MODULE *Boulanger*

This is a *PlusCal* encoding of the *Boulangerie* Algorithm of *Yoram Moses* and *Katia Patkin*–a variant of the *Bakery* Algorithm–and a proof that it implements mutual exclusion. The bakery algorithm appeared in

Leslie *Lamport* A New Solution of *Dijkstra*'s Concurrent Programming Problem Communications of the *ACM* 17, 8 (August 1974), $453 - 455$

The *PlusCal* encoding differs from the Moses-Patkin algorithm in one respect. To enter the critical section, the *PlusCal* version examines other processes one at a time–in the while loop at label $w1$. The Moses-Patkin algorithm performs those examinations in parallel. Because *PlusCal* does not allow sub-processes, it would be difficult (but not impossible) to express that algorithm in *PlusCal*. It would be easy to express their version in TLA+ (for example, by modifying the TLA+ translation of the *PlusCal* code), and it should be straightforward to convert the invariance proof presented here to a proof of the more general version. I will leave that as an exercise for others.

I started with a *PlusCal* encoding and invariance proof of the *Bakery* Algorithm. The only non-obvious part of that encoding is how it represented the safe registers assumed by the algorithm, which are registers in which reads and writes are not atomic. A safe register is represented by a variable r whose value is written by performing some number of atomic writes of non-deterministically chosen "legal" values to r followed by a single write of the desired value. A read of the register is performed by a single atomic read of r. It can be shown that this captures the semantics of a safe register.

Starting from the *PlusCal* version of the *Bakery* Algorithm, it was easy to modify it to the *Boulangerie* Algorithm (with the simplification described above). I model checked the algorithm on some small models to convince myself that there were no trivial errors that would be likely to arise from an error in the encoding. I then modified the invariant by a combination of a bit of thinking and a fair amount of trial and error, finding errors in the invariant by model checking very small models. (I checked it on two processes with chosen numbers restricted to be at most 3.)

When checking on a small model revealed no error in the invariant, I checked the proof with *TLAPS* (the TLA+ proof system). The high level proof, consisting of steps $\langle 1\rangle 1 - \langle 1\rangle 4$, are standard and are the same as for the *Bakery* Algorithm. *TLAPS* checks this simple four-step proof for the *Bakery* Algorithm with terminal BY proofs that just tell it to use the necessary assumptions and to expand all definitions. This didn't work for the hard part of the *Boulangerie* Algorithm–step $\langle 1\rangle 2$ that checks inductive invariance.

When a proof doesn't go through, one keeps decomposing the proof of the steps that aren't proved until one sees what the problem is. This decomposition is done with little thinking and no typing using the *Toolbox*'s Decompose Proof command. (The *Toolbox* is the *IDE* for the TLA+ tools.) Step $\langle 1\rangle 2$ has the form $A \land B \Rightarrow C$, where B is a disjunction, and the Decompose Proof command produces a level $- \langle 2\rangle$ proof consisting of subgoals essentially of the form $A \land Bi \Rightarrow C$ for the disjuncts Bi of B. Two of those subgoals weren't proved. I decomposed them both for several levels until I saw that in one of them, some step wasn't preserving the part of the invariant that asserts type-correctness. I then quickly found the culprit: a silly error in the type invariant in which I had in one place written the set *Proc* of process numbers instead of the set *Nat* of natural numbers. After correcting that error, only one of the level $- \langle 2\rangle$ subgoals remained unproved: step $\langle 2\rangle 5$. Using the Decompose Proof command as far as I could on that step, one substep remained unproved. (I think it was at level $\langle 5\rangle$.) Looking at what the proof obligations were, the obvious decomposition was a two-way case split, which I did by manually entering another level of subproof. One of those cases wasn't proved, so I tried another two-way case split on it. That worked. I then made that substep to the first step of the (level $\langle 3\rangle$) proof of $\langle 2\rangle 5$, moving its proof with it. With that additional fact, *TLAPS* was able to prove $\langle 2\rangle 5$ in one more step (the QED step).

The entire proof now is about 70 lines. I only typed 20 of those 70 lines. The rest either came from the original *Bakery* Algorithm (8-line) proof or were generated by the Decompose Proof Command.

