Mutual Exclusion

The following SMV program models a (buggy) two-users mutual exclusion algorithm.

```
-----------------------------------------------
---- The User ---------------------------------
-----------------------------------------------

MODULE User(auth)
  -- Parameter "auth" is the authorization to enter the critical section
  -- that the user receives from the arbiter.
VAR status: { NonCritical, Trying, Critical };
  -- Variable "status" codifies the current status of the user
  -- - NonCritical: outside the critical section;
  -- - Trying: trying to enter the critical section;
  -- - Critical: inside the critical section.
ASSIGN init(status) := NonCritical;
  -- Initially, the user is outside the critical section.
  next(status) := case status = NonCritical : { NonCritical, Trying };
    -- The user is currently outside the critical section;
    -- it can non-deterministically switch to status "Trying" and
    -- require to enter the critical section.
    status = Trying :
       case
         next(auth) = 0 : Trying;
         next(auth) = 1 : Critical;
       esac;
    -- The user is trying to enter the critical section;
    -- in the next state it will be inside the critical section
    -- only if it receives the authorization by the arbiter.
    status = Critical : { Critical, NonCritical};
    -- The user is currently inside the critical section;
    -- it can non-deterministically leave the critical section.
  esac;
VAR req: boolean;
  -- Variable "req" is true when the user requires access to the
  -- critical section, namely it is in status "Trying" or "Critical".
ASSIGN
  req := status in { Trying, Critical };

-----------------------------------------------
----  The Arbiter  -----------------------------
-----------------------------------------------
MODULE Arbiter(req0, req1)
  -- Parameters "req#" are the requests of the different users to
```

```
  -- access to the critical section.
VAR auth0: boolean;
  -- Authorization to access the critical section for user 0.
ASSIGN
  init(auth0) := 0;
  -- Initially the user in not authorized.
  next(auth0) := req0 & !auth1;
  -- In the next step, the user is authorized to access to the critical
  -- section only if:
    -- - the user requires access to the critical section; and
    -- - the other user is not already authorized to enter.
VAR auth1: boolean;
  -- Authorization to access the critical section for user 1.
ASSIGN
  init(auth1) := 0; next(auth1) := req1 & !auth0;

-------------------------------------------------
----  The main module  --------------------------
-------------------------------------------------
MODULE main
VAR
  U0: User(Ar.auth0); -- User 0
  U1: User(Ar.auth1); -- User 1
  Ar: Arbiter(U0.req, U1.req); -- The arbiter
```

In the program we have two users `U0` and `U1`, and an arbiter Ar. Each user can be either `NonCritical`, `Trying` or `Critical`:

- from `NonCritical`, it can nondeterministically go to Trying;
- from `Trying`, it can go to `Critical` when authorized by the arbiter;
- from `Critical`, it can nondeterministically go back to `NonCritical`.

The aim of the arbiter is to guarantee that the two users are not in status `Critical` at the same time.

1. Formalize the property: the two users cannot be at the same time in their critical section and show that it is false.
2. Fix the arbiter so that the property becomes true.
3. Formalize the property: if a user tries to enter its critical section, it will eventually succeed and show that it is false.
4. Add a fairness constraint that guarantees that the user does not stay forever in critical session.
5. Check if both properties are now true in extended model and, if not, fix the model.
6. Extend the model to the case of three (or four) users.