Self-Stabilization in Iteration Systems

Howard E. Motteler
Research Assocociate Professor
Maryland Center for Telecommunications Research
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore MD 21250
email: motteler@umbc.edu
phone: 410-455-1936
fax: 410-455-1291

Deepinder Sidhu
Professor, Computer Science and Electrical Engineering
Director, Maryland Center for Telecommunications Research
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore MD 21250
email: sidhu@umbc.edu
phone: 410-455-3028
fax: 410-455-3969

Abstract

An *iteration system* is a model of parallel computation in which a state vector is repeatedly updated by a non-deterministically selected set of functions. Such a system is *self stabilizing* if it is guaranteed to converge to a desirable state or set of states in a finite number of steps, from any initial state. In this paper, existing criteria for convergence to a fixpoint are extended, convergence for a large class of iteration systems is shown to be decidable, for three distinct types of fairness, and convergence properties for two different versions of the iteration system model are related.

Keywords: iteration system, convergence, self-stabilization

1. **Introduction**

A parallel or distributed system is *self stabilizing* if it is guaranteed to converge to a desirable state or set of states in a finite number of steps, from any initial state. Dijkstra has discussed the importance of self-stabilization in distributed systems [7,9], and examples of systems with self-stabilizing properties are given in [1,5,6,8].

In this paper, we study convergence and self-stabilization in a particular model of computation, *iteration systems*. An iteration system repeatedly updates a state vector with a non-deterministically selected set of functions. It can serve as a model either of shared memory updated by a set of concurrent processes, or of distributed computation synchronized by barriers. We will consider both convergence to a fixpoint and to a set of values, for several different fairness assumptions.

In Section 2, we define iteration systems and give an example of Dijkstra's Dining Philosophers [3] represented as an iteration system. Convergence of iteration systems to a fixpoint was studied by Arora et. al., in [10,13], and in Section 3, we extend some of those results. In Section 4, convergence for a broad class of iteration systems is shown to be testable, for both convergence to a set and to a fixpoint, for several definitions of fairness. In Section 5, convergence properties for two different version of the iteration systems model are considered, and we give conditions under which convergence to a fixpoint corresponds to convergence to a set in the more general model.

2. **Iteration Systems**

We begin with the formal definition of an iteration system, give an example of Dijkstra's 'Dining Philosophers' represented as such a system, and then discuss dependency graphs, a key tool in analyzing such systems.

2.1. **Definitions**

Computation in an iteration system proceeds as a sequence of non-deterministic updates to a global state vector. Let V be an n-vector of variables over some domain D, and F a set of n functions from D^n to D. An *iteration system* $\mathcal{I}$ is a pair (V, F), where each variable $v_i \in V$ has exactly one associated *update function* $f_i \in F$. An *update-sequence* is an infinite sequence $S_0, S_1, \ldots$ of non-empty subsets of V, and represents a sequence of choices of variables to be updated. In the iteration system model, these choices do not depend on the state of the system.

The *state* of an iteration system is an vector in D^n. For each element $d \in D^n$, an update-sequence $S_0, S_1, \ldots$ defines an infinite sequence of states, $d_0, d_1, \ldots$, where $d_0 = d$, and for $n \geq 0$, d_{n+1} is the result of simultaneously applying the update functions for the variables in S_n to the state d_n.

An update-sequence can be constrained by various forms of fairness. An update-sequence is *k-fair* if no variable is bypassed more than $k-1$ times; it is is *p-fair* if there

is a well defined probability associated with each subset choice; and *weakly-fair* if every variable appears infinitely often in the sequence of sets. If an update-sequence is k-fair or p-fair, then it is also weakly-fair.

A fair update-sequence can be represented as a sequence of *rounds*. A round is a contiguous subsequence of a update-sequence containing every variable. Starting with S_0, an update sequence can be partitioned into a sequence of minimal-length rounds such that if the update-sequence is k-fair, then no round has length greater than k.

An update-sequence *converges* to $A \subseteq D^n$ if for every initial state $d \in D^n$, there is some j such that for all $k \geq j$, $d_k \in A$. An iteration system converges to A if every fair update-sequence converges to A. If A is a singleton, we say the iteration system converges to a fixpoint. The set A may be thought of as an "attractor"—it is reachable from any initial state, and closed, i.e., when a computation enters a state in A, then all subsequent states are in A. If A is a set of desirable states, then we would say that the iteration system is self-stabilizing.

2.2. **An Example**

Dijkstra's dining philosopher problem has a straightforward representation as an iteration system. (For other examples, Aurora et al. give iteration systems for greatest common divisor, minimum of a bag, and shortest path.) For n philosophers, we need n variables representing philosophers, and n variables representing forks. Forks have values L, R, or I, depending on whether they are being used by the philosopher to the left or right, or are idle. Philosophers have values S, H, and E, for sleeping, hungry, and eating, respectively.

