Test 1.b: Vending Machine (with abstracted counterexample)

1 Test instructions

This test consists in debugging a given system which does not respect a given safety property because of a bug in the specification. We ask you to discover the bug and describe its cause in an email that you will have to send us. We provide you the following information (in this pdf): (i) a short system description; (ii) the system specification written in LNT code; (iii) a short property description and the property in MCL code; (iv) a counterexample; (v) an abstracted counterexample.

The abstracted counterexample is a version of the counterexample where we highlighted choices in the model that can contribute to the cause of the bug. These choices correspond to states in the model where: (a) there is a transition leading to a correct behaviour (that always satisfies the property) (b) there is a transition leading to an incorrect behaviour (that always violates the property), (c) there is a transition leading to both correct and incorrect behaviour.

You do not need to use any software tools to perform this test, it should be sufficient to exploit the information contained in this document. You will have a maximum amount of time of 30 minutes to understand the specification and perform the debugging. At the end, you must write your results in an email. The email must contains the following fields:

- TEST NAME: the name of the test ('Test 1.b Vending Machine' in this case);
- BUG DESCRIPTION: describe the cause of the bug in about two or three sentences (in English language); if you can point out a precise line in the LNT code that caused the bug, please indicate its number in the email; if you reach the time limit of 30 minutes without finding the bug, write "Bug not found";
- TIME: measure and provide the time (in minutes) you spent to discover the bug; you must start measuring time when you start reading Section 2 below; note that you must not take into account time spent in writing the email; if you reach 30 minutes without finding the bug, you can leave this field empty;

- ABSTRACTED COUNTEREXAMPLE: tell in one sentence if and why the abstracted counterexample helped you in understanding the bug.

You must send us the email at the following address:
gianluca.barbon AT inria DOT fr

2 The Vending Machine: system description

The system provided with this test case is composed of two processes: a *vending machine* and a *customer*. The vending machine contains 4 bottles of water, 4 bottles of soda and 4 bottles of beer. Money is represented by single unit coins. Each drink has a different cost in terms of coins: 1 coin for the water, 2 for the soda and 3 for the beer. The customer has a wallet containing 4 coins, and he continues buying drinks until he finishes his money or the machine is out of stock. We present the LNT code of the specification in Appendix A.

3 The property

The provided property states that if the customer has only 2 coins in his wallet, he should not be able to buy a beer, since it costs 3 coins. Figure 1 shows the property in MCL. The `CUSTOMER_WALLET !2` label checks the content of the customer's wallet, `DRINK_CHOICE !BEER !+1` expresses the beer choice and `PROVIDE_DRINK !BEER` shows the beer distribution by the machine.

```
[   true*.'CUSTOMER_WALLET !2'.
    true*.'DRINK_CHOICE !BEER !+1'.
    true*.'PROVIDE_DRINK !BEER'.
    true*
] false
```

Figure 1: MCL property

The property in Figure 1 is not satisfied by the system. This means that, even if the customer only has 2 coins, he is able to buy a beer, which should not be possible. Figure 2 shows a counterexample generated with the Evaluator tool (from the CADP toolbox). Note that the code provided with this test case is syntactically correct and compiles. The bug arise from the violation of the given property.