I don't know how much time I actually spent writing the algorithm and its proof. Except for the final compaction of the (correct) proof of $\langle 2 \rangle 5$, the entire exercise took me two days. However, most of that was spent tracking down bugs in the *Toolbox*. We are in the process of moving the *Toolbox* to a new version of Eclipse, and there are many bugs that must be fixed before it's ready to be released. I would estimate that it would have taken me less than 4 hours without *Toolbox* bugs. I find it remarkable how little thinking the whole thing took.

This whole process was a lot easier than trying to write a convincing hand proof–a proof that I would regard as adequate to justify publication of the proof.

EXTENDS *Integers*, *TLAPS*

We first declare N to be the number of processes, and we assume that N is a natural number.

CONSTANT N
ASSUME $N \in Nat$

We define *Procs* to be the set $\{1, 2, \ldots, N\}$ of processes.

$$Procs \triangleq 1 \,..\, N$$

$\prec$ is defined to be the lexicographical less-than relation on pairs of numbers.

$$a \prec b \triangleq \begin{array}{l} \lor\ a[1] < b[1] \\ \lor\ (a[1] = b[1]) \land (a[2] < b[2]) \end{array}$$

** this is a comment containing the *PlusCal* code *

```
--algorithm Boulanger
{ variable num = [i ∈ Procs ↦ 0], flag = [i ∈ Procs ↦ FALSE] ;
  fair process ( p ∈ Procs )
    variables unchecked = {}, max = 0, nxt = 1, previous = −1 ;
    { ncs:- while ( TRUE )
            { e1:   either { flag[self] := ¬flag[self] ;
                             goto e1 }
                    or     { flag[self] := TRUE ;
                             unchecked := Procs \ {self} ;
                             max := 0
                           } ;
              e2:   while ( unchecked ≠ {} )
                      { with ( i ∈ unchecked )
                          { unchecked := unchecked \ {i} ;
                            if ( num[i] > max ) { max := num[i] }
                          }
                      } ;
              e3:   either { with ( k ∈ Nat ) { num[self] := k } ;
                             goto e3 }
                    or     { num[self] := max + 1 } ;
```

```
        e4:   either { flag[self] := ¬flag[self] ;
                       goto e4 }
              or     { flag[self] := FALSE ;
                         unchecked := IF num[self] = 1
                                      THEN 1 .. (self − 1)
                                      ELSE Procs \ {self}
                     } ;
        w1:   while ( unchecked ≠ {} )
                {     with ( i ∈ unchecked ) { nxt := i } ;
                      await ¬flag[nxt] ;
                      previous := −1 ;
                  w2: if (  ∨ num[nxt] = 0
                            ∨ ⟨num[self], self⟩ ≺ ⟨num[nxt], nxt⟩
                            ∨  ∧ previous ≠ −1
                               ∧ num[nxt] ≠ previous )
                          { unchecked := unchecked \ {nxt} ;
                            if ( unchecked = {} ) { goto cs }
                            else { goto w1 }
                          }
                      else { previous := num[nxt] ;
                             goto w2 }

                } ;
        cs:   skip ;  the critical section;
        exit: either { with ( k ∈ Nat ) { num[self] := k } ;
                       goto exit }
              or     { num[self] := 0 }
      }
  }
}
```

*** this ends the comment containg the pluscal code *********

BEGIN TRANSLATION (this begins the translation of the *PlusCal* code)

