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 \texttt{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 nondeterministically 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 \( \forall Y \) 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 – (2) proof consisting of subgoals essentially of the form \( A \land B_i \Rightarrow C \) for the disjuncts \( B_i \) 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 \texttt{Proc} of process numbers instead of the set \texttt{Nat} of natural numbers. After correcting that error, only one of the level – (2) 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 \texttt{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 \( 2 \rangle 5 \), 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** \textit{Integers, TLAPS}

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

**CONSTANT** \( N \)

**ASSUME** \( N \in \text{Nat} \)

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

\( \text{Procs} \triangleq 1 \ldots N \)

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

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

---

**algorithm** \textit{Boulanger}

\{ 
  \textbf{variable} num = \{i \in \text{Procs} \mapsto 0\}, flag = \{i \in \text{Procs} \mapsto \text{FALSE}\};

  \textbf{fair process} ( p \in \text{Procs} )

  \textbf{variables} unchecked = \{\}, max = 0, nxt = 1, previous = -1;

  \{ 
    ncs:- \textbf{while} ( \text{TRUE} )
    \{ 
      e1: \textbf{either} \{ 
        flag[self] := \neg flag[self];
        \text{goto} e1
      \} 
      \textbf{or} \{ 
        flag[self] := \text{TRUE};
        unchecked := \text{Procs} \setminus \{self\};
        max := 0
      \};

      e2: \textbf{while} ( unchecked \neq \{\} )
      \{ 
        with ( i \in unchecked )
        \{ 
          unchecked := unchecked \setminus \{i\};
          if ( num[i] > max ) \{ max := num[i] \}
        \}
      \};

      e3: \textbf{either} \{ 
        with ( k \in \text{Nat} ) \{ num[self] := k \}
        \text{goto} e3
      \} 
      \textbf{or} \{ 
        num[self] := max + 1
      \};
  \}
\}
BEGIN TRANSLATION (this begins the translation of the PlusCal code)

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

vars ≜ (num, flag, pc, unchecked, max, nxt, previous)

ProcSet ≜ (Procs)

Init ≜ Global variables
  ∧ num = [i ∈ Procs ↦ 0]
  ∧ flag = [i ∈ Procs ↦ FALSE]
  ∧ unchecked = [self ∈ Procs ↦ {}]
  ∧ max = [self ∈ Procs ↦ 0]
  ∧ nxt = [self ∈ Procs ↦ 1]
  ∧ previous = [self ∈ Procs ↦ -1]

