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, Austin
2. Department of Computer Sciences, The University of Texas at Austin

Abstract

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.

1 Introduction

Iteration systems are a useful abstraction for computational, physical and biological systems that involve "truly concurrent" events. In computing science, they can be used to represent self-stabilizing programs, neural networks, transition systems and array processors. 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 [Hoa], UNITY [CM] and I/O Automata [Lyn]), whereas maximal parallelism requires that all enabled events are executed at every step (see, for instance, the work on systolic arrays [KL] and cellular automata [Wol]).

A property of interest in iteration systems is convergence. This property is useful in studying the self-stabilization of distributed programs (cf. [Dij1], [BGM], [BGW], [Dij] and [GE]), convergence of iterative methods in numerical analysis (cf. [Rob] and [BT]), and self-organization in neural networks (cf. [Koh] and [Arb]). 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. [Gri]) whose value is bounded from below and is decreased by 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 Section 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 Section 3, we identify two classes of iteration systems, namely those whose dependency graphs are acyclic or self-looping, and present a theorem that establishes efficient proof obligations for verifying the convergence of these two classes. This theorem is then extended to general, deterministic iteration systems in Section 4. In Section 5, we show that, with minor modifications, our results continue to hold in nondeterministic iteration systems. In Section 6, we extend our analysis of convergence based on dependency graphs for individual states. Concluding remarks are in Section 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. 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 empty, 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 concatenation of two sequences $S = S'; S''$, 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 $\langle \textit{variable} \rangle := \langle \textit{corresponding update function} \rangle$. 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

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

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

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

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 $z = 0$. $\square$

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 $x[1], x[2], \ldots, x[i]$. $\square$

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 two integer components, $first[i]$ and $second[i]$. The assignment statements

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

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

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

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

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 $\{n_v | v \in V\}$ and its set of directed edges is $\{(n_v, n_w) | v \in V,\ w \in V,$ and v *depends on* $w\}$.

The dependency graphs for 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. □

Proof:

To prove the theorem we need to introduce the following new concepts: "*rank*", "*stable*", "*#steps*", "↑", and "↓".

For an acyclic or self-looping iteration system, 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 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* in 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 in 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 in 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 in 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:

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

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, the sequence of values it takes in successive updates 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 variables whose rank is 1 is stable in $(C \uparrow \#steps(C', q, 1)) \circ q$.

Induction Step: $k > 1$.

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

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

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

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

From (1), all variables of rank $\leq k-1$ are stable in 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),

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

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,

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

and the inductive hypothesis is established for rank k. □

The examples in Section 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 [Dij] and [Rob].

In fact, it is straightforward to show that Theorem 1 cannot be made to apply to any class of iteration systems that properly includes 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 2:
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 those of I'.
- Every computation of I' is a computation of I.

Proof: Replace each maximally strongly connected component that has two or more nodes in I by a single node with a self-loop in the dependency graph of I'. This is accomplished by replacing all the variables in the component with one variable of type record; the components of this record correspond, in a one-to-one manner, to the replaced variables. □

System I' in Lemma 2 is self-looping; hence, its convergence can be established 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:

$$(x, y) := (\underline{\text{if}}\ x > y\ \underline{\text{then}}\ x - y\ \underline{\text{else}}\ x\ ,\ \underline{\text{if}}\ x < y\ \underline{\text{then}}\ y - x\ \underline{\text{else}}\ y\ )$$
$$z := \underline{\text{if}}\ x = y\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ z + 1.$$

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 is not in D but some variable in D depends on it.
- 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 $V_D \setminus D$ 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

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

$$y := \underline{\text{if}}\ x < y\ \underline{\text{then}}\ y - x\ \underline{\text{else}}\ y.$$

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 3:

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 [Har]). 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 in which every variable of rank lower than k is stable, there exists a finite prefix S of C such that all variables of rank k will be stable in $S \circ q$.

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 in q), then v is clearly stable after the first round.

- 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) guarantees 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 on 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 4), or a single variant function for the whole district (Lemma 5).

The intuition underlying Lemma 4 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 5 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), and

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

$V_{D'}$ 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 4: *(Local Variant)*

If each directed cycle in D has a variable v and a variant function $\# : Q_v \to N$ such that for each state q,

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

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

Lemma 5: *(Global Variant)*

If there is a variant function $\# : Q_D \to N$ such that for each state q, and for each strongly connected component D' in D,

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

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

Proofs of Lemmas:

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 4:

The proof obligation is to show that for an arbitrary computation, C, of (V_D, F_D) and an arbitrary state, q, 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 $V_D \setminus D$.