One possible set of update functions is as follows. Let philosophers have an associated update function $\mathbf{p}(p, f_L, f_R)$, where p is the philosopher's state, f_L is the state of the fork to the left, and f_R is the state of the fork to the right. The assignment $p := \mathbf{p}(p, f_L, f_R)$ updates the philosopher's state variable p. The philosopher update function is defined as

$$\mathbf{p}(p, f_L, f_R) = \begin{cases} S & \text{if } p = E, \\ H & \text{if } p = S, \\ E & \text{if } p = H \text{ and } f_L = R \text{ and } f_R = L, \\ p & \text{otherwise (no change).} \end{cases}$$

Forks have an associated update function $\mathbf{f}(f, p_L, p_R)$, where f is the fork's state, p_L is the state of the philosopher to the left, and p_R is the state of the philosopher to the right. The assignment $f := \mathbf{f}(f, p_L, p_R)$ updates the fork's state variable f. The fork update function is defined as

$$\mathbf{f}(f, p_L, p_R) \begin{cases} L & \text{if } f = I \text{ and } p_L = H, \\ R & \text{if } f = I \text{ and } p_R = H \text{ and } p_L \neq H, \\ I & \text{if } (f = L \text{ and } p_L = S) \text{ or } (f = R \text{ and } p_R = S), \\ f & \text{otherwise (no change).} \end{cases}$$

In this particular example, when philosophers on both sides of a fork are hungry, the fork goes to the philosopher on the left; otherwise the first philosopher to request an idle

fork gets it. This particular implementation has a well-know deadlock, with well-know fixes, e.g., even-numbered forks might go preferentially to philosophers on the left, and odd-numbered forks to philosophers on the right.

Since all the philosophers cannot eat at once, convergence to any single state implies starvation of at least some philosophers, and convergence to a state where all forks have the value L gives a deadlock that starves them all. Convergence to a set of states without such deadlocks, together the assumption of some form of fair update sequences, would guarantee a solution.

2.3. **The Dependency Graph of an Iteration System**

The idea of a *dependency graph* is central to our analysis of iteration systems. In general, an update function f_v may depend on only a subset of the variables in V. An update function f_x depends on y iff there are vectors d_1 and d_2, differing only in component y, such that $f_x(d_1) \neq f_x(d_2)$. The *dependency graph* for a given set of update functions contains a node for each element of V, and a directed edge (y, x) iff f_x depends on y.

We introduce some definitions concerning dependency graphs as follows. Our terminology is taken from Harary [11] and Bondy and Murty [4]. A *strongly connected component* in a directed graph is a maximal subgraph in which there is a directed path between every pair of nodes. Suppose G has strongly connected components $G_1, G_2, \ldots G_k$. The *condensation* of G has a node n_i for each component G_i, and an edge from n_i to n_j iff there is an edge from some node in G_i to some node in G_j. The condensation of G is acyclic, since the subgraphs G_i are maximal. In a directed acyclic graph, a *source node* is a node with in-degree 0 (i.e., no in-edges), and the *rank* of a node n is the length of the longest path from a source node to n. A cycle of length 1 is called a *self-loop*.

3. **Criteria for Convergence of Iteration Systems**

One approach to determining convergence is to find properties easily recognized or proved, that guarantee it. For example, for $B \subseteq D^n$, let $F(B)$ be the result of applying all the subsets of functions of F to vectors of B, i.e., the result of applying all possible single steps of the iteration system (V, F) to B. If we could show that for some $A \subset D^n$ and all B such that $A \subset B \subseteq D^n$ that $F(B) \subset B$, then the system would converge to A. This very general criterion amounts to little more than a restatement of the definition of convergence, without taking into account fairness, and by itself is not very helpful.

Without some form of fairness, even very simple systems may not converge; for example the iteration system with $V = \{x, y\}$, F defined as $f_x = 0$ (a constant function), and

$$f_y(x, y) = \begin{cases} y + 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

For any initial state with $x \neq 0$, the system will not converge until x is updated, and this is guaranteed only for fair update sequences.

In this section we investigate criteria for convergence to a fixpoint, in terms of properties of the dependency graph of an iteration system. Results in this section apply for any sort of fairness, since they take the form of the convergence of parts guaranteeing the convergence of the whole.

Arora et. al. [3] showed that if an iteration system is "self-looping," i.e., the longest cycle in the dependency graph has length 1, then the iteration system is convergent iff it has *some* convergent update-sequence. To show convergence for self-looping iteration systems, it is thus sufficient to select any update-sequence, and either prove or test it for convergence. They also show that this property (convergence iff there is some convergent update-sequence) is not true in general for iteration systems that have cycles of length 2 or more.

We show that to guarantee convergence it is sufficient to show either that each strong component of an iteration system is convergent, or more generally, to show that at least one value on each cycle of the dependency graph eventually remains unchanged. We present two theorems which extend results which first appeared in [3].

