Choice Coordination with Multiple Alternatives (Preliminary Version)

David S. Greenberg[1], Gadi Taubenfeld[2], and Da-Wei Wang[3]

[1] Sandia National Labs, P.O. Box 5800, Albuquerque, NM 87185
[2] AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
[3] Yale University, New Haven, CT 06520

Abstract. The Choice Coordination Problem with k alternatives (k-CCP) was introduced by Rabin in 1982 [Rab82]. The goal is to design a wait-free protocol for n asynchronous processes which causes all correct processes to agree on one out of k possible alternatives. The agreement on a single choice is complicated by the fact that there is no *a priori* agreement on names for the alternatives. Furthermore processes must state their choice and do all communication via registers associated with the alternatives. We exactly characterize when the k-CCP can be solved deterministiclly, prove upper and lower space bounds for deterministic solutions, and provide a randomized protocol which is significantly better than the deterministic lower bound.

1 Introduction

1.1 The Choice Coordination Problem

A central issue in distributed computing is how to coordinate the actions of asynchronous processes. Coordination becomes even more difficult if as many as $n-1$ of the n processes can fail. The Choice Coordination Problem (CCP) [Rab82] highlights many of the difficulties inherent in such *wait-free* situations. Solutions to the CCP thus lend insight into how to coordinate asynchronous actions.

In the k-CCP, n asynchronous processes must choose between k alternatives. Each process has its own naming convention for the alternatives. A solution to the k-CCP is a protocol which guarantees that all correct processes terminate having chosen the same alternative. A slightly more concrete version of the k-CCP associates a shared register with each alternative. All inter-process communication must be accomplished by writing in these registers. However, the registers do not have global names; the first register examined and the subsequent order in which registers are scanned may be different for each process. A special symbol must be written in exactly one register and all correct processes must terminate pointing to this register. The efficiency of the protocol is defined by the number of different symbols which may be written in the registers.

It seemed intuitively obvious that adding more alternatives and hence more registers would just make the coordination more difficult. One of our results

is that, surprisingly, having more alternatives can lead to requiring fewer symbols. Besides giving protocols which take advantage of additional registers we exactly characterize the values of k and n for which the k-CCP can be solved by n deterministic processes, prove lower bounds on the number of symbols required by deterministic protocols, and provide randomized protocols which are significantly better than the deterministic lower bounds.

1.2 Computational Model

Our model of computation consists of a fully asynchronous collection of n processes. It is assumed that each process has an identifier but the identifiers need not be unique. Processes may fail only by crashing; that is, they fail only by never entering the protocol or by leaving the protocol at some point and thereafter permanently refraining from writing the shared registers. We require that all protocols be wait-free. That is, they can tolerate up to $n-1$ process failures.

All inter-process communications are via finite sized shared registers which are initially in a known state. Access to the shared registers is via atomic "read-modify-write" instructions which, in a single indivisible step, read the value in a register and then write a new value that can depend on the value just read. In the k-CCP each of the k alternatives has an associated register shared by all processes. The registers do not have global names; a single register may be considered the fifth register by one process and the eighth by another. Even the order of the names may be different. Thus one process may scan four alternatives in order 3, 2, 1, 4 while another scans 2, 4, 1, 3.

The lack of global names for the registers makes it is convenient to think of each process as being assigned an initial register and an ordering of the registers which determines how it scans the registers. An interesting special case is when all the orderings coincide. If all processes are assigned the same ordering (though potentially different initial registers) we say that the alternatives are arranged as a unidirectional ring. If all processes are assigned either one particular ordering or its inverse then the alternatives are said to be arranged as a bidirectional ring. Although in the bidirectional ring case the protocol can use the fact that all processes use a single ordering or its inverse there is no *a priori* agreement on which is the ordering and which the inverse.

Given the above definitions we now can formally define a solution to the k-CCP. A protocol is a solution to the general k-CCP if, for all possible orderings and initial registers assigned to the asynchronous processes, eventually the special symbol, e, is written in exactly one register and all correct processes terminate with a pointer to the register containing the e. A protocol is a solution to the unidirectional (bidirectional) k-CCP if the protocol solves the k-CCP when all processes are assigned the same ordering (or its inverse). Protocols requiring the least number of values in the shared registers are considered optimal. Other papers [BBD89] also measure internal memory size of the processes but we will not address this measure here.