We show that for an arbitrary variable v in D there exists a positive integer j such that v is stable in $(C \uparrow j) \circ q$. 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 in 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 in $q' = ((C \uparrow k) \circ q)$.

- $v \in (V_D \setminus W)$: 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 in 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 in 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 in $(C \uparrow j) \circ q$ where $j = k + l$.

Proof of Lemma 5:
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,

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

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

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 [Har]). Take the condensation of G_W. As observed in the proof of Theorem 3, 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 $rank = i, (1 \leq i \wedge i \leq k)$. 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 $\#(W \circ q) < \#(q)$.
$\square$

As an example, both Lemma 4 and Lemma 5 can be used to show that the iteration system associated with the district in Example 1 is convergent. In using Lemma 4, let the variant functions for both x and y be their respective identity functions. In using Lemma 5, 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 3.

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. This choice is arbitrary except for the requirement 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:

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

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

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

$v[3] := \underline{\text{if}}\ \ first[1] + c \leq first[3] \vee second[3] = 1\ \underline{\text{then}}\ (first[1] + c, 1)\ \underline{\text{else}}\ v[3]$

$\quad\quad \mid\ \underline{\text{if}}\ \ first[2] + d \leq first[3] \vee second[3] = 2\ \underline{\text{then}}\ (first[2] + d, 2)\ \underline{\text{else}}\ v[3]$

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 nearest neighbor to node i along this path. □

We now redefine four 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, $f_v(q) = q_v$.
- 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 6:

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 7:

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 that is very similar to the one in the proof of Theorem 1(b) can be used to exhibit the required convergence. □

Lemma 8: *(Local Variant)*

If each directed cycle in D has a variable v and a variant function $\# : Q_v \to N$ such that for each state q, and each update function f_v in F_v

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

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

Lemma 9: *(Global Variant)*

If there is a variant function $\# : Q_D \to N$ such that for all states q, for all strongly connected components D' in D, and for every choice of update functions for the variables in $V_{D'}$,

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

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

6 State Dependency Graphs

For some applications, using dependency graphs yields an analysis that is coarser than necessary. For instance, it is possible that the dependency graph of an iteration system is cyclic even though the dependencies between variables at each state of the system are not cyclic (we present such an example shortly). In this section, we extend our analysis of convergence by first defining the notion of dependency at individual states. We then state and prove a theorem about the convergence of iteration systems that have acyclic or self-looping dependency graphs at each state.

For simplicity sake, we consider determinisitic iteration systems in this section. We note that it is straightforward to extend our analysis to accomodate non-deterministic iteration systems.

Let v, w be variables in V, and let q be a state of I. We say v *depends on* w *at* q iff there exists a state q' of I such that the following hold:

- $f_v(q) \neq f_v(q')$, and
- $v = w \Rightarrow (\forall u : u \neq v \Rightarrow q_u = q'_u)$, and
- $v \neq w \Rightarrow (q_v = q'_v \ \wedge \ f_v(q) = f_v(q' \,|^{w}_{q_w}))$

It is straightforward to show that if v *depends on* w *at* a state q then v *depends on* w as defined in Section 3. The proof involves considering the two cases $v = w$ and $v \neg w$ separately. In the first case, we have two states q and q' such that $f_v(q) \neq f_v(q')$ and $(\forall u : u \neq v \Rightarrow q_u = q'_u)$; thus, v *depends on* v. In the second case, we have two states q' and $q' \,|^{w}_{q_w}$ such that such that $f_v(q) \neq f_v(q')$ and $f_v(q) = f_v(q' \,|^{w}_{q_w}$ which implies $f_v(q') \neq f_v(q' \,|^{w}_{q_w}$. From this fact and the fact that q' and $q' \,|^{w}_{q_w}$ differ only in the value of w, we conclude that v *depends on* w in the second case.

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

Example 5:
Let *give* and *take* be integer variables, and let *st* be a variable that ranges over $\{g, t\}$. Then, the three assignment statements

$$\begin{aligned}
&st \quad := t \\
&take := \underline{\text{if}}\ (st{=}t \ \wedge \ give{-}take > 0)\ \underline{\text{then}}\ take{+}1\ \underline{\text{else}}\ take \\
&give := \underline{\text{if}}\ (st{=}g \ \wedge \ give{-}take < W)\ \underline{\text{then}}\ give{+}1\ \underline{\text{else}}\ give
\end{aligned}$$

where W is an arbitrary integer constant, define a convergent iteration system. At fixed point, $(st = t)$ and $(give \leq take)$ holds. The dependency graph of the iteration system is contrasted with the dependency graphs for individual states below.

