Cliques in Hypergraphs and Mutual Exclusion using Tokens

Edward T. Ordman
Department of Mathematical Sciences
University of Memphis*, Memphis, TN 38152 U.S.A.

Abstract

Token-passing algorithms are a well-known way of solving distributed mutual exclusion problems in computer networks. A simple abstraction of the concept of tokens allows the use of elementary constructions in general hypergraphs to show that certain sets of tokens are minimal. This suggests other problems about hypergraphs worthy of exploration. As an application, we introduce a new mutual exclusion problem, the *Excluded Taxpayer Problem*, which requires exponentially many tokens even though it can be solved in linear time by other methods.

1 Introduction.

The aim of this paper is twofold: first, it describes a computer science problem in sufficent detail to raise some graph theory problems, and takes small steps toward solutions. Second, it demonstrates the utility of these techniques by introducing a new mutual exclusion problem (the *Excluded Taxpayer Problem*, in section 5) which distinguishes the power of two programming techniques; the graph theory results show that this problem requires exponential effort to solve using token systems, although it can be solved with linear effort using shared variables.

In a collection of computer processes, it may be that not all the processes can be active at one time. For example, several processes want to use the printer but only one at a time can do so. Many studies of this problem in

*A recent name change from Memphis State University.

JCMCC 19 (1995), pp. 209-224

the computer science literature restrict themselves to the special cases in which the pattern of mutual exclusion can be modelled by a graph. (e.g., [2, 6, 8].) In this model, two processes are connected by an edge if and only if the two cannot be active at the same time. (If five processes are trying to use the same printer, of course, no two can do so at the same time, so all of the edges are present: the graph is a clique.) There has been a productive interaction between graph theory and computer science resulting from the search for classes of graphs for which nice mutual exclusion algorithms exist, e.g. [2, 8, 10].

There are a few studies of cases not modelled by a graph. The best known is *the k of n entry problem* (see section 4.2), in which there are n processes and at any time, at most k of them are allowed to be active. See, for example, [7, 12, 13]. This is not modelled by a graph, since no two processes exclude each other. It is easily modelled by a hypergraph.

There are relatively few studies of more general cases. One reason for this is that the general case seems to present mathematically difficult questions. One purpose of this paper is to phrase some of the interesting examples and questions in terms of hypergraphs, illustrate some useful results, and ask some questions about hypergraphs whose answers would help with the computer science problems.

The reader with even a modest acquaintence with graphs and hypergraphs may find it useful to read section 2.3 first, for motivation.

This research was initially motivated by discussions with Abdelmadjid Bouabdallah and Mohammed Naimi which took place during a visit by the author to the University of Paris at Orsay. The visit was supported by grant INT-9115870 of the National Science Foundation.

2 Definitions and motivating examples.

2.1 Graphs.

Definition 1. *Our graphs will always be simple undirected graphs without loops or multiple edges. If G is a graph a typical edge of G will be denoted as a pair (a, b); note that $(a, b) = (b, a)$. A clique or complete subgraph in G will typically be denoted (a, b, c, d) (the number of vertices is two or more); this is a subgraph, not necessarily maximal, such that any two vertices in it are connected by an edge in G.*

We will speak of a clique quite casually, as if it consists of the vertices $a, b, \ldots$ OR as if it consists of the edges $(a, b), (a, c), (b, c), \ldots$; we say it

contains both the vertices and the edges. By the *size* of a clique we mean the number of vertices in it; a clique of size n has $n(n-1)/2$ edges.

Definition 2. *A* clique covering *of a graph G is a set of cliques $C = \{C_1, C_2, \ldots, C_k\}$ such that (1) every edge of any C_i is an edge of G, and (2) every edge of G is an edge of at least one C_i.*

A mutual exclusion problem (these will be defined below) often arises in the form of a list of sets of processes no two of which can perform some action "at the same time". This situation may sometimes be modelled by a graph given by means of a clique covering. Thus, instead of representing a graph by a set of edges, such as $G = \{(a,b),(a,c),(b,c),(b,d),(c,d)\}$, we may choose to represent it by a set of cliques of a clique covering. The following are both notations for the graph just given: $\{(a,b,c),(b,d),(c,d)\}$ and $\{(a,b,c),(b,c,d)\}$. The last of these includes two overlapping cliques, which is expressly allowed. Note that $G \neq \{(a,b,c,d)\}$ since (a,d) is not an edge of G.

