MODULE *BakeryDeconstructedAtomic*

This is the *PlusCal* specification of the deconstructed bakery algorithm in the paper

Deconstructing the Bakery to Build a Distributed State Machine

There is one simplification that has been made in the *PlusCal* version: the registers $localCh[i][j]$ have been made atomic, a read or write being a single atomic action. This doesn't affect the derivation of the distributed bakery algorithm from the deconstructed algorithm, which also makes the simplifying assumption those registers are atomic because they disappear from the final algorithm.

Here are some of the changes made to the paper's notation to conform to *PlusCal*/TLA+. Tuples are enclosed in $\langle\rangle$, so we write $\langle i, j\rangle$ instead of (i,j). There's no upside down "?" symbol in TLA+, so that's replaced by the identifier qm.

The pseudo-code for main process i has two places in which subprocesses (i, j) are forked and process i resumes execution when they complete. *PlusCal* doesn't have subprocesses. This is represented in *PlusCal* by having a single process $\langle i, j\rangle$ executing concurrently with process i, synchronizing appropriately using the variable pc.

Here is the basic idea:

This pseudo-code for process i:

```
main code ;
process j # i \in S
  s1: subprocess code
end process
p2: more main code
```

is expressed in *PlusCal* as follows:

In process i

```
    main code ;
p2: await \A j # i : pc[<<i,j>>] = "s2"
    more main code
```

In process $\langle i, j\rangle$

```
s1: await pc[i] = "p2"
    subprocess code ;
s2: ...
```

Also, processes have identifiers and, for reasons that are not important here, we can't use i as the identifier for process i, so we use $\langle i\rangle$. So, $pc[i]$ in the example above should be $pc[\langle i\rangle]$. In the pseudo-code, process i also launches asynchronous processes (i, j) to set $localNum[j][i]$ to 0. In the code, these are another set of processes with ids $\langle i, j, \text{"wr"}\rangle$.

We could simplify this algorithm by not waiting for $localNum[j][i]$ to equal 0 in subprocess $\langle i, j\rangle$ and having the asynchronous write of 0 not do anything if process i has begun the write to $localCh[i][j]$ that sets its value to $number[i]$. However, *I* think *I* like the algorithm in the paper the way it is because it makes the pseudo-code more self-contained.

Like the pseudo-code shown in the paper, this version of the algorithm represents the M action as an atomic step.

EXTENDS *Data*

****

--algorithm *Decon*{
  variables $number = [p \in Procs \mapsto 0]$,

```
            localNum  = [p ∈ Procs ↦ [q ∈ OtherProcs(p) ↦ 0]],
            localCh   = [p ∈ Procs ↦ [q ∈ OtherProcs(p) ↦ 0]] ;

fair process ( main ∈ ProcIds )
{
 ncs:- while ( TRUE ) {
            skip ;   noncritical section
        M: await ∀ p ∈ SubProcsOf(self[1]) : pc[p] = "test" ;
            with ( v ∈ {n ∈ Nat \ {0} : ∀ j  ∈ OtherProcs(self[1]) :
                        localNum[self[1]][j] ≠ qm ⇒ n > localNum[self[1]][j]} ) {
               number[self[1]] := v ;
               localNum := [j ∈ Procs ↦
                              [i ∈ OtherProcs(j) ↦
                                IF i = self[1] THEN qm
                                               ELSE localNum[j][i]]] ;
            } ;
        L:  await ∀ p ∈ SubProcsOf(self[1]) : pc[p] = "ch" ;
       cs:  skip ;   critical section
        P:  number[self[1]] := 0 ;
            localNum := [j ∈ Procs ↦
                           [i ∈ OtherProcs(j) ↦
                             IF i = self[1] THEN qm
                                            ELSE localNum[j][i]]] ;
         }
 }

fair process ( sub ∈ SubProcs ) {
  ch: while ( TRUE ) {
          await pc[⟨self[1]⟩] = "M" ;
          localCh[self[2]][self[1]] := 1 ;
  test:   await pc[⟨self[1]⟩] = "L" ;
          localNum[self[2]][self[1]] := number[self[1]] ;
    Lb:   localCh[self[2]][self[1]] := 0 ;
    L2:   await localCh[self[1]][self[2]] = 0 ;
    L3:-   See below for an explanation of why there is no fairness here.
          await (localNum[self[1]][self[2]] ∉ {0, qm}) ⇒
                  (⟨number[self[1]], self[1]⟩ ≪
                     ⟨localNum[self[1]][self[2]], self[2]⟩)
```