BEGIN TRANSLATION (this begins the translation of the PlusCal code)
\[ \begin{align*}
\land pc &= [self \in ProcSet \mapsto \text{"ncs"}] \\
ncs(self) &\triangleq \land pc[self] = \text{"ncs"} \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e1"}] \\
&\text{UNCHANGED } \langle \text{num, flag, unchecked, max, nxt, previous} \rangle \\
e1(self) &\triangleq \land pc[self] = \text{"e1"} \\
&\land \lor \land \text{flag}' = [\text{flag EXCEPT } ![self] = \neg\text{flag}[self]] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e1"}] \\
&\text{UNCHANGED } \langle \text{unchecked, max} \rangle \\
&\lor \land \text{flag}' = [\text{flag EXCEPT } ![self] = \text{TRUE}] \\
&\land \text{unchecked}' = [\text{unchecked EXCEPT } ![self] = Procs \setminus \{self\}] \\
&\land max' = [\text{max EXCEPT } ![self] = 0] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e2"}] \\
&\text{UNCHANGED } \langle \text{num, flag, unchecked, max, nxt, previous} \rangle \\
e2(self) &\triangleq \land pc[self] = \text{"e2"} \\
&\land \text{IF } \text{unchecked}[self] \neq \{\} \\
&\land \exists i \in \text{unchecked}[self] : \\
&\land \text{num}' = [\text{num EXCEPT } ![self] = \text{num}[i]] \\
&\land \text{if } \text{num}[i] > \text{max}[self] \\
&\land \text{max}' = [\text{max EXCEPT } ![self] = \text{num}[i]] \\
&\text{ELSE } \land \text{TRUE} \\
&\land \text{max}' = \text{max} \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e2"}] \\
&\text{ELSE } \land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e3"}] \\
&\text{UNCHANGED } \langle \text{unchecked, max} \rangle \\
e3(self) &\triangleq \land pc[self] = \text{"e3"} \\
&\land \lor \land \exists k \in \text{Nat} : \\
&\land \text{num}' = [\text{num EXCEPT } ![self] = k] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e3"}] \\
&\lor \land \text{num}' = [\text{num EXCEPT } ![self] = \text{max}[self] + 1] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e4"}] \\
&\text{UNCHANGED } \langle \text{flag, unchecked, max, nxt, previous} \rangle \\
e4(self) &\triangleq \land pc[self] = \text{"e4"} \\
&\land \lor \land \text{flag}' = [\text{flag EXCEPT } ![self] = \neg\text{flag}[self]] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"e4"}] \\
&\text{UNCHANGED } \text{unchecked} \\
&\lor \land \text{flag}' = [\text{flag EXCEPT } ![self] = \text{FALSE}] \\
&\land \text{unchecked}' = [\text{unchecked EXCEPT } ![self] = \text{IF } \text{num}[self] = 1 \\
&\text{THEN } 1 \ldots (\text{self} - 1) \\
&\text{ELSE } \text{Procs} \setminus \{\text{self}\}] \\
&\land pc'[pc] = [pc \text{ EXCEPT } ![self] = \text{"w1"}] \\
\end{align*} \]
\[ w1(self) \triangleq \land pc[self] = "w1" \\
\land \text{IF } unchecked[self] \neq \{\} \\
\text{THEN } \land \exists i \in unchecked[self]: \\
\land \text{next } = \text{next except } \neg [self] = i \\
\land \neg \text{flag}[\text{next }[self]] \\
\land \text{previous } = \text{previous except } \neg [self] = 1 \\
\land pc' = [pc \text{ except } [self] = "w2"] \\
\text{ELSE } \land pc' = [pc \text{ except } [self] = "cs"] \\
\land \text{UNCHANGED } (\text{next, previous}) \\
\land \text{UNCHANGED } (\text{num, max, nxt, previous}) \]

\[ w2(self) \triangleq \land pc[self] = "w2" \\
\land \text{IF } \lor \text{num}[\text{next }[self]] = 0 \\
\lor \langle \text{num }[self], \text{self} \rangle < \langle \text{num }[\text{next }[self]], \text{next }[self] \rangle \\
\lor \land \text{previous } = \neg 1 \\
\lor \land \text{num }[\text{next }[self]] \neq \text{previous }[self] \\
\text{THEN } \land \text{unchanged } = \text{unchanged except } \neg [self] = \text{unchanged }[self] \setminus \{\text{next }[self]\}] \\
\land \text{IF unchanged }[self] = \{\} \\
\text{THEN } \land pc' = [pc \text{ except } [self] = "cs"] \\
\text{ELSE } \land pc' = [pc \text{ except } [self] = "w1"] \\
\land \text{UNCHANGED previous} \\
\text{ELSE } \land \text{previous } = \text{previous except } \neg [self] = \text{num }[\text{next }[self]] \\
\land pc' = [pc \text{ except } [self] = "w2"] \\
\land \text{UNCHANGED unchecked} \\
\land \text{UNCHANGED } (\text{num, flag, max, nxt}) \]

\[ cs(self) \triangleq \land pc[self] = "cs" \\
\land \text{TRUE} \\
\land pc' = [pc \text{ except } [self] = "exit"] \\
\land \text{UNCHANGED } (\text{num, flag, unchecked, max, nxt, previous}) \]

\[ exit(self) \triangleq \land pc[self] = "exit" \\
\land \lor \land \exists k \in \text{Nat} : \\
\land \text{num }' = \text{num except } \neg [self] = k \\
\land pc' = [pc \text{ except } [self] = "exit"] \\
\lor \land \text{num }' = \text{num except } \neg [self] = 0 \\
\land pc' = [pc \text{ except } [self] = "ncs"] \\
\land \text{UNCHANGED } (\text{flag, unchecked, max, nxt, previous}) \]

