LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Analysing mutual exclusion algorithms using CCS

by

David Walker

Analysing mutual exclusion algorithms
using CCS

ECS-LFCS-88-45
(also published as CSR-257-88

LFCS Report Series

January 1988

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Analysing mutual exclusion algorithms using CCS

David Walker

Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh

December 1987

Abstract

A number of algorithms intended to ensure mutually exclusive execution of certain 'critical sections' of code are studied by representing them as CCS agents and employing an automated tool, the `Concurrency Workbench`, to analyse the representations. The problem of determining whether or not each of the algorithms does indeed preserve mutual exclusion is considered.

Introduction

Milner's Calculus of Communicating Systems, CCS, provides a semantic basis for reasoning about concurrent and communicating systems. In [M] Milner describes a means of representing as CCS agents programs expressed in a simple concurrent imperative language. The purpose of the present paper is to report the results of an experiment in using such representations to reason about the behaviour of a class of concurrent algorithms. The algorithms considered are all intended to ensure mutually exclusive execution of the 'critical sections' of a number of concurrent processes, and the particular problem posed is to determine whether or not each of these algorithms does indeed preserve mutual exclusion. These well-known algorithms were selected for study as, in addition to their being amenable to analysis using the technique of translation into CCS (at least as far as the problem of preservation of mutual exclusion is concerned), and their being sufficiently 'small' to be tackled using the available tools, direct comparison with other methods which have been used to analyse the algorithms would become possible. It is possible that the technique considered here may be used to establish further properties of the algorithms, but this question is not pursued in the present paper.

In order that the analytical technique examined in this paper be at all viable it is essential that suitable automated tools be available. In performing the analyses reported here, I have made extensive use of the `Concurrency Workbench`, a prototype automated tool for reasoning about CCS agents[1]. The availability of such tools by no means ensures that this technique will be widely applicable. For in representing algorithms as CCS agents we face the problem that the state-spaces of such representing agents are often very large, even for simple algorithms, and consequently analyses of such representations can involve very substantial amounts of computation. However it is important to note that in establishing that certain of the algorithms do indeed preserve mutual exclusion, no attempt has been made in this paper to exploit any understanding as to *why* we expect this to be so. The study of appropriate means of so doing, in particular in conjunction with the use of automated tools such as the Concurrency Workbench, remains a subject for research.

Familiarity with CCS is assumed. Section 1 contains a brief description of the method of translating simple concurrent algorithms into CCS used throughout the paper (see also [M, chapter 9]). The remaining sections contain descriptions and analyses of five mutual exclusion algorithms due, respectively, to Dijkstra, Hyman, Dekker, Lamport and Peterson.

1. Translating imperative algorithms into CCS

In chapter 9 of [M], Milner describes a means of representing as CCS agents programs expressed in a simple concurrent imperative language This method underlies the translation into the language of CCS of the mutual exclusion algorithms considered in this paper. On account of the relative simplicity of the problem of representing these algorithms in CCS, it was considered

[1]The Concurrency Workbench is currently under development in a joint SERC-funded project between the Laboratory for Foundations of Computer Science in the University of Edinburgh and the Department of Computer Science in the University of Sussex. The version of the Concurrency Workbench used in preparing this paper was version 2.2 written by Joachim Parrow.

unnecessary to perform the translation in complete detail — the agents obtained by doing so would be very similar to the representations considered here. In this section we illustrate the translation method by considering how to represent Dijkstra's mutual exclusion algorithm [D].

It is assumed that there are $n(\geq 2)$ processes and global variables

$$\begin{aligned} b, c &: \textbf{array } [1..n] \textbf{ of boolean} \\ k &: \textbf{integer}, \end{aligned}$$

and that initially all entries in b and c have value **true** and the value of k is one of 1, 2, ..., n. The i^{th} process $(1 \leq i \leq n)$ may be represented as follows:

```
var j : integer ;
while true do
begin
   ⟨ noncritical section ⟩ ;
   b[i] := false ;
   L_i : if k ≠ i then begin
                             c[i] := true ;
                             if b[k] then k := i ;
                             goto L_i
                          end
        else begin
                 c[i] := false ;
                 for j := 1 to n do
                     if ( j ≠ i and ¬(c[j])) then goto L_i
              end ;
   ⟨ critical section ⟩ ;
   c[i] := true ;
   b[i] := true
end ;
```

In translating into CCS, each of the array variables b and c is represented as a family of n variables, $b_1, \ldots, b_n$ and $c_1, \ldots, c_n$, and each of these variables is represented as a CCS agent.

A variable v of some type D is represented as a family of agents V_d $(d \in D)$ with 'access' sorts $\{\overline{\text{read}_d} \mid d \in D\} \cup \{\text{write}_d \mid d \in D\}$ defined as follows: for $d \in D$

$$V_d =_{df} \overline{\mathrm{read}_d}.\, V_d \; + \; \sum_{e \in D} \mathrm{write}_e.\, V_e$$

Thus if the current value of the variable v is d then that value may be read by a process which may perform the action read_d (thus communicating with V_d), and any value e of type D may be written to v by a process which may communicate with V_d by performing the action $\overline{\mathrm{write}_e}$. (Note that, in general, in describing the access sort of a variable it is necessary to use names characteristic of that particular variable, rather than simply 'read' and 'write.' We will achieve this by incorporating an appropriate distinctive part of the name of the variable into the names of the actions comprising its access sort (see below). Although of no significance in this paper, it is interesting to note that, as described by Milner in [M], the restriction operator of CCS provides a pleasant means of dealing with the scoping of variables.)

