

# 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 have 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.

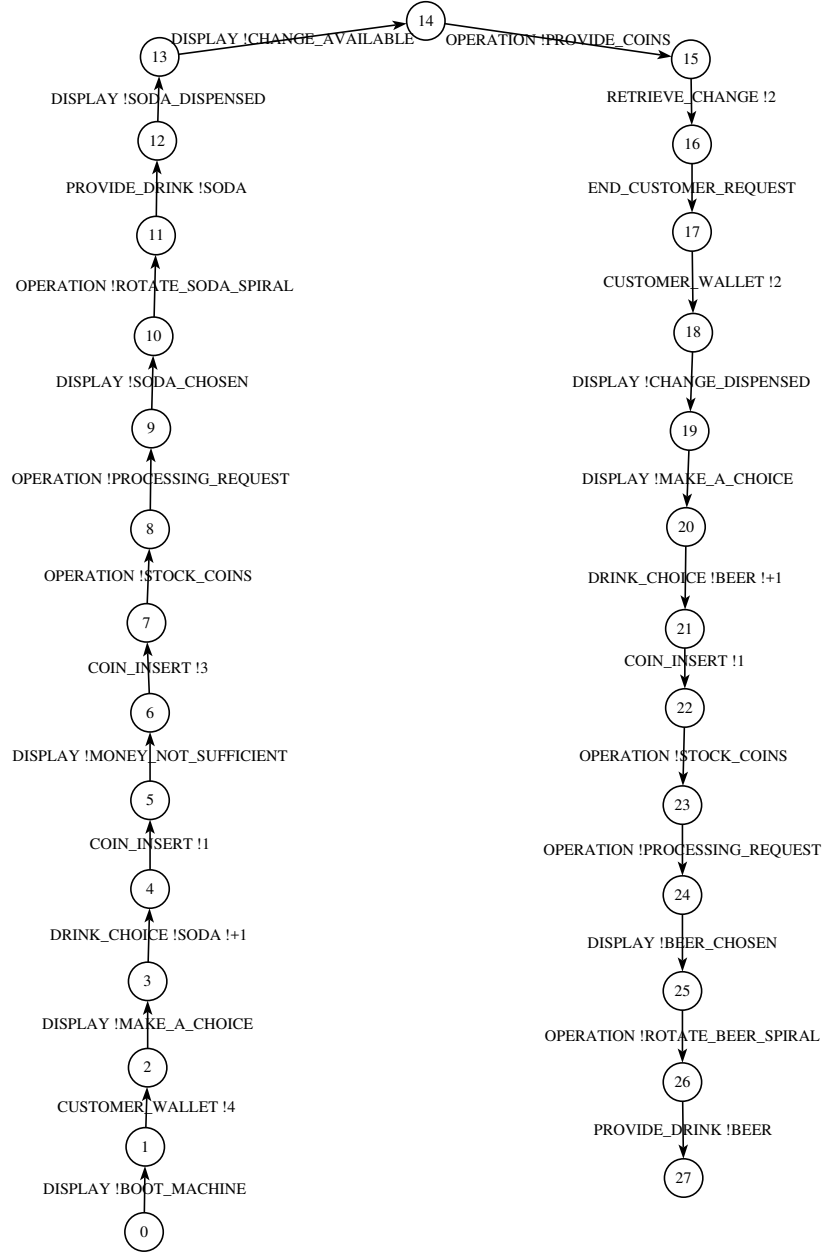


Figure 2: Counterexample.