The await condition is written in the form A $\Rightarrow B$ rather than A $\vee B$ because when *TLC* is finding new states, when evaluating A $\vee B$ it evaluates B even when A is true, and in this case that would produce an error if $localNum[self[1]][self[2]]$ equals qm.

```
        }
 }
```

We allow process $\langle i, j, \text{"wr"} \rangle$ to set $localNum[j][i]$ to 0 only if it has not already been set to qm by process $\langle i \rangle$ in action $M0$. We could also allow it to write 0 after that write of qm but before process $\langle i, j \rangle$ executes statement test. Such a write just decreases the possible executions, so eliminating this possibility doesn't forbid any possible executions.

$$
\begin{array}{l}
\textbf{fair process } (\ wrp \in WrProcs\ )\ \{ \\
\quad wr\!: \textbf{while } (\ \text{TRUE}\ )\ \{ \\
\qquad\quad \textbf{await } \land\ localNum[self[2]][self[1]] = qm \\
\qquad\qquad\qquad\ \ \land\ pc[\langle self[1] \rangle] \in \{\text{"ncs"}, \text{"M"}\}\,; \\
\qquad\quad localNum[self[2]][self[1]] := 0\,; \\
\qquad\quad \} \\
\quad \} \\
\}
\end{array}
$$

****

BEGIN TRANSLATION ($chksum(pcal) =$ "$7827c38d$" $\land$ $chksum(tla) =$ "$83cb6c12$")

$\text{VARIABLES } number,\ localNum,\ localCh,\ pc$

$$vars \triangleq \langle number, localNum, localCh, pc \rangle$$

$$ProcSet \triangleq (ProcIds) \cup (SubProcs) \cup (WrProcs)$$

$$
\begin{array}{ll}
Init \triangleq & \text{Global variables} \\
& \land\ number = [p \in Procs \mapsto 0] \\
& \land\ localNum = [p \in Procs \mapsto [q \in OtherProcs(p) \mapsto 0]] \\
& \land\ localCh = [p \in Procs \mapsto [q \in OtherProcs(p) \mapsto 0]] \\
& \land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in ProcIds \rightarrow \text{"ncs"} \\
& \qquad\qquad\qquad\qquad\qquad\quad \Box\ \ self \in SubProcs \rightarrow \text{"ch"} \\
& \qquad\qquad\qquad\qquad\qquad\quad \Box\ \ self \in WrProcs \rightarrow \text{"wr"}]
\end{array}
$$

$$
\begin{array}{ll}
ncs(self) \triangleq & \land\ pc[self] = \text{"ncs"} \\
& \land\ \text{TRUE} \\
& \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"M"}] \\
& \land\ \text{UNCHANGED } \langle number, localNum, localCh \rangle
\end{array}
$$

$$
\begin{array}{ll}
M(self) \triangleq & \land\ pc[self] = \text{"M"} \\
& \land\ \forall p \in SubProcsOf(self[1]) : pc[p] = \text{"test"} \\
& \land\ \exists v \in \{n \in Nat \setminus \{0\} : \forall j \in OtherProcs(self[1]) : \\
& \qquad\qquad localNum[self[1]][j] \neq qm \Rightarrow n > localNum[self[1]][j]\} : \\
& \qquad \land\ number' = [number \text{ EXCEPT } ![self[1]] = v] \\
& \qquad \land\ localNum' = [j \in Procs \mapsto \\
& \qquad\qquad\qquad\qquad [i \in OtherProcs(j) \mapsto \\
& \qquad\qquad\qquad\qquad\quad \text{IF } i = self[1] \text{ THEN } qm \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{ELSE } localNum[j][i]]] \\
& \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"L"}] \\
& \land\ \text{UNCHANGED } localCh
\end{array}
$$