VARIABLES num, $flag$, pc, $unchecked$, max, nxt, $previous$

$vars \triangleq \langle num, flag, pc, unchecked, max, nxt, previous \rangle$

$ProcSet \triangleq (Procs)$

$Init \triangleq$ Global variables

$\land\ num = [i \in Procs \mapsto 0]$

$\land\ flag = [i \in Procs \mapsto \text{FALSE}]$

Process p

$\land\ unchecked = [self \in Procs \mapsto \{\}]$

$\land\ max = [self \in Procs \mapsto 0]$

$\land\ nxt = [self \in Procs \mapsto 1]$

$\land\ previous = [self \in Procs \mapsto -1]$

$$\land\ pc = [self \in ProcSet \mapsto \text{"ncs"}]$$

$$
\begin{array}{lll}
ncs(self) & \triangleq & \land\ pc[self] = \text{"ncs"} \\
 & & \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e1"}] \\
 & & \land\ \textsc{unchanged}\ \langle num, flag, unchecked, max, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
e1(self) & \triangleq & \land\ pc[self] = \text{"e1"} \\
 & & \land\ \lor\ \land\ flag' = [flag\ \textsc{except}\ ![self] = \neg flag[self]] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e1"}] \\
 & & \qquad\ \ \land\ \textsc{unchanged}\ \langle unchecked, max \rangle \\
 & & \quad\ \lor\ \land\ flag' = [flag\ \textsc{except}\ ![self] = \textsc{true}] \\
 & & \qquad\ \ \land\ unchecked' = [unchecked\ \textsc{except}\ ![self] = Procs \setminus \{self\}] \\
 & & \qquad\ \ \land\ max' = [max\ \textsc{except}\ ![self] = 0] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e2"}] \\
 & & \land\ \textsc{unchanged}\ \langle num, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
e2(self) & \triangleq & \land\ pc[self] = \text{"e2"} \\
 & & \land\ \textsc{if}\ unchecked[self] \neq \{\} \\
 & & \qquad \textsc{then}\ \ \land\ \exists\, i \in unchecked[self] : \\
 & & \qquad\qquad\qquad \land\ unchecked' = [unchecked\ \textsc{except}\ ![self] = unchecked[self] \setminus \{i\}] \\
 & & \qquad\qquad\qquad \land\ \textsc{if}\ num[i] > max[self] \\
 & & \qquad\qquad\qquad\qquad \textsc{then}\ \ \land\ max' = [max\ \textsc{except}\ ![self] = num[i]] \\
 & & \qquad\qquad\qquad\qquad \textsc{else}\ \ \ \land\ \textsc{true} \\
 & & \qquad\qquad\qquad\qquad\qquad\quad\ \land\ max' = max \\
 & & \qquad\qquad\quad\ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e2"}] \\
 & & \qquad \textsc{else}\ \ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e3"}] \\
 & & \qquad\qquad\quad\ \land\ \textsc{unchanged}\ \langle unchecked, max \rangle \\
 & & \land\ \textsc{unchanged}\ \langle num, flag, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
e3(self) & \triangleq & \land\ pc[self] = \text{"e3"} \\
 & & \land\ \lor\ \land\ \exists\, k \in Nat : \\
 & & \qquad\qquad\quad num' = [num\ \textsc{except}\ ![self] = k] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e3"}] \\
 & & \quad\ \lor\ \land\ num' = [num\ \textsc{except}\ ![self] = max[self] + 1] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e4"}] \\
 & & \land\ \textsc{unchanged}\ \langle flag, unchecked, max, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
e4(self) & \triangleq & \land\ pc[self] = \text{"e4"} \\
 & & \land\ \lor\ \land\ flag' = [flag\ \textsc{except}\ ![self] = \neg flag[self]] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"e4"}] \\
 & & \qquad\ \ \land\ \textsc{unchanged}\ unchecked \\
 & & \quad\ \lor\ \land\ flag' = [flag\ \textsc{except}\ ![self] = \textsc{false}] \\
 & & \qquad\ \ \land\ unchecked' = [unchecked\ \textsc{except}\ ![self] = \textsc{if}\ num[self] = 1 \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{then}\ 1 \,..\, (self - 1) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{else}\ \ Procs \setminus \{self\}] \\
 & & \qquad\ \ \land\ pc' = [pc\ \textsc{except}\ ![self] = \text{"w1"}]
\end{array}
$$

$$\land \text{UNCHANGED } \langle num, max, nxt, previous \rangle$$

$$
\begin{array}{lll}
w1(self) & \triangleq & \land\ pc[self] = \text{"w1"} \\
& & \land\ \text{IF } unchecked[self] \neq \{\} \\
& & \qquad \text{THEN } \land\ \exists\, i \in unchecked[self] : \\
& & \qquad\qquad\qquad nxt' = [nxt \text{ EXCEPT } ![self] = i] \\
& & \qquad\qquad\quad \land\ \neg flag[nxt'[self]] \\
& & \qquad\qquad\quad \land\ previous' = [previous \text{ EXCEPT } ![self] = -1] \\
& & \qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"w2"}] \\
& & \qquad \text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"cs"}] \\
& & \qquad\qquad\quad \land\ \text{UNCHANGED } \langle nxt, previous \rangle \\
& & \land\ \text{UNCHANGED } \langle num, flag, unchecked, max \rangle
\end{array}
$$