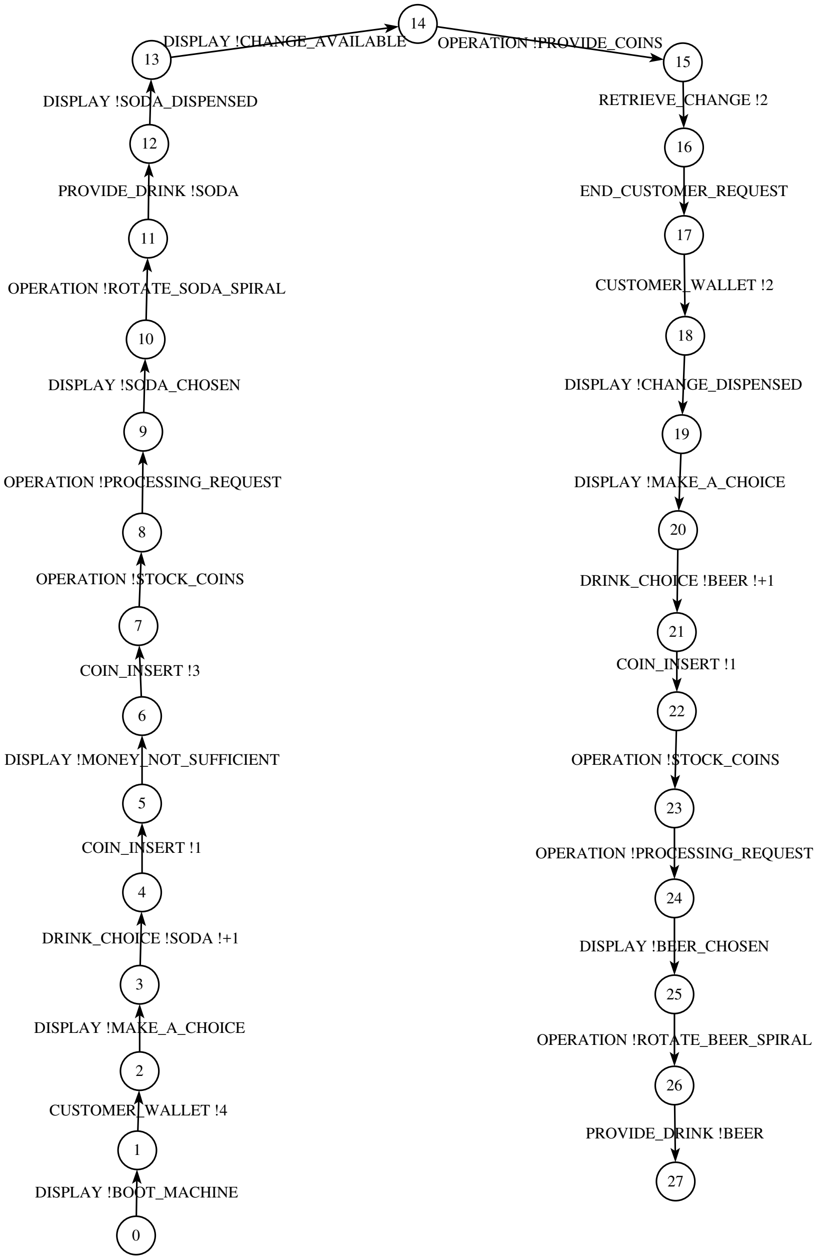

Figure 2: Counterexample.

4 Abstracted Counterexample

Figure 3 shows the abstraction of the counterexample depicted in Figure 2. The corresponding choices are shown on the right-hand side of the Figure.

