be some point in α by which all the processes in P have already reached label M; note that they never thereafter drop back to any point in the code prior to label M. Let α_1 be a suffix of α in which all processes in P are in the final for loop, after label M.

We claim that there is at least one process in P. Specifically, the process with the smallest index among all the contenders is not blocked from reaching label M.

Let i be the largest index of a process in P. We claim that eventually in α_1, any process $j \in Q$ such that $j > i$ has $flag(j)$ set permanently to 0. This is because each time j executes one of the first two for loops, it discovers the presence of a smaller index contender and returns to L. Whenever it does this, it sets $flag(j) := 0$, and once it has done this, it can never progress far enough to set $flag(j) := 1$. So let α_2 be a suffix of α_1 in which all processes in Q with indices $> i$ always have their *flag*s equal to 0.

Now in α_2, there is nothing to stop process i from reaching C: every larger-index process j has $flag(j) = 0$, so i will complete the third for loop successfully. Thus, i enters C, which is a contradiction. □

Theorem 10.23 *BurnsME solves the mutual exclusion problem.*

10.7 The Bakery Algorithm

In this section we present the *Bakery algorithm* for mutual exclusion. It works somewhat the way a bakery does, where customers draw tickets when they enter and are served in the order of their ticket numbers.

The *Bakery* algorithm only uses single-writer/multi-reader shared registers. In fact, it also works using a weaker form of register known as a *safe register*, in which the registers are allowed to provide arbitrary responses to reads that are performed concurrently with writes.

The *Bakery* algorithm guarantees lockout-freedom and a good time bound. It guarantees bounded bypass and also a related condition—it is "FIFO after a wait-free doorway" (to be defined below). An unattractive property of the *Bakery* algorithm is that it uses unbounded size registers.

The code follows. We remark that the code given here can be simplified if we are only interested in the usual sort of registers (and not weaker types of registers such as safe registers). We leave this simplification for an exercise.

***Bakery* algorithm:**

Shared variables:
for every i, $1 \leq i \leq n$:
  $choosing(i) \in \{0, 1\}$, initially 0, writable by i and readable by all $j \neq i$
  $number(i) \in \mathbb{N}$, initially 0, writable by i and readable by all $j \neq i$

Process *i*:

```
** Remainder region **

try_i
choosing(i) := 1
number(i) := 1 + max_{j≠i} number(j)
choosing(i) := 0
for j ≠ i do
    waitfor choosing(j) = 0
    waitfor number(j) = 0 or (number(i), i) < (number(j), j)
crit_i

** Critical region **

exit_i
number(i) := 0
rem_i
```

In the *Bakery* algorithm, the first part of the trying region, until the point where process i sets $choosing(i) := 0$, is designated as the *doorway*. While in the doorway, process i chooses a *number* that is greater than all the numbers that it reads for the other processes. It reads the other processes' *number*s one at a time, in any order, then writes its own *number*. While it is reading and choosing numbers, i makes sure that $choosing(i) = 1$, as a signal to the other processes.

Note that it is possible for two processes to be in the doorway at the same time, which can cause them to choose the same number. To break such ties, processes compare not just their *number*s, but their $(number, index)$ pairs. This comparison is done lexicographically, thus breaking ties in favor of the process with the smaller index.

In the rest of the trying region, the process waits for the other processes to finish choosing and also waits for its $(number, index)$ pair to become the lowest.

To prove correctness, let D denote the doorway (i.e., the set of process states in which the process is in the doorway), and let $T - D$ denote the rest of the trying region. Well-formedness is easy to see. To show the mutual exclusion condition, we use a lemma.

Lemma 10.24 *In any reachable system state of the Bakery algorithm, and for any processes i and j, $i \neq j$, the following is true. If i is in C and j is in $(T - D) \cup C$, then $(number(i), i) < (number(j), j)$.*

We give an operational proof, since it can be extended more easily to the safe register case.