## 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.

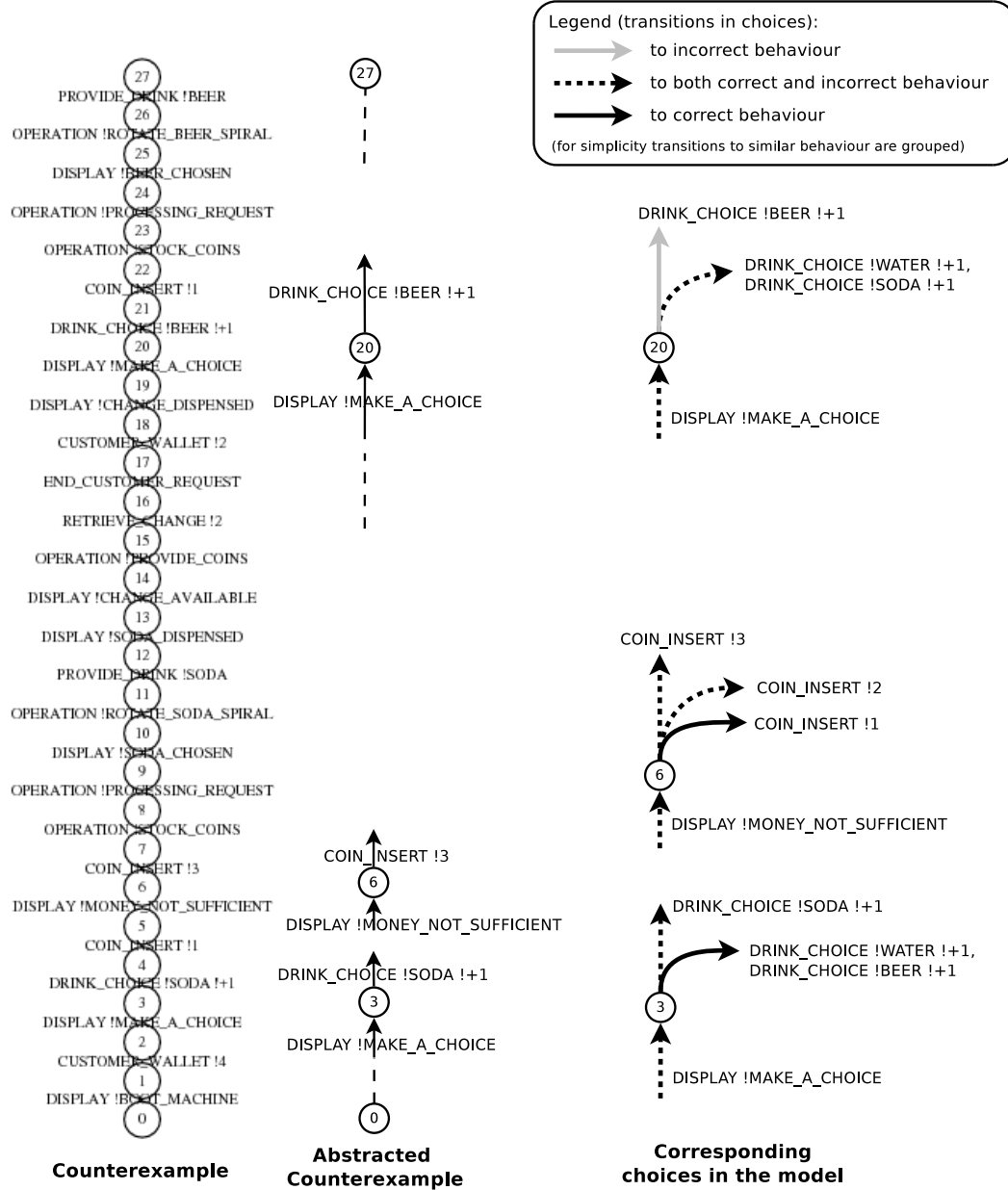


Figure 3: Abstracted counterexample and corresponding choices.