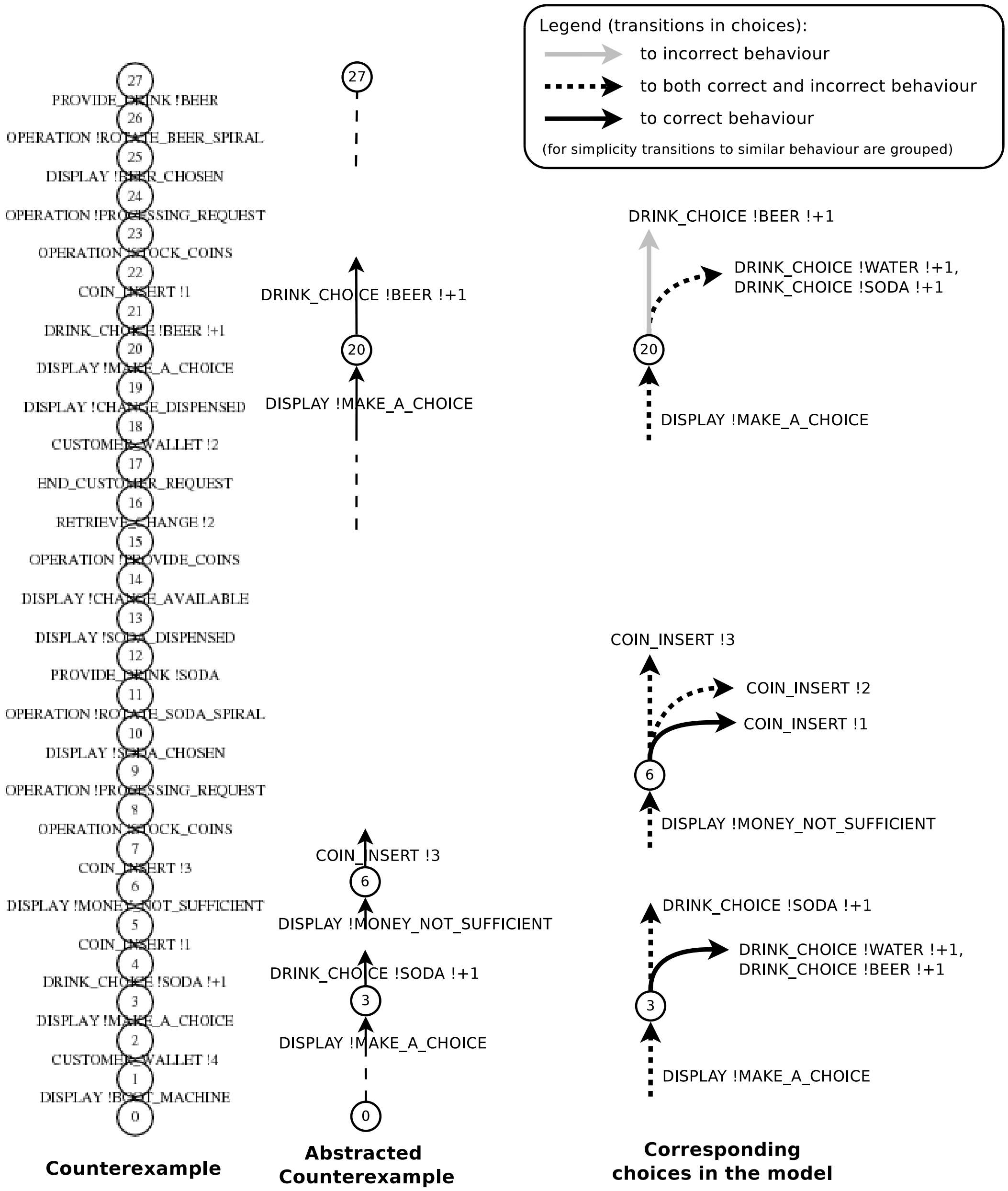

Figure 3: Abstracted counterexample and corresponding choices.

Appendix A LNT code

```

module VENDING_MACHINE is

--------------------------------------------------------------------------------

   (* Type of drinks provided by the machine. *)
   type DRINK_TYPE is
      SODA,
      BEER,
      WATER
   end type

   (* Messages displayed by the machine display. *)
   type DISPLAY_MSG is
      BOOT_MACHINE,
      MAKE_A_CHOICE,
      MONEY_NOT_SUFFICIENT,
      CHANGE_AVAILABLE,
      NO_CHANGE,
      CHANGE_DISPENSED,
      SODA_CHOSEN,
      BEER_CHOSEN,
      WATER_CHOSEN,
      SODA_DISPENSED,
      BEER_DISPENSED,
      WATER_DISPENSED,
      SODA_OUT_OF_STOCK,
      BEER_OUT_OF_STOCK,
      WATER_OUT_OF_STOCK,
      ALL_DRINKS_OUT_OF_STOCK,
      MACHINE_OUT_OF_ORDER
   end type

   (* Internal operations performed by the machine. *)
   type INTERNAL_OP is
      STOCK_COINS,
      PROVIDE_COINS,
      ROTATE_SODA_SPIRAL,
      ROTATE_BEER_SPIRAL,
      ROTATE_WATER_SPIRAL,
      PROCESSING_REQUEST
   end type

   (* Drink choice channel (Customer->Machine).
      It allows to express the drink type and its quantity. *)
   channel DRINK_CHOICE_C is
      (DRINK_TYPE, int)
   end channel

   (* Coins channel (Customer<->Machine).
      It used by the customer to pay and by the machine to provide change. *)
   channel COINS_C is
      (nat)
   end channel

   (* Machine display channel. *)
   channel DISPLAY_C is
      (DISPLAY_MSG)
   end channel

   (* Machine internal operation channel. *)
   channel INTERNAL_OP_C is
      (INTERNAL_OP)
   end channel