2.2 Hypergraphs.

A warning: the notation given here for hypergraphs, and particularly for cliques in hypergraphs, is nonstandard. We introduce this nonstandard construction expressly to simplify the representation of mutual exclusion problems.

Definition 3. *Given a set S, a* hypergraph *H on S is a collection of subsets of S such that each $H_i \in H$ has at least two elements.*

An element of H is called a hyperedge *of H. We will say a hyperedge H_i* contains *a hyperedge H_j if $H_i \supseteq H_j$. The* cardinality *of a hyperedge is the number of vertices in it.*

As with graphs, we will casually denote a hyperedge as (a,b,c,d) rather than the more formally correct $\{a,b,c,d\}$. A hypergraph is *uniform* if all its hyperedges have the same cardinality.

Definition 4. *A* hyperclique of size k and cardinality e *will be typically denoted $A = ((a_1, \ldots, a_k), e)$. It includes every hyperedge of cardinality e selected from the set of vertices $\{a_1, \ldots, a_k\}$.*

Thus a graph clique (a,b,c,d,e) is a special case $((a,b,c,d,e),2)$. A hyperclique is a uniform hypergraph. A hyperclique of size k and cardinality e contains k *Choose* e edges.

The hypergraph H *contains* the hyperclique A if every vertex of A is in H and every hyperedge of A is in H. In particular, if B and B' are two sets

of at least e vertices each and $B \supset B'$, then the hyperclique (B, e) contains the hyperclique (B', e).

Definition 5. *A set $A = \{A_1, \ldots, A_k\}$ of hypercliques will be called a* hyperclique covering *of the hypergraph H if*
1. *Every hyperedge of each A_i is a hyperedge of H.*
2. *Every hyperedge of H is a hyperedge of some A_i.*

We can now represent a hypergraph, like a graph, by listing *either* the hyperedges *or* any hyperclique covering of the hyperedges. For example, let $S = \{a, b, c, d\}$ and $H = \{(a, b), (a, b), (b, c), (b, c, d)\}$ Then H could also be represented as $\{((a, b, c), 2), ((b, c, d), 3)\}$.

2.3 Mutual Exclusion Protocols.

We take the following formulation of the *mutual exclusion problem* as a motivation below. Suppose that there is a set S of processes $S_1, S_2, \ldots, S_n$. For purposes of this paper we may suppose each is a program running on its own computer. Each process has a special part called a *critical section*. We suppose that certain sets of these processes may not all be in their critical sections at one time. We will call such a set of processes *mutally excluding*. We suppose that if a set of processes is mutually excluding, so is any larger set (superset).

Informally, we will say that *process k runs* as an abbreviation for *process S_k is in its critical section.*

A mutual exclusion problem may in practice be described in a variety of ways; we will see some examples below. Clique coverings of graphs were stressed above because often the description of a problem contains, or can be rephrased as, something resembling "processes 1 and 2 cannot run at the same time; processes 1 and 3 cannot run at the same time; no two of processes 3, 4, and 5 may run at the same time." This is of course just a description of an *exclusion graph* with clique covering $\{(1, 2), (1, 3), (3, 4, 5)\}$. The statement "no four of processes 2, 4, 6, 8, 10, or 12 can run at once" is a description of an *exclusion hypergraph* containing all sets of four processes from that set – that is, the hyperclique $((2, 4, 6, 8, 10, 12), 4)$.

In a common way of managing this problem, each process goes through a preliminary algorithm called an *entry protocol* before entering its critical section, to determine whether it can enter without causing a mutually excluding set of processes to be running at once. On finishing its critical section, it usually executes an *exit protocol* which may, for example,

involve telling another process it may now enter. It may then perhaps engage in other activities (its *remainder section*) before possibly deciding to try to enter its critical section again. Entry and exit protocols may involve accessing shared variables, passing messages, or other ways of exchanging status information. For more detailled discussions and motivations see, e.g., [1].