1.3 Related Work

There are only two published papers about the k-CCP problem. Rabin's paper which introduced the problem [Rab82], and a paper by Bar-Noy, Ben-or and Dolev [BBD89]. Rabin is mainly interested in the case of $k = 2$ and $t = n - 1$. (k is the number of alternatives and t is the possible number of faulty processes.) He shows a deterministic protocol using $m = n + 2$ symbols (for each register) and a lower bound of $m \geq (n/8)^{1/3}$ for deterministic protocols. The upper bound can be modified to hold for any k. Also, as mentioned in [BBD89], the lower bound can be immediately extended to any t, obtaining $m = \Omega(t^{1/3})$. Rabin contrasts these deterministic results with a randomized protocol which, for m symbols, terminates correctly with probability $1 - 1/2^{m/2}$.

Bar-noy, Ben-or and Dolev extend Rabin's analysis to arbitrary t, examine the local storage requirements, and study a semi-synchronous model. They present deterministic and randomized protocols which both use $O(t^2)$ symbols. The randomized solution, which is for identical processes, terminates with probability 1 and is always correctly. The deterministic solution requires $O(2^n \log t)$ bits of internal memory. Another deterministic solution which requires only $O(\log n)$ bits of internal memory but $O(t^2 \log n)$ symbols is given. For $k = 3$ they showed that $n/2 + 3$ symbols is sufficient for a deterministic wait-free solution, and that for every prime k only $n/(k-1) + 3$ symbols are needed under the assumption that the registers are arranged as a unidirectional ring.

The k-CCP is related to the classic consensus problem[Fis83]. When all processes use the same set of names for the alternatives and the first register examined by a process in the k-CCP corresponds to the process' input in the consensus problem then a solution to the k-CCP is similar to a solution to consensus. However, in the consensus problem there is the additional requirement that the chosen alternative be the input of some process.

Impossibility results about the well-studied consensus problem therefore lend insight into the k-CCP. For example, Fischer, Lynch, and Paterson show that no consensus protocol can tolerate even a single crash failure in an asynchronous message-passing model [FLP85]. A similar result also holds for a shared memory model which supports only atomic read and write operations [LA87]. This last result can be used to show that: the k-CCP is not solvable in the presence of even a single crash failure if only atomic read and write operations are assumed. The Loui and Abu-Amara impossibility proof does not use the requirement that the chosen alternative be the input of some process but instead uses the weaker requirement that there are two runs in which different alternatives are chosen. Since it is easy to ensure that the k-CCP meets this weaker condition the impossibility for k-CCP follows.

1.4 Summary of Results

As mentioned, most of existing results examine only two alternatives (i.e., $k = 2$), and assume unique identifiers for deterministic protocols. We solve the problem for any number of alternatives, and also study the case where processes do not have unique identifiers. The two main questions that we try to answer are:

1. Assuming that there is no limitation on the size of each register, under what circumstances is the k-CCP solvable?
2. How many symbols for each of the k registers are necessary and sufficient to solve the k-CCP, as a function of k and n?

The answers to both these questions give a measure of the communication-space complexity of the problem and also provide a way of assessing the cost of achieving reliability. We give a brief overview of our results below.

Solvability Conditions for the k-CCP: Assuming that there is no limitation on the size of each register, we show that the k-CCP is solvable by a deterministic protocol if and only if the maximal number of processes having the same identifier is smaller than the least prime divisor of k. In proving this result, no assumption is made about the arrangement of the registers.

Deterministic Solutions – Upper Bound: When the registers are arranged on a ring we give a protocol which makes use of additional alternatives' registers to reduce the number of symbols required. For n processes and k alternatives (i.e., registers) our protocols use $O(n\ \eta(k)/k + \eta(k))$ symbols, where $\eta(k)$ is the number of prime factors of k counting duplicates as separate. Furthermore, we give a simple protocol which for $k > n^2$ uses at most 6 symbols.

Deterministic Solutions – Lower Bound: When the registers are arranged in a ring we show that all correct protocols must use at least $\sqrt[3]{n/k^3}$ symbols. When no assumption is made about the arrangement of the registers, we tighten the bound to $m \geq \sqrt[3]{n/4k}$.