The main result of this section now follows. **Theorem 10**:

If for each state of I, the dependency graph is

(a) acyclic, then I is convergent, and if it is

(b) either self-looping or acyclic, and there exists one convergent computation, then I is convergent. $\square$

Proof

Proof of part (a):

The proof is based on the observation:

If there is no variable w such that variable v *depends on* w *at* some state, then f_v is a constant function. (1)

To prove (1), we need to show that if there does not exist w such that v *depends on* w *at* some state q then $f_v(q) = f_v(r)$ for every state r. The proof is by induction on the number of variables in which q and r differ. For the base case, let q and r differ in their value of exactly one variable, say u. Since v does not depend on u at q, it follows from the definition of the depends on relation at q that $f_v(q) = f_v(r)$. For the induction step, let q and r differ in their value of k ($k \leq |V|$) variables. Let $w, w \neq v$, be one of these variables. By induction hypothesis, we have that $f_v(q) = f_v(r \mid^{w}_{q_w})$. Since v *depends on* w *at* q does not hold, it now follows $f_v(q) = f_v(r)$. This completes the proof of (1).

We are now ready to prove part (a); in particular, we show that after the application of the first k rounds ($k \leq |V|$) of an arbitrary computation C to an arbitrary state q, at least k variables are stable in the resulting state $(C \uparrow k) \circ q$. This shows that, the system is at a fixed point after $|V|$ rounds. The proof is by induction on k.

Base Case: $k = 1$.

From (1), it follows that each variable with rank 1 in the dependency graph for q is updated by a constant function. Hence, after the first round each such variable is stable in the resulting state $(C \uparrow 1) \circ q$.

Induction Step: $k > 1$.

Let $q' = (C \uparrow (k-1)) \circ q$, and let W be the set of variables that are stable in q'. By induction hypothesis, $|W| \geq (k-1)$.

Define R to be the set of states $\{r \mid (\forall v \in W : q'_v = r_v)\}$. Now, pick any variable v that has lowest rank in the dependency graph for q', among variables not in W. By a straightforward extension of (1), it follows that $(\forall r \in R : f_v(q') = f_v(r))$. Hence, v is stable in $(C\uparrow k) \circ q$.

Proof of part (b):

The proof is based on the following (two) observations:

For $v \in V$, and $q, q' \in Q$ such that $q_v = q'_v$

$$(\exists w\, v \neq w : v \textit{ depends on } w \textit{ at } q) \equiv (\exists w\, v \neq w : v \textit{ depends on } w \textit{ at } q') \qquad (2)$$

If the dependency graph for each state of I is either self-looping or acyclic, then there exists a variable that has rank 1 in the dependency graph for each state of I. (3)

To verify (2) it suffices, by symmetry, to prove the implication from left to right. Therefore, assume that v *depends on* w *at* q for some w, $v \neq w$. By definition of the depends on relation at q, there exists a state r such that $(q_v = r_v \;\wedge\; f_v(q) \neq f_v(r) \;\wedge\; f_v(q) = f_v(r \mid {}^{w}_{q_w}))$. Two conclusions are now immediate:

- $q'_v = r_v$
- there exists a state r' such that $r_v = r'_v$ and $f_v(q') \neq f_v(r')$.

Let the set of variables in which q' and r' differ be U. For any u in U, if $f_v(q') = f_v(r' \mid {}^{u}_{q'_u})$ holds then v *depends on* u *at* q'. Else, repeat this argument for q' and $(r' \mid {}^{u}_{q'_u})$ using the set $U \setminus \{u\}$, and so on, until a variable (other than v) on which v depends at q' is established (end proof of (2)).

We prove (3), by contradiction, as follows. Assume that the dependency graph for each state of I is either self-looping or acyclic and that there is no variable which has rank 1 in the dependency graph for each state of I. Consider for each state of I the set of variables that have rank 1 in the dependency graph for that state. Let W be the union of the sets for all such states. By our assumption, for all $v \in W$, there must be a state $q.v$ such that $(\exists w : v \textit{ depends on } w \textit{ at } q.v)$. Let q be a state such that $(\forall v \in W : q_v = q.v_v)$. By observation (2), it follows that $(\forall v \in W : (\exists w : v \textit{ depends on } w \textit{ at } q))$, which contradicts the assumption that q is either acyclic or self-looping (end proof of (3)).