For the present paper, in order to stress the hypergraph questions, we propose a drastically simplified model of an entry protocol. We suppose the existence of a multiset (set with repetitions allowed) T of *tokens* which will be used to regulate entry into the critical sections. Each process S_i has associated with it a set T_i, a subset of T, such that S_i may enter its critical section if and only if it has obtained the tokens in T_i. Thus we suppose the existence of, let us say, a *bowl* B initially containing all the tokens of T. One process S_i at a time examines the bowl; if it finds the tokens T_i it needs, it takes them from B and enters its critical section; if not, it waits until a later time to try again. Once S_i is done examining the bowl, another process has a turn to do so. Any process exiting its critical section returns its tokens to the bowl, so that they may be used by another process. Only one process at a time may work with the bowl. More formally:

Algorithm: **Mutual exclusion using tokens**
For each Process:
Global shared data *Bowl* {*a multiset of tokens*}
Local variable *Success initially True*
Begin
entry protocol:
    Lock(*Bowl*);
    for each token in my set do
        if present in bowl
            then remove it from bowl;
            else set *Success* to *False*;
    end for
    Unlock(*Bowl*);
    if *Success* then enter Critical Section
        else wait and try again later;

...
critical section
...
exit protocol:
    Lock(*Bowl*);
    return my tokens to *Bowl*;
    Unlock(*Bowl*);

end.

Given an appropriately careful definition, any mutual exclusion problem may be solved (i.e., the exclusion enforced) by an appropriate system of tokens; we prove a very simple version of this in the next section and leave a more complete discussion for elsewhere. The primary purposes of this paper are to establish some conditions under which a set of tokens is minimal, to present an interesting example, and to state some problems about hypergraphs that arise along the way.

One part of the search for efficient mutual exclusion algorithms involves finding classes of mutual exclusion problems and functions that will rapidly compute whether a set of processes P contains any mutually excluding set. By "rapidly" we mean, loosely, in time polynomial (preferably of low degree) as a function of the number of processes or as a function of the information available about the collection of mutually excluding sets. Often, finding such a function turns on finding a concise description of the collection of mutually excluding sets. The collection of mutually excluding sets, of course, is a hypergraph. We summarize as follows:

Problem 1. *Find quick exclusion functions.* Given a set of vertices S and a hypergraph H on those vertices, find a function $\mathcal{I}_H$ which, given a subset S' of S, will quickly determine whether S' contains any hyperedge of H. (We may regard $\mathcal{I}_H(S')$ as a boolean function, with the value *true* if S' contains a hyperedge.)

The Taxpayer Exclusion Problem in section 5 below illustrates this well; there we construct a hypergraph on $n = 300$ vertices, with a great many hyperedges (3^{100} hyperedges of cardinality 100) and give an algorithm which determines whether a set of vertices contains a hyperedge and runs in time and space linear in n.

Problem 2. *Find quickly excluding classes of hypergraphs.* Find classes of hypergraphs for which there exist quick exclusion functions, and find the functions.

Hypercliques are clearly such classes. Among graphs, threshold graphs (discussed in section 4.3 below) are an excellent example; see for example [6].

Problem 3. *Identify and parametrize a hypergraph.* Given a hypergraph, discover quickly if it is in one of the quickly excluding classes, and determine its quick exclusion function.

For graphs, for example, we know that threshold graphs can be recognized in linear time [3, 6, 11]. In section 4.3 we define threshold hypergraphs similarly, so as to satisfy Problem 2; but it is less clear whether one can recognize a threshold hypergraph quickly enough to usefully solve Problem 3.

The rest of this paper consists of showing how some particular mutual exclusion problems look when presented this way, and then poses some questions on hypergraphs whose answers would allow the easy solution of more extensive classes of mutual exclusion problems.