$$
\begin{array}{lll}
w2(self) & \triangleq & \land\ pc[self] = \text{"w2"} \\
& & \land\ \text{IF } \lor\ num[nxt[self]] = 0 \\
& & \qquad \lor\ \langle num[self], self \rangle \prec \langle num[nxt[self]], nxt[self] \rangle \\
& & \qquad \lor\ \ \land\ previous[self] \neq -1 \\
& & \qquad\quad\ \ \land\ num[nxt[self]] \neq previous[self] \\
& & \qquad \text{THEN } \land\ unchecked' = [unchecked \text{ EXCEPT } ![self] = unchecked[self] \setminus \{nxt[self]\}] \\
& & \qquad\qquad\quad \land\ \text{IF } unchecked'[self] = \{\} \\
& & \qquad\qquad\qquad\quad \text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"cs"}] \\
& & \qquad\qquad\qquad\quad \text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"w1"}] \\
& & \qquad\qquad\quad \land\ \text{UNCHANGED } previous \\
& & \qquad \text{ELSE } \land\ previous' = [previous \text{ EXCEPT } ![self] = num[nxt[self]]] \\
& & \qquad\qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"w2"}] \\
& & \qquad\qquad\quad \land\ \text{UNCHANGED } unchecked \\
& & \land\ \text{UNCHANGED } \langle num, flag, max, nxt \rangle
\end{array}
$$

$$
\begin{array}{lll}
cs(self) & \triangleq & \land\ pc[self] = \text{"cs"} \\
& & \land\ \text{TRUE} \\
& & \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"exit"}] \\
& & \land\ \text{UNCHANGED } \langle num, flag, unchecked, max, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
exit(self) & \triangleq & \land\ pc[self] = \text{"exit"} \\
& & \land\ \lor\ \land\ \exists\, k \in Nat : \\
& & \qquad\qquad\quad num' = [num \text{ EXCEPT } ![self] = k] \\
& & \qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"exit"}] \\
& & \qquad \lor\ \land\ num' = [num \text{ EXCEPT } ![self] = 0] \\
& & \qquad\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"ncs"}] \\
& & \land\ \text{UNCHANGED } \langle flag, unchecked, max, nxt, previous \rangle
\end{array}
$$

$$
\begin{array}{lll}
p(self) & \triangleq & ncs(self) \lor e1(self) \lor e2(self) \lor e3(self) \lor e4(self) \\
& & \quad \lor\ w1(self) \lor w2(self) \lor cs(self) \lor exit(self)
\end{array}
$$

$$Next \triangleq (\exists\, self \in Procs : p(self))$$

$$Spec \triangleq \land\ Init \land \Box[Next]_{vars}$$

$\land \forall self \in Procs : \mathrm{WF}_{vars}((pc[self] \neq \text{"ncs"}) \land p(self))$

END TRANSLATION (this ends the translation of the *PlusCal* code)

MutualExclusion asserts that two distinct processes are in their critical sections.

$$MutualExclusion \triangleq \forall i, j \in Procs : (i \neq j) \Rightarrow \neg \begin{array}{l} \land pc[i] = \text{"cs"} \\ \land pc[j] = \text{"cs"} \end{array}$$

The Inductive Invariant

TypeOK is the type-correctness invariant.

$$TypeOK \triangleq \begin{array}{l} \land num \in [Procs \to Nat] \\ \land flag \in [Procs \to \text{BOOLEAN}] \\ \land unchecked \in [Procs \to \text{SUBSET } Procs] \\ \land max \in [Procs \to Nat] \\ \land nxt \in [Procs \to Procs] \\ \land pc \in [Procs \to \{\text{"ncs"}, \text{"e1"}, \text{"e2"}, \text{"e3"}, \\ \qquad\qquad\qquad \text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}, \text{"exit"}\}] \\ \land previous \in [Procs \to Nat \cup \{-1\}] \end{array}$$

Before(i, j) is a condition that implies that $num[i] > 0$ and, if j is trying to enter its critical section and i does not change $num[i]$, then j either has or will choose a value of $num[j]$ for which

$\langle num[i], i \rangle \prec \langle num[j], j \rangle$

is true.

