extend this definition to allow each variable to be updated by more than one update function. More specifically, we associate with each variable $v$ a finite, non-empty set of update functions $F_v$. At each step in which $v$ is updated, one of the functions in $F_v$ is chosen to update $v$. The only requirement we impose on the choice method is that each update function in $F_v$ is chosen infinitely often in every computation. (Note that this is possible because each variable is updated infinitely often in every computation.)

In the next example, we represent a nondeterministic iteration system by a set of assignment statements (with

choice), one for each variable. If $F_v = \{f, g, \ldots, h\}$ then the assignment statement that updates $v$ has the form:

$$v := f \mid g \mid \ldots \mid h$$

*Example 4* (Nondeterministic Shortest Path). We exhibit a nondeterministic iteration system for the directed graph in Example 3. The nondeterminism makes it possible to reduce the 'atomicity' of the update functions. In fact, every update function refers uniquely to one edge in the graph, as follows:

$v[0] := (0, 0)$

$v[1] := (a, 0)$

$v[2] := (b, 0)$

$v[3] :=$ **if** $first[1] + c \leq first[3] \vee second[3] = 1$
**then** $(first[1] + c, 1)$ **else** $v[3]$
$\mid$ **if** $first[2] + d \leq first[3] \vee second[3] = 2$
**then** $(first[2] + d, 1)$ **else** $v[3]$

This system is convergent to some fixed point, and when it is at a fixed point, each $first[i]$ is the length of the shortest path from node $i$ to node 0, and each $second[i]$ is the node immediately following node $i$ along this path.

We now redefine five concepts that were introduced earlier in order to accommodate the extension to nondeterminism.

- A state $q$ of an iteration system is a *fixed point* iff for each variable $v$ and each update function $f_v$ in $F_v$, $f_v(q) = q_v$.
- The *application* of a finite sequence of steps $S$ to a state $q$, denoted $S \circ q$, is the state $q'$ defined inductively as follows:
  - if $S$ is the empty sequence, then $q' = q$
  - if $S$ is a single step, then for every variable $v$ in $V$,

$$q'_v = \begin{cases} f_v(q), & \text{if } v \in S \text{ and } f_v \text{ in } F_v \text{ is chosen} \\ q_v, & \text{otherwise} \end{cases}$$

  - if $S = S'; S''$, then $q' = S'' \circ (S' \circ q)$.
- A computation is said to be *convergent* iff for each state $q$ and for each choice of update functions in the computation there exists a finite prefix $S$ of the computation such that $S \circ q$ is a fixed point.
- The *depends on* relation is redefined as follows: variable $v$ *depends on* variable $w$ iff there exist two states $q$ and $q'$ such that $q$ and $q'$ differ only in their value of $w$ and $f_v(q) \neq f_v(q')$ for some $f_v$ in $F_v$.
- We augment the notion of the *iteration system* $(V_D, F_D)$ *associated with* a district $D$ in the dependency graph of an iteration system $I$. With each variable $v$ in $V_D$ that is also in $D$, we now associate its set of update functions in $I$, i.e. $F_v$. The set of update functions for every variable in $V_D$ but not in $D$ is defined to be the set that contains only the identity function for that variable.

Next, we extend the previous results to nondeterministic iteration systems.

**Theorem 3.** *If a nondeterministic iteration system is*

(a) *acyclic and has a fixed point, then it is convergent, and if it is*
(b) *self-looping and has one convergent computation, then it is convergent.*

*Proof Sketch.* The assumption that a fixed point exists can be used to show that the induction argument for the deterministic case continues to hold. In particular, the assumption is needed to assert that once all the variables of rank lower than that of variable $v$ are stable then all the update functions of $f_v$ compute the same value. □

**Theorem 4.** *If a district-convergent nondeterministic iteration system is*

(a) *0-cyclic and has a fixed point, it is convergent, and if it is*
(b) *1-cyclic and has one convergent computation, then it is convergent.*

*Proof Sketch.* To prove the convergence of an arbitrary computation with an arbitrary choice of functions at each step (that respects the selection restriction outlined above), we consider the convergent computation with the same choice of functions. Now a counting argument similar to the one in the proof of Theorem 1(b) can be used to exhibit the required convergence. □

**Lemma 4 (Local Variant).** *If each directed cycle in $D$ has a variable $v$ and a variant function $\# : Q_v \longrightarrow N$ such that for each state $q$ in $Q_D$, and each update function $f_v$ in $F_v$*

$$\#(f_v(q)) < \#(q_v) \vee (f_v(q) = q_v)$$

*then the iteration system $(V_D, F_D)$ is convergent provided it has a fixed point.*