Real protocols are of course often more complex, but the model suggested here is of interest since (1) it raises interesting combinatorial and hypergraph questions, and (2) it can be used to show some limitations of the use of tokens. The reader wanting to know about more realistic algorithms may want to look at a distributed processing textbook, e.g. [1].

3 Tokens for general hypergraphs.

We need a bit of terminology and notation. A *set* has no repeated elements, while a *multiset* may have repeated elements: $\{a, b, c\}$ is a set of three elements and $\{a, b, b, c\}$ is a multiset of four elements, with three distinct elements. When we must distinguish multiset union from set union, we denote it $\cup^*$, so that $\{a, b\} \cup \{b, c\} = \{a, b, c\}$ while $\{a, b\} \cup^* \{b, c\} = \{a, b, b, c\}$.

Unless otherwise specified, $\subset$ and similar symbols refer to multisets whenever a multiset is involved in the comparison; hence it is *false* that $\{a, b, b, c\} \subseteq \{a, b, c\}$.

The repeated elements of a multiset are said to be *of the same type*; when we use a multiset of tokens, we use the same phrase to denote indistinguishable tokens.

We begin by showing that, given an arbitrary mutual exclusion hypergraph, there is a (possibly very large) token system that will enforce the mutual exclusion.

Theorem 3.1 *Let H be a hypergraph on a set S. There is a multiset T and there are sets T_i such that $T_a \cup^* \ldots \cup^* T_b \subseteq T$ if and only if $\{S_a, \ldots, S_b\}$ contains no hyperedge of H.*

PROOF: For each hyperedge H_k of cardinality f of H we introduce $f - 1$ tokens t_k. (The $f - 1$ tokens associated with a given hyperedge are of the

same type). All these tokens together form T. For each S_i, let T_i contain one copy of t_k if and only if S_i is a vertex of the hyperedge H_k. Now no set of S_i's containing a hyperedge H_k can obtain all their tokens at the same time, since there are not enough t_k's available. But any set of S_i's not containing a hyperedge can get enough tokens; simply pass out the tokens to them "one hyperedge at a time" to see that there are enough. □

The above method may use far too many tokens. For example, if five processes are trying to share one printer, this would assign ten tokens – one for each edge in the graph. Each process would demand four tokens (to lock out the other four processes.) But the graph is a clique, and only one one token is needed! This fact easily generalizes.

Proposition 3.2 *Let H be a hypergraph on S and let D be a hyperclique covering of H. Then mutual exclusion can be enforced by a system of tokens with one token type per hyperclique, and each token having one fewer copies than the cardinality of its hyperclique.*

PROOF: For each hyperclique C_k of cardinality f, create $f-1$ tokens T_k. Let T_i include a single t_k if and only if S_i is in C_k. It is easy to check that a set of S_i's can each obtain their required tokens if and only if the set intersects every hyperclique C_k in a set of vertices less than the cardinality of C_k, as desired. □

There is a converse to this result. Any token system gives rise to a clique covering. Given a set S of processes, a multiset T of tokens, and a set T_i of tokens wanted for each S_i, we define a hypergraph as follows: For each set of indistinguishable tokens t_k in T (suppose there are j copies of t_k), and each S_i having a copy of t_k in T_i, we let the set of those S_i form a hyperclique of cardinality $j+1$. The set of all such hypercliques form a hyperclique covering of a hypergraph H.

Theorem 3.3 *Let H be a hypergraph on S corresponding to a mutual exclusion problem. If there is a hyperclique covering of H by exactly c hypercliques, then there is a token system for the mutual exclusion problem with exactly c types of tokens, and conversely.*

The reader should be warned that this theorem was relatively easy because of the restriction that each process wanted only a *set* and not a *multiset* of tokens: that is, it wanted no more than one token of each type. If we replace the sets T_i with multisets, the sets of indistinguishable tokens no longer induce a covering by hypercliques, but rather by threshold

hypergraphs; see section 4.3 below. The author has explored [8, 10] the relationship between these two coverings for graphs, but the situation for hypergraphs appears to be more difficult.