In the present paper we need consider only program variables of type **boolean** and of type 1..n. A variable b of type **boolean** with initial value **true** may, for example, be represented as an agent $B_{\mathbf{true}}$ as follows:

$$\begin{aligned} B_{\mathbf{true}} &=_{df} \overline{\mathrm{read}_{\mathbf{true}}}.\, B_{\mathbf{true}} + \mathrm{write}_{\mathbf{true}}.\, B_{\mathbf{true}} + \mathrm{write}_{\mathbf{false}}.\, B_{\mathbf{false}} \\ B_{\mathbf{false}} &=_{df} \overline{\mathrm{read}_{\mathbf{false}}}.\, B_{\mathbf{false}} + \mathrm{write}_{\mathbf{false}}.\, B_{\mathbf{false}} + \mathrm{write}_{\mathbf{true}}\, .B_{\mathbf{true}} \end{aligned}$$

The variables b_i and c_i $(1 \le i \le n)$ with initial value **true** may then be represented as the processes Bit and Cit obtained from $B_{\mathbf{true}}$ by relabelling as follows:

$$\begin{aligned} \mathrm{Bit} &=_{df} B_{\mathbf{true}}[\mathrm{birf}/\mathrm{read}_{\mathbf{false}}, \mathrm{birt}/\mathrm{read}_{\mathbf{true}}, \mathrm{biwf}/\mathrm{write}_{\mathbf{false}}, \mathrm{biwt}/\mathrm{write}_{\mathbf{true}}] \\ \mathrm{Cit} &=_{df} B_{\mathbf{true}}[\mathrm{cirf}/\mathrm{read}_{\mathbf{false}}, \mathrm{cirt}/\mathrm{read}_{\mathbf{true}}, \mathrm{ciwf}/\mathrm{write}_{\mathbf{false}}, \mathrm{ciwt}/\mathrm{write}_{\mathbf{true}}] \end{aligned}$$

Here we use abbreviations to reduce the length of process names and action names. Thus, e.g., B1t represents variable b_1 with current value **true**, b1rf may be read "read value **false** from variable B1" and c3wt as "write value **true** to variable C3."

We could give a similar representation of the variable k of type 1..n. However, as we shall see below, in order to model faithfully the assumption that any read or write of a program variable is an atomic event, an assumption that is made throughout this paper, it is sometimes necessary (and in particular it is necessary in the case of the variable k in Dijkstra's algorithm) to use a slightly more complicated representation of a program variable, and thus we shall defer doing so for the present.

It is possible to represent each of the n processes as an agent using n as a parameter of the representation. However in order to carry out analysis of the resulting agents using the Concurrency Workbench we must consider a specific value of n. Thus we consider here only the case $n = 3$. The i^{th} process (i = 1, 2, 3) is represented as a process Pi as follows:

$$\begin{array}{lll}
\mathrm{P1} & =_{df} & \overline{\mathrm{b1wf}}.\,\mathrm{P11} \\
\mathrm{P11} & =_{df} & \mathrm{kr1}.\,\mathrm{P12} + \mathrm{kr2}.\,\overline{\mathrm{c1wt}}.\,\mathrm{P12} + \mathrm{kr3}.\,\overline{\mathrm{c1wt}}.\,\mathrm{P12} \\
\mathrm{P12} & =_{df} & \overline{\mathrm{get}}.\,(\mathrm{k1r1}.\,(\mathrm{b1rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw1}}.\,\mathrm{P11} + \mathrm{b1rf}.\,\overline{\mathrm{put}}.\,\mathrm{P11}) \\
 & & \quad + \mathrm{k1r2}.\,(\mathrm{b2rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw1}}.\,\mathrm{P11} + \mathrm{b2rf}.\,\overline{\mathrm{put}}.\,\mathrm{P11}) \\
 & & \quad + \mathrm{k1r3}.\,(\mathrm{b3rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw1}}.\,\mathrm{P11} + \mathrm{b3rf}.\,\overline{\mathrm{put}}.\,\mathrm{P11})) \\
\mathrm{P13} & =_{df} & \overline{\mathrm{c1wf}}.\,(\mathrm{c2rf}.\,\mathrm{P11} + \mathrm{c2rt}.\,\mathrm{P14}) \\
\mathrm{P14} & =_{df} & \mathrm{c3rf}.\,\mathrm{P11} + \mathrm{c3rt}.\,\mathrm{P15} \\
\mathrm{P15} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{c1wt}}.\,\overline{\mathrm{b1wt}}.\,\mathrm{P1}
\end{array}$$

$$\begin{array}{lll}
\mathrm{P2} & =_{df} & \overline{\mathrm{b2wf}}.\,\mathrm{P21} \\
\mathrm{P21} & =_{df} & \mathrm{kr2}.\,\mathrm{P23} + \mathrm{kr1}.\,\overline{\mathrm{c2wt}}.\,\mathrm{P22} + \mathrm{kr3}.\,\overline{\mathrm{c2wt}}.\,\mathrm{P22} \\
\mathrm{P22} & =_{df} & \overline{\mathrm{get}}.\,(\mathrm{k2r1}.\,(\mathrm{b1rt}.\,\overline{\mathrm{k2v}}.\,\overline{\mathrm{put}}.\,\mathrm{P21} + \mathrm{b1rf}.\,\overline{\mathrm{put}}.\,\mathrm{P21}) \\
 & & \quad + \mathrm{k2r2}.\,(\mathrm{b2rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw2}}.\,\mathrm{P21} + \mathrm{b2rf}.\,\overline{\mathrm{put}}.\,\mathrm{P21}) \\
 & & \quad + \mathrm{k2r3}.\,(\mathrm{b3rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw2}}.\,\mathrm{P21} + \mathrm{b3rf}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw2}}.\,\mathrm{P21})) \\
\mathrm{P23} & =_{df} & \overline{\mathrm{c2wf}}.\,(\mathrm{c1rf}.\,\mathrm{P21} + \mathrm{c1rt}.\,\mathrm{P24}) \\
\mathrm{P24} & =_{df} & \mathrm{c3rf}.\,\mathrm{P21} + \mathrm{c3rt}.\,\mathrm{P25} \\
\mathrm{P25} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{c2wt}}.\,\overline{\mathrm{b2wt}}.\,\mathrm{P2}
\end{array}$$

$$\begin{array}{lll}
\mathrm{P3} & =_{df} & \overline{\mathrm{b3wf}}.\,\mathrm{P31} \\
\mathrm{P31} & =_{df} & \mathrm{kr3}.\,\mathrm{P33} + \mathrm{kr1}.\,\overline{\mathrm{c3wt}}.\,\mathrm{P32} + \mathrm{kr2}.\,\overline{\mathrm{c3wt}}.\,\mathrm{P32} \\
\mathrm{P32} & =_{df} & \overline{\mathrm{get}}.\,(\mathrm{k3r1}.\,(\mathrm{b1rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw3}}.\,\mathrm{P31} + \mathrm{b1rf}.\,\overline{\mathrm{put}}.\,\mathrm{P31}) \\
 & & \quad + \mathrm{k3r2}.\,(\mathrm{b2rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw3}}.\,\mathrm{P31} + \mathrm{b2rf}.\,\overline{\mathrm{put}}.\,\mathrm{P31}) \\
 & & \quad + \mathrm{k3r3}.\,(\mathrm{b3rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw3}}.\,\mathrm{P31} + \mathrm{b3rf}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw3}}.\,\mathrm{P31})) \\
\mathrm{P33} & =_{df} & \overline{\mathrm{c3wf}}.\,(\mathrm{c1rf}.\,\mathrm{P31} + \mathrm{c1rt}.\,\mathrm{P34}) \\
\mathrm{P34} & =_{df} & \mathrm{c2rf}.\,\mathrm{P31} + \mathrm{c2rt}.\,\mathrm{P35} \\
\mathrm{P35} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{c3wt}}.\,\overline{\mathrm{b3wt}}.\,\mathrm{P3}
\end{array}$$

