The bakery algorithm originally appeared in:

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

The code for the algorithm given in that paper is:

```plaintext
begin integer j;
    L1: choosing [i] := 1 ;
        number[i] := 1 + maximum (number[1],..., number[N]);
        choosing[i] := 0;
        for j = 1 step 1 until N do
            begin
                L2: if choosing[j] /= 0 then goto L2;
                L3: if number[j] /= 0 and (number[j], j) < (number[i],i)
                    then goto L3;
            end;
        critical section;
        number[i] := 0;
        noncritical section;
        goto L1;
end
```

What makes the bakery algorithm interesting is that it doesn’t assume that reading or writing a memory register is an atomic operation. Instead it assumes safe registers, which ensure only that a read that doesn’t overlap a write obtains the current value of the register, but allows a read that overlaps a write to obtain any value of the correct type. This is modeled in TLA+ by letting the read be atomic but having a write operation perform a sequence of atomic writes of arbitrary type-correct values before atomically writing the desired value. (Only the shared registers number[i] and choosing[i] need be to be modeled in this way; operations to a process’s local variables can be taken to be atomic.)

This PlusCal version of the Atomic Bakery algorithm is one in which variables whose initial values are not used are initialized to particular type-correct values. If the variables were left uninitialized, the PlusCal translation would initialize them to a particular unspecified value. This would complicate the proof because it would make the type-correctness invariant more complicated, but it would be more efficient to model check. We could write a version that is more elegant and easy to prove, but less efficient to model check, by initializing the variables to arbitrarily chosen type-correct values.

EXTENDS Naturals, TLAPS

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

```
CONSTANT N
ASSUME N ∈ Nat
```

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

```
Procs ≡ 1 .. N
```

< is defined to be the lexicographical less-than relation on pairs of numbers.

```
```

** this is a comment containing the PlusCal code *
--algorithm Bakery
{ variables num = [i ∈ Procs ↦ 0], flag = [i ∈ Procs ↦ false] ; fair process ( p ∈ Procs ) variables unchecked = {}, max = 0, nxt = 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 } ;
go to e3 }
or { with ( i ∈ {j ∈ Nat : j > max} )
 { num[self] := i }
}
} ;
e4: either { flag[self] := ¬flag[self] ;
goto e4 }
or { flag[self] := false ;
unchecked := Procs \ {self} }
} ;
w1: while ( unchecked ≠ {} )
{ with ( i ∈ unchecked ) { nxt := i } ;
await ¬flag[nxt] ;
w2: await ∨ num[nxt] = 0 
∨ ⟨num[self] , self⟩ ∼ ⟨num[nxt] , nxt⟩ ;
unchecked := unchecked \ {nxt} ;
}
} ;
cs: skip ; the critical section;
ext: either { with ( k ∈ Nat ) { num[self] := k } ;
go to exit }
or { num[self] := 0 }
}
}

*** this ends the comment containing the pluscal code *******
BEGIN TRANSLATION (this begins the translation of the PlusCal code)

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

$ProcSet \triangleq$ (Procs)

Init $\triangleq$ Global variables
$\wedge$ num $= [i \in$ Procs $\mapsto 0$
$\wedge$ flag $= [i \in$ Procs $\mapsto$ FALSE$]$

Process $p$
$\wedge$ unchecked $= [self \in$ Procs $\mapsto \{\}]$
$\wedge$ max $= [self \in$ Procs $\mapsto \{\}]$
$\wedge$ nxt $= [self \in$ Procs $\mapsto 1$
$\wedge$ pc $= [self \in$ ProcSet $\mapsto "$ncs"]$

ncs(self) $\triangleq$ pc[self] = "$ncs$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e1"]
$\wedge$ UNCHANGED $\langle$ num, flag, unchecked, max, nxt $\rangle$

$e1$(self) $\triangleq$ pc[self] = "$e1$
$\wedge$ $\forall$ flag$'$ = [flag EXCEPTION ![self] = $\neg$flag[self]]
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e1$]
$\wedge$ UNCHANGED $\langle$ unchecked, max $\rangle$
$\lor$ flag$'$ = [flag EXCEPTION ![self] = TRUE$]
$\wedge$ unchecked$'$ = $\langle$unchecked EXCEPTION ![self] = Procs \{self$\}$$\rangle$
$\wedge$ max$'$ = $\langle$max EXCEPTION ![self] = 0$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e2$]
$\wedge$ UNCHANGED $\langle$ num, nxt $\rangle$