Randomized Solutions: We present a randomized protocol that solves the k-CCP, and terminates with probability greater than $1 - 2^{-(m - \log k - 2\log\log k - 3)/2}$, where m is the number of symbols used by a protocol. Thus, if $k \leq 2^{100}$ a probability of success greater than $1 - 2^{-100}$ can be achieved for any number of processes using registers which are just 9 bits wide. In this protocol the processes are identical (i.e., have the same identifier) and no assumption is made about the arrangement of the registers.

2 Solvability Conditions for the k-CCP

A first question concerning the k-CCP is: Under what circumstances is the k-CCP solvable deterministically? In this section we show that the k-CCP is solvable if and only if the maximum number of processes having the same identifier is smaller than the least prime divisor of k. It is assumed that there is no limitation on the size of each register. In later sections the relation of register size to number of processes for which the k-CCP is solvable will be investigated.

More precisely, assume each process has an identifier but that the identifiers need not be unique. Let $\mathcal{N}$ be the maximum number of processes with the same identifier. Notice that when the processes are identical $\mathcal{N} = n$, and when they have unique identifiers $\mathcal{N} = 1$. Now recall that k is the number of registers and define $\ell(k)$ to be the least divisor of k which is greater than 1. The theorem below, gives a complete characterization for the solvable cases. (Notice that no assumption is made about the arrangement of the registers.)

Theorem 1. *For any k, n, and $\mathcal{N}$, there exists a deterministic protocol for n processes which solves the k-CCP if and only if $\mathcal{N} < \ell(k)$.*

Proof. We first assume that $\mathcal{N} \geq \ell(k)$, and prove that no solution exists. Pick a cyclic ordering of the k registers and divide it into $\ell(k)$ segments each containing exactly $k/\ell(k)$ registers. Pick $\ell(k)$ processes with the same identifier and assign one to the first register in each of the $\ell(k)$ segments. Now, schedule these and only these processes in a round robin fashion; each process, in turn, moves to the next register in the ordering and executes one read-modify-write operation. Because the processes are placed symmetrically, each operation of the first scheduled process is always followed by the *same* operation, on a different register, by each other process. Thus the original symmetry is restored after each round of the round robin. Therefore, if one process writes e into some register then at the end of the round each of these processes will write e in a different register.

Next, we assume that $\mathcal{N} < \ell(k)$, and show how to solve the k-CCP on a general graph. For simplicity we will start by assuming that $n = \mathcal{N}$, that is all processes have the same identifier. The general case of several different identifiers each associated with at most $\mathcal{N}$ processes is discussed at the end of the proof. The protocol for all processes having the same identifier is given in Figure 1.

In the protocol the constructs **lock** and **unlock** mark the beginning and end of atomic, exclusive access to the shared register at which pointer p is pointing. Each process can lock only one register at a time. The fact that the read-modify-write operation is wait-free and atomic is reflected by the assumption that a process does not fail between executing **lock** and the next **unlock**, and that any non-faulty process that reaches a **lock** instruction eventually executes it.

The goal of Procedure A is to break the symmetry of the processes. The full protocol logically divides the registers into n tracks and uses Procedure A repeatedly on different tracks in order to give each process a unique value. The unique values can then be used in a last track to identify a unique decision register.

In this abstract we only state without proof the main lemmas required by the proof of the theorem.

Lemma 1. *All processes scheduled k times in during a call to Procedure A terminate. Not all processes using Procedure A on the same track halt with the same value of c.*

An execution of Procedure A partitions the processes into groups depending on their final value of c. Although no process can know any other process' group number each can determine from the final configuration how many processes are in each possible group. The number of processes in group $i \geq 0$ (i.e., the number of processes where $c = i$) is simply the number of shared registers containing the value i minus the number of registers containing $i + 1$. Let $f(i)$ be the total number of processes in groups greater than i and $\lambda(i)$ be the number of processes in group i (let $f(-1) = n$).

```
shared C: ring of k registers range over n-tuples of integers;;
local S: array [1..k] of n-tuples of integers;
local i: integer {1...k};                    % pointer into S
local c: integer {0...k};                    % counter
local t: integer {1...n};                    % track
local r: integer {1...n};                    % minimum rank
local p: pointer which initially points to some arbitrary register of C;
t := 1; r := 1;
Repeat
    Execute Procedure A on track t returning c and S
    if f(c) = 0 then t = t + 1 else t := t + f(c);
    r := r + f(c);
Until λ(c) = 1
Write r on track n of any one register containing ⊥ if any exists;
Write 0 on track n of all registers containing ⊥ if any exists;
Write e on the register containing the maximum value in track n
end-protocol

% Using only track t of S divide processes into at least two groups
% Input: S,p,t, Output: S,p,c
Procedure A
    c := 0;
    % Write in tth track of as many registers as possible
    For i from 1 to k do
        lock
            if p↑= ⊥ then {c:=c+1 ; p↑:= c}; % change track t of p↑ only
            S[i] := p↑;
        unlock
        move p to the next register;
    end-for
end-procedure
```