**Lemma 5 (Global Variant).** *If there is a variant function $\# : Q_D \longrightarrow N$ such that for all states $q$ in $Q_D$, for all strongly connected components $D'$ in $D$, and for every choice of update functions for the variables in $V_{D'}$,*

$$\#(V_{D'} \circ q) < \#(q) \vee (V_{D'} \circ q = q)$$

*then the iteration system $(V_D, F_D)$ is convergent.*

## 6 Finer Dependencies

As defined so far, the dependency graph of an iteration system requires considering the entire state space of the system. This requirement can result in a dependency graph that is "coarser" than desirable. For instance, the dependency graph of an iteration system may contain cycles, but if we were to consider only some states of the system then a definition of the dependency graph restricted to that set of states may result in a acyclic or self-looping graph, thereby enabling a simpler analysis of convergence.

In this section, we define a restricted notion of dependency graphs, based on which we discuss methods for establishing the convergence of iteration systems.

Let $I$ be an iteration system $(V, F)$ whose state space is $Q$, i.e., $Q$ is the cartesian product of the domains of all variables in $V$. Let $v$ and $w$ variables in $V$, and $P \subseteq Q$. We say that $v$ *depends on* $w$ *at* $P$ iff there exist two states $q \in P$ and $q' \in Q$ sich that $q$ and $q'$ differ only in their value of $w$ and $f_v(q) \neq f_v(q')$. The *dependency graph for* $P$ is a directed graph whose nodes correspond to the variables in $V$ and whose directed edges correspond to the depends-on-at-$P$ relation. Note that our earlier notion of the dependency graph of $I$ can now be referred to as the dependency graph for $Q$.

We say that $P$ is *serially closed* iff for all variables $v$ in $V$ and all states $q$ in $P$, $\{v\} \circ q \in P$.

Let $Q$ be partitioned into disjoint sets $P_1, \ldots, P_K$. The *partitioning graph* of $I$ for $\{P_1, \ldots, P_K\}$ is a directed graph that has a node corresponding to each partition in $\{P_1, \ldots, P_K\}$ and that has a directed edge from partition $P_i$ to partition $P_j$ iff $i \neq j$ and there exist two states $q \in P_i$ and $q' \in P_j$ such that for some step $W$, $W \subseteq V$, $W \circ q = q'$.

A computation $C$ of $I$ *reaches* a state $q'$ from a state $q$ iff there exists a finite prefix $S$ of $C$ such that $S \circ q = q'$. A computation $C$ of $I$ is said to be *serial* iff each step in $C$ contains exactly one variable.

**Theorem 5.** *Let $I$ be an iteration system and $\{P_1, \ldots, P_K\}$ be a partitioning of the state space of $I$ such that the partitioning graph of $I$ for $\{P_1, \ldots, P_K\}$ is acyclic.*

*$I$ is convergent if, for each $P_i$, at least one of the following three conditions hold.*

(a) *The dependency graph for $P_i$ is acyclic.*
(b) *The dependency graph for $P_i$ is self-looping, $P_i$ is serially closed, and there exists a computation that upon starting from an arbitrary state in $P_i$ reaches a fixed point in $P_i$.*
(c) *Every computation, upon starting from an arbitrary state in $P_i$, reaches a state not in $P_i$.*

*Proof.* We need to show that an arbitrary computation $C$ reaches a fixed point from an arbitrary state $q$. First, note that the partitioning $\{P_1, \ldots, P_K\}$ is acyclic. Hence, there exists a partition $P_j$ and a natural number $m$ such that the computation $C \downarrow m$, upon starting from state $(C \uparrow m) \circ q$, reaches states in $P_j$ only. From this, it follows that condition (c) does not hold for $P_j$. We consider separately the cases that conditions (a) or (b) hold for $P_j$, next. For the rest of the proof, let $G$ be the dependency graph for $P_j$, $ST$ be an arbitrary step, and $r$ be an arbitrary state in $P_j$ such that the state $ST \circ r$ is in $P_j$.

*Condition (a) holds for $P_j$.* Let $v$ be an arbitrary variable of rank $k$ in $G$. We claim:
If $G$ is acyclic and, for each variable $x$ of rank less than $k$, $r_x = (ST \circ r)_x$ holds, then $f_v(r) = f_v(ST \circ r)$ (1).