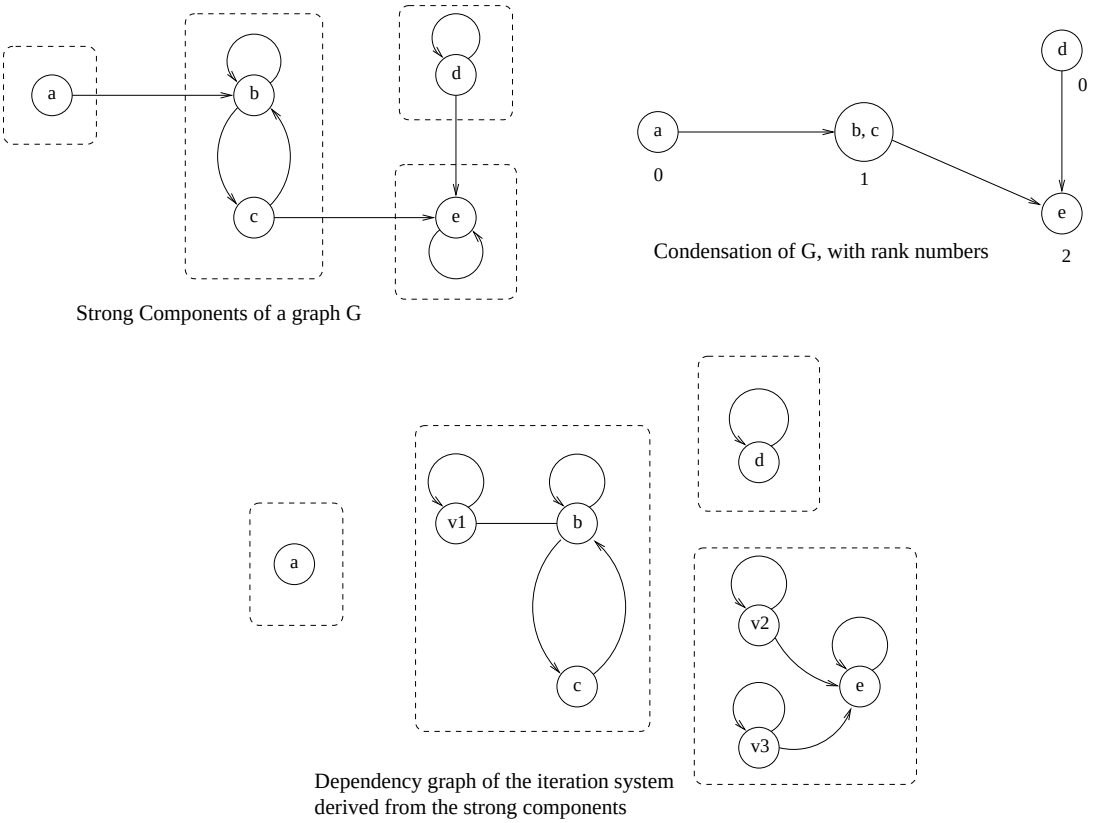


Figure 1. The dependency graph of an iteration system, its strong components, and their dependency graphs.

The convergence behavior of an iteration system can be characterized in terms of the behavior of its strongly connected components. We can form a new iteration system from each strong component of an iteration system, as follows. Let U be the set of nodes from a strong component of an iteration system. The new iteration system consists of every variable in U as well as a new set of source variables, one for each edge into the strong component. The update function for each new variable v is the identity function $f_v(v) = v$. A strong component is said to be convergent iff its associated iteration system is convergent. Figure 1 gives an example of the dependency graph of an iteration system, its strong components, and the dependency graphs of the iteration systems derived from those components.

The following result is a generalization of a theorem presented in [3]. In particular, we do not restrict components to cycles of 2 or more. We also present a much shorter proof, and show that the converse is false.

Theorem 1. If all strong components of an iteration system $\mathcal{I}$ are convergent, then $\mathcal{I}$ is convergent.

Proof. Let G be the dependency graph of $\mathcal{I}$, and G' the condensation of G. A *source node* is a node with no in-edges. The *rank* of a node n in G' is the length of the longest path in G' from a source node to n. The rank of a variable $v \in V$ (a node of G) is the rank of the strong component in G' that contains v. Suppose a particular update-sequence C of $\mathcal{I}$ is partitioned into rounds. We prove by induction on the rank of variables $v \in V$ that if v has rank k, then there is some round r (depending on k) such that for all rounds greater or equal to r, v remains unchanged.

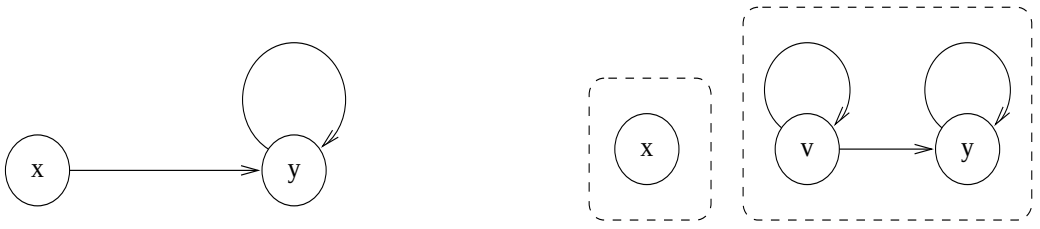
For the basis, suppose v has rank 0. Then v is the member of a strong component with no in-edges in G'. By hypothesis that component converges, so there is some round r after which all its variables stop changing.

Suppose the inductive hypothesis is true for all variables v of rank k, let v' be a variable of rank $k+1$, and let A be the set of variables that are in-adjacent to any member of the strong component containing v'. Each variable $v_i \in A$ has rank at most k, and so by the inductive hypothesis, there is some round r_i such that for all rounds greater or equal to r_i, v_i stops changing. Let r be the maximum of these values r_i, so after r rounds, all variables in A stop changing. Thus, after round r, every value on a variable in-adjacent to the component containing v' stops changing.

By hypothesis, the component containing v' is convergent, i.e., for any fixed set of values on variables in-adjacent to that component, there is some number r' of rounds such that the variables of the component stop changing after r' rounds. Therefore, after $r + r'$ rounds, v' will remain unchanged. ☒

The converse of the theorem is false: an iteration system can be convergent, and have strong components that are not convergent. Consider the simple iteration system we saw earlier, where $V = \{x, y\}$, F defined as $f_x = 0$ (a constant function), and

$$f_y(x, y) = \begin{cases} y + 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$



Dependency graph of an interation system that is convergent for fx and fy as given in the text

Iteration system derived from strong components of the dependency graph

Figure 2. An iteration system which is convergent but has strong components that are not convergent.