Some remarks concerning these representations are in order. Firstly note that the local variables j have been omitted since they are unnecessary: the transitions of the agents accurately describe the behaviour of the process without the use of a counter variable. Secondly, and more significantly, as mentioned above in order to model the assumption that in executing the statement "**if** $b[k]$ **then** $k := i$" the read of the variable $b[k]$ by the i^{th} process is an atomic action, it is necessary to use a more complex representation of the variable k such as the following family of processes K1, K2, K3:

$$\begin{aligned}
\mathrm{K1} &=_{df} \mathrm{get}.(\overline{\mathrm{k1r1}}.\,\mathrm{put}.\,\mathrm{K1} + \overline{\mathrm{k2r1}}.\,\mathrm{put}.\,\mathrm{K1} + \overline{\mathrm{k3r1}}.\,\mathrm{put}.\,\mathrm{K1}) \\
&\quad + \mathrm{kw1}.\,\mathrm{K1} + \mathrm{kw2}.\,\mathrm{K2} + \mathrm{kw3}.\,\mathrm{K3} \\
\mathrm{K2} &=_{df} \mathrm{get}.(\overline{\mathrm{k1r2}}.\,\mathrm{put}.\,\mathrm{K2} + \overline{\mathrm{k2r2}}.\,\mathrm{put}.\,\mathrm{K2} + \overline{\mathrm{k3r2}}.\,\mathrm{put}.\,\mathrm{K2}) \\
&\quad + \mathrm{kw2}.\,\mathrm{K2} + \mathrm{kw3}.\,\mathrm{K3} + \mathrm{kw1}.\,\mathrm{K1} \\
\mathrm{K3} &=_{df} \mathrm{get}.(\overline{\mathrm{k1r3}}.\,\mathrm{put}.\,\mathrm{K3} + \overline{\mathrm{k2r3}}.\,\mathrm{put}.\,\mathrm{K3} + \overline{\mathrm{k3r3}}.\,\mathrm{put}.\,\mathrm{K3}) \\
&\quad + \mathrm{kw3}.\,\mathrm{K3} + \mathrm{kw1}.\,\mathrm{K1} + \mathrm{kw2}.\,\mathrm{K2}
\end{aligned}$$

(I am indebted to Robin Milner for suggesting this representation.) Here, e.g., kw1 may be read "write value 1 to variable K" and k3r2 as "process P3 may read value 2 from variable K." Note that if in the course of executing the statement "**if** $b[k]$ **then** $k := i$" the i^{th} process has read the variable k but not yet read the variable $b[k]$, then on account of the 'get' and 'put' actions, no other process may access (and in particular change the value of) the variable k. Thirdly, the noncritical section of each process is omitted from the representation, as we are concerned principally with the problem of mutual exclusion. And finally, the critical section of each process is represented as a pair of actions, 'enter' and 'exit,' representing entry to and exit from the critical section respectively. (If appropriate we can replace 'enter' and 'exit' in the i^{th} process by 'enter$_i$' and 'exit$_i$' respectively to model the execution of the critical section of that particular process.)

Thus, taking the initial value of k to be 1 we may represent Dijkstra's algorithm in the case of 3 processes by the agent Dijkstra–3 defined by:

$$\text{Dijkstra–3} =_{df} (\mathrm{P1} \mid \mathrm{P2} \mid \mathrm{P3} \mid \mathrm{B1t} \mid \mathrm{B2t} \mid \mathrm{B3t} \mid \mathrm{C1t} \mid \mathrm{C2t} \mid \mathrm{C3t} \mid \mathrm{K1})\backslash \mathrm{L}$$

where L is the set of all actions used for synchronization among the agents in the composition. Thus the agent Dijkstra–3 has sort {enter, exit}.

2. Dijkstra's algorithm

In this section we consider the first published solution to the mutual exclusion problem, due to Dijkstra [D]. In the preceding section this algorithm was described and a representation in CCS in the case of 3 processes presented. Before considering the analysis of this representation we consider the algorithm in the case of 2 processes. The agent Dijkstra−2 below provides a representation of the algorithm in this case.