We prove Claim 1 by induction on the size of $ST$.

Base case: $|ST| = 1$.

Let $ST = \{w\}$. We consider two cases.

- *rank* $(w) < k$: Since $r_w = (\{w\} \circ r)_w$ by assumption, $r = \{w\} \circ r$; hence, $f_v(r) = f_v(ST \circ r)$.
- *rank* $(w) \geq k$: Since $G$ is acyclic, $v$ does not depend on $w$ at $P_j$; hence, $f_v(r) = f_v(ST \circ r)$ holds by definition of the depends-on-at-$P_j$ relation.

Induction Step: $|ST| > 1$.

Let $ST = \{w\} \cup ST'$, where $w$ is a variable of rank $\geq$ the rank of each variable in $ST'$. Hence, for each variable $w' \in ST'$, $f_{w'}(r) = f_{w'}(\{w\} \circ r)$ and, consequently, $ST \circ r = ST' \circ (\{w\} \circ r)$. Assume that for each variable $x$ of rank less than $k$, $(r_x = (ST \circ r)_x)$ holds. We appeal to the induction hypothesis twice:

1) Note that since $r_x = (ST \circ r)_x$, $r_x = (\{w\} \circ r)_x$ holds. We can, therefore, instantiate $ST$ and $r$ to $\{w\}$ and $r$, respectively, in the induction hypothesis to assert

- $f_v(r) = f_v(\{w\} \circ r)$.

2) Note that since $ST \circ r = ST' \circ (\{w\} \circ r)$, $r_x = (ST' \circ (\{w\} \circ r))_x$ holds. Recall from 1) that $r_x = (\{w\} \circ r)_x$; hence, $(\{w\} \circ r)_x = (ST' \circ \{w\} \circ r)_x)$. We can, therefore, instantiate $ST$ and $r$ to $ST'$ and $\{w\} \circ r$, respectively, in the induction hypothesis to assert

- $f_v(\{w\} \circ r) = f_v(ST' \circ (\{w\} \circ r))$.

From 1) and 2), by transitivity of equality, $f_v(r) = f_v(ST' \circ (\{w\} \circ r)$. Since $ST \circ r = ST' \circ (\{w\} \circ r)$, it follows that $f_v(r) = f_v(ST \circ r)$ (end proof of Claim 1).

From Claim 1, we can conclude that upon the application of one round of $C \downarrow m$ to the state $(C \uparrow m) \circ q$, the value of the rank 1 variables is the same in all states reached by the computation $C \downarrow (m+1)$ from the state $(C \uparrow (m+1)) \circ q$. By induction on $k$, the value of the variables of rank $\leq k$ is the same in all states reached by computation $C \downarrow (m+k)$ from the state $(C \uparrow (m+k)) \circ q$. Thus, all states reached by the computation $C \downarrow (m+|V|)$ from the state $(C \uparrow (m+|V|)) \circ q$ are identical to $(C \uparrow (m+|V|)) \circ q$; that is, $(C \uparrow (m+|V|)) \circ q$ is a fixed point.

*Condition (b) holds for $P_j$.* We claim:
There exists a sequence $S$ of distinct steps such that each step contains exactly one variable which is in $ST$, and $S \circ r = ST \circ r$ (2).

We prove Claim 2 by induction on the size of $ST$.
Base case: $|ST| = 1$.
The sequence $S$, where $S = ST$, satisfies the observation.
Induction step: $|ST| > 1$.

Let $w$ be a variable in $ST$ of highest rank among the variables in $ST$. For each variable $v \in (ST - \{w\})$, therefore, $v$ does not depend on $w$ at $P_j$. It follows that, $f_v(r) = f_v(\{w\} \circ r)$ and, hence, $ST \circ r = (ST - \{w\}) \circ (\{w\} \circ r)$. Also, since $r \in P_j$, $(ST \circ r) \in P_j$, $ST \circ r = (ST - \{w\}) \circ (\{w\} \circ r)$, and the partitioning $\{P_1, \ldots, P_K\}$ is acyclic, it follows that the state $(\{w\} \circ r)$ is in $P_j$. We can, therefore, instantiate $ST$ and $r$ to $ST - \{w\}$ and $\{w\} \circ r$, respectively, in the induction hypothesis to assert that there exists a sequence $S'$ of distinct steps such that each step contains exactly one variable which is in $(ST - \{w\})$, and $S' \circ (\{w\} \circ r) = (ST - \{w\}) \circ (\{w\} \circ r)$. Now, the sequence $S$, where $S = \{w\}; S'$, satisfies the observation (end proof of Claim 2).