The dependency graph is illustrated in Figure 2. This iteration system is convergent, as any fairness assumption guarantees that x eventually becomes 0 so that y stops increasing. The iteration system formed from the component consisting of y and its associated 1-cycle is not convergent, as for values other that 0 on its source variable v, y increases without limit.

A strong component consisting of a single variable v with no 1-cycle (i.e., no self-loop) is always convergent. If values on variables in-adjacent to v do not change, then after no more than 1 update, v will not change. If G is acyclic, then its strong components are single variables with no 1-cycles. Therefore if the dependency graph of an iteration system is acyclic, then it is convergent. (This was proved directly in [3], however it is simply a corollary of Theorem 1.)

We now give an intuitive generalization of Theorem 1: to guarantee convergence it is sufficient to show that one variable on each directed cycle will eventually stop changing. Suppose U is a subset of V. An update-sequence is called *U-convergent* iff for every initial state $d \in D^n$, there is some j such that for all $k \geq j$, the components of states d_k and d_j corresponding to elements of U are identical. An iteration system is U-convergent iff every fair update-sequence is U-convergent.

Theorem 2. Let G be the dependency graph of an iteration system $\mathcal{I}$ and let U be a subset of V such at least one element of U lies on every directed cycle of G. Then $\mathcal{I}$ is convergent iff it is U-convergent.

Proof. It is clear that convergence implies U-convergence, so we must show the converse. Suppose a particular update-sequence C of $\mathcal{I}$ is U-convergent, and is partitioned into rounds, starting with round 0. If the update-sequence is U-convergent, then there is some round k after which none of the variables of U change. Suppose the last set of variables in round k is S_j, with d_j the corresponding state.

We form a new iteration system $\mathcal{I}'$ from $\mathcal{I}$ as follows. Replace each variable $u \in U$ with two variables u' and u''. The function f_u now writes to u', and is denoted as $f_{u'}$

in the new iteration system. All functions that read u now read u''. The new iteration system $\mathcal{I}'$ has a dependency graph G', where each variable $u \in U$ is partitioned into two variables: u' with the *in* arcs of u incident, and u'', with the *out* arcs of u incident. The new dependency graph G' is acyclic.

Let d'_j be a state of the new iteration system, identical to d_j, but with each $u' = u'' = u$. Let C' be an update-sequence of $\mathcal{I}'$ corresponding to C, but updating only u', when C updates u. Since variables in U do not change after round k, the state sequences $d_{j+1}, d_{j+2}, \ldots$ and $d'_{j+1}, d'_{j+2}, \ldots$ will correspond, in both $\mathcal{I}$ and $\mathcal{I}'$. Since $\mathcal{I}'$ has an acyclic dependency graph, the update-sequence converges by Theorem 1, and so the original update-sequence converges. ☒

In [3], iteration systems implementations of greatest common divisor, minimum of a multiset, and a simple version of shortest path are shown to converge as an application of Theorem 1. For the dining philosopher example, however, all points are strongly connected, so that Theorem 1 does not apply. Further, Theorem 2 tells us only that the system converges to a fixpoint (and so the philosophers starve) if, e.g., all forks stop changing, which we already knew. For such problems a different technique is needed, and we consider one possible approach in the next section.

4. **Testing Convergence of Iteration Systems**

In this section, we show that if an iteration system has a finite state space, then it can be effectively tested for convergence. Even with a finite state space, depending on the fairness assumptions used, there may be no fixed length within which computations of a particular iteration system converge. This rules out a straightforward (though very slow) testing scheme, where all computations up to a given length are checked for convergence.

We give a method for testing iteration systems based on the idea of finding the "fair transitive closure" of the state space graph. Details of the test method vary with the form of fairness assumed. Unfortunately, the complexity of the test is exponential in the number of variables. We believe the approach is nonetheless relevant for several reasons: we found it interesting that some method exists to test convergence, since direct tests do not work, as we cannot always bound time to convergence; it is at least possible that faster tests for convergence exist, and finding some method could be a first step towards finding a faster method; and although slow, the method might still be useful for moderate-sized problems.

Many iteration systems of interest have a finite state space, including many versions of the dining philosopher example given in Section 2. If all functions in an iteration system have finite range, then after one round (after every variable has been updated at least once) the system has a finite state space. We can give a more general criterion for finite state space, as follows.

Lemma 1. Suppose that for every cycle in the dependency graph G of an iteration system $\mathcal{I}$, some variable on that cycle is updated by a function that has finite range. Then after at most r rounds, $\mathcal{I}$ has a finite state space, where r is bounded by the diameter of G.

Proof. Form a new graph G' from G, by deleting in-edges to each of the variables updated by finite functions. G' is acyclic, and has some maximum rank r. The rank of each variable $v \in V$ is its rank in G'. Each variable v has a set of ancestors in G'. Since the longest path through G' has length r, after r rounds no value has an ancestor that is an initial value.

For each variable v, define the set D_v as the set of all possible values that v can take on, after round r, in any update-sequence of $\mathcal{I}$. We construct the sets D_v by induction on the rank of variables v in G'. If a variable v has rank 0, it is updated either by a constant function (if v is a source node in G), or by a finite function.

In either case, D_v is the range of f_v, and is finite. Now, suppose v has rank $k+1$, and is updated by $f_v(v_1, \ldots, v_k)$. Then $v_1, \ldots, v_k$ have rank at most k, and the associated sets $D_{v_1}, \ldots, D_{v_k}$ are defined. D_v is the set of values obtained by applying f_v to all combinations of values from $D_{v_1}, \ldots, D_{v_k}$. Since there is some maximum rank r, the construction terminates, and each set D_v is finite. ⊠

The state space (after r rounds) for $\mathcal{I}$ is $Q = D_{v_1} \times D_{v_2} \times \cdots \times D_{v_n}$, where $n = |V|$. The set Q contains every possible state of $\mathcal{I}$ after the first r rounds, in any update-sequence. Given an update-sequence C, let a be the number of steps in the first r rounds, and let $b = |Q|$. Then within $a + b + 1$ steps, some state must be repeated.