4 Simple examples.

4.1 Cliques in Graphs: Simple Mutual Exclusion.

Example 1. *Classical Mutual Exclusion.* [5] There are n computers wanting to use the printer; only one can do so at a time. There is exactly one token, $T = \{t\}$, every $T_i = \{t\}$, and the time needed for a process to determine if the token is available is $O(1)$.

The above example works because the minimal mutually excluding sets include all pairs of processes. The mutual exclusion graph is a clique. The following result, then, is trivial.

Proposition 4.1 *The set of cliques is a quickly excluding set of graphs. For any clique C and any subset S' of S, $\mathcal{I}_C(S') = true$ if S' contains at least two elements.*

Can we easily recognize members of this class? While finding a minimal clique cover of an arbitrary graph is NP-hard, it is in fact easy to check whether a given graph is a single clique. How long it takes, however, may be a function of how the graph is given to us. If it is given as a single clique, $\{(1, 2, \ldots, n)\}$, we merely need read the description (of length $O(n)$) to check; the work involved is $O(n)$. If we are given a full list of edges, $\{(1,2),(1,3),\ldots,(n-1,n)\}$ then the length of the description in $O(n^2)$ and we must do that much work merely to read it. In fact, if we are given an unreasonably large clique covering of a graph (say, the entire power set of the vertex set, less singletons, with all cliques that include vertex n placed at the end of the list) we may have to read nearly the entire list (surely at least $O(2^n)$ characters) before we can confirm that all edges are present.

Let us suppose that D is a description of a graph G given by a clique covering (or of a hypergraph given by a hyperclique covering.) Let d be the length of D, in characters. The above argument suggests that it is probably unreasonable to try to establish whether a graph G is a clique in less work than $O(d)$, for some D with d in the range $O(n^2) < d < O(2^n)$. On the other hand, it may take somewhat more than $O(d)$ steps, if D has many

cliques that must be expanded. Can this be made more precise? Here is a small step:

Algorithm 1. *Let D be a clique covering description of a graph G on n vertices, and suppose D has k cliques. Then we can determine if G is a single clique in time no more than $O(kn^2)$.*

Here is an algorithm. Note that if n is not given initially, a first phase may be added: read the entire description D and set n to be the largest vertex label found. If the names of vertices used in D are not consecutive natural numbers, count the labels that are found and make a substitution.

Algorithm: **Does a clique covering describe a clique?**
integer *counter initially 0*;
array $E[i,j]$; $1 \leq i < j \leq n$ *initially 0*;
begin
  for each clique c in D **do**
    for each edge $(r,s),(r<s)$, of the clique c **do**
      begin if $E[r,s]=0$ **then**
        begin *counter*=*counter*+1; $E[r,s] := 1$;
          if *counter* $= n(n-1)/2$ **then** THE GRAPH IS A CLIQUE;
        end;
      end;
end.

Loosely, we examine each clique in turn, making an entry in the adjacency matrix for each edge, counting each edge when it first occurs.

The difficulty is in counting the number of edges generated, i.e., the number of times we must look in the table E. A clique covering may cover each edge many times. For example, if our clique listing had $O(n)$ cliques each of cardinality $O(n)$, we would examine $O(n^2)$ edges per clique, for a total of $O(n^3)$ edges. The work would thus exceed the length $O(n^2)$ of D. Can a clique covering with description of length only $O(n^2)$ ever require $O(n^4)$ work to check?

4.2 Hypercliques: k of n problems.

Example 2. *The k of n entry problem.* [7, 12, 13] Suppose there are n processes, S_1 through S_n. At most k are allowed to enter their critical section at one time. The minimal mutually excluding sets are now the

n *Choose* $(k+1)$ possible sets of size $k+1$. The length of the full hyperedge list is now $O(k(n \text{ Choose } k))$, while the hyperclique description contains exactly one hyperclique of size n and cardinality $k+1$. The hyperclique description is of length $O(n)$ and it is very easy to test any proposed set of active processes against it.

We can build a token system easily: provide k identical tokens and have each process require exactly one token to enter.