$$L(self) \triangleq \land\ pc[self] = \text{"L"}$$

$$
\begin{array}{l}
\land \forall p \in SubProcsOf(self[1]) : pc[p] = \text{“ch”} \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“cs”}] \\
\land \text{UNCHANGED } \langle number, localNum, localCh \rangle
\end{array}
$$

$$
cs(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“cs”} \\
\land \text{TRUE} \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“P”}] \\
\land \text{UNCHANGED } \langle number, localNum, localCh \rangle
\end{array}
$$

$$
P(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“P”} \\
\land number' = [number \text{ EXCEPT } ![self[1]] = 0] \\
\land localNum' = [j \in Procs \mapsto \\
\qquad\qquad [i \in OtherProcs(j) \mapsto \\
\qquad\qquad\quad \text{IF } i = self[1] \text{ THEN } qm \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE } localNum[j][i]]] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“ncs”}] \\
\land \text{UNCHANGED } localCh
\end{array}
$$

$$
main(self) \triangleq ncs(self) \lor M(self) \lor L(self) \lor cs(self) \lor P(self)
$$

$$
ch(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“ch”} \\
\land pc[\langle self[1] \rangle] = \text{“M”} \\
\land localCh' = [localCh \text{ EXCEPT } ![self[2]][self[1]] = 1] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“test”}] \\
\land \text{UNCHANGED } \langle number, localNum \rangle
\end{array}
$$

$$
test(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“test”} \\
\land pc[\langle self[1] \rangle] = \text{“L”} \\
\land localNum' = [localNum \text{ EXCEPT } ![self[2]][self[1]] = number[self[1]]] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“Lb”}] \\
\land \text{UNCHANGED } \langle number, localCh \rangle
\end{array}
$$

$$
Lb(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“Lb”} \\
\land localCh' = [localCh \text{ EXCEPT } ![self[2]][self[1]] = 0] \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“L2”}] \\
\land \text{UNCHANGED } \langle number, localNum \rangle
\end{array}
$$

$$
L2(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“L2”} \\
\land localCh[self[1]][self[2]] = 0 \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“L3”}] \\
\land \text{UNCHANGED } \langle number, localNum, localCh \rangle
\end{array}
$$

$$
L3(self) \triangleq
\begin{array}{l}
\land pc[self] = \text{“L3”} \\
\land (localNum[self[1]][self[2]] \notin \{0, qm\}) \Rightarrow \\
\quad (\langle number[self[1]], self[1] \rangle \ll \\
\qquad \langle localNum[self[1]][self[2]], self[2] \rangle) \\
\land pc' = [pc \text{ EXCEPT } ![self] = \text{“ch”}]
\end{array}
$$

$\land$ UNCHANGED $\langle number, localNum, localCh \rangle$

$sub(self) \triangleq ch(self) \lor test(self) \lor Lb(self) \lor L2(self) \lor L3(self)$

$$
\begin{aligned}
wr(self) \triangleq\ & \land pc[self] = \text{"wr"} \\
& \land \ \land localNum[self[2]][self[1]] = qm \\
& \quad\ \land pc[\langle self[1] \rangle] \in \{\text{"ncs"}, \text{"M"}\} \\
& \land localNum' = [localNum \ \text{EXCEPT}\ ![self[2]][self[1]] = 0] \\
& \land pc' = [pc \ \text{EXCEPT}\ ![self] = \text{"wr"}] \\
& \land \text{UNCHANGED}\ \langle number, localCh \rangle
\end{aligned}
$$

$wrp(self) \triangleq wr(self)$

$$
\begin{aligned}
Next \triangleq\ & (\exists\, self \in ProcIds : main(self)) \\
& \lor (\exists\, self \in SubProcs : sub(self)) \\
& \lor (\exists\, self \in WrProcs : wrp(self))
\end{aligned}
$$

$$
\begin{aligned}
Spec \triangleq\ & \land Init \land \Box[Next]_{vars} \\
& \land \forall\, self \in ProcIds : \mathrm{WF}_{vars}((pc[self] \neq \text{"ncs"}) \land main(self)) \\
& \land \forall\, self \in SubProcs : \mathrm{WF}_{vars}((pc[self] \neq \text{"L3"}) \land sub(self)) \\
& \land \forall\, self \in WrProcs : \mathrm{WF}_{vars}(wrp(self))
\end{aligned}
$$

