Verification Assignment

Filippo Nevi

VR458510

System Verification & Testing

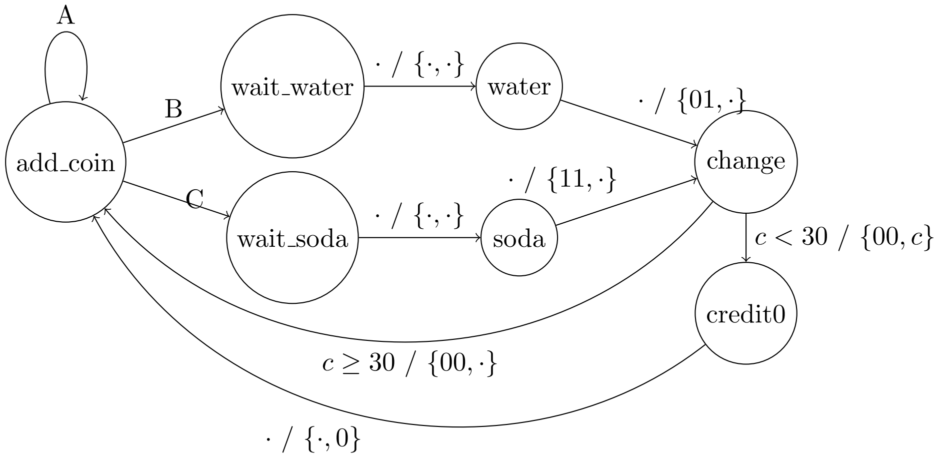
Computer Engineering for Robotics and Smart Industry
University of Verona
2021/2022

1 RTL Design

1.1 Finite State Machine

The output is represented as the set: {beverage_out,change_out}, where the symbol $\cdot$ represents an unchanged signal.

In order to keep the graphical representation as clean as possible, I've decided to label with letters the transition that have long descriptions and explain them in a bulleted list.


A
B
C
add_coin
wait_water
wait_soda
$\cdot$ / $\{\cdot, \cdot\}$
$\cdot$ / $\{\cdot, \cdot\}$
water
soda
$\cdot$ / $\{01, \cdot\}$
$\cdot$ / $\{11, \cdot\}$
change
$c < 30$ / $\{00, c\}$
credit0
$c \geq 30$ / $\{00, \cdot\}$
$\cdot$ / $\{\cdot, 0\}$

Where:

- **A**: coin_in $\in \{10c, 20c, 50c, 1\text{€}, 2\text{€}\}$ / $\{\cdot, \cdot\}$
- **B**: $c \geq 30$ && button_in == 01 / $\{\cdot, \cdot\}$
- **C**: $c \geq 50$ && button_in == 11 / $\{\cdot, \cdot\}$

1.2 Implementation choices

- `wait_water` and `wait_soda` implement the delay between the selection and the delivery of a beverage. Without them, the machine would be forced to stay in one of the other states: for example, if the delay was computed in `add_coin`, the FSM would accept coins while the delay is passing; if it was computed in `water` or `soda`, the signal `beverage_out` would be high for too long
- `credit0` is necessary because `change_out <= c` and `c <= 0` can't be computed in the same clock cycle: the change would be set to zero every time
- If there is no change after the delivery of a beverage, the state machine accepts an input (a new coin or a pressed button) without the delay `M`

2 Verilog-style Test Bench

After initializing the machine and adding some coins, the test bench inputs an illegal button code but it gets ignored. Then, it starts acting like a normal user until the final part, in which it inputs a coin while the vending machine is returning the change. Just like the other occurrence, the machine manages to ignore the wrong timing of the input.

3 Constrained Random Test Bench.

3.1 Transaction constraints

- `coin_in` has to be in the set of admitted coins: $\{10c, 20c, 50c, 1\text{€}, 2\text{€}\}$
- `button_in` = 01 or 11 implies `coin_in` = 0: this allows the FSM's state to evolve correctly
- `button_in` is set to be *no choice* (`00`), *water* (`01`) or *soda* (`11`)

3.2 Asserted properties

1. `G((state = water)->(nexttime[1] beverage_out = 01))`
2. `G((state = soda)->(nexttime[1] beverage_out = 11))`
3. `G((state = add_coin && c < 30)->(nexttime[1] state = add_coin))`
4. `G((state = credit0)->(change_out > 0 && change_out < 30))`
5. `G((c ≥ 30 && state = change)->(nexttime[1] change_out = 0))`

3.3 Contingency tables

The contingency tables were computed performing the negation of the antecedent and the consequent using De Morgan's laws. To implement them in the code, there are four instances of each property. For example, regarding the first one:

- `p0` : ATCT
- `p0N_` : AFCT (N indicates the antecedent is negated, _ as the second character indicates the consequent is untouched)
- `p0_N` : ATCF
- `p0NN` : AFCF

The results are divided in two cases: the first one has the standard constraints for the input values. The second one uses a *subset* of allowed coins in which 1 € and 2 € coins were removed. This choice was taken to allow a higher number of cases in which some change has to be given. With them included in the set, the credit in the machine would be always higher than 30 cents and the ATCT of `p3` is zero.