Fig. 1. Solution to the k-CCP when all processes have the same id.

Procedure A breaks some of the initial symmetry, producing at least two non-empty groups. The full protocol applies Procedure A recursively to each group of processes until each process is in a group of its own at which point it can be assigned a unique identifier.

If each instantiation of Procedure A used the same registers then they might interfere badly. It is not possible to agree *a priori* on a partition of the registers in order to avoid interference. Instead each register is logically divided into n tracks; ie. the values in the registers can be thought of as n-tuples of track values. The full protocol arranges that all processes participating in a given instantiation of Procedure A use the unique track reserved for this instantiation. Since each instantiation uses its own track Lemma 1 will hold for all instantiations.

Lemma 2. *Each correct process eventually exits the main loop of the protocol in Figure 1 with a unique value of r.*

The proof of Theorem 1 follows easily from Lemma 2. The termination of the entire protocol is clear since the main loop terminates. The correctness follows from the fact that each process exits the main loop with a unique value of r ($1 \leq r \leq n$). In track n each process then writes its value at most once and at least one process writes its value. Thus there must be a unique maximum value in track n when it no longer contains any non-bottom symbols. Hence the value e is written exactly one.

We started the proof by assuming that $n = \mathcal{N}$, that is all processes have the same identifier. When there are several identifier values, each assigned to fewer than $\ell(k)$ processes then separate tracks must be assigned *a priori* for each potential identifier value.

An immediate consequence of Theorem 1 is that for any k and n, there exists a solution to the k-CCP, when the n processes have unique identifiers; and when k is even there is a solution only if the processes have unique identifiers.

A quick calculation shows that the protocol of Figure 1 requires a total of $(n+2)(k+1)^{n-1}$ symbols. There are several ways of saving symbols. Perhaps the easiest is to have processes not increment their counter above $k/2+1$ since this value must be unique. However the best solutions still use $(k/c)^n$ symbols for some constant c. When the registers are constrained to be visited as a unidirectional or bidirectional ring the number of symbols can be reduced to $\lceil (\#id \cdot \mathcal{N})/(\ell(k)-1) \rceil + 3$, where $\#id$ is the number of different identifiers.

3 Upper Bounds

In this section we examine the k-CCP when each process has its own unique identifier and the alternatives are arranged as a unidirectional ring. We present a protocol which, for a fixed n, reduces the number of symbols required as k increases. No more than $O((n \log k)/k + \log k)$ symbols are ever used by the protocol. When k has few factors then even fewer symbols are needed. In addition, when $k > n^2$ at most 6 symbols are used.

We write k's prime factorization as $k = \prod \rho_i^{e_i}$ (where $\forall i, \rho_i$ is prime and $\forall i \neq j, \rho_i \neq \rho_j$.) Let $\eta(k) = \sum_i e_i$; thus $\eta(k)$ is the number of prime factors of k, counting duplicates as separate. Note that if k is a power of 2 then $\eta(k) = \log_2 k$, and that $\log_2 k$ is the maximum possible value of $\eta(k)$.

Theorem 2. *The k-CCP can be solved by n processes having unique identifiers on a unidirectional ring using m symbols if*

1. $m \geq \lceil n/(k-1) \rceil + 3\eta(k) + \lceil 2n/(k-2) \rceil (\eta(k)-1)$, or
2. $k > n^2$ and $m \geq 6$.

In this abstract we give only a sketch of the proof of Part 1 of Theorem 2. The complete protocol on which the proof is based is given in Figure 2. The efficiency

of the protocol could be increased in several ways which would, however, made the proof less transparent. For example, it is not necesary to check if $i \leq \eta(k)$ in the guard of the main **while** loop and a process can often skip ahead to a later phase once it has determined that it has fallen behind.