By Claim 2, the states reached by the computation $C\downarrow m$ from the state $(C\uparrow m)\circ q$ are a subset of the states reached by some serial computation $C'$ from the state $(C\uparrow m)\circ q$. Also, since the computation $C\downarrow m$, upon starting from the state $(C\uparrow m)\circ q$, reaches states in $P_j$ only, and the partitioning $\{P_1,\ldots,P_K\}$ is acyclic, it follows that $C'$, upon starting from $(C\uparrow m)\circ q$, reaches states in $P_j$ only.

Let $n$ be a natural number and $S$ be an arbitrary prefix of $C'$. Define $n*\{v\}$ to be the sequence of $n$ steps, each of which in $\{v\}$; and $\#occ(S,v)$ to be the number of steps in $S$ that contain $v$. We claim:
If $G$ is self-looping and each variable $x$ of rank less than $k$ remains unchanged in all states that computation $C'$ reaches from $r$, then $((n*\{v\})\circ r)_v=(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$ for all $n>\#occ(S,v)$ (3).

We prove Claim 3 by induction on the length of $S$.
Base case: $S=\{\ \}$.
$\#occ(S,v)=0$ and $(S\circ r)=r$, hence the claim is trivially true.
Induction case: $S=S';\{w\}$ where $S'$ is a sequence of steps and $w\in V$.

The induction hypothesis is:
$((n*\{v\})\circ r)_v=(((n-\#occ(S',v))*\{v\})\circ(S'\circ r))_v$.
If $w=v$, then $1+\#occ(S',v)=\#occ(S,v)$. It follows that $(((n-\#occ(S',v))*\{v\})\circ(S'\circ r))_v=$
$(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$.
From the induction hypothesis, therefore,
$((n*\{v\})\circ r)_v=(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$.
Else $w\neq v$, and therefore $\#occ(S',v)=\#occ(S,v)$ holds. If the rank of $w$ is $<$ the rank of $v$ then, by the antecedent of Claim 3, $S'\circ r=S\circ r$ and, hence,
$(((n-\#occ(S',v))*\{v\})\circ(S'\circ r))_v=$
$(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$. If the rank of $w$ is $\geq$ the rank of $v$, then $v$ does not depend on $w$, since $G$ is self-looping. Since $P_j$ is serially closed, it follows by induction on $n$ that
$(((n-\#occ(S',v))*\{v\})\circ(S'\circ r))_v=$
$(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$.
From the induction hypothesis, therefore,
$((n*\{v\})\circ r)_v=(((n-\#occ(S,v))*\{v\})\circ(S\circ r))_v$
(end proof of Claim 3).

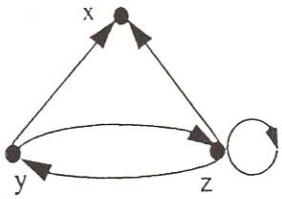
From Claim 3, we can conclude that once all variables of rank $<k$ remain unchanged in the states reached by the computation, the sequence of values that a rank $k$ variable takes is a function only of the number of updates to it. From condition (b), there exist a computation $C''$ such that $C''\circ r$ reaches a fixed point. The counting argument over $C$, $q$, and $C'$ in the proof of Theorem 1(b) can now be repeated using $C'$, $(C\uparrow m)\circ q$, and $C''$ respectively. Hence, the computation $C\downarrow m$ reaches a fixed point from the state $(C\uparrow m)\circ q$. □

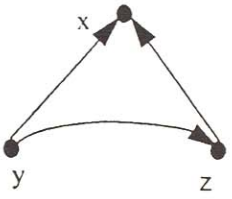
*Example 5.* Let $y$ and $z$ be boolean variables, and let $x$ be a variable that ranges over $\{left, right\}$. Then, the three assignment statements

$x:=right$

$y:=$ **if** $x=right$ **then** $\neg z$ **else** *false*

$z:=$ **if** $x=left$ **then** $y\vee z$ **else** *false*




define a convergent iteration system. At fixed point, $(x=right)$, $(y\equiv true)$ and $(z\equiv false)$ hold.