## Appendix A LNT code

```
1
2 module VENDING_MACHINE is
3
4 -----
5
6   (* Type of drinks provided by the machine. *)
7   type DRINK_TYPE is
8     SODA,
9     BEER,
10    WATER
11  end type
12
13   (* Messages displayed by the machine display. *)
14   type DISPLAY_MSG is
15     BOOT.MACHINE,
16     MAKE_A_CHOICE,
17     MONEY_NOT_SUFFICIENT,
18     CHANGE_AVAILABLE,
19     NO_CHANGE,
20     CHANGE_DISPENSED,
21     SODA_CHOSEN,
22     BEER_CHOSEN,
23     WATER_CHOSEN,
24     SODA_DISPENSED,
25     BEER_DISPENSED,
26     WATER_DISPENSED,
27     SODA_OUT_OF_STOCK,
28     BEER_OUT_OF_STOCK,
29     WATER_OUT_OF_STOCK,
30     ALL_DRINKS_OUT_OF_STOCK,
31     MACHINE_OUT_OF_ORDER
32  end type
33
34   (* Internal operations performed by the machine. *)
35   type INTERNAL_OP is
36     STOCK_COINS,
37     PROVIDE_COINS,
38     ROTATE_SODA_SPIRAL,
39     ROTATE_BEER_SPIRAL,
40     ROTATE_WATER_SPIRAL,
41     PROCESSING_REQUEST
42  end type
43
44   (* Drink choice channel (Customer->Machine).
45      It allows to express the drink type and its quantity. *)
46   channel DRINK_CHOICE_C is
47     (DRINK_TYPE, int)
48  end channel
49
50   (* Coins channel (Customer<->Machine).
51      It used by the customer to pay and by the machine to provide change. *)
52   channel COINS_C is
53     (nat)
54  end channel
55
56   (* Machine display channel. *)
57   channel DISPLAY_C is
58     (DISPLAY_MSG)
59  end channel
60
61   (* Machine internal operation channel. *)
62   channel INTERNAL_OP_C is
63     (INTERNAL_OP)
64  end channel
65
```

```

66      (* Machine dispenser channel (Machine→Customer). *)
67      channel DISPENSER_C is
68          (DRINK_TYPE)
69      end channel
70
71      (* User wallet money channel. *)
72      channel CUSTOMER_WALLET_C is
73          (nat)
74      end channel
75
76
77      (* It checks if the current drink stock is sufficient for the given quantity.
78         DRINK_STOCK : the current drink stock;
79         REQ_QUANT : the required quantity. *)
80      function CHECK_QUANT(DRINK_STOCK, REQ_QUANT : int) : bool is
81          if ((DRINK_STOCK - REQ_QUANT) < 0) then
82              return false
83          else
84              return true
85          end if
86      end function
87
88      (* It retrieves the money required for a given drink. *)
89      function DRINK_VALUE(DRINK : DRINK_TYPE) : nat is
90          case DRINK of DRINK_TYPE in
91              WATER → return 1
92              | SODA → return 2
93              | BEER → return 3
94          end case
95      end function
96
97      (* It checks if the given amount of money is sufficient for the given drink.
98         DRINK : the given drink;
99         COIN_QUANT : the given amount of money. *)
100     function CHECK_MONEY(DRINK : DRINK_TYPE, COIN_QUANT : nat) : bool is
101         if (COIN_QUANT < DRINK_VALUE(DRINK)) then
102             return false
103         else
104             return true
105         end if
106     end function
107
108
109
110     (*
111     * Customer process: it starts with the amount of money contained in its
112     * wallet. The process can choose between three different kind of drinks:
113     * water, soda and beer. After the buying step, the process requires change.
114     * If the process still has money, it will continue buying drinks from the
115     * machine, until he finish his money or the machine is out of stock.
116     *)
117     process CUSTOMER [ COIN_INSERT, RETRIEVE_CHANGE : COINS_C,
118                       DRINK_CHOICE : DRINK_CHOICE_C,
119                       PROVIDE_DRINK : DISPENSER_C,
120                       CUSTOMER_OUT_OF_MONEY, END_CUSTOMER_REQUEST : NONE,
121                       CUSTOMER_WALLET : CUSTOMER_WALLET_C]
122         (wallet : nat) — The amount of money owned by the customer.
123         is
124         var coins, curr_wallet, change : nat,
125            collected_drink : DRINK_TYPE in
126             — The customer will continue until he finish his money
127             if (wallet > 0) then
128                 CUSTOMER_WALLET(wallet);
129                 curr_wallet := wallet;
130                 change := 0;
131                 coins := 1;
132                 select
133                     COIN_INSERT(!coins);