\[ p(self) \triangleq \text{ncs(self)} \lor e1(self) \lor e2(self) \lor e3(self) \lor e4(self) \\
\lor w1(self) \lor w2(self) \lor \text{cs(self)} \lor \text{exit(self)} \]

\[ \text{Next} \triangleq (\exists self \in \text{Procs} : p(self)) \]

\[ \text{Spec} \triangleq \land \text{Init} \land \Box [\text{Next}]_{\text{vars}} \]
\[ \forall \text{self} \in \text{Procs} : \text{WF\textsubscript{vars}}((\text{pc}[\text{self}] \neq \text{"ncs"}) \land p(\text{self})) \]

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

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

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

The Inductive Invariant

*TypeOK* is the type-correctness invariant.

\[ \text{TypeOK} \triangleq \land \text{num} \in [\text{Procs} \rightarrow \text{Nat}] \land \text{flag} \in [\text{Procs} \rightarrow \text{BOOLEAN}] \land \text{unchecked} \in [\text{Procs} \rightarrow \text{SUBSET Procs}] \land \text{max} \in [\text{Procs} \rightarrow \text{Nat}] \land \text{nat} \in [\text{Procs} \rightarrow \text{Procs}] \land \text{pc} \in [\text{Procs} \rightarrow \{ \text{"ncs"}, \text{"e1"}, \text{"e2"}, \text{"e3"}, \\
\text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}, \text{"exit"} \}] \land \text{previous} \in [\text{Procs} \rightarrow \text{Nat} \cup \{-1\}] \]

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

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

*Inv* is the complete inductive invariant.

\[ \text{Inv} \triangleq \land \text{TypeOK} \land \forall i \in \text{Procs} : \land (\text{pc}[i] \in \{ \text{"ncs"}, \text{"e1"}, \text{"e2"} \}) \Rightarrow (\text{num}[i] = 0) \land (\text{pc}[i] \in \{ \text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"} \}) \Rightarrow (\text{num}[i] \neq 0) \land (\text{pc}[i] \in \{ \text{"e2"}, \text{"e3"} \}) \Rightarrow \text{flag}[i] \]
\( \land (pc[i] = \text{"w2"}) \Rightarrow (nxt[i] \neq i) \)
\( \land (pc[i] \in \{\text{"e2"}, \text{"w1"}, \text{"w2"}\}) \Rightarrow i \notin \text{unchecked}[i] \)
\( \land (pc[i] \in \{\text{"w1"}, \text{"w2"}\}) \Rightarrow \forall j \in (\text{Procs} \setminus \text{unchecked}[i]) \setminus \{i\} : \text{Before}(i, j) \)
\( \land (pc[i] = \text{"w2"}) \land \forall (pc[nxt[i]] = \text{"e2"}) \land (i \notin \text{unchecked}[nxt[i]]) \land \neg pc[nxt[i]] = \text{"e3"} \Rightarrow \max[nxt[i]] \geq \text{num}[i] \)
\( \land (pc[i] = \text{"w2"}) \land \text{previous}[i] \neq -1 \land \text{previous}[i] \neq \text{num}[nxt[i]] \land pc[nxt[i]] \in \{\text{"e4"}, \text{"w1"}, \text{"w2"}, \text{"cs"}\} \Rightarrow \text{Before}(i, nxt[i]) \)
\( (pc[i] = \text{\text{cs"})} \Rightarrow \forall j \in \text{Procs} \setminus \{i\} : \text{Before}(i, j) \)

Proof of Mutual Exclusion

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

\textbf{THEOREM Spec} \(\Rightarrow\) \(\square\text{MutualExclusion}\)

\(\langle 1 \rangle\) use \(N \in \text{Natdefs Procs, Inv, TypeOK, Before, }\prec, \text{ProcSet} \)

\(\langle 1\rangle.1. \text{Init} \Rightarrow \text{Inv} \)
BY SMT DEF \text{Init}

