Justin Milliman - Akshay Kumar Dosapati
CS4710: Model-Driven Software Development, Fall 2022
SPIN Project 1 - Parallel Swap - Report

Description:

This model stands up N number of processes to perform N swaps in an array of size N, where A is all distinct non-negative integers. Each process will perform one swap as long as no other process is actively accessing the same cell(s) within A, and the resulting array A has no duplicate or omitted values from the original array A.

Explanation:

Once the program is initialized, we assign values from [0 - (N - 1)] to array A. Simultaneously, we stand up N processes to perform swaps. These processes wait on a 'start' condition to evaluate to true, where the start condition is activated only once A has been initialized. Once started, each process will actively try to perform one swap, and then terminate. Each process uses its own PID as the initial cell to swap from [cell "i"]. If someone is actively swapping with our cell, wait for them to finish, otherwise attempt to set ourselves up for and perform the swap.

Each process will attempt to perform the following actions to get itself setup for and then perform a swap. If it is blocked by other cells, it will simply wait and keep trying to perform the swap.

Setup steps:

- Check that the cell "i" is not engaged in a swap [ActivePids[i] = false]. Iff it's not, reserve it for us by setting the corresponding reservedCells[i] to be the current processes' PID [reservedCells[i] = _pid].

- If our current value of 'j' is not a good swap, find a better value 'j' within the range [0 - (N - 1)] ('j' is not allowed to equal 'i', as that would be useless).

- Check that the cell "j" is not engaged in a swap [ActivePids[j] = false]. Iff it's not, reserve it for us by setting the corresponding reservedCells[j] to be the current processes' PID [reservedCells[j] = _pid]

Setup Swap Step:

- If we have a valid swap, where we have reserved two cells [i, j], set ourselves as swapping [ActivePids[_pid] = true]. This tells other processes that when they look at the cells ‘i’ and ‘j’, that we are actively swapping them and they should wait to perform a swap with those cells until we have completed our swap.

Swap Steps:

If we are waiting on ourselves to perform a swap, we can ignore all of the setup steps. If for some reason the swap failed, the process will resume the setup steps and try again.

If the setup went well and we successfully told all other processes to wait for us before accessing the cells we want to swap, then we can now perform the swap.

Once the current process has finished a successful swap, we need to release our reserved cells for other processes to use, and set ourselves globally as not swapping, as well as increment the global swap counter. We release the cells by doing the following:

- [ActivePids[_pid] = false]

The ‘reservedCells’ reference to the cells we were accessing will now be freed. This is because of how we use the reservedCells as the index to check the activity status of cells [ActivePids[reservedCells[<someCellValue]] = true/false]. So once the current process has performed a swap, the cells within reservedCells both now point to an inactive process. This means any other process can commandeer the cell and reserve it for a swap.

Finishing Steps:

Once all processes have performed their swap and terminated, we now check the resulting array A for any duplicates or omitted elements. We do this in a brute force fashion, comparing every cell within A to every other cell. If any duplicates are found, we know something went wrong, so we break and fail LTL.

If all went well, the program terminates as expected, as we have successfully performed a set of N asynchronous swaps.

Verification Steps:

LTL:

[] ((! (((np_ == 0)) && ((duplicates == 0)))) || ((sCount == N)))

(np_ == 0) -> All processes terminate elegantly
(duplicates == 0) -> No duplicates exist within the resulting array A
(sCount == N) -> Resulting swaps should be equal to the number of processes

Definition: All processes always terminate as intended, no duplicates exist within A, and the resulting total number of swaps performed is equal to the N number of processes (as each process only performs one swap).

Verification:

When running the model with [spin -run -a Swap.pml] the process never terminates. This tells us that the program can access an infinite number of states (run an infinite number of times) and never violate the above LTL.

Example run (stopped early to get the “full” logging): (Next Page)

```
➜  Project1 - Parallel Swap git:(main) ✗ spin -run -a Swap.pml
ltl SwapLTL: [] ((! (((np_==0)) && ((duplicates==0)))) || ((sCount==10)))
Depth=    2032 States=    1e+06 Transitions= 1.48e+06 Memory=   445.624 t=     1.06 R=   9e+05
Depth=    3862 States=    2e+06 Transitions=    3e+06 Memory=   762.519 t=     2.14 R=   9e+05
Depth=    9616 States=    3e+06 Transitions= 4.62e+06 Memory=  1079.706 t=     3.25 R=   9e+05
error: max search depth too small
Depth=    9999 States=    4e+06 Transitions= 6.93e+06 Memory=  1400.507 t=     4.55 R=   9e+05
Depth=    9999 States=    5e+06 Transitions= 9.41e+06 Memory=  1721.698 t=     5.95 R=   8e+05
Depth=    9999 States=    6e+06 Transitions= 1.18e+07 Memory=  2042.401 t=      7.4 R=   8e+05
Depth=    9999 States=    7e+06 Transitions= 1.41e+07 Memory=  2363.105 t=     8.91 R=   8e+05
Depth=    9999 States=    8e+06 Transitions= 1.59e+07 Memory=  2682.050 t=     10.2 R=   8e+05
^CInterrupted

(Spin Version 6.5.2 -- 9 June 2022)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             + (SwapLTL)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 372 byte, depth reached 9999, errors: 0
  8133120 states, stored
  7984707 states, matched
 16117827 transitions (= stored+matched)
   639068 atomic steps
hash conflicts:    853505 (resolved)

Stats on memory usage (in Megabytes):
 3102.539       equivalent memory usage for states (stored*(State-vector + overhead))
 2600.788       actual memory usage for states (compression: 83.83%)
                state-vector as stored = 307 byte + 28 byte overhead
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
    5.087       memory lost to fragmentation
 2724.237       total actual memory usage



pan: elapsed time 10.4 seconds
pan: rate 784293.15 states/second
➜  Project1 - Parallel Swap git:(main) ✗
```

*** Notice the 784,293 checked states per second. Not really relevant but my computer was on fire because of it*

Extra Credit:

After you verify the safety and liveness properties, revise your Promela code such that each proctype performs the swapping for an unbounded number of times. Note that your code must be non-terminating.

- Are the LTL properties still satisfied? Explain what you experience.
- Should the proctypes be synchronized after each round of swapping? How would you revise the model to make it work? Provide your revised Promela model that works correctly.