Correspondence

Dear Sir, The proof of Lemma 2 (p. 247) seems to be incomplete, i.e. invalid; the increasing of $t^k_{i,N}$ does not guarantee unity as limit, only (as bounded by the sum being always constant unity) *some* value *between* zero and one as limit.

One has to show that (except $t^n_{j,N} > 0$ also by conditions $\Sigma t^{k-r}_{i,j} < 1$) the limit of all $t^k_{i,j}$ for $j < N$ tends to zero. (Otherwise one would have proved that *any* Markov chain does *not cycle* in the limit indefinitely, if only there exists a state reachable from all others – which assertion apparently is not true). Also I cannot follow the statement concerning (18): right side is equal to $\dfrac{100}{1 - \dfrac{\wedge}{E_s}\left(\wedge - \dfrac{\sigma}{A\nu p}\right)}$ and, with $\sigma < A\nu p$, $E_s > 1$ it tends to some value not easy to be recognised when $p \rightarrow 1$.

Yours faithfully
K. BROKATE
Rubezahlweg 20
D-725 Leonberg
Republic of Western Germany

Combinations in lexicographic order

Dear Sir,
The standard method of generating all the combinations of k^{th} order out of $\{1, 2, \ldots, N\}$, $N \geq k$, in lexicographic order is[1,2]

- to start with combination $\{1, 2, \ldots, k\}$. If an integer array $A[1\ ..\ N]$ contains the generated combinations, then $A[i] = i$ initially;
- if combination $A[1], A[2], \ldots, A[k]$ is the last one generated, then the next one in lexicographic order is

$$A[1], A[2], \ldots, A[F-1], A[F]+1,$$
$$A[F]+2, \ldots, A[F]+k-F+1,$$

where F is the last position, where $A[F]$ can be increased by 1 and still in the last position k

$$A[F]+k-F+1 \leq N.$$

Thus, we have to look for the first position F in $k, k-1, \ldots$ such that

$$A[F] < N - k + F.$$

This can be improved considerably by complete removal of search[3]. If a combination is generated, then there are two possible cases only. If F is the position of the last ascent ($A[F] \rightarrow A[F]+1$), then

- either $A[k] = N$ in the combination generated. Then no further ascent in F is possible and as the next ascent position $F - 1$ is to be used. If this new F equals to zero, then there is no next combination;
- or $A[k] < N$; then the next ascent is to be performed in position k and we set $F = k$ for the next combination to be generated.

The procedure given[3] can be simplified further if we set on the first entry $F := 1$, $A[1] := 0$. With N, k, F and integer array $A[1\ ..]$ considered global the whole procedure becomes then

```
procedure LexComb;
var i: integer;
begin
  A[F]:= A[F] + 1; for i: = F + 1 to k do
    A[i]:= A[i − 1] + 1;
  if A[k] < N then F: = k else F := F − 1
end;
```

by making use of Pascal language.

Yours faithfully
STANISLAV DVOŘÁK
Tesla Rožnov,
Rožnov pod Radhoštěm,
Czechoslovakia

REFERENCES

1. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice.* Prentice–Hall, Englewood Cliffs, N.J. (1977).
2. C. J. Mifsud, Alg. 154, Combinations in Lexicographic Order. *Comm. ACM,* **6**, p. 103 (1963).
3. S. Dvořák and A. Musset, *Basic in Action.* Butterworth, London (1984).

Dear Sir,
The article 'A distributed algorithm for mutual exclusion' by J. M. Helary *et al* (*The Computer Journal*, **31** (4), 289–295 (1988)) raises a couple of points which probably need to be mentioned. Neither in Section 2 – Assumptions, nor in any of the assumptions in the remainder of the article is it made explicit that all of the procedures of the algorithm in one process must run on the same processor if mutual exclusion is to be achieved, the procedures can not take advantage of a network node having multiple processors.

The stipulation is made in the article that each of the procedures used in the algorithm must be atomic. This takes care of the problem which would arise if the algorithm were run on a single processor machine which utilized time slicing but does nothing to prevent the failure of the algorithm were a number of the procedures at one node to run concurrently. In this latter case a process which holds the token may enter its own critical section (via enter_CS procedure, as token_here is true) at the same time as it is transmitting the token (via transmit_token procedure, again as token_here is true), once the process exits its critical section it will again transmit the token (via exit_CS procedure) thus mutual exclusion will not be supported and multiple tokens will have been generated to circulate in the network.

It may be argued that the requirement that each procedure is atomic implies they should not run concurrently but this is not integral to the definition of an atomic procedure and thence should be stated explicitly.

As a lecturer I endeavour to get my students to use meaningful data names in their algorithms. These efforts are somewhat undermined when I study articles such as this one with the students and they want to know what is meaningful about 'elec' and 'lud'. If these are meaningful in French could you please give their English equivalents, at least it will help the students feel they are not the only ones who are subject to this type of discipline.

Yours faithfully.
C. G. BURGESS
Dept. Computer Science,
Univ. Southern Mississippi,
S.S. Box 5106, Hattiesburg,
MS 39406, U.S.A.

Dear Sir,
It continues to surprise me that so many people persist in using the so-called 'classical' method of performing two's-complementation on binary numbers whereas the modern method is not only faster, but more accurate.

The classical method requires two stages: a direct complement which involves changing all 1 bits to 0 and all 0 bits to 1, followed by adding binary 1 to the complemented number. It is the latter stage where most students tend to introduce errors, especially where multiple carries are involved.

The modern method, which I use daily in my work, performs a two's-complement in one stage using the following algorithm:

1. If the number is zero (0), the complement is obviously zero.
2. Scan the binary number from RIGHT to LEFT up to and including the FIRST 1 bit. Copy down the bits unchanged.
3. Complement the remaining bits.

For example, to two's-complement the number 0110100100, you would copy down the first three bits on the right hand side of the number, 100, unchanged and complement the remaining seven bits to produce 1001011100.

This technique has the advantage that any binary number of any length can be complemented as fast as it is written down.

Since I have not seen this technique documented in any computer related literature, I would be grateful if you could pass on the above information to readers of your journal.

Yours faithfully
STEVEN W. PALMER
Systems Programmer
L. J. Technical Systems Ltd,
Francis Way,
Bowthorpe Industrial Estate,
Norwich, NORFOLK NR5 9JA