Let S be the sequence $s_0, ..., s_{k-1}$. The ith rotation ($0 \leq i < k$) of S is the sequence $\mathrm{R}_i(S) = s_i, ..., s_k, s_1, ..., s_{i-1}$. A *lexicographically maximal rotation* (abbv. lmr) of S is any rotation $\mathrm{R}_j(S)$ such that for all $0 \leq i < k$, $\mathrm{R}_j(S) \geq \mathrm{R}_i(S)$ according to the standard lexicographic order.

Lemma 3. *If a sequence of length k has L lmrs then L divides k and each symbol occurs a multiple of L times.*

Recall that $\ell(k)$ is the least divisor of k which is greater than 1 (when k is prime $\ell(k) = k$). It follows from Lemma 3 that in a sequence of length k where some symbol occurs fewer than $\ell(k)$ times, there is a unique lmr. Furthermore, in a sequence of length k where some symbol occurs a number of times relatively prime to k, there is a unique lmr.

Ideas Behind the Protocol

We are now ready for an intuitive description of the protocol. Initially the k registers all contain the same known initial symbol and thus there are k lmrs. The protocol proceeds in phases; each phase reduces the number of lmrs. Lemma 3 shows that the number of lmrs is always a divisor of k, thus the number of lmrs must actually decrease in each phase by a multiple of prime factors of k. Even if each phase divides the number of lmrs by only a single prime factor of k then after $\eta(k)$ phases the number of lmrs would have to be reduced to one. Once a single lmr is achieved the processes choose the register at the start of the single lmr.

A different set of symbols is used in each phase. This allows processes which have been delayed to quickly determine the correct phase. In particular it prevents processes last scheduled in an earlier phase from interfering with a later phase. The protocol is nonetheless efficient because there are not many phases and relatively few symbols are used in each phase.

During each phase the processes must use these limited number of symbols to reduce the number of lmrs. In order to ensure that only a single lmr remains would require ensuring that some symbol occurs fewer than $\ell(k)$ times. Fortunately we need only to reduce the number of lmrs in each phase. In the first phase we only need to guarantee that some symbol occurs less than k times. Consequently, the first phase uses only $\lceil n/(k-1) \rceil + 2$ symbols.

It might seem that as the number of lmrs decreases, more symbols will be needed to further reduce the number of lmrs. For example, if every symbol must occur less often than the number of lmrs and the number of lmrs is 2 then n symbols are necessary. The protocol avoids this growth of number of symbols by taking advantage of symbol placement around the ring of registers.

shared C: ring of k registers range over pairs of integers, initially all $= (1, -1)$;
local S: array $[1..k]$ of pairs of integers;
local $i, j, val, pos, lmrs, bl, index$: integer;
local p: pointer which initially points on some *arbitary* register of C;
boolean $writeactive$: flag specifying whether active symbol needs to be written;
constant $marker = \infty$; $mfill = -1$; $afill = 0$; $e = (\infty, \infty)$;

$i := 1$; $val := \lceil id_p/(k-1) \rceil$; $lmrs := k$; $writeactive := true$;
while $(lmrs \neq 1)$ and $(i \leq \eta(k))$ **do**
  % `Ensure that an active symbol is written in this phase`
  lock if $writeactive$ and $(p\uparrow = (i, mfill))$ **then** $p\uparrow := (i, val)$ **unlock**;

  % `Loop A: Ensure completion of active phase by writing fill symbols`
  for j **from** 1 **to** k **do**
    move p to the next register;
    lock if $p\uparrow = (i, mfill)$ **then** $p\uparrow := (i, afill)$ **unlock** ;
    $S[j] := p\uparrow$ **end-for**

  % `If` S `holds only phase` i `symbols then`
  % `found an active configuration and must write a marker symbol`
  if $\forall j, S[j] < (i+1, -1)$ **then**
    $index :=$ index in S of the beginning of an lmr;
    move p forward $index$ times;
    lock if $p\uparrow < (i+1, -1)$ **then** $p\uparrow := (i+1, marker)$ **unlock**;

