CS705 Assignment 2

Q1.) Model-checking LTL properties using SPIN

ltl properties:

```
/* Safety Property: Multiple processes cannot enter the critical section
simultaneously. */
ltl safety { [](ncrit <= 1) }

/* Liveness Property: If a process is waiting, it will eventually enter
the critical section. */
//ltl liveness1 { [](flag[_pid] -> <> (ncrit == 1)) }
ltl liveness1_pid0 { [](flag[0] -> <> (ncrit == 1) && (_pid == 0)) }
ltl liveness1_pid1 { [](flag[1] -> <> (ncrit == 1) && (_pid == 1)) }

/* Liveness Property: Any process not in the critical section will
eventually enter it. */
//ltl liveness2 { [](flag[_pid] == 0 -> <> (ncrit == 1)) }
ltl liveness2_pid0 { [](flag[0] == 0 -> <> (ncrit == 1) && (_pid == 0)) }
ltl liveness2_pid1 { [](flag[1] == 0 -> <> (ncrit == 1) && (_pid == 1)) }
```

— test liveness for process `0` and `1`

Results:

```
admin@debian:~/repo/cs705_assignments/cs705_as2/Q1$ ./pan -a -f
pan: ltl formula safety

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

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

State-vector 36 byte, depth reached 55, errors: 0
       44 states, stored
       27 states, matched
       71 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
```

```
Stats on memory usage (in Megabytes):
    0.003       equivalent memory usage for states (stored*(State-vector +
overhead))
    0.283       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage


unreached in proctype user
        petersons_muetx.pml:19, state 11, "-end-"
        (1 of 11 states)
unreached in claim safety
        _spin_nvr.tmp:8, state 10, "-end-"
        (1 of 10 states)
unreached in claim liveness1_pid0
        _spin_nvr.tmp:18, state 13, "(!((ncrit==1)))"
        _spin_nvr.tmp:22, state 17, "-end-"
        (2 of 17 states)
unreached in claim liveness1_pid1
        _spin_nvr.tmp:32, state 13, "(!((ncrit==1)))"
        _spin_nvr.tmp:36, state 17, "-end-"
        (2 of 17 states)
unreached in claim liveness2_pid0
        _spin_nvr.tmp:46, state 13, "(!((ncrit==1)))"
        _spin_nvr.tmp:50, state 17, "-end-"
        (2 of 17 states)
unreached in claim liveness2_pid1
        _spin_nvr.tmp:60, state 13, "(!((ncrit==1)))"
        _spin_nvr.tmp:64, state 17, "-end-"
        (2 of 17 states)

pan: elapsed time 0 seconds
```

No assertion violations: The model passes without errors, meaning the model satisfies the LTL properties of safety and liveness.

States explored: 44 states were explored, 27 were revisited as duplicates (matched states), meaning the model was analysed for multiple execution paths.

No atomic steps: Your model doesn't include any atomic operations, which is expected for Peterson's algorithm as it's implemented through traditional control structures.

Unreached states in `user` process: The implicit termination state (-end- state) is never reached. This is due to the process loops using goto again indefinitely.

Unreached states in ltl properties:

- `(!((ncrit==1)))` This means the critical section always only one process at a time.

- The implicit termination state (-end- state) is never reached. This is due to the process loops using goto again indefinitely.

Q2.) SMT solvers for hardware verification

Majority voter equation:

```
Y = (!ABC) + (A!BC) + (AB!C) + (ABC)
```

Equation:

```
Y' = AB + BC + AC
```

Result:

```
admin@debian:~/repo/cs705_assignments$ python -u
"/home/admin/repo/cs705_assignments/cs705_as2/Q2/smt_sovler.py"
Equations are equivalent
```

Simplification steps to prove `!ABC + A!BC + AB!C + ABC == BC + AB + AC`:
Majority voter equation:
`(!ABC) + (A!BC) + (AB!C) + (ABC)`

1. distributive law: `ABC + !ABC = BC(!A + A)`
   `BC(!A + A) + A!BC + AB!C`
2. complement law: `(!A + A) = 1`
   `BC + A!BC + AB!C`
3. distributive law: `BC + A!BC = C(A!B + B)`
   `C(A!B + B) + AB!C`
4. absorption law: `A!B +B = A + B`
   `C(A + B) + AB!C`
5. expand: `C(A + B) = AC + BC`
   `AC + BC + AB!C`
6. distributive law: `AC + AB!C = A(B!C + C)`
   `BC + A(B!C + C)`
7. absorption law: `A(B!C + C) = A(B + C)`
   `BC + A(B + C)`
8. expand: `A(B + C) = AB + AC`
   `BC + AB + AC`