Proposition 4.2 *The set of hypercliques is a quickly excluding set of hypergraphs. We can test the ability of a process to enter its critical section in time* $O(k) \leq O(n)$.

This is true even though the hyperedge description of a hyperclique is potentially very large (there are n *Choose* k hyperedges.) Again, we ask:

Problem 4. *Can we quickly recognize a hyperclique, if the hypergraph is given in some other form?* The question seems much harder than for graphs.

4.3 Threshold graphs: readers and writers.

Example 3. *The readers-writers problem.* Suppose we are given n processes R_i and n processes W_i. Each W_i is attempting to *write* the same data record $\mathcal{A}$; each R_i is attempting to *read* it. The mutual exclusion problem is this: No other process can access the data while a writer is writing. That is, any number of readers may work if no writer is working, or exactly one writer may work. Fixing $n = 3$ to simplify notation, the mutual exclusion problem is modelled by the graph

$$\{(W_1,W_2),(W_1,W_3),(W_2,W_3),(W_1,R_1),(W_1,R_2),(W_1,R_3),(W_2,R_1),$$
$$(W_2,R_2),(W_2,R_3),(W_3,R_1),(W_3,R_2),(W_3,R_3)\}$$

which could equally well be written

$$\{(W_1,W_2,W_3,R_1),(W_1,W_2,W_3,R_2),(W_1,W_2,W_3,R_3)\}$$

Here we clearly see the advantage of using clique coverings instead of full graph descriptions. For a system with n readers and n writers, the exclusion graph has $n(n-1)/2 + n^2 = O(n^2)$ edges; the corresponding clique covering has exactly n cliques. Since the cliques are of size $n+1$ but

of cardinality 2, we need only one token for each clique; each writer will try to obtain all n tokens while a reader will settle for only one.

This example has another particularly nice feature: there is no need to label the tokens. That is, all n tokens can be treated as interchangeable. This departs from our standard model, in that it requires the writers to demand a multiset, rather than a set, of tokens. But it puts the problem into a well-known class: the graph is a *threshold graph* [3, 6].

Definition 6. *A graph is called a* threshold graph *if each vertex v_k has a positive integer label t_k and there is a positive integer t (called the* threshold*) such that any pair of distinct vertices v_i and v_j is connected by an edge if and only if $t_i + t_j > t$.*

A hypergraph is called a threshold hypergraph *if each vertex v_k has a positive integer label t_k and there is a positive integer t (called the* threshold*) such that any set $\{v_i, \ldots, v_j\}$ of vertices contains a hyperedge if and only if $t_i + \cdots + t_j > t$.*

A threshold graph and a threshold hypergraph share the property that a mutual exclusion problem modelled by them can be implemented in the following easy way: Give each process the label t_i corresponding to its place in the (hyper)graph. Given a set of processes, add their labels: if the sum exceeds t, the set is mutually excluding. In the computer science literature, algorithms for such exclusion problems are often implemented using computer operations called $\mathbf{PV_{chunk}}$ operations [6].

It is NP-complete to determine even whether a graph can be covered by two threshold graphs [4], much less three or more; but it is very quick to determine if it is a threshold graph [6, 10, 11].

Problem 5. *Can we recognize if a given hypergraph is a threshold hypergraph, and determine the threshold and vertex labels?* Again, we suppose the graph is given by a hyperclique covering (possibly a full hyperedge list) and would like the answer to be found in a time polynomial in the length of the covering.

Threshold graphs (and hypergraphs) may have very long (hyper)edge lists. We are most often interested in the ones with relatively small (hyper)clique covers, since they are the ones with small token systems. A threshold graph on n vertices always has a clique cover of at most $n - 1$ cliques, and it is easy to find [6, 8, 10].

Problem 6. *Does a threshold hypergraph necessarily have a small hyperclique cover?*

5 A new example: The Excluded Taxpayer Problem.

It is now natural to look for a more pathological example. Are there natural problems where no reasonably-sized token system will do? Can we find evidence that there are classes of hypergraphs which are tractable, even though they have no small hyperclique coverings? The following example gives affirmative answers.