  % `Loop B: Ensure completion of marker phase by writing fill symbols`
  for j **from** 1 **to** k **do**
    move p to the next register;
    lock if $p\uparrow < (i+1, -1)$ **then** $p\uparrow := (i+1, mfill)$ **unlock**;
    $S[j] := p\uparrow$ **end-for**

  % S `holds a copy of` C`, possibly the phase` $i+1$ `marker configuration`
  % `Prepare to write active symbol in next phase if necessary`
  $i := i + 1$; $writeactive := false$;
  if $\forall j, S[j] = (i, marker)$ or $S[j] = (i, mfill)$ **then**
    $lmrs :=$ the number lmrs of S;
    if $lmrs = 1$ **then** $pos := 0$;
    else { $bl := k/lmrs$; $pos := 1 + id_p \bmod (lmrs - 1)$;
      $val := \lceil id_p/((lmrs-1)(bl-1)) \rceil$; $writeactive :=$ true} ;
    $index :=$ index in S of the beginning of an lmr;
    move p forward $index + pos$ times;
end-while
lock $p\uparrow := e$ **unlock**.

This protocol has several non-optimal features which were included in order to simplify the proof.

Fig. 2. Solution to the k-CCP — program for process p (with identifier id_p).

A sequence containing L lmrs can be divided into L equal blocks of $b = k/L$ registers such that the cyclic sequence beginning at the start of each block is an lmr. Instead of a phase ensuring that some symbol occurs fewer than L times it ensures that not all blocks are the same. Since all blocks will not be the same, not all blocks can still begin lmrs and the number of lmrs is reduced.

One detail omitted in the previous paragraph is that we need to guarantee that lmrs begin only at registers which are the beginning of lmrs of the previous phase. Thus we add a special marking subphase at the beginning of each phase. The marking subphase marks, with the largest symbol possible for the phase, some of the registers at the beginnings of lmrs in the previous phase; thus only these marked registers can be at the start of lmrs in the new phase.

It is also possible to show that at most $log_{k/n}k$ symbols are needed, when $k > n$. In this case, in each phase the number of lmrs must decrease by a factor of n/k and thus the number of phases is at most $\log_{k/n} k$. In addition each phase requires at most one active symbol.

Part 2 of Theorem 2 depends on a very simple protocol. The remainder of this section contains a description of the protocol along with intuitive reasons why it works.

The six symbols are denoted $\perp, a, \texttt{fill-a}, b, \texttt{fill-b}$, and e. Initially all registers contain $\perp$. Each process then writes a in its first register if the register still contains $\perp$ and writes `fill-a` in every other register which still contains $\perp$. Having done a Read-modify-write on each register once it now has a record of all registers in which a will ever be written.

Since $k > n^2$ and at most n registers contain an a (each process writes a at most once) there must be at least one register containing a followed by n registers containing `fill-a`. Process i moves to the ith register containing `fill-a` in such a gap and writes b if the register still contains `fill-a`. Thereafter it writes `fill-b` on every register still containing `fill-a`. Now, having performed a Read-modify-write on each register twice the process has a record of the final contents of all the registers before e is written.

There is a unique b which is furthest from its preceeding a (and hence an unique lmr). Since the identity of b is determined by relative distance from its preceeding a all processes agree on its identity despite there different names for the registers. Thus each can write e in the register containing this special b.

4 A Lower Bound

In this section, we establish a lower bound on the number of symbols required to solve the k-CCP. Our bound generalizes Rabin's lower bound for $k = 2$ to apply to general k. We examine both the case where it is known that the registers are arranged as a ring and the case where nothing is known about the registers' arrangement.

Theorem 3. *Let P be a protocol for n processes that solves the k-CCP. Let m be the number of symbols used by P. Then,*

1. $m \geq \sqrt[3]{n/k^3}$, *assuming the registers are arranged as a unidirectional ring;*
2. $m \geq \sqrt[3]{n/(4k \log k)}$, *when no assumption is made about the arrangement of the registers.*

The proof relies on two key properties of the protocols. The wait-free property ensures that any process run sufficiently long will eventually write a new symbol until it finally writes e. The "anonymity" of the registers ensures that if all registers contain the same symbol then a process will take the same steps regardless of its initial register and, in the general case, regardless of the order in which the registers are visited.