Define the state space graph $\mathbf{G} = (Q, \mathbf{E})$ as follows. Nodes of $\mathbf{G}$ are the states Q of $\mathcal{I}$, after the first r rounds. There is an edge (p, q) in $\mathbf{E}$ iff for some nonempty $S \subseteq V$, updating the variables in S can change state p to state q. Because computations are infinite, every state p has at least one out edge. Edges (p, q) will have labels which will depend on the particular assumptions about fairness. Paths through $\mathbf{G}$ correspond to computations of $\mathcal{I}$, including computations that are not fair.† A fixpoint p of $\mathcal{I}$ is a node in $\mathbf{G}$ where the only out-edge is the 1-cycle (p, p).

The transitive closure of $\mathbf{G}$ has an edge (p, q) iff there is a path from p to q in $\mathbf{G}$ [2]. Unfortunately, in general only some of these paths correspond to fair computations. What we want is a "fair transitive closure," which would have an edge (p, q) iff there is a fair path from p to q in $\mathbf{G}$.

If we do not restrict $\mathcal{I}$ to fair computations, then we can test for convergence to a fixpoint more easily. In this case, $\mathcal{I}$ converges to a fixpoint iff every cycle of $\mathbf{G}$ is a 1-cycle, as some (possibly unfair) computation could follow a k-cycle, for any $k \geq 1$. If we assume some form of fairness, it is still the case that if $\mathbf{G}$ has only 1-cycles then $\mathcal{I}$ converges to a fixpoint.

We use the semiring transitive closure algorithm [2] to find fair paths, for every pair of states $p, q \in Q$. In this algorithm, a *label* for a given path is the semiring product $\otimes$ of the edge labels along the path. The value $c(p, q)$ for vertices p and q is the semiring sum

† Our paths, as in Aho, Hopcroft, and Ullman, may repeatedly visit vertices and edges [2]. Harary would call this a walk [11].

$\oplus$ of the labels for all paths from p to q. The transitive closure algorithm finds the values $c(p,q)$ for all vertices $p, q \in Q$.

Depending on the particular definition of fairness, we can either define a set of semiring operators which give the fair transitive closure directly, or we can first find a non-fair closure, and then discard non-fair paths. If we can find the fair transitive closure, its connected components determine the convergence properties of the iteration system.

In the remainder of this section, we present three theorems (one for each of the three forms of fairness we have considered) which show that when we have a finite state space, we can effective determine the set of values to which an iteration system can converge. Define a *finite iteration system* as an iteration system with finite state space.

Theorem 3. If all update-sequences of a finite iteration system $\mathcal{I}$ are p-fair, then we can effectively determine the convergence set.

Proof. For p-fair update-sequences, we can compute a fair transitive closure directly. We use the semiring $(\mathcal{R}, +, \cdot, 0, 1)$, where $+$ and $\cdot$ are the usual operations on real numbers. In a p-fair update-sequence, each choice of a non-empty subset of V has a non-zero associated probability, and the set of these probabilities sum to 1. Each edge (q,r) of $\mathbf{G}$ is labeled with the sum of the probabilities associated with those subsets that take q to r. Since any subset of variables may be updated in a given state, the probabilities on out-edges of each node of $\mathbf{G}$ sum to 1.

For $x \in (0,1]$, we define $x^* = \sum_{i=0}^{\infty} x^i = 1/(1-x)$, for $x \neq 1$, and 1 if $x = 1$. Since x is greater than 0, the associated cycle is eventually exited and a non-cycle path is taken. The effect of the value of x^* is to normalize the non-cycle paths.

Define the fair transitive closure $\mathbf{G}^F$ of G as having an edge (q,r) iff $c(q,r) > 0$. Consider the strongly connected components of $\mathbf{G}^F$. If such a component has an out-edge, that path has an associated probability greater than 0, and will eventually be taken. Let A be the union of all states in strongly connected components with no out-edges. Then the iteration system $\mathcal{I}$ converges to A. $\boxtimes$

Corollary. For p-fair update-sequences, $\mathcal{I}$ converges to a fixpoint iff the strongly connected components of $\mathbf{G}^F$ that have no out-edges are singletons.

Proof. Clearly, if the strongly connected components of $\mathbf{G}^F$ with no out-edges are singletons, then $\mathcal{I}$ converges. Suppose some strongly connected component of $\mathbf{G}^F$ with no out-edges has 2 or more states; then there is an update-sequence that visits more than 1 state infinitely often. $\boxtimes$

Probabilistic fairness excludes some non-fair update-sequences. With no fairness assumption, an iteration system converges to a fixpoint iff every cycle of $\mathbf{G}$ is a 1-cycle. A non-fair update-sequence can travel forever around a cycle, even when that cycle has an out-edge.

Theorem 4. If all update-sequences of a finite iteration system $\mathcal{I}$ are weakly fair, then we can effectively determine the convergence set.