$$Before(i, j) \triangleq \begin{array}{l} \land num[i] > 0 \\ \land \lor pc[j] \in \{\text{"ncs"}, \text{"e1"}, \text{"exit"}\} \\ \quad \lor \land pc[j] = \text{"e2"} \\ \quad\quad \land \lor i \in unchecked[j] \\ \quad\quad\quad \lor max[j] \geq num[i] \\ \quad\quad\quad \lor (j > i) \land (max[j] + 1 = num[i]) \\ \quad \lor \land pc[j] = \text{"e3"} \\ \quad\quad \land \lor max[j] \geq num[i] \\ \quad\quad\quad \lor (j > i) \land (max[j] + 1 = num[i]) \\ \quad \lor \land pc[j] \in \{\text{"e4"}, \text{"w1"}, \text{"w2"}\} \\ \quad\quad \land \langle num[i], i \rangle \prec \langle num[j], j \rangle \\ \quad\quad \land (pc[j] \in \{\text{"w1"}, \text{"w2"}\}) \Rightarrow (i \in unchecked[j]) \\ \quad \lor \land num[i] = 1 \\ \quad\quad \land i < j \end{array}$$

Inv is the complete inductive invariant.

$$Inv \triangleq \begin{array}{l} \land TypeOK \\ \land \forall i \in Procs : \\ \quad \land (pc[i] \in \{\text{"ncs"}, \text{"e1"}, \text{"e2"}\}) \Rightarrow (num[i] = 0) \\ \quad \land (pc[i] \in \{\text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}\}) \Rightarrow (num[i] \neq 0) \\ \quad \land (pc[i] \in \{\text{"e2"}, \text{"e3"}\}) \Rightarrow flag[i] \end{array}$$

$\land (pc[i] = \text{"w2"}) \Rightarrow (nxt[i] \neq i)$
$\land (pc[i] \in \{\text{"e2"}, \text{"w1"}, \text{"w2"}\}) \Rightarrow i \notin unchecked[i]$
$\land (pc[i] \in \{\text{"w1"}, \text{"w2"}\}) \Rightarrow$
$\quad \forall j \in (Procs \setminus unchecked[i]) \setminus \{i\} : Before(i, j)$
$\land \ \land pc[i] = \text{"w2"}$
$\quad \land \ \lor (pc[nxt[i]] = \text{"e2"}) \land (i \notin unchecked[nxt[i]])$
$\quad\quad \lor pc[nxt[i]] = \text{"e3"}$
$\quad \Rightarrow max[nxt[i]] \geq num[i]$
$\land \ \land pc[i] = \text{"w2"}$
$\quad \land previous[i] \neq -1$
$\quad \land previous[i] \neq num[nxt[i]]$
$\quad \land pc[nxt[i]] \in \{\text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}\}$
$\quad \Rightarrow Before(i, nxt[i])$
$\land (pc[i] = \text{"cs"}) \Rightarrow \forall j \in Procs \setminus \{i\} : Before(i, j)$

Proof of Mutual Exclusion

This is a standard invariance proof, where $\langle 1 \rangle 2$ asserts that any step of the algorithm (including a stuttering step) starting in a state in which *Inv* is true leaves *Inv* true. Step $\langle 1 \rangle 4$ follows easily from $\langle 1 \rangle 1 - \langle 1 \rangle 3$ by simple temporal reasoning, checked by the *PTL* (Propositional Temporal Logic) backend prover.

THEOREM $Spec \Rightarrow \Box MutualExclusion$
$\langle 1 \rangle$ USE $N \in Nat$ DEFS $Procs, Inv, TypeOK, Before, \prec, ProcSet$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$
BY SMT DEF $Init$

$\langle 1 \rangle 2.$ $Inv \land [Next]_{vars} \Rightarrow Inv'$
$\langle 2 \rangle$ SUFFICES ASSUME Inv,
$[Next]_{vars}$
PROVE Inv'
OBVIOUS
$\langle 2 \rangle 1.$ ASSUME NEW $self \in Procs$,
$ncs(self)$
PROVE Inv'
BY $\langle 2 \rangle 1$, $Z3$ DEF $Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars$
$\langle 2 \rangle 2.$ ASSUME NEW $self \in Procs$,
$e1(self)$
PROVE Inv'
BY $\langle 2 \rangle 2$, $Z3$ DEF $Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars$
$\langle 2 \rangle 3.$ ASSUME NEW $self \in Procs$,
$e2(self)$
PROVE Inv'
BY $\langle 2 \rangle 3$, $Z3$ DEF $Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars$
$\langle 2 \rangle 4.$ ASSUME NEW $self \in Procs$,
$e3(self)$
PROVE Inv'