$$
\begin{array}{rcl}
\mathrm{B1f} & =_{df} & \overline{\mathrm{b1rf}}.\,\mathrm{B1f} + \mathrm{b1wf}.\,\mathrm{B1f} + \mathrm{b1wt}.\,\mathrm{B1t} \\
\mathrm{B1t} & =_{df} & \overline{\mathrm{b1rt}}.\,\mathrm{B1t} + \mathrm{b1wt}.\,\mathrm{B1t} + \mathrm{b1wf}.\,\mathrm{B1f} \\
\mathrm{B2f} & =_{df} & \overline{\mathrm{b2rf}}.\,\mathrm{B2f} + \mathrm{b2wf}.\,\mathrm{B2f} + \mathrm{b2wt}.\,\mathrm{B2t} \\
\mathrm{B2t} & =_{df} & \overline{\mathrm{b2rt}}.\,\mathrm{B2t} + \mathrm{b2wt}.\,\mathrm{B2t} + \mathrm{b2wf}.\,\mathrm{B2f} \\
\mathrm{C1f} & =_{df} & \overline{\mathrm{c1rf}}.\,\mathrm{C1f} + \mathrm{c1wf}.\,\mathrm{C1f} + \mathrm{c1wt}.\,\mathrm{C1t} \\
\mathrm{C1t} & =_{df} & \overline{\mathrm{c1rt}}.\,\mathrm{C1t} + \mathrm{c1wt}.\,\mathrm{C1t} + \mathrm{c1wf}.\,\mathrm{C1f} \\
\mathrm{C2f} & =_{df} & \overline{\mathrm{c2rf}}.\,\mathrm{C2f} + \mathrm{c2wf}.\,\mathrm{C2f} + \mathrm{c2wt}.\,\mathrm{C2t} \\
\mathrm{C2t} & =_{df} & \overline{\mathrm{c2rt}}.\,\mathrm{C2t} + \mathrm{c2wt}.\,\mathrm{C2t} + \mathrm{c2wf}.\,\mathrm{C2f} \\
\mathrm{K1} & =_{df} & \overline{\mathrm{kr1}}.\,\mathrm{K1} + \mathrm{kw1}.\,\mathrm{K1} + \mathrm{kw2}.\,\mathrm{K2} \\
 & & + \mathrm{get}.\,(\overline{\mathrm{k1r1}}.\,\mathrm{put}.\,\mathrm{K1} + \overline{\mathrm{k2r1}}.\,\mathrm{put}.\,\mathrm{K1}) \\
\mathrm{K2} & =_{df} & \overline{\mathrm{kr2}}.\,\mathrm{K2} + \mathrm{kw2}.\,\mathrm{K2} + \mathrm{kw1}.\,\mathrm{K1} \\
 & & + \mathrm{get}.\,(\overline{\mathrm{k1r2}}.\,\mathrm{put}.\,\mathrm{K2} + \overline{\mathrm{k2r2}}.\,\mathrm{put}.\,\mathrm{K2})
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{P1} & =_{df} & \overline{\mathrm{b1wf}}.\,\mathrm{P11} \\
\mathrm{P11} & =_{df} & \mathrm{kr1}.\,\mathrm{P13} + \mathrm{kr2}.\,\overline{\mathrm{c1wt}}.\,\mathrm{P12} \\
\mathrm{P12} & =_{df} & \overline{\mathrm{get}}.\,(\mathrm{k1r1}.\,(\mathrm{b1rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw1}}.\,\mathrm{P11} + \mathrm{b1rf}.\,\overline{\mathrm{put}}.\,\mathrm{P11}) \\
 & & + \mathrm{k1r2}.\,(\mathrm{b2rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw1}}.\,\mathrm{P11} + \mathrm{b2rf}.\,\overline{\mathrm{put}}.\,\mathrm{P11})) \\
\mathrm{P13} & =_{df} & \overline{\mathrm{c1wf}}.\,(\mathrm{c2rf}.\,\mathrm{P11} + \mathrm{c2rt}.\,\mathrm{P14}) \\
\mathrm{P14} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{c1wt}}.\,\overline{\mathrm{b1wt}}.\,\mathrm{P1}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{P2} & =_{df} & \overline{\mathrm{b2wf}}.\,\mathrm{P21} \\
\mathrm{P21} & =_{df} & \mathrm{kr2}.\,\mathrm{P23} + \mathrm{kr1}.\,\overline{\mathrm{c2wt}}.\,\mathrm{P22} \\
\mathrm{P22} & =_{df} & \overline{\mathrm{get}}.\,(\mathrm{k2r1}.\,(\mathrm{b1rt}.\,\overline{\mathrm{put}}.\,\overline{\mathrm{kw2}}.\,\mathrm{P21} + \mathrm{b1rf}.\,\overline{\mathrm{put}}.\,\mathrm{P21}) \\
 & & + \mathrm{k2r2}.\,(\mathrm{b2rt}.\,D\overline{\mathrm{put}}.\,\overline{\mathrm{kw2}}.\,\mathrm{P21} + \mathrm{b2rf}.\,\overline{\mathrm{put}}.\,\mathrm{P21})) \\
\mathrm{P23} & =_{df} & \overline{\mathrm{c2wf}}.\,(\mathrm{c1rf}.\,\mathrm{P21} + \mathrm{c1rt}.\,\mathrm{P24}) \\
\mathrm{P24} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{c2wt}}.\,\overline{\mathrm{b2wt}}.\,\mathrm{P2}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{Dijkstra{-}2} & =_{df} & (\mathrm{P1}|\mathrm{P2}|\mathrm{K1}|\mathrm{B1f}|\mathrm{C1f}|\mathrm{B2f}|\mathrm{C2f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt}, \\
 & & \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}, \mathrm{c1rf}, \mathrm{c1rt}, \mathrm{c1wf}, \mathrm{c1wt}, \mathrm{c2rf}, \mathrm{c2rt}, \\
 & & \mathrm{c2wf}, \mathrm{c2wt}, \mathrm{kr1}, \mathrm{kr2}, \mathrm{kw1}, \mathrm{kw2}, \mathrm{get}, \mathrm{put}, \mathrm{k1r1}, \mathrm{k1r2}, \mathrm{k2r1}, \mathrm{k2r2}\}
\end{array}
$$

The agents P1 and P2 represent the two processes, entry to and exit from their critical sections being represented by the actions 'enter' and 'exit' respectively. The agents B1f, B2f, C1f, C2f and K1 represent the program variables b_1, b_2, c_1, c_2 of type **boolean** and k of type 1..2 respectively. (The algorithm is described in the previous section.)

How can we express within CCS that the algorithm preserves mutual exclusion? The solution adopted in this paper is show that Dijkstra–2 is bisimulation equivalent to the agent X where

$$\mathrm{X} =_{df} \mathrm{enter.exit.X}.$$

Note that this implies that mutual exclusion is preserved and that the agent Dijkstra-2 may never reach a 'deadlocked' state.

Using the Concurrency Workbench the state-space of the agent Dijkstra–2 was computed[2]: it contains 310 states. Moreover the Workbench was used to determine[3] that

$$\text{Dijkstra–2} \approx \mathrm{X}.$$

It is interesting to note that if in the definitions of P1 and P2 the actions enter and exit are replaced by enter_1 and exit_1 and by enter_2 and exit_2 respectively, then a minimal agent bisimulation-equivalent to the modified Dijkstra–2 has 42 states.

Having met with some success we now return to considering Dijkstra's algorithm in the case of 3 processes. Here we meet in a stark form perhaps the most serious limitation of the technique discussed in this paper, namely the problem of large state spaces. For using the Workbench it was established that the agent Dijkstra–3 from the previous section which provides a representation of the algorithm in the case of 3 processes has 4255 states. The prospect of analysing such an agent without the assistance of some form of automated tool is not an inviting one. However even with a tool such as the Concurrency Workbench, the amount of computation required to effect analyses such as determining whether or not Dijkstra–3 $\approx$ X (where $\mathrm{X} =_{df}$ enter.exit.X) is substantial[4]. Indeed at the time of writing it has not been possible using the Workbench to determine whether or not

$$\text{Dijkstra–3} \approx \mathrm{X}$$

[2]using the 'state-space' command.