Proof. For weakly fair update-sequences, we compute the transitive closure of G and then discard non-fair paths. Let L be the union of all sets of variables that take p to q; then the edge (p, q) is labeled $\{L\}$. For weak fairness, the semiring domain is sets of sets of variables. The additive and multiplicative identities are $\mathbf{0} = \emptyset$ and $\mathbf{1} = \{\emptyset\}$. Operators $\oplus$ and $\otimes$ are union and pairwise union, respectively.

$$s_1 \oplus s_2 = s_1 \cup s_2$$
$$s_1 \otimes s_2 = \{\, a \cup b \mid a \in s_1 \text{ and } b \in s_2 \,\}$$
$$s^* = \{\, \cup s \,\}$$

The transitive closure algorithm computes values $c(p, q)$, where each element of $c(p, q)$ is a maximal set of variables encountered on a path from p to q. The definition of s^* reflects the fact that repeated trips from s back to s can eventually visit all edge label sets on paths from s to s. Sets of sets of vertices are needed because multiple paths between pairs of states that do not lie on a cycle may involve disjoint sets of variables.

Consider a strongly connected component S of $\mathbf{G}$, and let p be an element of S. If $c(p, p)$ contains V, i.e., if all vertices can be visited in S, then an infinite fair update-sequence can remain indefinitely within S. Let A be the union of states in all such strongly connected components S. Then the iteration system $\mathcal{I}$ converges to A. ☒

Corollary. For weakly-fair update-sequences, $\mathcal{I}$ converges to a fixpoint iff whenever there is a strongly connected components S such that for $p \in S$, $c(p, p)$ contains V, then S is a singleton.

Proof. Suppose $\mathcal{I}$ always converges to a fixpoint and there is a strongly connected components S such that for $p \in S$, $c(p, p)$ contains V. Since a fair path exists through S, S must be a singleton. Suppose that $\mathcal{I}$ fails to converge to a fixpoint. Then some weakly fair paths visits two or more states infinitely often, so some strongly connected component of $\mathbf{G}$ contains 2 or more states. ☒

Theorem 5. If all update-sequences of a finite iteration system $\mathcal{I}$ are k-fair, then we can effectively determine the convergence set.

Proof. For k-fairness, the semiring domain is sets of sequences of sets of variables. Edge labels of $\mathbf{G}$ are the same as for weak fairness. Semiring operators $\oplus$ and $\otimes$ are union and pairwise concatenation, $\mathbf{0}$ is $\emptyset$, and $\mathbf{1}$ is $\{\epsilon\}$.

$$s_1 \oplus s_2 = s_1 \cup s_2$$
$$s_1 \otimes s_2 = \{a \cdot b \mid a \in s_1 \text{ and } b \in s_2\}$$
$$s^* = \mathbf{1} \oplus s$$

The transitive closure algorithm computes values $c(p, q)$, where each element of $c(p, q)$ is a sequence of sets of variables encountered on a path from p to q. The definition of s^* reflects the fact that repeated paths around a cycle will encounter the same set of edge sequences.

Consider a strongly connected component S of $\mathbf{G}$. If there is an element p in S such that $c(p,p)$ satisfies the k-fairness condition, then an infinite k-fair update-sequence can remain indefinitely within S. Let A be the union of all such strongly connected components S. Then the iteration system $\mathcal{I}$ converges to A. ☒

For k-fairness, it may not be the case that all elements in A occur infinitely often in some computation. Certain states in a strongly connected component might be reachable in a weakly-fair but not a k-fair computation. Thus the fixpoint convergence corollary of Theorem 5 is not an "if and only if" result.

Corollary. For k-fair update-sequences, $\mathcal{I}$ converges to a fixpoint if whenever there is a strongly connected components S and state $p \in S$, such that $c(p,p)$ satisfies the k-fairness condition, then then S is a singleton.

Proof. Similar to previous corollary. ☒

The complexity of the procedures we have outlined is dominated by the size of the state space, which is exponential in $|V|$. The basic transitive closure procedure is $O(b^3)$ for b states. Testing convergence for weak fairness is more complex than testing for p-fairness, and testing convergence for k-fairness is more complex than testing for weak fairness.

As noted, the tests we are considering do not give us the time to convergence. With p-fairness an expected time to convergence might be computed, but for weak and k-fairness, in general no bound can be given. After the first r rounds, the system can be in any one of at most b states. Recall a was the number of steps in the first r rounds, and that r is the rank of the dependency graph G, so $r < n$. If we assume $\mathcal{I}$ is weakly-fair, all we can say is that a is finite, and we have no fixed time bound. If we assume every update-sequence is k-fair, then $a \leq rk < nk$. Even with k-fairness, if a state p is in a strongly connected component with an out-edge, then p can be visited an arbitrary number of times before converging to a fixpoint q.

5. Generalized Iteration Systems

Generalized iteration systems (the "non-deterministic iteration systems" of [3]) are closely related to standard iteration systems. A generalized iteration system has two forms of non-determinism: variable-choice, and function-choice. A step in a generalized update-sequence can be represented as first choosing a set of variables, and then choosing a function (from the associated set) for each of the selected variables.

We give a construction which allows a standard iteration system to simulate a generalized iteration system in much the same way that a DFA can be constructed to simulate an NDFA. We then show that convergence to a fixpoint in the standard iteration system corresponds to convergence to a set in the generalized iteration system. The construction can be used to apply results of previous sections to generalized iteration systems; an example is given of a version of Theorem 2 for generalized iteration systems.

An update-sequence in a generalized iteration system is a sequence of sets of pairs, where each pair consists of a variable and the particular function that updates it. More

precisely, for each variable v, we have a *set* of update functions, $\{f_{v,1}, \ldots, f_{v,\ell}\}$. A *update-sequence* is a sequence of sets $T_0, T_1, \ldots$, where each T_j is a set of pairs of the form $(v, f_{v,i})$, with v a variable and $f_{v,i}$ one of its update functions. Each variable v can appear in at most one of the pairs in a set T_j.