```

```

134         select
135             DRINK_CHOICE(!WATER, !1 of int)
136         []
137         DRINK_CHOICE(!SODA, !1 of int)
138         []
139         DRINK_CHOICE(!BEER, !1 of int)
140     end select
141 []
142     select
143         DRINK_CHOICE(!WATER, !1 of int)
144     []
145         DRINK_CHOICE(!SODA, !1 of int)
146     []
147         DRINK_CHOICE(!BEER, !1 of int)
148     end select;
149     COIN_INSERT(!coins)
150 end select;
151 curr_wallet := curr_wallet - coins;
152 loop L in -- loop if more money is needed
153     select
154         PROVIDE_DRINK(?collected_drink);
155         break L
156     []
157         coins := any nat
158             where ((coins <= curr_wallet) and (coins > 0));
159         COIN_INSERT(!coins);
160         curr_wallet := curr_wallet - coins
161     end select
162 end loop;
163 RETRIEVE_CHANGE(?change);
164 curr_wallet := curr_wallet + change;
165 END_CUSTOMER_REQUEST;
166 CUSTOMER[COIN_INSERT, RETRIEVE_CHANGE, DRINK_CHOICE,
167     PROVIDE_DRINK, CUSTOMER_OUT_OF_MONEY,
168     END_CUSTOMER_REQUEST, CUSTOMER_WALLET](curr_wallet)
169 else
170     CUSTOMER_OUT_OF_MONEY
171 end if
172 end var
173 end process
174
175 (*
176  * Vending machine: it waits for the user choice and/or money. If money if
177  * inserted first, it then asks to choose the drink. If the drink is chosen
178  * first, it then asks to provide money. If money is not sufficient, it
179  * will continue to ask money. When the inserted amount of money it
180  * sufficient, it check if the drink is available. If it is the case,
181  * the drink is provided and the cost of the drink is decreased from the
182  * inserted money and the drink quantity is updated. Otherwise, the money
183  * remains available in the machine.
184  * At the end, if change is available in the machine, it can be
185  * returned to the customer.
186  *)
187 process MACHINE [ DISPLAY : DISPLAY_C,
188     PROVIDE_DRINK : DISPENSER_C,
189     DRINK_CHOICE : DRINK_CHOICE_C,
190     COIN_INSERT, RETRIEVE_CHANGE : COINS_C,
191     OPERATION : INTERNAL_OP_C]
192     -- Remaining quantities for each drink.
193     (water_stock, soda_stock, beer_stock : int,
194     rem_change : nat) -- Remaining money in the machine.
195     is
196 var    quant, water_q, soda_q, beer_q : int, -- Required quantities and stock.
197     inserted_coins, total_coins : nat,
198     drink : DRINK_TYPE in
199     inserted_coins := 0;
200     total_coins := rem_change;
201     water_q := water_stock;