[3]using either the 'equivalent' command, which takes two agents and (assuming that their state-spaces are finite) determines whether or not they are bisimulation equivalent, or the 'minimize' command, which takes a finite-state agent and finds an minimal (in terms of the number of states) bisimulation-equivalent agent.

[4]It is worth noting that the state-space partitioning algorithm used to determine whether or not two agents are bisimulation equivalent runs in time which is bounded by a small polynomial in the size of the state-spaces of the agents [KS]. This may be compared with, e.g., the fact that the problem of determining whether or not two agent are "failures-equivalent" is PSPACE-complete [KS].

despite the devotion of several hours of CPU time of a SUN 4 to the problem.

3. Hyman's algorithm

The purpose of this section is to illustrate how the Concurrency Workbench was used to analyse an algorithm which does not preserve mutual exclusion. The algorithm considered is that published by Hyman [H] as a simplification of Dijkstra's algorithm of the preceding section in the case of 2 processes (see also [K]). The following formulation of Hyman's algorithm (with some change of notation) is taken from [PS].

There are two processes, two variables b_1 and b_2 of type **boolean** with initial value **false** and a variable k of type 1..2 with arbitrary initial value. The i^{th} process ($i = 1, 2$) may be described as follows (where j is the index of the other process).

```
while true do
begin
    ⟨ noncritical section ⟩ ;
    while $k \neq i$ do begin
                          while $b_j$ do skip ;
                          $k := i$
                      end ;
    ⟨ critical section ⟩ ;
    $b_i$ := false
end ;
```

The CCS representation of the algorithm (assuming the initial value of k to be 1) is given below as the agent Hyman−2.

$$
\begin{array}{rcl}
\mathrm{B1f} & =_{df} & \overline{\mathrm{b1rf}}.\,\mathrm{B1f} + \mathrm{b1wf}.\,\mathrm{B1f} + \mathrm{b1wt}.\,\mathrm{B1t} \\
\mathrm{B1t} & =_{df} & \overline{\mathrm{b1rt}}.\,\mathrm{B1t} + \mathrm{b1wt}.\,\mathrm{B1t} + \mathrm{b1wf}.\,\mathrm{B1f} \\
\mathrm{B2f} & =_{df} & \overline{\mathrm{b2rf}}.\,\mathrm{B2f} + \mathrm{b2wf}.\,\mathrm{B2f} + \mathrm{b2wt}.\,\mathrm{B2t} \\
\mathrm{B2t} & =_{df} & \overline{\mathrm{b2rt}}.\,\mathrm{B2t} + \mathrm{b2wt}.\,\mathrm{B2t} + \mathrm{b2wf}.\,\mathrm{B2f} \\
\mathrm{K1} & =_{df} & \overline{\mathrm{kr1}}.\,\mathrm{K1} + \mathrm{kw1}.\,\mathrm{K1} + \mathrm{kw2}.\,\mathrm{K2} \\
\mathrm{K2} & =_{df} & \overline{\mathrm{kr2}}.\,\mathrm{K2} + \mathrm{kw2}.\,\mathrm{K2} + \mathrm{kw1}.\,\mathrm{K1}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{P1} & =_{df} & \overline{\mathrm{b1wt}}.\,\mathrm{P11} \\
\mathrm{P11} & =_{df} & \mathrm{kr1}.\,\mathrm{P13} + \mathrm{kr2}.\,\mathrm{P12} \\
\mathrm{P12} & =_{df} & \mathrm{b2rt}.\,\mathrm{P12} + \mathrm{b2rf}.\,\overline{\mathrm{kw1}}.\,\mathrm{P13} \\
\mathrm{P13} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{b1wf}}.\,\mathrm{P1}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{P2} & =_{df} & \overline{\mathrm{b2wt}}.\,\mathrm{P21} \\
\mathrm{P21} & =_{df} & \mathrm{kr2}.\,\mathrm{P23} + \mathrm{kr1}.\,\mathrm{P22} \\
\mathrm{P22} & =_{df} & \mathrm{b1rt}.\,\mathrm{P22} + \mathrm{b1rf}.\,\overline{\mathrm{kw2}}.\,\mathrm{P23} \\
\mathrm{P23} & =_{df} & \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{b2wf}}.\,\mathrm{P2}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{Hyman{-}2} & =_{df} & (\mathrm{P1}|\mathrm{P2}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt}, \\
 & & \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}, \mathrm{kr1}, \mathrm{kr2}, \mathrm{kw1}, \mathrm{kw2}\}
\end{array}
$$

The agent Hyman−2 has 71 states and a minimal bisimulation-equivalent agent has 9 states. Using the Concurrency Workbench it was determined that mutual exclusion is not preserved. Indeed we can exhibit a path through the state-transition diagram of Hyman−2 which shows that it is possible for the agent Hyman−2 to perform two enter actions without an intervening exit action:

$$
\begin{array}{rcl}
\mathrm{Hyman{-}2} & \xrightarrow{\tau} & (\mathrm{P1}|\mathrm{P21}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\tau} & (\mathrm{P1}|\mathrm{P22}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\tau} & (\mathrm{P1}|\overline{\mathrm{kw2}}.\,\mathrm{P23}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\tau} & (\mathrm{P11}|\overline{\mathrm{kw2}}.\,\mathrm{P23}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\tau} & (\mathrm{P13}|\overline{\mathrm{kw2}}.\,\mathrm{P23}|\mathrm{K1}|\mathrm{B1t}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\mathrm{enter}} & (\mathrm{exit}.\,\overline{\mathrm{b1wf}}.\,\mathrm{P1}|\overline{\mathrm{kw2}}.\,\mathrm{P23}|\mathrm{K1}|\mathrm{B1t}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\tau} & (\mathrm{exit}.\,\overline{\mathrm{b1wf}}.\,\mathrm{P1}|\mathrm{exit}.\,\overline{\mathrm{b2wf}}.\,\mathrm{P2}|\mathrm{K2}|\mathrm{B1t}|\mathrm{B2t})\backslash\mathrm{L} \\
 & \xrightarrow{\mathrm{enter}} & (\mathrm{exit}.\,\overline{\mathrm{b1wf}}.\,\mathrm{P1}|\mathrm{exit}.\,\overline{\mathrm{b2wf}}.\,\mathrm{P2}|\mathrm{K2}|\mathrm{B1t}|\mathrm{B2t})\backslash\mathrm{L}
\end{array}
$$

where L is the set of actions restricted in the definition of Hyman−2.