Fairness criterion can be given for both variable and function selection. For example, for weak fairness, we require that each function $f_{v,i}$ (and so each variable v) appear infinitely often in the sequence. The dependency graph of a generalized iteration system has an edge set given by the union of the dependency relations of the individual functions, i.e., an edge from x to y iff at least one of the functions that update y depend on x.

Let $\mathcal{I}$ be a generalized iteration system with n variables, over domain D. We construct a corresponding standard iteration system $\hat{\mathcal{I}}$ as follows. Let $\{v_1, \ldots, v_n\}$ be the variables of $\mathcal{I}$, and $\{w_1, \ldots, w_n\}$ be the variables of $\hat{\mathcal{I}}$. The domain of variables in $\hat{\mathcal{I}}$ is 2^D, and functions in $\hat{\mathcal{I}}$ have type $F_w : (2^D)^n \mapsto 2^D$. For each variable v and each set of update functions $\{f_{v,1}, \ldots, f_{v,\ell}\}$ in $\mathcal{I}$, define F_w, the corresponding update function in $\hat{\mathcal{I}}$, as

$$F_w(w_1, \ldots, w_n) = \{f_{v,i}(a_1, \ldots, a_n) | a_j \in w_j, 1 \leq j \leq n, 1 \leq i \leq \ell\}.$$

Let $C = T_0, T_1, \ldots$ be an update-sequence of a generalized iteration system $\mathcal{I}$. Define $\pi(T_j) = \{v | (v, f_{v,i}) \in T_j, \text{ for some } i\}$, and $\Pi(C) = \pi(T_0), \pi(T_1), \ldots$ Thus $\Pi(C)$ is the sequence of sets of variables updated in C. Define the equivalence relation $\sim$ on update-sequences of a generalized iteration system $\mathcal{I}$ as $C_1 \sim C_2$ iff $\Pi(C_1) = \Pi(C_2)$, and define $[C_1] = \{C_2 | C_1 \sim C_2\}$.

A update-sequence of a generalized iteration system is *variable-fair* iff it is either weakly- or k-fair with respect to variable choice, with no restrictions made on function choice. In the following discussion, we assume that update-sequences of the generalized iteration system are only variable-fair. The following lemma shows how a standard iteration system can simulate such a generalized iteration system.

Lemma 2. Let $\mathcal{I}$ be a generalized iteration system, and $\hat{\mathcal{I}}$ the corresponding standard iteration system. Let $[C]$ be a class of variable-fair update-sequences of $\mathcal{I}$, and $\Pi(C)$ a update-sequence of $\hat{\mathcal{I}}$. Suppose all computations in $[C]$ start in state $(a_1, \ldots, a_n)$, and the computation $\Pi(C)$ starts in state $(\{a_1\}, \ldots, \{a_n\})$. Then state $(b_1, \ldots, b_n)$ is reachable in k steps by a computation in $[C]$ iff $b_j \in B_j, 1 \leq j \leq n$, for the state $(B_1, \ldots, B_n)$ reachable in k steps in the computation $\Pi(C)$

Proof. The proof is by induction on steps of $\Pi(C)$. For the basis, at step 0 we have $a_i \in \{a_i\}$, by assumption. Suppose the hypothesis is true at step k, and let $(B_1, \ldots, B_n)$ and $(B'_0, \ldots, B'_n)$ be the states of computation $\Pi(C)$ at steps k and $k+1$.

Let $(b'_1, \ldots, b'_n)$ be the state of a computation in $[C]$ at step $k+1$. Each b'_i was updated by some choice of function acting on a state $(b_1, \ldots, b_n)$ at step k of a computation in $[C]$. By the inductive hypothesis, each $b_j \in B_j$. By the definition of next state of a computation of $\hat{\mathcal{I}}$, B'_i contains all combinations of v_i's update functions applied to elements of $(B_1, \ldots, B_n)$. Therefore, $b'_i \in B'_i$.

Now, suppose $(b'_1, \ldots, b'_n)$ is any state such that $b'_i \in B'_i$. The value b'_i is the result of applying one of v_i's update functions to some state $(b_1, \ldots, b_n)$ such that $b_j \in B_j$. By the inductive hypothesis, $(b_1, \ldots, b_n)$ is reachable in k steps, in some computation of $[C]$. This computation will have next states corresponding to every possible choice of update functions, so that $(b'_1, \ldots, b'_n)$ will be the state of a computation of $[C]$ at step $k+1$. ☒

Suppose $\mathcal{I}$ is a generalized iteration system, and $\hat{\mathcal{I}}$ the corresponding standard iteration system. Suppose that from any initial state of the form $(\{a_1\}, \ldots, \{a_n\})$, $\hat{\mathcal{I}}$ converges to a single state $(B_1, \ldots, B_n)$. Then $\mathcal{I}$ converges to $A = \{(b_1, \ldots, b_n) | b_i \in B_i\}$. If $\hat{\mathcal{I}}$ converges to a single state of the form $(\{b_1\}, \ldots, \{b_n\})$ then the generalized iteration system $\mathcal{I}$ converges to the single state $(b_1, \ldots, b_n)$.

We can use Lemma 2 to apply theorems from the preceding sections to generalized iteration systems. For example, the following is a version of Theorem 2 for generalized iteration systems.