Informally, the proof employs an adversary which progressively extends a run to lead from one configuration in which all registers contain the same symbol to another such configuration. Each configuration uses a symbol not used in the previous configurations. Eventually a configuration is reached with two registers containing e. The wait-free property allows the adversary to find processes which, when added to the run, will write new symbols and the anonymity property allows it to force these processes to write the symbol in every register.

In order to extend the run to the next configuration the adversary needs many processes which have not yet been used in the run. Protocols using more symbols require the adversary to go through more intermediate configurations before reaching one containing two e's. Thus if more symbols are used than more processes can be allowed to participate without including enough for the adversary to force the protocol to fail.

The details of the adversary, which is a modified version of the one used by Rabin[Rab82], are omitted from this abstract. In order to achieve bounds when $k > 2$ we needed new combinatorial lemmas which bound the cost to the adversary of forcing processes to write in particular registers.

5 Randomized Protocol

We have seen that for large values of k, compared to n, the number of symbols used by a deterministic protocol in the ring case can be kept relatively small. Our lower bounds show, however, that the number of symbols must grow as a function of n. It is natural to ask if the number of symbols can be reduced if a small probability of non-termination is allowed. In his seminal paper Rabin showed that randomization could reduce the number of symbols when $k = 2$. In this section we show that, for any value of k, randomization can reduce the number of symbols needed.

Our strategy for randomized protocols draws many ideas from our deterministic protocols. Randomization allows us to simultaneously reduce the number of symbols required, dispense with process identifiers, and succeed regardless of whether or not the registers are arranged in a ring. For example, if $k \leq 2^{100}$ then

a probability of success greater than $1 - 2^{-100}$ can be achieved for any number of identical processes using registers which are just 9 bits wide.

Theorem 4. *For any k, n, and security parameter q, there exists a randomized protocol for n processes using $m = 2q + \log \min\{k, n\} + 2 \log \log \min\{k, n\} + 3$ symbols which solves the k-CCP for n processes and terminates with probability greater than $1 - 2^{-q}$.*

The proof of Theorem 4 is based on the analysis of the protocol given in Figure 3. This protocol was designed to make the analysis of the worst case simple. There are many obvious optimizations which improve its behavior on fortuitous runs. Before analysing the protocol we give an informal description of it and an intuitive idea of why it works.

The Protocol

The randomized protocol, like the deterministic one, works in phases. Each phase uses some new symbols in an attempt to reduce the number of registers which may end up containing e. If in a predetermined number of phases the number of *live* registers (ones in which e may eventually be written) is reduced to one then the protocol succeeds. The use of more symbols per phase or more phases will increase the probability of success. Thus given a bound on allowable failure probability we can ask what is the smallest number of symbols which can attain this failure probability.

More specifically, in each phase each process picks a register which survived the previous phase (we call these registers *live*), writes in this register a random symbol from those symbols assigned this phase, and then attempts to write the *kill* symbol in all registers containing symbols from earlier phases. Intuitively, it has ensured that at least one register survives the phase and then tried to kill all other registers. As in the deterministic protocol we use a lock construct to denote the read-modify-write. Inside the lock the process may generate a random value. The choice being made inside the lock is meant to denote that we assume that the scheduling of the read-modify-write cannot depend on the choice of random value.

A phase completes when all registers contain symbols from the same phase or the kill symbol. All registers containing the maximum-value, least-frequently-occurring symbol are considered to have survived the phase. If only one register survives (recall that at least one must survive) then the protocol has completed successfully and this register is chosen. Otherwise, if more phases remain the next phase is begun. If this was the last phase then the protocol fails.

As in the deterministic protocols, care must be taken to make sure that all processes, no matter how long they are delayed, make consistent decisions. In particular any processes deciding to go to the next phase must agree on which registers survived the previous phase.