```

```
    (* Machine dispenser channel (Machine->Customer). *)
    channel DISPENSER_C is
       (DRINK_TYPE)
    end channel

    (* User wallet money channel. *)
    channel CUSTOMER_WALLET_C is
       (nat)
    end channel
--------------------------------------------------------------------------------

    (* It checks if the current drink stock is sufficient for the given quantity.
       DRINK_STOCK : the current drink stock;
       REQ_QUANT : the required quantity.  *)
    function CHECK_QUANT(DRINK_STOCK, REQ_QUANT : int) : bool is
       if ((DRINK_STOCK - REQ_QUANT) < 0) then
          return false
       else
          return true
       end if
    end function

    (* It retrieves the money required for a given drink. *)
    function DRINK_VALUE(DRINK : DRINK_TYPE) : nat is
       case DRINK of DRINK_TYPE in
          WATER -> return 1
          | SODA -> return 2
          | BEER -> return 3
       end case
    end function

    (* It checks if the given amount of money is sufficient for the given drink.
       DRINK : the given drink;
       COIN_QUANT : the given amount of money. *)
    function CHECK_MONEY(DRINK : DRINK_TYPE, COIN_QUANT : nat) : bool is
       if (COIN_QUANT < DRINK_VALUE(DRINK)) then
          return false
       else
          return true
       end if
    end function

--------------------------------------------------------------------------------

    (*
     * Customer process: it starts with the amount of money contained in its
     * wallet. The process can choose between three different kind of drinks:
     * water, soda and beer. After the buying step, the process requires change.
     * If the process still has money, it will continue buying drinks from the
     * machine, until he finish his money or the machine is out of stock.
     *)
    process CUSTOMER  [  COIN_INSERT,RETRIEVE_CHANGE : COINS_C,
                         DRINK_CHOICE : DRINK_CHOICE_C,
                         PROVIDE_DRINK : DISPENSER_C,
                         CUSTOMER_OUT_OF_MONEY, END_CUSTOMER_REQUEST : NONE,
                         CUSTOMER_WALLET : CUSTOMER_WALLET_C]
                      (wallet : nat)  -- The amount of money owned by the customer.
                      is
       var   coins, curr_wallet, change : nat,
             collected_drink : DRINK_TYPE in
          -- The customer will continue until he finish his money
          if (wallet > 0) then
             CUSTOMER_WALLET(wallet);
             curr_wallet := wallet;
             change := 0;
             coins := 1;
             select
                COIN_INSERT(!coins);
```

```
            select
               DRINK_CHOICE(!WATER, !1 of int)
            []
               DRINK_CHOICE(!SODA, !1 of int)
            []
               DRINK_CHOICE(!BEER, !1 of int)
            end select
         []
            select
               DRINK_CHOICE(!WATER, !1 of int)
            []
               DRINK_CHOICE(!SODA, !1 of int)
            []
               DRINK_CHOICE(!BEER, !1 of int)
            end select;
            COIN_INSERT(!coins)
         end select;
         curr_wallet := curr_wallet - coins;
         loop L in  -- loop if more money is needed
            select
               PROVIDE_DRINK(?collected_drink);
               break L
            []
               coins := any nat
                        where ((coins <= curr_wallet) and (coins > 0));
               COIN_INSERT(!coins);
               curr_wallet := curr_wallet - coins
            end select
         end loop;
         RETRIEVE_CHANGE(?change);
         curr_wallet := curr_wallet + change;
         END_CUSTOMER_REQUEST;
         CUSTOMER[COIN_INSERT,RETRIEVE_CHANGE, DRINK_CHOICE,
                  PROVIDE_DRINK, CUSTOMER_OUT_OF_MONEY,
                  END_CUSTOMER_REQUEST, CUSTOMER_WALLET](curr_wallet)
      else
         CUSTOMER_OUT_OF_MONEY
      end if
   end var
end process

(*
 * Vending machine: it waits for the user choice and/or money. If money if
 * inserted first , it then asks to choose the drink. If the drink is chosen
 * first , it then asks to provide money. If money is not sufficient , it
 * will continue to ask money. When the inserted amount of money it
 * sufficient , it check if the drink is available. If it is the case,
 * the drink is provided and the cost of the drink is decreased from the
 * inserted money and the drink quantity is updated. Otherwise, the money
 * remains available in the machine.
 * At the end, if change is available in the machine, it can be
 * returned to the customer.
 *)
process MACHINE   [ DISPLAY : DISPLAY_C,
                    PROVIDE_DRINK : DISPENSER_C,
                    DRINK_CHOICE : DRINK_CHOICE_C,