Proof. Fix some point s in an execution in which i is in C and j is in $(T - D) \cup C$. (Formally, s is an occurrence of a system state.) Call the values of $number(i)$ and $number(j)$ at point s the *correct* values of these variables.

Process i must read $choosing(j) = 0$ in its first waitfor loop, prior to entering C. Let π denote this reading event; thus, π precedes s. When π occurs, j is not in the "choosing region" (i.e., the portion of the doorway after setting $choosing(j) := 1$). But since j is in $(T - D) \cup C$ at point s, j must pass through the choosing region at some point. There are two cases to consider.

1. j enters the choosing region after π. Then the correct $number(i)$ is chosen before j starts choosing, ensuring that j sees the correct $number(i)$ when it chooses. Therefore, at point s, we have $number(j) > number(i)$, which suffices.

2. j leaves the choosing region before π. Then whenever i reads j's number in its second waitfor loop, it gets the correct $number(j)$ But since i decides to enter C anyhow, it must be that $(number(i), i) < (number(j), j)$. This again suffices.

□

Lemma 10.25 *The Bakery algorithm satisfies mutual exclusion.*

Proof. Suppose that, in some reachable state, two processes, i and j, are both in C. Then by Lemma 10.24 applied twice, we must have both $(number(i), i) < (number(j), j)$ and $(number(j), j) < (number(i), i)$. This is a contradiction. □

Lemma 10.26 *The Bakery algorithm guarantees progress.*

Proof. The exit region is easy, as usual. For the trying region, we again argue by contradiction. Suppose that progress is not guaranteed. Then eventually a point is reached after which all processes are in T or R, and no new region changes occur. By the code, all of the processes in T eventually complete the doorway and reach $T - D$. Then the process with the lowest $(number, index)$ pair is not blocked from reaching C. □

Lemma 10.27 *The Bakery algorithm guarantees lockout-freedom.*

Proof. Consider a particular process i in T and suppose it never reaches C. Process i eventually completes the doorway and reaches $T - D$. Thereafter, any new process that enters the doorway sees i's latest *number* and so chooses a higher number. Thus, since i doesn't reach C, none of these new processes reach C either, since each is blocked by the test of $number(i)$ in its second wait loop.

But repeated use of Lemma 10.26 implies that there must be continuing progress, including infinitely many *crit* events, which contradicts the fact that all new entrants to the trying region are blocked. □

Theorem 10.28 *The Bakery algorithm solves the mutual exclusion problem and is lockout-free.*

Complexity analysis. An upper bound for the time from when a process i enters the trying region until it enters the critical region is $(n-1)c + O(n^2\ell)$. This is not so easy to show; we just give a brief sketch and leave the details for an exercise.

First, it only takes time $O(n\ell)$ for process i to complete the doorway; we must bound the length of the time interval I that i spends in $T - D$. Let P be the set of other processes already in T at the moment i enters $T - D$. Then only processes in P can enter C before i does, and each of these can only do so once. It follows that the total time within interval I during which some process is in C is at most $(n-1)c$, and that the total time within interval I during which some process is in the doorway is at most $O(n^2\ell)$.

It remains to bound the *residual time* within interval I, that is, the total time within I during which no process is either in C or in the doorway. We bound the residual time by considering the progress of processes in $P \cup \{i\}$. During the residual time, note that none of these processes is ever blocked in its first waitfor loop, since all the *choosing* variables are 0. Moreover, some process in $P \cup \{i\}$ will not be blocked at any step of its second waitfor loop either, and so, within residual time $O(n\ell)$, will enter C. After it finishes, some other process in $P \cup \{i\}$ will not be blocked, and so, within an additional residual time $O(n\ell)$, will enter C, and so on. This continues until i enters C, for a total residual time of $O(n^2\ell)$.

FIFO after a wait-free doorway. The *Bakery* algorithm guarantees a high-level-fairness condition that is somewhat stronger than lockout-freedom. Namely, if process i completes the doorway before j enters T, then j cannot enter C before i does. Note that the algorithm is not actually FIFO based on the time of entry