4. Dekker's algorithm

In this section we consider an algorithm due to Dekker (see [PS]). There are two processes, two variables b_1 and b_2 of type **boolean** with initial value **false**, and one variable k of type 1..2 with arbitrary initial value. The i^{th} process ($i = 1, 2$) may be described as follows (where j is the index of the other process):

```
while true do
begin
    ⟨ noncritical section ⟩ ;
    $b_i$ := true ;
    while $b_j$ do
            if $k = j$ then begin
                                $b_i$ := false ;
                                while $k = j$ do skip ;
                                $b_i$ := true
                            end ;
    ⟨ critical section ⟩ ;
    $k$ := $j$ ;
    $b_i$ := false
end ;
```

The algorithm may be represented (assuming the initial value of k to be 1) as the agent Dekker−2 below:

$$\begin{array}{rcl}
\mathrm{B1f} & =_{df} & \overline{\mathrm{b1rf}}.\mathrm{B1f} + \mathrm{b1wf}.\mathrm{B1f} + \mathrm{b1wt}.\mathrm{B1t} \\
\mathrm{B1t} & =_{df} & \overline{\mathrm{b1rt}}.\mathrm{B1t} + \mathrm{b1wt}.\mathrm{B1t} + \mathrm{b1wf}.\mathrm{B1f} \\
\mathrm{B2f} & =_{df} & \overline{\mathrm{b2rf}}.\mathrm{B2f} + \mathrm{b2wf}.\mathrm{B2f} + \mathrm{b2wt}.\mathrm{B2t} \\
\mathrm{B2t} & =_{df} & \overline{\mathrm{b2rt}}.\mathrm{B2t} + \mathrm{b2wt}.\mathrm{B2t} + \mathrm{b2wf}.\mathrm{B2f} \\
\mathrm{K1} & =_{df} & \overline{\mathrm{kr1}}.\mathrm{K1} + \mathrm{kw1}.\mathrm{K1} + \mathrm{kw2}.\mathrm{K2} \\
\mathrm{K2} & =_{df} & \overline{\mathrm{kr2}}.\mathrm{K2} + \mathrm{kw2}.\mathrm{K2} + \mathrm{kw1}.\mathrm{K1}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{P1} & =_{df} & \overline{\mathrm{b1wt}}.\mathrm{P11} \\
\mathrm{P11} & =_{df} & \mathrm{b2rf}.\mathrm{P14} + \mathrm{b2rt}.\mathrm{P12} \\
\mathrm{P12} & =_{df} & \mathrm{kr1}.\mathrm{P11} + \mathrm{kr2}.\overline{\mathrm{b1wf}}.\mathrm{P13} \\
\mathrm{P13} & =_{df} & \mathrm{kr2}.\mathrm{P13} + \mathrm{kr1}.\overline{\mathrm{b1wt}}.\mathrm{P11} \\
\mathrm{P14} & =_{df} & \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{kw2}}.\overline{\mathrm{b1wf}}.\mathrm{P1}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{P2} & =_{df} & \overline{\mathrm{b2wt}}.\mathrm{P21} \\
\mathrm{P21} & =_{df} & \mathrm{b1rf}.\mathrm{P24} + \mathrm{b1rt}.\mathrm{P22} \\
\mathrm{P22} & =_{df} & \mathrm{kr2}.\mathrm{P21} + \mathrm{kr1}.\overline{\mathrm{b2wf}}.\mathrm{P23} \\
\mathrm{P23} & =_{df} & \mathrm{kr1}.\mathrm{P23} + \mathrm{kr2}.\overline{\mathrm{b2wt}}.\mathrm{P21} \\
\mathrm{P24} & =_{df} & \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{kw1}}.\overline{\mathrm{b2wf}}.\mathrm{P2}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{Dekker{-}2} & =_{df} & (\mathrm{P1}|\mathrm{P2}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt}, \\
 & & \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}, \mathrm{kr1}, \mathrm{kr2}, \mathrm{kw1}, \mathrm{kw2}\}
\end{array}$$

The agent Dekker−2 has 127 states and is bisimulation equivalent to X where X $=_{df}$ enter.exit.X. A minimal agent bisimulation-equivalent to the modification of Dekker−2 with 'enter$_i$' replacing 'enter' for $i = 1, 2$ etc., has 14 states.

5. Lamport's algorithm

In this section we consider an algorithm due to Lamport [L] and referred to by him as 'the one-bit algorithm.' There are $n(\geq 2)$ processes and n variables $b_1, \ldots b_n$ of type **boolean**, each initially having value **false**. The i^{th} process $(1 \leq i \leq n)$ may be described as follows:

```
var $j$ : integer ;
while true do
begin
    ⟨ noncritical section ⟩ ;
    $L_i$ : $b_i$ := true ;
    for $j := 1$ to $i-1$ do
        if $b_j$ then begin
                        $b_i$ := false ;
                        while $b_j$ do skip ;
                        goto $L_i$
                    end ;
    for $j := i+1$ to $n$ do
        while $b_j$ do skip ;
    ⟨ critical section ⟩ ;
    $b_i$ := false
end ;
```

As in the case of Dijkstra's algorithm this algorithm could be represented in CCS using the number of processes, n, as a parameter of the representation. Here, however, we consider only the cases $n = 2$ and $n = 3$. In the case of 2 processes the CCS representation is the agent Lamport−2 below:

$$\begin{aligned}
\mathrm{B1f} &=_{df} \overline{\mathrm{b1rf}}.\,\mathrm{B1f} + \mathrm{b1wf}.\,\mathrm{B1f} + \mathrm{b1wt}.\,\mathrm{B1t}\\
\mathrm{B1t} &=_{df} \overline{\mathrm{b1rt}}.\,\mathrm{B1t} + \mathrm{b1wt}.\,\mathrm{B1t} + \mathrm{b1wf}.\,\mathrm{B1f}\\
\mathrm{B2f} &=_{df} \overline{\mathrm{b2rf}}.\,\mathrm{B2f} + \mathrm{b2wf}.\,\mathrm{B2f} + \mathrm{b2wt}.\,\mathrm{B2t}\\
\mathrm{B2t} &=_{df} \overline{\mathrm{b2rt}}.\,\mathrm{B2t} + \mathrm{b2wt}.\,\mathrm{B2t} + \mathrm{b2wf}.\,\mathrm{B2f}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P1} &=_{df} \overline{\mathrm{b1wt}}.\,\mathrm{P11}\\
\mathrm{P11} &=_{df} \mathrm{b2rf}.\,\mathrm{P12} + \mathrm{b2rt}.\,\mathrm{P11}\\
\mathrm{P12} &=_{df} \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{b1wf}}.\,\mathrm{P1}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P2} &=_{df} \overline{\mathrm{b2wt}}.\,\mathrm{P21}\\
\mathrm{P21} &=_{df} \mathrm{b1rf}.\,\mathrm{P23} + \mathrm{b1rt}.\,\overline{\mathrm{b2wf}}.\,\mathrm{P22}\\
\mathrm{P22} &=_{df} \mathrm{b1rt}.\,\mathrm{P22} + \mathrm{b1rf}.\,\mathrm{P2}\\
\mathrm{P23} &=_{df} \mathrm{enter}.\,\mathrm{exit}.\,\overline{\mathrm{b2wf}}.\,\mathrm{P2}
\end{aligned}$$

$$\mathrm{Lamport{-}2} =_{df} (\mathrm{P1}|\mathrm{P2}|\mathrm{B1f}|\mathrm{B2f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt}, \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}\}$$

The agent Lamport-2 has 27 states and is bisimulation equivalent to X where X $=_{df}$ enter.exit.X. Thus mutual exclusion is preserved. A minimal agent bisimulation equivalent to the modified Lamport-2 obtained by replacing 'enter' and 'exit' in Pi by 'enter_i' and 'exit_i' for $i = 1, 2$, has 7 states.

In the case of 3 processes the CCS representation is the agent Lamport-3 below:

$$\begin{aligned}
\mathrm{B1f} &=_{df} \overline{\mathrm{b1rf}}.\mathrm{B1f} + \mathrm{b1wf}.\mathrm{B1f} + \mathrm{b1wt}.\mathrm{B1t}\\
\mathrm{B1t} &=_{df} \overline{\mathrm{b1rt}}.\mathrm{B1t} + \mathrm{b1wt}.\mathrm{B1t} + \mathrm{b1wf}.\mathrm{B1f}\\
\mathrm{B2f} &=_{df} \overline{\mathrm{b2rf}}.\mathrm{B2f} + \mathrm{b2wf}.\mathrm{B2f} + \mathrm{b2wt}.\mathrm{B2t}\\
\mathrm{B2t} &=_{df} \overline{\mathrm{b2rt}}.\mathrm{B2t} + \mathrm{b2wt}.\mathrm{B2t} + \mathrm{b2wf}.\mathrm{B2f}\\
\mathrm{B3f} &=_{df} \overline{\mathrm{b3rf}}.\mathrm{B3f} + \mathrm{b3wf}.\mathrm{B3f} + \mathrm{b3wt}.\mathrm{B3t}\\
\mathrm{B3t} &=_{df} \overline{\mathrm{b3rt}}.\mathrm{B3t} + \mathrm{b3wt}.\mathrm{B3t} + \mathrm{b3wf}.\mathrm{B3f}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P1} &=_{df} \overline{\mathrm{b1wt}}.\mathrm{P11}\\
\mathrm{P11} &=_{df} \mathrm{b2rf}.\mathrm{P12} + \mathrm{b2rt}.\mathrm{P11}\\
\mathrm{P12} &=_{df} \mathrm{b3rf}.\mathrm{P13} + \mathrm{b1rt}.\mathrm{P12}\\
\mathrm{P13} &=_{df} \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{b1wf}}.\mathrm{P1}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P2} &=_{df} \overline{\mathrm{b2wt}}.\mathrm{P21}\\
\mathrm{P21} &=_{df} \mathrm{b1rf}.\mathrm{P23} + \mathrm{b1rt}.\overline{\mathrm{b2wf}}.\mathrm{P22}\\
\mathrm{P22} &=_{df} \mathrm{b1rf}.\mathrm{P2} + \mathrm{b1rt}.\mathrm{P22}\\
\mathrm{P23} &=_{df} \mathrm{b3rf}.\mathrm{P24} + \mathrm{b3rt}.\mathrm{P23}\\
\mathrm{P24} &=_{df} \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{b2wf}}.\mathrm{P2}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P3} &=_{df} \overline{\mathrm{b3wt}}.\mathrm{P31}\\
\mathrm{P31} &=_{df} \mathrm{b1rf}.\mathrm{P33} + \mathrm{b1rt}.\overline{\mathrm{b3wf}}.\mathrm{P32}\\
\mathrm{P32} &=_{df} \mathrm{b1rf}.\mathrm{P3} + \mathrm{b1rt}.\mathrm{P32}\\
\mathrm{P33} &=_{df} \mathrm{b2rf}.\mathrm{P35} + \mathrm{b2rt}.\overline{\mathrm{b3wf}}.\mathrm{P34}\\
\mathrm{P34} &=_{df} \mathrm{b2rt}.\mathrm{P34} + \mathrm{b2rf}.\mathrm{P3}\\
\mathrm{P35} &=_{df} \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{b3wf}}.\mathrm{P3}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Lamport{-}3} &=_{df} (\mathrm{P1}|\mathrm{P2}|\mathrm{P3}|\mathrm{B1f}|\mathrm{B2f}|\mathrm{B3f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt},\\
&\qquad \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}, \mathrm{b3rf}, \mathrm{b3rt}, \mathrm{b3wf}, \mathrm{b3wt}\}
\end{aligned}$$

The agent Lamport-3 has 237 states and again is bisimulation equivalent to X where X $=_{df}$ enter.exit.X. A minimal agent bisimulation equivalent to the modified Lamport-3 (with 'enter_i' replacing 'enter' for $i = 1, 2, 3$ etc.) has 26 states.