END TRANSLATION

In statement $L3$, the await condition is satisfied if process $\langle i, j \rangle$ reads $localNum[self[1]][self[2]]$ equal to qm. This is because that's a possible execution, since the process could "interpret" the qm as 0. For checking safety (namely, mutual exclusion), we want to allow that because it's a possibility that must be taken into account. However, for checking liveness, we don't want to require that the statement must be executed when $localNum[self[1]][self[2]]$ equals qm, since that value could also be interpreted as $localNum[self[1]][self[2]]$ equal to 1, which could prevent the wait condition from being true. So we omit that fairness condition from the formula *Spec* produced by translating the algorithm, and we add weak fairness of the action when $localNum[self[1]][self[2]]$ does not equal qm. This produces the TLA+ specification *FSpec* defined here.

$$
\begin{aligned}
FSpec \triangleq\ & \land Spec \\
& \land \forall\, q \in SubProcs : \mathrm{WF}_{vars}(L3(q) \land (localNum[q[1]][q[2]] \neq qm))
\end{aligned}
$$

$$
\begin{aligned}
TypeOK \triangleq\ & \land number \in [Procs \to Nat] \\
& \land \ \land \text{DOMAIN}\ localNum = Procs \\
& \quad\ \land \forall\, i \in Procs : localNum[i] \in [OtherProcs(i) \to Nat \cup \{qm\}] \\
& \land \ \land \text{DOMAIN}\ localCh = Procs \\
& \quad\ \land \forall\, i \in Procs : localCh[i] \in [OtherProcs(i) \to \{0, 1\}]
\end{aligned}
$$

$MutualExclusion \triangleq \forall\, p, q \in ProcIds : (p \neq q) \Rightarrow (\{pc[p], pc[q]\} \neq \{\text{"cs"}\})$

$StarvationFree \triangleq \forall\, p \in ProcIds : (pc[p] = \text{"M"}) \leadsto (pc[p] = \text{"cs"})$

Checking the invariant in the appendix of the paper.

$inBakery(i, j) \triangleq \lor pc[\langle i, j \rangle] \in \{\text{"Lb"}, \text{"L2"}, \text{"L3"}\}$

$$\begin{array}{l} \lor \land pc[\langle i, j\rangle] = \text{"ch"} \\ \quad \land pc[\langle i\rangle] \in \{\text{"L"}, \text{"cs"}\} \end{array}$$

$$inCS(i) \triangleq pc[\langle i\rangle] = \text{"cs"}$$

In TLA+, we can't write both $inDoorway(i, j, w)$ and $inDoorway(i, j)$, so we change the first to $inDoorwayVal$. Its definition differs from the definition of $inDoorway(i, j, w)$ in the paper to avoid having to add a history variable to remember the value of $localNum[self[1]][j]$ read in statement $M0$. It's a nicer definition, but it would have required more explanation than the definition in the paper.

The definition of $inDoorway(i, j)$ is equivalent to the one in the paper. It is obviously implied by $\exists\, w \in Nat : inDoorwayVal(i, j, w)$, and type correctness implies the opposite implication.

$$inDoorwayVal(i, j, w) \triangleq \begin{array}{l} \land pc[\langle i\rangle] = \text{"L"} \\ \land pc[\langle i, j\rangle] = \text{"test"} \\ \land number[i] > w \end{array}$$

$$inDoorway(i, j) \triangleq \begin{array}{l} \land pc[\langle i\rangle] = \text{"L"} \\ \land pc[\langle i, j\rangle] = \text{"test"} \end{array}$$

$$Outside(i, j) \triangleq \lnot(inDoorway(i, j) \lor inBakery(i, j))$$

$$passed(i, j, LL) \triangleq \textsc{if } LL = \text{"L2"} \textsc{ then } \begin{array}{l} \lor pc[\langle i, j\rangle] = \text{"L3"} \\ \lor \land pc[\langle i, j\rangle] = \text{"ch"} \\ \quad \land pc[\langle i\rangle] \in \{\text{"L"}, \text{"cs"}\} \end{array} \textsc{ else } \begin{array}{l} \land pc[\langle i, j\rangle] = \text{"ch"} \\ \land pc[\langle i\rangle] \in \{\text{"L"}, \text{"cs"}\} \end{array}$$