Theorem 6. Let G be the dependency graph of a generalized iteration system $\mathcal{I}$ and let U be a subset of V such at least one element of U lies on every directed cycle of G. If a variable-fair computation is U-convergent, then it converges to a set A.

Proof. Let $\hat{\mathcal{I}}$ be the corresponding standard iteration system. By hypothesis, for any initial state $(a_1, \ldots, a_n)$ the generalized iteration system will eventually reach a state where the components in U do not change. Let $v_j \in U$ be such a component. Then the corresponding standard iteration system will reach a state where $B_j = \{b_j\}$, a singleton. Thus the standard iteration system is also U-convergent, and so is convergent. Therefore the generalized iteration system converges to a set A. ☒

We can also represent any standard iteration system with a simple generalized iteration system, where variable-choice is replaced by function choice, as shown in the following proposition. The one-node, multiple-loop generalized iteration system used for the representation is very similar to the vector addition systems used to represent Petri nets [12].

Proposition 1. Every standard iteration system has an equivalent generalized system, with a single vector valued variable.

Proof. States in the original system are values in the multiple-update system. Given a regular iteration system $\mathcal{I}$, for each subset S of V, define $F_S(d_1) = d_2$ iff firing S in state d_1 results in state d_2. Every computation in the original iteration system has a corresponding computation in the multiple update system, where we choose functions instead of subsets. ☒

The relationship between fairness in the two models is as follows. Suppose the original computation is, for example, k-fair. There may still be some particular subsets of V that never appear in the computation, and so the corresponding generalized computation will not be fair. If a computation in the generalized system is k-fair (every subset appears within k steps), then the corresponding computation in the regular system is also k-fair.

6. **Conclusions and Future Work**

We have extended existing criteria for convergence of iteration systems, shown that convergence for a large class of iteration systems is decidable, for three distinct types of fairness, and shown how the convergence properties for two different version of the iteration system model are closely related. Convergence criteria are convenient, but are not always applicable, as we saw with the dining philosophers example. The testing method we introduce, though of exponential complexity in the number of variables, is relevant for several reasons: we found it interesting that some method exists to test convergence, since direct tests do not work, as we cannot always bound time to convergence; it is at least possible that faster tests for convergence exist, and finding some method could be a first step towards finding a faster method; and although slow, the method might still be useful for moderate-sized problems.

There are a number of directions for further work. It would be interesting to find further criteria for convergence, and faster algorithm for convergence testing. Questions about reachability can be posed for this model, as each class F of update functions has an associated reachability problem. For example, Lemma 1 can be used to show that reachability is decidable if at least one function per cycle has finite range. Finally, it seems possible that a more general theory of the closure of relations with fairness constraints could be developed, that would generalize these and similar results.

7. **Bibliography**

1. Afek, Y., and Brown, G. M. "Self-Stabilization of the Alternating Protocol." Submitted for publication.
2. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
3. Arora, A., Attie, P., Evangelist, M., and Gouda, M. "Convergence of iteration systems." In *Proceedings of the MCC Workshop on Self-Stabilizing systems* (Austin, Texas). MCC Corp., August, 1989.
4. Bondy, J. A., and Murty, U. S. R. *Graph Theory with Applications*. North Holland, 1976.
5. Brown, G., Gouda, M., and Wu, C. Token Systems that Self-Stabilize. *IEEE Transactions on Computers* 38, 6 (June 1989), 845–852.
6. Burns, J. E., and Pachl, J. Uniform Self-Stabilizing Rings. *ACM Transactions on Programming Languages and Systems,* 11, 2 (April 1989), 330–344.
7. Dijkstra, E. W. Self-stabilizing systems in spite of distriubted control. *CACM* 17, 11 ACM, (1974).
8. Dijkstra, E. W. Termination detection for diffusing computations. *Information Processing Letters* 11, 1 (August, 1980).

9. Dijkstra, E. W. A belated proof of self-stabilization. *Distributed Computing* 1, 5 Springer-Verlag, (1986).
10. Gouda, M. G. "The stabilizing philosopher: asymmetry by memory and by action." University of Texas at Austin, TR-87-12, 1987.
11. Harary, F. *Graph Theory*. Addison-Wesley, 1969.
12. Kosaraju, S. R. "Decidability of reachability in vector addition systems." (preliminary version), Johns Hopkins University, 1982.
13. Lehmann, D., and Rabin, M. O. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosopher's problem. ACM, 1981.

8. TEXmacro info

8.1. Table of Contents

1. Introduction 2
2. Iteration Systems 2
2.1. Definitions 2
2.2. An Example 3
2.3. The Dependency Graph of an Iteration System 4
3. Criteria for Convergence of Iteration Systems 4
4. Testing Convergence of Iteration Systems 8
5. Generalized Iteration Systems 12
6. Conclusions and Future Work 14
7. Bibliography 15
8. TEXmacro info 17
8.1. Table of Contents 17

8.2. Figure List

Figure Definitions

Figure 1 (“depgraph)
Figure 2 (“converse)

Figure Inserts

Figure 1 (“depgraph) pp 5
Figure 2 (“converse) pp 6

(Inserted figures should be in order!!)

8.3. Result List

Theorem 1. (“components) pp 6
Theorem 2. (“uconverge) pp 7
Lemma 1. (“finite) pp 8
Theorem 3. (“probconv) pp 10
Theorem 4. (“weakconv) pp 10
Theorem 5. (“kfair) pp 11
Lemma 2. (“general) pp 13
Theorem 6. (“genuconv) pp 14
Proposition 1. (“oneloop) pp 14