```

```

202     soda_q := soda_stock;
203     beer_q := beer_stock;
204     select
205         COIN_INSERT(?inserted_coins);
206         total_coins := inserted_coins + total_coins;
207         DISPLAY(MAKE_A_CHOICE);
208         DRINK_CHOICE(?drink, ?quant);
209         loop L in
210             if CHECK_MONEY(drink, total_coins) then
211                 OPERATION(STOCK_COINS);
212                 break L
213             else
214                 DISPLAY(MONEY_NOT_SUFFICIENT)
215             end if;
216             COIN_INSERT(?inserted_coins);
217             total_coins := inserted_coins + total_coins
218         end loop
219     []
220     DISPLAY(MAKE_A_CHOICE);
221     DRINK_CHOICE(?drink, ?quant);
222     loop L in
223         COIN_INSERT(?inserted_coins);
224         total_coins := inserted_coins + total_coins;
225         if CHECK_MONEY(drink, total_coins) then
226             OPERATION(STOCK_COINS);
227             break L
228         else
229             DISPLAY(MONEY_NOT_SUFFICIENT)
230         end if
231     end loop
232 end select;
233 — The inserted money is sufficient for the chosen drink.
234 — The vending machine will now process the request.
235 OPERATION(PROCESSING_REQUEST);
236 case drink of DRINK_TYPE in
237     WATER ->
238         DISPLAY(WATER_CHOSEN);
239         if CHECK_QUANT(water_q, quant) then
240             OPERATION(ROTATE_WATER_SPIRAL);
241             PROVIDE_DRINK(!WATER);
242             water_q := water_q - quant;
243             total_coins := total_coins - DRINK_VALUE(drink);
244             DISPLAY(WATER_DISPENSED)
245         else
246             DISPLAY(WATER_OUT_OF_STOCK)
247         end if
248     | SODA ->
249         DISPLAY(SODA_CHOSEN);
250         if CHECK_QUANT(soda_q, quant) then
251             OPERATION(ROTATE_SODA_SPIRAL);
252             PROVIDE_DRINK(!SODA);
253             soda_q := soda_q - quant;
254             total_coins := total_coins - DRINK_VALUE(drink);
255             DISPLAY(SODA_DISPENSED)
256         else
257             DISPLAY(SODA_OUT_OF_STOCK)
258         end if
259     | BEER ->
260         DISPLAY(BEER_CHOSEN);
261         if CHECK_QUANT(beer_q, quant) then
262             OPERATION(ROTATE_BEER_SPIRAL);
263             PROVIDE_DRINK(!BEER);
264             beer_q := beer_q - quant;
265             total_coins := total_coins - DRINK_VALUE(drink);
266             DISPLAY(BEER_DISPENSED)
267         else
268             DISPLAY(BEER_OUT_OF_STOCK)
269         end if

```



```

270     end case;
271     -- The customer ask for change.
272     -- If there is some, the machine will provide it.
273     if (total_coins == 0) then
274         DISPLAY(NO.CHANGE);
275         RETRIEVE.CHANGE(!total_coins)
276     else
277         DISPLAY(CHANGE.AVAILABLE);
278         OPERATION(PROVIDE.COINS);
279         RETRIEVE.CHANGE(!total_coins);
280         DISPLAY(CHANGE.DISPENSED)
281     end if;
282     -- The vending machine can work until it finishes all the drink stocks.
283     if ((water_q!=0) or (soda_q!=0) or (beer_q!=0)) then
284         MACHINE[ DISPLAY, PROVIDE.DRINK, DRINK.CHOICE, COIN.INSERT,
285             RETRIEVE.CHANGE, OPERATION]
286             (water_q, soda_q, beer_q, total_coins)
287     else
288         DISPLAY(ALL.DRINKS.OUT.OF.STOCK);
289         DISPLAY(MACHINE.OUT.OF.ORDER)
290     end if
291 end var
292 end process
293
294 (*
295  * Parallelization of the Vending Machine and the Customer.
296  * The two processes will interact until the end of the stock in the machine
297  * or the end of customer money.
298  *)
299 process MAIN[ DISPLAY : DISPLAY_C,
300     PROVIDE.DRINK : DISPENSER_C,
301     DRINK.CHOICE : DRINK.CHOICE_C,
302     COIN.INSERT, RETRIEVE.CHANGE : COINS_C,
303     OPERATION : INTERNAL.OP_C,
304     CUSTOMER.OUT.OF.MONEY, END.CUSTOMER.REQUEST : NONE,
305     CUSTOMER.WALLET : CUSTOMER.WALLET_C] is
306     par COIN.INSERT, DRINK.CHOICE, PROVIDE.DRINK, RETRIEVE.CHANGE in
307         DISPLAY(BOOT.MACHINE);
308         MACHINE[ DISPLAY, PROVIDE.DRINK, DRINK.CHOICE, COIN.INSERT,
309             RETRIEVE.CHANGE, OPERATION] (4, 4, 4, 0)
310     ||
311         CUSTOMER[COIN.INSERT, RETRIEVE.CHANGE, DRINK.CHOICE, PROVIDE.DRINK,
312             CUSTOMER.OUT.OF.MONEY, END.CUSTOMER.REQUEST, CUSTOMER.WALLET] (4)
313     end par
314 end process
315
316 -----
317
318 end module

```