\(\langle 1\rangle.2. \text{Inv} \land [\text{Next}]_{\text{vars}} \Rightarrow \text{Inv}'\)
(2) SUFFICES ASSUME \text{Inv}, [\text{Next}]_{\text{vars}}
PROVE \text{Inv}'

OBVIOUS

\(\langle 2\rangle.1. \text{ASSUME NEW self} \in \text{Procs, ncs(self)} \)
PROVE \text{Inv}'
BY \(\langle 2\rangle.1, Z3 \text{ DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars} \)

\(\langle 2\rangle.2. \text{ASSUME NEW self} \in \text{Procs, e1(self)} \)
PROVE \text{Inv}'
BY \(\langle 2\rangle.2, Z3 \text{ DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars} \)

\(\langle 2\rangle.3. \text{ASSUME NEW self} \in \text{Procs, e2(self)} \)
PROVE \text{Inv}'
BY \(\langle 2\rangle.3, Z3 \text{ DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars} \)

\(\langle 2\rangle.4. \text{ASSUME NEW self} \in \text{Procs, e3(self)} \)
PROVE \text{Inv}'}
BY (2) 4, Z3 DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 5. ASSUME NEW self ∈ Procs,
   e4(self)
   PROVE Inv'

⟨3⟩ ASSUME NEW i ∈ Procs', (pc[i] ∈ {"w1", "w2"})'
   PROVE (∀ j ∈ (Procs \ unchecked[i]) \ {i} : Before(i, j))'
   ⟨4⟩ 1.CASE self = i
   ⟨5⟩ SUFFICES ASSUME NEW j ∈ ((Procs \ unchecked[i]) \ {i})'
   PROVE Before(i, j)'
   OBVIOUS
   ⟨5⟩ 1.CASE i < j
   BY ⟨4⟩ 1, ⟨5⟩ 1, (2) 5, Z3 DEF ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
   ⟨5⟩ 2.CASE j ≤ i
   ⟨6⟩ unchecked'[i] = 1..(i - 1)
   BY ⟨4⟩ 1, (2) 5 DEF e4
   ⟨6⟩ QED
   BY ⟨4⟩ 1, ⟨2⟩ 5, Z3 DEF ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
   ⟨5⟩ 3. QED
   BY ⟨5⟩ 1, ⟨5⟩ 2
   ⟨4⟩ 2.CASE self ≠ i
   BY ⟨4⟩ 2, (2) 5, Z3 DEF ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
   ⟨4⟩ 3. QED
   BY ⟨4⟩ 1, ⟨4⟩ 2
⟨3⟩ QED
   BY ⟨2⟩ 5, Z3 DEF ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 6. ASSUME NEW self ∈ Procs,
   w1(self)
   PROVE Inv'
   BY ⟨2⟩ 6, Z3 DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 7. ASSUME NEW self ∈ Procs,
   w2(self)
   PROVE Inv'
   BY ⟨2⟩ 7, Z3 DEF ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 8. ASSUME NEW self ∈ Procs,
   cs(self)
   PROVE Inv'
   BY ⟨2⟩ 8, Z3 DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 9. ASSUME NEW self ∈ Procs,
   exit(self)
   PROVE Inv'
   BY ⟨2⟩ 9, Z3 DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 10. CASE UNCHANGED vars
   BY ⟨2⟩ 10, Z3 DEF Next, ncs, p, e1, e2, e3, e4, w1, w2, cs, exit, vars
(2) 11. QED
   BY ⟨2⟩ 1, ⟨2⟩ 10, ⟨2⟩ 2, ⟨2⟩ 3, ⟨2⟩ 4, ⟨2⟩ 5, ⟨2⟩ 6, ⟨2⟩ 7, ⟨2⟩ 8, ⟨2⟩ 9 DEF Next, p

8
(1)3. Inv ⇒ MutualExclusion
   BY SMT DEF MutualExclusion
(1)4. QED
   BY (1)1, (1)2, (1)3, PTL DEF Spec

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

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