$e2$(self) $\triangleq$ pc[self] = "$e2$
$\wedge$ IF unchecked[self] $\neq \{\}$
$\lor$ THEN $\exists$ i $\in$ unchecked[self] :
$\wedge$ unchecked$'$ = $\langle$unchecked EXCEPTION ![self] = unchecked[self] \{i$\}$
$\wedge$ IF num[i] $>$ max[self]
$\lor$ THEN $\exists$ i $\in$ Nat $:$
num$'$ = $\langle$num EXCEPTION ![self] = k$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e3$]
$\wedge$ ELSE $\exists$ i $\in$ Nat $:$:
num$'$ = $\langle$num EXCEPTION ![self] = i$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e4$]
$\lor$ max$'$ = max
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e2$]
$\lor$ ELSE $\exists$ i $\in$ Nat $:$
num$'$ = $\langle$num EXCEPTION ![self] = i$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e3$]
$\wedge$ UNCHANGED $\langle$unchecked, max $\rangle$

$e3$(self) $\triangleq$ pc[self] = "$e3$
$\wedge$ $\forall$ $\exists$ k $\in$ Nat$:$
num$'$ = $\langle$num EXCEPTION ![self] = k$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e3$]
$\lor$ $\exists$ i $\in$ Nat $:$ j $>$ max[self]$:$
num$'$ = $\langle$num EXCEPTION ![self] = i$\rangle$
$\wedge$ pc$'$ = [pc EXCEPTION ![self] = "$e4$]
∧ UNCHANGED (flag, unchecked, max, nxt)

e4(self) \triangleq ∧ pc[self] = "e4"
∧ ∨ ∧ flag' = [flag EXCEPT ![self] = ¬flag[self]]
∧ pc' = [pc EXCEPT ![self] = "e4"]
∧ UNCHANGED unchecked
∨ ∧ flag' = [flag EXCEPT ![self] = FALSE]
∧ unchecked' = [unchecked EXCEPT ![self] = Procs \ {self}]
∧ pc' = [pc EXCEPT ![self] = "w1"]
∧ UNCHANGED (num, max, nxt)

w1(self) \triangleq ∧ pc[self] = "w1"
∧ IF unchecked[self] ≠ {}
THEN ∧ ∃ i ∈ unchecked[self] :
    nxt' = [nxt EXCEPT ![self] = i]
    ∧ ¬flag[nxt'[self]]
∧ pc' = [pc EXCEPT ![self] = "w2"]
ELSE ∧ pc' = [pc EXCEPT ![self] = "cs"]
∧ nxt' = nxt
∧ UNCHANGED (num, flag, unchecked, max)

w2(self) \triangleq ∧ pc[self] = "w2"
∧ ∨ num[nxt[self]] = 0
    ∨ (num[self], self) ≺ (num[nxt[self]], nxt[self])
∧ unchecked' = [unchecked EXCEPT ![self] = unchecked[self] \ {nxt[self]}]
∧ pc' = [pc EXCEPT ![self] = "w1"]
∧ UNCHANGED (num, flag, unchecked, max, nxt)

cs(self) \triangleq ∧ pc[self] = "cs"
∧ TRUE
∧ pc' = [pc EXCEPT ![self] = "exit"]
∧ UNCHANGED (num, flag, unchecked, max, nxt)

exit(self) \triangleq ∧ pc[self] = "exit"
∧ ∨ ∧ ∃ k ∈ Nat :
    num' = [num EXCEPT ![self] = k]
∧ pc' = [pc EXCEPT ![self] = "exit"]
∨ ∧ num' = [num EXCEPT ![self] = 0]
∧ pc' = [pc EXCEPT ![self] = "ncs"]
∧ UNCHANGED (flag, unchecked, max, nxt)

p(self) \triangleq ncs(self) \lor e1(self) \lor e2(self) \lor e3(self) \lor e4(self)
\lor w1(self) \lor w2(self) \lor cs(self) \lor exit(self)

Next \triangleq (∃ self ∈ Procs : p(self))

Spec \triangleq ∧ Init \land □[Next]_{vars}
\[ \forall \text{self} \in \text{Procs} : \WF_{\text{vars}}((\text{pc}[\text{self}] \neq \text{"ncs"}) \land p(\text{self})) \]
\[ \forall \text{self} \in \text{Procs} : \WF_{\text{vars}}(\land e1(\text{self}) \lor e3(\text{self}) \lor e4(\text{self}) \lor \text{exit}(\text{self}) \land (\text{pc}[\text{self}] \neq \text{pc}[\text{self}])) \]