BY $\langle 2\rangle 4$, $Z3$ DEF Next, ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 5.$ ASSUME NEW $\mathit{self} \in \mathit{Procs}$,
$e4(\mathit{self})$
PROVE Inv'

$\langle 3\rangle$ ASSUME NEW $i \in \mathit{Procs}'$, $(\mathit{pc}[i] \in \{\text{“w1”}, \text{“w2”}\})'$
PROVE $(\forall j \in (\mathit{Procs} \setminus \mathit{unchecked}[i]) \setminus \{i\} : \mathit{Before}(i, j))'$

$\langle 4\rangle 1.$CASE $\mathit{self} = i$

$\langle 5\rangle$ SUFFICES ASSUME NEW $j \in ((\mathit{Procs} \setminus \mathit{unchecked}[i]) \setminus \{i\})'$
PROVE $\mathit{Before}(i, j)'$

OBVIOUS

$\langle 5\rangle 1.$CASE $i < j$

BY $\langle 4\rangle 1$, $\langle 5\rangle 1$, $\langle 2\rangle 5$, $Z3$ DEF ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 5\rangle 2.$CASE $j \leq i$

$\langle 6\rangle$ $\mathit{unchecked}'[i] = 1 \,..\, (i - 1)$

BY $\langle 4\rangle 1$, $\langle 2\rangle 5$ DEF $e4$

$\langle 6\rangle$ QED

BY $\langle 4\rangle 1$, $\langle 2\rangle 5$, $Z3$ DEF ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 2.$CASE $\mathit{self} \neq i$

BY $\langle 4\rangle 2$, $\langle 2\rangle 5$, $Z3$ DEF ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 4\rangle 3.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle$ QED

BY $\langle 2\rangle 5$, $Z3$ DEF ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 6.$ ASSUME NEW $\mathit{self} \in \mathit{Procs}$,
$w1(\mathit{self})$
PROVE Inv'

BY $\langle 2\rangle 6$, $Z3$ DEF Next, ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 7.$ ASSUME NEW $\mathit{self} \in \mathit{Procs}$,
$w2(\mathit{self})$
PROVE Inv'

BY $\langle 2\rangle 7$, $Z3$ DEF ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 8.$ ASSUME NEW $\mathit{self} \in \mathit{Procs}$,
$\mathit{cs}(\mathit{self})$
PROVE Inv'

BY $\langle 2\rangle 8$, $Z3$ DEF Next, ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 9.$ ASSUME NEW $\mathit{self} \in \mathit{Procs}$,
$\mathit{exit}(\mathit{self})$
PROVE Inv'

BY $\langle 2\rangle 9$, $Z3$ DEF Next, ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 10.$CASE UNCHANGED vars

BY $\langle 2\rangle 10$, $Z3$ DEF Next, ncs, p, $e1$, $e2$, $e3$, $e4$, $w1$, $w2$, cs, exit, vars

$\langle 2\rangle 11.$ QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 10$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$, $\langle 2\rangle 9$ DEF Next, p

$\langle 1 \rangle 3.$ $Inv \Rightarrow MutualExclusion$
BY SMT DEF $MutualExclusion$

$\langle 1 \rangle 4.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, PTL DEF $Spec$

$Trying(i) \triangleq pc[i] = \text{“e1”}$
$InCS(i) \triangleq pc[i] = \text{“cs”}$
$DeadlockFree \triangleq (\exists\, i \in Procs : Trying(i)) \leadsto (\exists\, i \in Procs : InCS(i))$
$StarvationFree \triangleq \forall\, i \in Procs : Trying(i) \leadsto InCS(i)$

* Modification History
* Last modified *Tue Dec* 18 12:08:37 *PST* 2018 by *lamport*
* Created *Thu Nov* 21 15:54:32 *PST* 2013 by *lamport*