It is in general NP–complete to find a minimal hyperclique cover (or even a clique cover). However, there are interesting special cases in which we can recognize that a hyperclique cover (or a token system) is minimal, by *ad hoc* methods. The example below is of interest, as showing how bad the pathology can be, and also showing that clever entry protocols can succeed.

Example 4. *The Excluded Taxpayer Problem.* The name of this problem is suggested by the somewhat obscure American joke, *Don't tax you, don't tax me, tax the fellow behind the tree.* Suppose we have a legislature containing n members, each of whom represents one of k interest groups. If at least one representative of each interest group is present at a meeting, no tax plan can be passed. However, if all representatives of interest group i are absent, the members present can agree to tax group i. Accordingly, we need an algorithm which prevents a member from entering if her entrance would mean all groups would be represented.

Assuming that k divides n and exactly n/k members represent each group, we denote representative i from group j by $r_{j,i}$. (for concreteness, consider $n = 300$ and $k = 100$, with 3 representatives for each of 100 interest groups.) Process $r_{j,i}$ may enter the meeting (its critical section) if and only if there is some $t, t \neq j$ such that *no* $r_{t,s}$ is present. Clearly, the minimal mutually excluding sets consist of exactly k members, one from each group. Thus the number of such sets is $(n/k)^k$. No set contains another. No collection of these sets form a hyperclique of cardinality greater than k, since any set larger than k contains two members of the same group and they are not together in any minimal mutually excluding set.

We first argue that this is not an unreasonably hard mutual exclusion problem. In fact, using standard tools such as shared-memory algorithms, it can easily be solved in linear time and space. Suppose at the entrance to the legislative chamber we place a chalkboard (shared memory) with k labelled rows, each with a space for n/k marks. Each legislator, before entering the legislative chamber, checks the chalkboard. If a member of her

group is in, she adds her mark in the appropriate row and goes in. If no member of her group is in, she checks to be sure that at least one other group is unrepresented and then adds her mark and goes in. If hers is the only group unrepresented, she must go away and try later. On exiting the chamber, any legislator must erase one mark from her group's row. This solution clearly involves shared memory of space exactly n bits and time not exceeding n.

To meet the conditions promised in the discussion of Problem 1, in Section 2.3 above, here is a more formal description of a similar algorithm executed by each process $R(j,i), j = 1 \ldots 100, i = 1 \ldots 3$. This algorithm uses integer shared variables, runs in constant time (assuming we can increment an integer of size n, or test it for being zero, in constant time) and thus allows us to test whether a set of points of cardinality n contains a hyperedge in time essentially $O(n)$.

Algorithm: **Excluded Taxpayer, with shared memory**
    Global integer N *initally* 0; { *number of groups represented* }
    Global array A[1..100] of 0..2 *initially all* 0;
Begin process R[j,i]:
entry protocol
    Lock(N, A); *guarantees unique access to data*
    if $A[j] > 0$ **then begin** $A[j] := A[j] + 1$; enter; **end**
        else if $N < 99$ **then**
            begin $N := N + 1$; $A[j] := 1$; enter; **end**
        else wait and try later;
    Unlock(N, A);
...
exit protocol
    Lock(N, A)
    $A[j] := A[j] - 1$;
    if $A[j] = 0$ **then** $N := N - 1$;
    exit;
    Unlock(N, A)
End.

Now let us try to solve this problem with a token system. Outside the door of the legislative chamber is a bowl of tokens; each member has a list of (distinctly labelled) tokens that must be taken from the bowl prior to entering the chamber and returned on leaving. How many tokens are needed? The solution of Theorem 3.1 would generate $k-1$ tokens for each of the $(n/k)^k$ hyperedges - in the present case, 99×3^{100} tokens.