$$Before(i, j) \triangleq \begin{array}{l} \land inBakery(i, j) \\ \land \lor Outside(j, i) \\ \quad \lor inDoorwayVal(j, i, number[i]) \\ \quad \lor \land inBakery(j, i) \\ \quad\quad \land \langle number[i], i\rangle \ll \langle number[j], j\rangle \\ \quad\quad \land \lnot passed(j, i, \text{"L3"}) \end{array}$$

$$Inv(i, j) \triangleq \begin{array}{l} \land inBakery(i, j) \Rightarrow Before(i, j) \lor Before(j, i) \\ \qquad\qquad\qquad\qquad \lor inDoorway(j, i) \\ \land passed(i, j, \text{"L2"}) \Rightarrow Before(i, j) \lor Before(j, i) \\ \land passed(i, j, \text{"L3"}) \Rightarrow Before(i, j) \end{array}$$

$$I \triangleq \forall\, i \in Procs : \forall\, j \in OtherProcs(i) : Inv(i, j)$$

The following is for testing. Since the spec allows the values of $number[n]$ to get arbitrarily large, there are infinitely many states. The obvious solution to that is to use models with a state constraint that $number[n]$ is at most some value $TestMaxNum$. However, TLC would still not be able to execute the spec because the with statement in action M allows an infinite number of possible values for $number[n]$. To solve that problem, we have the model redefine Nat to a finite set of numbers. The obvious set is $0 \,..\, TestMaxNum$. However, trying that reveals a subtle problem. Running the model produces a bogus counterexample to the $StarvationFree$ property.

This is surprising, since constraints on the state space generally fail to find real counterexamples to a liveness property because the counterexamples require large (possibly infinite) traces that are ruled out by the state constraint. The remaining traces may not satisfy the liveness property, but they are ruled out because they fail to satisfy the algorithm's fairness requirements. In this case, a behavior that didn't satisfy the liveness property *StarvationFree* but shouldn't have satisfied the fairness requirements of the algorithm did satisfy the fairness requirement because of the substitution of a finite set of numbers for *Nat*.

Here's what happened: In the behavior, two nodes kept alternately entering the critical section in a way that kept increasing their values of num until one of those values reached *TestMaxNum*. That one entered its critical section while the other was in its noncritical section, re-entered its noncritical section, and then the two processes kept repeating this dance forever. Meanwhile, a third process's subprocess was trying to execute action M. Every time it tried to execute that action, it saw that another process's number equaled *TestMaxNum*. In a normal execution, it would just set its value of num larger than *TestMaxNum* and eventually enter its critical section. However, it couldn't do that because the substitution of $0 \,..\, TestMaxNum$ for *Nat* meant that it couldn't set num to such a value, so the enter step was disabled. The fairness requirement on the enter action is weak fairness, which requires an action eventually to be taken only if it's continually enabled. Requiring strong fairness of the action would have solved this problem, because the enabled action kept being enabled and strong fairness would rule out a behavior in which that process's enter step never occurred. However, it's important that the algorithm satisfy starvation freedom without assuming strong fairness of any of its steps.

The solution to this problem is to substitute $0 \,..\, (TestMax + 1)$ for *Nat*. The state constraint will allow the enter step to be taken, but will allow no further steps from that state. The process still never enters its critical section, but now the behavior that keeps it from doing so will violate the weak fairness requirements on that process's steps.

$$TestMaxNum \triangleq 4$$
$$TestNat \triangleq 0 \,..\, (TestMaxNum + 1)$$

* Modification History
* Last modified *Tue Nov* 16 18:37:41 *CET* 2021 by *merz*
* Last modified *Thu Jul* 01 12:24:37 *CEST* 2021 by *merz*
* Last modified *Wed Apr* 28 18:06:24 *PDT* 2021 by *lamport*
* Created Sat *Apr* 24 09:45:26 *PDT* 2021 by *lamport*6