The rest of the proof is by an induction on the number of stable variables in the system state. We need to prove that if some computation C' 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 in state $(C\uparrow i) \circ q$; that is, $(C\uparrow i) \circ q$ is a fixed point.

Base Case: $k = 1$.

By (3), there exists a variable whose rank is 1 in the dependency graph of all states of I. Hence, it is necessarily stable after $\#steps(C', q, 1)$ updates to it in any computation starting in state q.

Induction Step: $k > 1$.

Let $q' = (C \uparrow i) \circ q$, where i is the least integer such that at least $k-1$ variables are stable in q'. Now, consider the variables that have lowest rank in the dependency graph for q', among the variables not stable in q'. Define R' to be the set of states $\{r \mid (\forall v \,\text{stable in}\, q' : r_v = q'_v)\}$. By a straightforward extension of (3), it follows that there exists a variable that depends only on itself at each state in R'; that is $(\forall r, r' \in R' : f_v(r) = f_v(r' \mid ^{v}_{r_v}))$. Thus, by a counting argument similar to the one presented in Theorem 1(b), eventually k variables are stable. $\square$

MARKER¿¿¿ Example 5 illustrates Theorem 10. The dependency graph for each state of the iteration system in the example is acyclic or self-looping and, therefore, by Theorem 10, the iteration system is convergent.

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, [GFMR] where the eventual enabledness of an action is reduced to termination.)

There are several issues to be investigated in extending this work. Firstly, more general sufficiency conditions need to be identified. Secondly, useful generalizations 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.

We note that methods for proving convergence to a closed set have been recently shown by Burns, Gouda and Miller [BGM]. Their methods employ state dependency relations also, although their definition of the state dependency relation is more complex. One of their main results, paraphrased in our notation, is as follows. If the dependency graph for each state in an iteration system is self-looping or acyclic, and if all sequential computations converge to a closed set R, then all computations converge to R. This result is also valid for our definition of the depends on relation at a state. (The proof, omitted here, is based on showing that if the dependency graph for each state in an iteration system is self-looping or acyclic, then the state resulting from an arbitrary step is equivalent to the state resulting from *some* sequence of sequential steps.)

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

[Arb] M.A. Arbib, *Brains, Machines and Mathematics*, Springer-Verlag, 1987.

[AAEG] A. Arora, P. Attie, M. Evangelist and M. Gouda, "Convergence of Iteration Systems", submitted for publication in *Distributed Computing* (Special Issue on Self-Stabilization).

[BGM] J.E. Burns, M.G. Gouda, and R.E. Miller, "On relaxing interleaving assumptions", *Proceedings of the MCC Workshop on Self-Stabilizing Systems*, MCC Technical Report #STP-379-89.

[BGW] G.M. Brown, M.G. Gouda, and C.-L. Wu, "Token Systems that Self-Stablize", *IEEE Transactions on Computers 38(6)*, pp. 845-852, 1989.

[BT] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, 1989.

[CM] K.M. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley Publishing, 1988.

[Dij] E.W. Dijkstra, EWD306 "The Solution to a Cyclic Relaxation Problem", 1973. Reprinted in *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, pp. 34-35, 1982.

[Dij1] E.W. Dijkstra,"Self-stabilizing Systems in Spite of Distributed Control", *Communications of the ACM 17(11)*, pp. 643-644, 1973.

[Gri] D.Gries, *The Science of Programming*, Springer-Verlag, 1981.

[GE] M.G. Gouda and M. Evangelist, "Convergence Response Tradeoffs in Concurrent Systems", *MCC Technical Report #STP-124-89*; also submitted to *ACM TOPLAS*.

[GFMR] O. Grumberg, N. Francez, J.A. Makovsky and W.P. deRoever, "A proof rule for fair termination of guarded commands", *Information and Control 66*, pp. 83-102, 1985.

[Har] F. Harary, *Graph Theory*, Addison-Wesley Publishing, 1972.

[Hoa] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, 1985.

[Koh] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, 1984.

[KL] H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI)," in *Sparse Matrix Proc.*, 1978.

[Lyn] N.A. Lynch, "I/O Automata: A Model for Discrete Event Systems," *Proc. of 22nd Annual Conference on Information Sciences and Systems*, 1988.

[Rob] F. Robert, *Discrete Iterations - A Metric Study*, Springer-Verlag, 1986.

[Wol] S. Wolfram, *Theory and Applications of Cellular Automata, Advanced Series on Complex Systems*, Vol.1, World Scientific Publishing, 1986.