Theorem 5.1 *For the Excluded Taxpayer Problem with n members evenly divided into k groups, the minimal token system has $(k-1)(n/k)^k$ tokens.*

PROOF: We have seen above that the mutual exclusion hypergraph has $(n/k)^k$ hyperedges each of cardinality k, and that these hyperedges constitute the only possible hyperclique covering of the hypergraph: that is, there is no hyperclique in the graph other than these hyperedges. Hence any token system must give rise, by Theorem 3.3, to at least these $(n/k)^k$ hypercliques. Further, since in each hyperclique it is possible for $k-1$ processes to enter at once, there must be at least $k-1$ tokens of each type. □

Again, this proof depends on the assumption that each process demands a *set* of distiguishable tokens, *not* a multiset. The same theorem is true with multisets allowed, but the proof is much more complex and is left to a later paper.

This example shows that, in an appropriate sense, there are hypergraphs on n vertices with no hyperclique covering of size polynomial in n. It also shows that there are mutual exclusion problems on n processes, solvable in linear time and space using shared variables, with no token system of size polynomial in n.

We rephrase the situation from a slightly different point of view. Let $H(n,k)$ denote the hypergraph of the excluded taxpayer problem for n processes in k groups of size n/k. This hypergraph has a very large number of hyperedges. Testing for mutual exclusion by tokens needs a great many tokens. But testing for mutual exclusion using shared variables is easy; so describing the hyperedges of the hypergraph is, in a sense, easy.

Algorithm 2. *Given a set S and a hypergraph $H(n,k)$ on S, and a subset H_1 of S, determine if H_1 contains a hyperedge of $H(n,k)$.* The method is as above: read the elements of H_1, making a tally by the group of each element as one goes. If at the end there is at least one tally by each of the k groups, then the set H_1 contains a hyperedge.

This gives rise to a rather ill-formed, but highly applicable, specialization of Problem 2: what are some other classes of hypergraphs, in addition to the class of $H(n,k)$, having such efficient test algorithms?

Problem 7. *Identify (classes of) hypergraphs which have "large" number of hyperedges, and no "small" hyperclique coverings, for which there are linear-time tests that will recognize sets of vertices which contain a hyperedge.*

References.

[1] G. Andrews, "Concurrent Programming: Principles and Practice", Benjamin/Cummings, Redwood City, Ca., 1991.

[2] K. M. Chandy and J. Misra, *The drinking philosophers problem*, ACM Trans. on Prog. Languages and Systems 6(1984), 632–646.

[3] V. Chvátal and P. Hammer, *Aggregation of inequalities in integer programming*, Ann. Discrete Math. 1 (1977), 145–162.

[4] M. B. Cozzens and R. Liebowitz, *Threshold dimension of graphs*, SIAM J. Algebraic Discrete Methods 5(1984), 579–595.

[5] E. W. Dijkstra, *Solution of a problem in concurrent programming control*, Comm. ACM 8(1965), 569.

[6] P. Henderson and Y. Zalcstein, *A graph theoretic characterization of the* $\mathbf{PV_{chunk}}$ *class of synchronizing primitives*, SIAM J. Comp. 6 (1977), 88–108.

[7] M. Naimi, *Distributed algorithm for K-entries to a critical section based on the directed graphs*, Operating Systems Review, 27(October 1993).

[8] E. T. Ordman, *Threshold coverings and resource allocation*, Proc. 16th Southeastern International Conf. on Graph Theory, Combinatorics, and Computing, Congr. Numer. 49(1985), 99–113.

[9] E. T. Ordman, *Dining philosophers and graph covering problems*, J. Combinatorial Mathematics and Combinatorial Computing 1(1987), 181–190.

[10] E. T. Ordman, *Minimal threshold separators and memory requirements for synchronization*, SIAM J. Computing 18(1989), 152-165.

[11] J. Orlin, *The minimal integral separator of a threshold graph*, Ann. Discrete Math. 1(1977), 415-419.

[12] K. Raymond, *A distributed algorithm for multiple entries to a critical section*, Inform. Process. Lett. 30(1989), 189–193.

[13] P. K. Srimani and R. L. N. Reddy, *Another distributed algorithm for multiple entries to a critical section*, Inform. Process. Lett. 41(1992), 51–57.