The dependency graph for this system (shown below) has cycles. One possibility for establishing the convergence of this system is to first establish district-convergence and then apply Theorem 2. Note that there is only one district in the dependency graph – between nodes $y$ and $z$ – and establishing the convergence of the iteration system associated with this district

$x:=x$

$y:=$ **if** $x=right$ **then** $\neg z$ **else** *false*

$z:=$ **if** $x=left$ **then** $y\vee z$ **else** *false*

is not easier than establishing the convergence of the original system.

On the other hand, we can partition the state space of the system into an acyclic partition $\{P_1,P_2\}$, where $P_1\equiv(x=left)$ and $P_2\equiv(x=right)$. Every computation, upon starting from an arbitrary state in $P_1$, is guaranteed to reach a state in $P_2$ when it first updates $x$. Moreover, the dependency graph for $P_2$ (shown below) is acyclic. Thus, by Theorem 5(b), the system is easily shown to be convergent.

Another method for establishing convergence based on finer dependencies has been presented by Burns, Gouda and Miller (cf. [2]). Their method uses a different notion of dependency which is defined for each state in the state space of the system. It is observed that, if the dependency graph of every state is acyclic or self-looping, then the effect of executing an arbitrary step is identical to the effect of executing some sequence of serial steps, i.e., steps that update exactly one variable. Consequently, a computation is convergent if a serial version of that computation is convergent. Establishing convergence is thus reduced to checking the convergence of all serial computations.

## 7 Conclusions

We have defined a very general model of computation that exhibits true concurrency, and have considered convergence as a typical property of systems expressed in this

model. We established several results that reduce the proof burden involved in establishing convergence.

When analyzing concurrent systems, convergence can be used to model termination. Since some progress (that is, eventuality) properties of a concurrent system can be reduced to the termination of a derived system, our techniques are applicable in verifying progress properties. (See, for example, [10], where the eventual enabledness of an action is reduced to termination.)

There are several issues to be investigated in extending this work. First, more general sufficiency conditions need to be identified. Second, useful extensions of the notion of convergence need to be considered. For instance, the requirement that each system computation necessarily reaches a fixed point can be weakened to the requirement that each system computation necessarily reaches some state in a set that is closed under system computation. Or, the requirement that the system start from an arbitrary state can be altered to requiring that the system start at any state in some distinguished set. For these extensions, methods similar to those discussed in Sect. 6 can be found.

Finally, problems for further research include comparing the 'rate' of convergence of various types of computations. Methods regarding other properties, such as closure, also deserve study.

## References


1. Arbib MA: Brains, machines and mathematics. Springer, Berlin Heidelberg New York 1987
2. Burns JE, Gouda MG, Miller RE: On relaxing interleaving assumptions. Proc MCC Workshop on Self-Stabilizing Systems. MCC Tech Rep #STP-379-89
3. Brown GM, Gouda MG, Wu CL: Token systems that self-stabilize. IEEE Trans Comput 38(6): 845–852 (1989)
4. Bertsekas DP, Tsitsiklis JN: Parallel and distributed computation. Prentice-Hall, New Jersey 1989
5. Chandy KM, Misra J: Parallel program design: a foundation. Addison-Wesley 1988
6. Dijkstra EW: The solution to a cyclic relaxation problem (1973) Reprinted in: Selected writings on computing: a personal perspective. Springer, Berlin Heidelberg New York 1982, pp 34–35
7. Dijkstra EW: Self-stabilizing systems in spite of distributed control. Commun ACM 17(11): 643–644 (1973)
8. Gries D: The science of programming. Springer, Berlin Heidelberg New York 1981
9. Gouda MG, Evangelist M: Convergence response tradeoffs in concurrent systems. MCC Tech Rep #STP-124-89 (also submitted to ACM TOPLAS)
10. Grumberg O, Francez N, Makovsky JA, deRoever WP: A proof rule for fair termination of guarded commands. Inf Contr 66: 83–102 (1985)
11. Harary F: Graph theory. Addison-Wesley 1972
12. Hoare CAR: Communicating sequential processes. Prentice-Hall International 1985
13. Kohonen T: Self-organization and associative memory. Springer, Berlin Heidelberg New York 1984
14. Kung HT, Leiserson CE: Systolic arrays (for VLSI). In: Sparse Matrix Proc, 1978
15. Lynch NA: I/O automata: a model for discrete event systems. Proc 22nd Annu Conf on Information Sciences and Systems, 1988
16. Robert F: Discrete iterations – a metric study. Springer, Berlin Heidelberg New York 1986
17. Wolfram S: Theory and applications of cellular automata, advanced series on complex systems, vol. 1. World Scientific, 1986