The simplest version of the protocol uses two symbols at each phase. By bounding the number of phases required to, with high probability, successively

```
shared C: ring of k registers range over pairs of integers, initially (0, 0);
local S: array [1..k] of pairs of integers;
local i: integer {1...k};                    % pointer into S
local phase: integer;                        % # of phases
local number: array[0..1] of {0...k};        % number of live registers
local live: boolean;                         % value of the live register
local p: pointer which initially points on some arbitary register of C;
constant H = q + log min{k, n} + log log min{k, n}; kill = (−1, −1);
function RANDOM;                             % Returns 0 0r 1 at random

phase := 1; number[0] = k; live = 0;
while number[live] ≠ 1 and phase ≤ H do
    % Try to write new value on a live register
    lock
        if p↑= (phase − 1, live) then p↑:= (phase,RANDOM);
        (phase, live) := p↑; % Note that both phase and live are set here.
    unlock
    S[1] := (phase, live); move p to the next register; i := 2;

    % Look for a post-phase configuration
    while (p↑≠ e) and (i ≠ 1) do
        lock
            if p↑< (phase, −1) then p↑:= kill
            elseif p↑≥ (phase + 1, 0) then i := 1;
            (phase, live) := p↑;
        unlock
        S[i] := (phase, live); move p to the next register;
        if i = k then i := 1 else i := i + 1;
    end-while

    % Compute the number and value of the live registers
    number[0] = 0; number[1] = 0;
    for i = 1 to k do
        if S[i] = (phase, 0) then number[0] := number[0] + 1
        elseif S[i] = (phase, 1) then number[1] := number[1] + 1
    end-for
    if number[1] ≥ number[0] then live = 1 else live = 0;

    % Move to a register which contained live when S was read
    while S[i] ≠ (phase, live) do
        move p to the next register;
        if i = k then i := 1 else i := i + 1;
    end-while
    phase := phase + 1;
end-while
lock if number[live] = 1 then p↑:= e unlock.
```

Fig. 3. Randomized Solution to the k-CCP.

halve the number of surviving registers the upper bound on number of symbols required for the entire protocol given in Theorem 4 is achieved.

The protocol can be fine-tuned by varying the number of symbols used by each phase. For example, in the $k = 2$ case studied by Rabin, the protocol in which processes randomly choose, at each phase, between three symbols to write yields a probability of termination which is slightly greater then the probability in Rabin's protocol. If the registers are known to be arranged on a ring the ideas from Section 3 can increase the termination probability even more.

Using a recent result of Karp[K91] it is possible improve our analysis to show that fewer phases and thus fewer symbols achieves the same success probability. Karp's result on probabilistic recurrence equations shows that $q + \log \min\{k, n\}$ phases suffice.

6 Conclusions and Open Questions

We have examined the k-CCP under varying assumptions about the structure of the alternatives and about the relative values of the number of alternatives and the number of processes. Our results generalize and extend most of the previously known results to the case where more than two alternatives are available. A major surprise of these extensions is that when more alternatives (and hence more registers) are added the complexity of the problem does not necessarily increase. In fact, in certain cases more alternatives means that fewer symbols are used for each register.

Several variants of the k-CCP remain as interesting open questions. Our deterministic protocols are correct even if the initial states of the registers are not known. However, the size of the registers is no longer bounded; can this be improved? When the atomic operations are just reads and writes (rather than read-modify-write) then the presence of even a single crash failure implies that there is no solution to k-CCP. What is the effect of using an intermediate strength operations such as test-and-set?[1] It is not difficult to extend our wait-free protocols to create protocols which tolerate up to $t < n - 1$ failures with costs which decrease as t decreases.

Acknowledgements

The authors would like to thank Sandeep Bhatt, Mike Fischer, Michael Merritt, Nick Reingold, Jeff Westbrook, and Lenore Zuck for helpful discussions.

This work was supported in part by NSF/DARPA grant CCR-8908285, NSF grants CCR-8807426, CCR-8910289, and IRI-9015570, AFOSR grant 89-0382, ONR contract N00014-89-J-1980, DOE contract DE-AC04-76DP00789, a Hebrew Technical Institute scholarship, and an IBM Graduate Fellowship.

[1] In the test-and-set operation the read together with the write is atomic but the the value written cannot depend on the value read.

References

[BBD89] A. Bar-Noy, M. Ben-Or, and D. Dolev. Choice coordination with limited failure. *Distributed Computing*, 3:61–72, 1989.

[Fis83] M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In M. Karpinsky, editor, *Foundations of Computation Theory*, pages 127–140. Lecture Notes in Computer Science, vol. 158, Springer-Verlag, 1983.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[K91] R. M. Karp. Probabilistic Recurrence Relations. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 190–197, 1991.

[LA87] C. M. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[Rab82] M. O. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.

This article was processed using the LaTeX macro package with LLNCS style