                    COIN_INSERT, RETRIEVE_CHANGE : COINS_C,
                    OPERATION : INTERNAL_OP_C]
                  -- Remaining quantities for each drink.
                  (water_stock, soda_stock, beer_stock : int,
                  rem_change : nat) -- Remaining money in the machine.
                  is
   var   quant, water_q, soda_q, beer_q : int, -- Required quantities and stock.
         inserted_coins, total_coins : nat,
         drink : DRINK_TYPE in
      inserted_coins := 0;
      total_coins := rem_change;
      water_q := water_stock;
```

```
soda_q := soda_stock;
beer_q := beer_stock;
select
   COIN_INSERT(?inserted_coins);
   total_coins := inserted_coins + total_coins;
   DISPLAY(MAKE_A_CHOICE);
   DRINK_CHOICE(?drink, ?quant);
   loop L in
      if CHECK_MONEY(drink,total_coins) then
         OPERATION(STOCK_COINS);
         break L
      else
         DISPLAY(MONEY_NOT_SUFFICIENT)
      end if;
      COIN_INSERT(?inserted_coins);
      total_coins := inserted_coins + total_coins
   end loop
[]
   DISPLAY(MAKE_A_CHOICE);
   DRINK_CHOICE(?drink, ?quant);
   loop L in
      COIN_INSERT(?inserted_coins);
      total_coins := inserted_coins + total_coins;
      if CHECK_MONEY(drink,total_coins) then
         OPERATION(STOCK_COINS);
         break L
      else
         DISPLAY(MONEY_NOT_SUFFICIENT)
      end if
   end loop
end select;
-- The inserted money is sufficient for the chosen drink.
-- The vending machine will now process the request.
OPERATION(PROCESSING_REQUEST);
case drink of DRINK_TYPE in
   WATER ->
      DISPLAY(WATER_CHOSEN);
      if CHECK_QUANT(water_q, quant) then
         OPERATION(ROTATE_WATER_SPIRAL);
         PROVIDE_DRINK(!WATER);
         water_q := water_q - quant;
         total_coins := total_coins - DRINK_VALUE(drink);
         DISPLAY(WATER_DISPENSED)
      else
         DISPLAY(WATER_OUT_OF_STOCK)
      end if
   | SODA ->
      DISPLAY(SODA_CHOSEN);
      if CHECK_QUANT(soda_q, quant) then
         OPERATION(ROTATE_SODA_SPIRAL);
         PROVIDE_DRINK(!SODA);
         soda_q := soda_q - quant;
         total_coins := total_coins - DRINK_VALUE(drink);
         DISPLAY(SODA_DISPENSED)
      else
         DISPLAY(SODA_OUT_OF_STOCK)
      end if
   | BEER ->
      DISPLAY(BEER_CHOSEN);
      if CHECK_QUANT(beer_q, quant) then
         OPERATION(ROTATE_BEER_SPIRAL);
         PROVIDE_DRINK(!BEER);
         beer_q := beer_q - quant;
         total_coins := total_coins - DRINK_VALUE(drink);
         DISPLAY(BEER_DISPENSED)
      else
         DISPLAY(BEER_OUT_OF_STOCK)
      end if
```

```
            end case;
            -- The customer ask for change.
            -- If there is some, the machine will provide it.
            if (total_coins == 0) then
                DISPLAY(NO_CHANGE);
                RETRIEVE_CHANGE(!total_coins)
            else
                DISPLAY(CHANGE_AVAILABLE);
                OPERATION(PROVIDE_COINS);
                RETRIEVE_CHANGE(!total_coins);
                DISPLAY(CHANGE_DISPENSED)
            end if;
            -- The vending machine can work until it finishes all the drink stocks.
            if ((water_q!=0) or (soda_q!=0) or (beer_q!=0))  then
                MACHINE[ DISPLAY, PROVIDE_DRINK, DRINK_CHOICE, COIN_INSERT,
                         RETRIEVE_CHANGE, OPERATION]
                       (water_q, soda_q, beer_q, total_coins)
            else
                DISPLAY(ALL_DRINKS_OUT_OF_STOCK);
                DISPLAY(MACHINE_OUT_OF_ORDER)
            end if
        end var
    end process

    (*
     * Parallelization of the Vending Machine and the Customer.
     * The two processes will interact until the end of the stock in the machine
     * or the end of customer money.
     *)
    process MAIN[   DISPLAY : DISPLAY_C,
                    PROVIDE_DRINK : DISPENSER_C,
                    DRINK_CHOICE : DRINK_CHOICE_C,
                    COIN_INSERT, RETRIEVE_CHANGE : COINS_C,
                    OPERATION : INTERNAL_OP_C,
                    CUSTOMER_OUT_OF_MONEY, END_CUSTOMER_REQUEST : NONE,
                    CUSTOMER_WALLET : CUSTOMER_WALLET_C] is
        par COIN_INSERT, DRINK_CHOICE, PROVIDE_DRINK, RETRIEVE_CHANGE in
            DISPLAY(BOOT_MACHINE);
            MACHINE[ DISPLAY, PROVIDE_DRINK, DRINK_CHOICE, COIN_INSERT,
                     RETRIEVE_CHANGE, OPERATION] (4, 4, 4, 0)
        ||
            CUSTOMER[COIN_INSERT, RETRIEVE_CHANGE, DRINK_CHOICE, PROVIDE_DRINK,
                     CUSTOMER_OUT_OF_MONEY, END_CUSTOMER_REQUEST, CUSTOMER_WALLET] (4)
        end par
    end process

--------------------------------------------------------------------------------

end module
```