6. Peterson's algorithm

The final algorithm considered in this paper is due to Peterson [PS]. We assume that there are two processes, two variables b_1 and b_2 of type **boolean** each having initial value **false**, and one variable k of type 1..2 with arbitrary initial value. The i^{th} process $(i = 1, 2)$ may be described as follows (where j is the index of the other process):

```
while true do
begin
    ⟨ noncritical section ⟩ ;
    $b_i$ := true ;
    $k := j$ ;
    while ($b_j$ and $k = j$) do skip ;
    ⟨ critical section ⟩ ;
    $b_i$ := false
end ;
```

The CCS representation of this algorithm (assuming the initial value of k to be 1) is the agent Peterson−2 below.

$$\begin{aligned}
\mathrm{B1f} &=_{df} \overline{\mathrm{b1rf}}.\mathrm{B1f} + \mathrm{b1wf}.\mathrm{B1f} + \mathrm{b1wt}.\mathrm{B1t}\\
\mathrm{B1t} &=_{df} \overline{\mathrm{b1rt}}.\mathrm{B1t} + \mathrm{b1wt}.\mathrm{B1t} + \mathrm{b1wf}.\mathrm{B1f}\\
\mathrm{B2f} &=_{df} \overline{\mathrm{b2rf}}.\mathrm{B2f} + \mathrm{b2wf}.\mathrm{B2f} + \mathrm{b2wt}.\mathrm{B2t}\\
\mathrm{B2t} &=_{df} \overline{\mathrm{b2rt}}.\mathrm{B2t} + \mathrm{b2wt}.\mathrm{B2t} + \mathrm{b2wf}.\mathrm{B2f}\\
\mathrm{K1} &=_{df} \overline{\mathrm{kr1}}.\mathrm{K1} + \mathrm{kw1}.\mathrm{K1} + \mathrm{kw2}.\mathrm{K2}\\
\mathrm{K2} &=_{df} \overline{\mathrm{kr2}}.\mathrm{K2} + \mathrm{kw2}.\mathrm{K2} + \mathrm{kw1}.\mathrm{K1}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P1} &=_{df} \overline{\mathrm{b1wt}}.\overline{\mathrm{kw2}}.\mathrm{P11}\\
\mathrm{P11} &=_{df} \mathrm{b2rt}.\mathrm{P11} + \mathrm{b2rf}.\mathrm{P12} + \mathrm{kr2}.\mathrm{P11} + \mathrm{kr1}.\mathrm{P12}\\
\mathrm{P12} &=_{df} \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{b1wf}}.\mathrm{P1}
\end{aligned}$$

$$\begin{aligned}
\mathrm{P2} &=_{df} \overline{\mathrm{b2wt}}.\overline{\mathrm{kw1}}.\mathrm{P21}\\
\mathrm{P21} &=_{df} \mathrm{b1rf}.\mathrm{P22} + \mathrm{b1rt}.\mathrm{P21} + \mathrm{kr1}.\mathrm{P21} + \mathrm{kr2}.\mathrm{P22}\\
\mathrm{P22} &=_{df} \mathrm{enter}.\mathrm{exit}.\overline{\mathrm{b2wf}}.\mathrm{P2}
\end{aligned}$$

$$\begin{aligned}
\text{Peterson}-2 &=_{df} (\mathrm{P1}|\mathrm{P2}|\mathrm{K1}|\mathrm{B1f}|\mathrm{B2f})\backslash\{\mathrm{b1rf}, \mathrm{b1rt}, \mathrm{b1wf}, \mathrm{b1wt},\\
&\qquad \mathrm{b2rf}, \mathrm{b2rt}, \mathrm{b2wf}, \mathrm{b2wt}, \mathrm{kr1}, \mathrm{kr2}, \mathrm{kw1}, \mathrm{kw2}\}
\end{aligned}$$

The agent Peterson−2 has 33 states and is bisimulation equivalent to X where X $=_{df}$ enter.exit.X. A minimal agent bisimulation equivalent to the modification of Peterson−2 with 'enter$_i$' replacing 'enter' in Pi ($i = 1,2$) etc., has 16 states.

Acknowledgments

I am indebted to Joachim Parrow, the author of the Concurrency Workbench (version 2.2), to Robin Milner who prompted me to write this report, and to Kevin Mitchell for patient ML-related assistance. This research has been supported by a grant from the Venture Research Unit of the British Petroleum Company.

References

[D] Dijkstra, E.W., *Solution of a Problem in Concurrent Programming Control*, Comm. A.C.M. 8/9 (1965).

[H] Hyman, H., *Comments on a Problem in Concurrent Programming Control*, Comm. A.C.M. 9/1 (1966).

[K] Knuth, D.E., *Additional Comments on a Problem in Concurrent Programming Control*, Comm. A.C.M. 9/5 (1966).

[KS] Kanellakis, P.C. and Smolka, S.A., *CCS Expressions, Finite State Processes, and Three Problems of Equivalence*, Proceedings of A.C.M. Symposium on Principles of Distributed Computing (1983).

[L] Lamport, L., *The Mutual Exclusion Problem Part II — Statement and Solutions*, J.A.C.M. 33/2 (1986).

[M] Milner, R., A Calculus of Communicating Systems, Springer-Verlag (1980).

[PS] Peterson, J.L. and Silberschatz, A., Operating System Concepts, 2nd ed., Addison Wesley (1985).

Postscript

Since completing this paper I have learned of some papers describing work involving the use of automated tools to analyse mutual exclusion algorithms. In a review paper entitled *Research on automatic verification of finite-state concurrent systems* (Ann. Rev. Comput. Sci. 1987, 2, 269-290), E. Clarke and O. Grümberg describe briefly an analysis of Lamport's algorithm in the case of two processes undertaken using a model-checker for the *fair computation tree logic* $\mathrm{CTL}^{\mathrm{F}}$. This analysis is described more fully in *Automatic verification of finite-state concurrent systems* by Clarke, E.A. Emerson and A.P. Sistla (ACM Toplas, 8/2, 1986). In a typewritten paper entitled *Design and synthesis of synchronization skeletons using branching time temporal logic*, Clarke and Emerson describe a method of automatically synthesizing concurrent programs from specifications expressed in the *computation tree logic* CTL, and illustrate the method by synthesizing an algorithm to ensure mutual exclusion for two processes. (See also *Using branching time temporal logic to synthesize synchronization skeletons*, Sci. Comput. Program., 2, 1982.) I wish to record here my gratitude to Edmund Clarke who kindly sent me a large number of papers describing work undertaken by him and his colleagues, including the papers cited above.

I believe that the AUTO system developed at INRIA (Sophia Antipolis) has been used to analyse some mutual exclusion algorithms and, in conjunction with the ESTEREL system, to analyse real-time systems.