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

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

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

The Inductive Invariant

**TypeOK** is the type-correctness invariant.

\[ \text{TypeOK} \triangleq \land \text{num} \in [\text{Procs} \rightarrow \text{Nat}] \land \text{flag} \in [\text{Procs} \rightarrow \text{BOOLEAN}] \land \text{unchecked} \in [\text{Procs} \rightarrow \text{SUBSET} \ \text{Procs}] \land \text{max} \in [\text{Procs} \rightarrow \text{Nat}] \land \text{nxt} \in [\text{Procs} \rightarrow \text{Procs}] \land \text{pc} \in [\text{Procs} \rightarrow \{\text{"ncs"}, \text{"e1"}, \text{"e2"}, \text{"e3"}, \text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}, \text{"exit"}\}] \]

\[ \text{Before}(i, j) \text{ is a condition that implies that } \text{num}[i] > 0 \text{ and, if } j \text{ is trying to enter its critical section and } i \text{ does not change } \text{num}[i], \text{ then } j \text{ either has or will choose a value of } \text{num}[j] \text{ for which } \langle \text{num}[i], i \rangle \prec \langle \text{num}[j], j \rangle \text{ is true.} \]

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

**Inv** is the complete inductive invariant.

\[ \text{Inv} \triangleq \land \text{TypeOK} \land \forall i \in \text{Procs} : \land \text{This conjunct is not needed for mutual exclusion, but it is needed to prove liveness.} \]
\[ (\text{pc}[i] \in \{\text{"ncs"}, \text{"e1"}, \text{"e2"}\}) \Rightarrow (\text{num}[i] = 0) \land (\text{pc}[i] \in \{\text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}\}) \Rightarrow (\text{num}[i] \neq 0) \land (\text{pc}[i] \in \{\text{"e2"}, \text{"e3"}\}) \Rightarrow \text{flag}[i] \]
This conjunct is not needed to prove mutual exclusion. It’s needed to prove liveness, but it could be removed if the \( \prec \) in the wait condition were changed to \( \leq \).

\[
\begin{align*}
& (pc[i] = "w2") \Rightarrow (nxt[i] \neq i) \\
& \land pc[i] \in \{"e2", "w1", "w2"\} \Rightarrow i \notin unchecked[i] \\
& \land (pc[i] \in \{"w1", "w2"\}) \Rightarrow \\
& \quad \forall j \in (Procs \setminus unchecked[i]) \setminus \{i\} : Before(i, j) \\
& \land (pc[i] = "w2") \\
& \land \lor (pc[nxt[i]] = "e2") \land (i \notin unchecked[nxt[i]]) \\
& \quad \lor pc[nxt[i]] = "e3" \\
& \Rightarrow \max[nxt[i]] \geq \num[i] \\
& \land (pc[i] = "cs") \Rightarrow \forall j \in Procs \setminus \{i\} : Before(i, j)
\end{align*}
\]

Proving Mutual Exclusion

This is a standard invariance proof, where (1)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 (1)4 follows easily from (1)1 – (1)3 by simple temporal reasoning, but TLAPS does not yet do any temporal reasoning.

**THEOREM** Spec \( \Rightarrow \Box \text{MutualExclusion} \)

(1)1. Init \( \Rightarrow \) Inv

(1)2. Inv \land [Next]_{vars} \Rightarrow Inv'

(1)3. Inv \Rightarrow \text{MutualExclusion}

(1)4. QED

\[
\begin{align*}
\text{Trying}(i) & \triangleq pc[i] = "e1" \\
\text{InCS}(i) & \triangleq pc[i] = "cs" \\
\text{DeadlockFree} & \triangleq (\exists i \in \text{Procs} : \text{Trying}(i)) \sim (\exists i \in \text{Procs} : \text{InCS}(i)) \\
\text{StarvationFree} & \triangleq \forall i \in \text{Procs} : \text{Trying}(i) \sim \text{InCS}(i)
\end{align*}
\]

\*

Modification History

\* Last modified Tue Dec 18 13:48:46 PST 2018 by lamport

\* Created Thu Nov 21 15:54:32 PST 2013 by lamport