Additionally, only property `p3` can be asserted 1000 times: the other properties get covered 999 times because they contain a reference to the following time instant, which doesn't exist in the last clock period of the simulation. Finally, property `p4` has a very high number of AFCTs because when there is no output change, its signal is set to zero.

3.3.1 Contingency tables with the whole set of coins

	CT	**CF**
AT	59	0
AF	0	940

Table 1: `p0`

	CT	**CF**
AT	60	0
AF	0	939

Table 2: `p1`

	CT	**CF**
AT	3	0
AF	520	476

Table 3: `p2`

	CT	**CF**
AT	0	0
AF	0	1000

Table 4: `p3` (Note: ATCT = 0)

	CT	**CF**
AT	119	0
AF	880	0

Table 5: `p4`

3.3.2 Contingency tables without 1 and 2 € coins

	CT	**CF**
AT	68	0
AF	0	931

Table 6: `p0`

	CT	**CF**
AT	68	0
AF	0	931

Table 7: `p1`

	CT	**CF**
AT	22	0
AF	415	562

Table 8: `p2`

	CT	**CF**
AT	8	0
AF	0	992

Table 9: `p3`

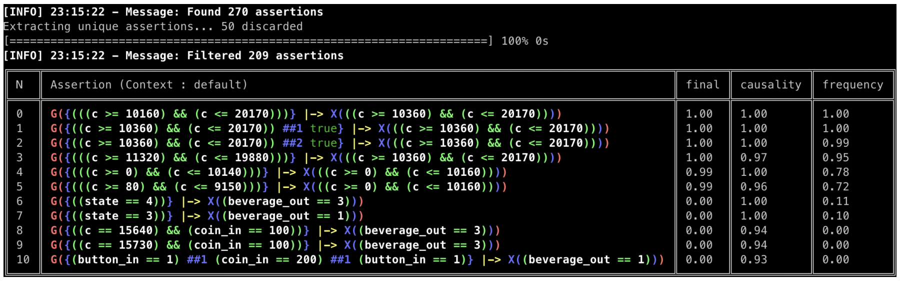
	CT	**CF**
AT	130	0
AF	861	8

Table 10: `p4`

4 HARM

The tool extracted the properties `p0` and `p1`, in their exact form. This was obtained by removing three templates and leaving only two that were similar to the manually defined assertions. A filter was applied in order to obtain a manageable set of results: it selects the assertions that have at least 90% of causality.

As expected, the credit quickly rises in the `.vcd` trace that includes every coin and the interesting assertions will be placed at rows 6 and 7, due to their low frequency.

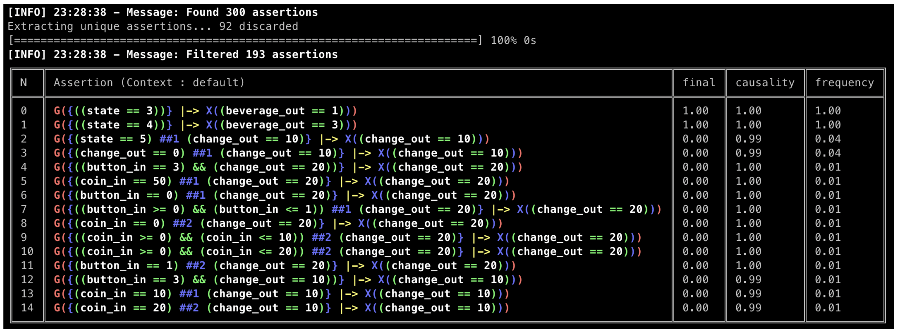


```
[INFO] 23:15:22 - Message: Found 270 assertions
Extracting unique assertions... 50 discarded
[=====================================================================] 100% 0s
[INFO] 23:15:22 - Message: Filtered 209 assertions
 N    Assertion (Context : default)                                                          final  causality  frequency

 0    G({(((c >= 10160) && (c <= 20170)))} |-> X(((c >= 10360) && (c <= 20170))))             1.00   1.00       1.00
 1    G({((c >= 10360) && (c <= 20170)) ##1 true} |-> X(((c >= 10360) && (c <= 20170))))      1.00   1.00       1.00
 2    G({((c >= 10360) && (c <= 20170)) ##2 true} |-> X(((c >= 10360) && (c <= 20170))))      1.00   1.00       0.99
 3    G({(((c >= 11320) && (c <= 19880)))} |-> X(((c >= 10360) && (c <= 20170))))             1.00   0.97       0.95
 4    G({(((c >= 0) && (c <= 10140)))} |-> X(((c >= 0) && (c <= 10160))))                     0.99   1.00       0.78
 5    G({(((c >= 80) && (c <= 9150)))} |-> X(((c >= 0) && (c <= 10160))))                     0.99   0.96       0.72
 6    G({((state == 4))} |-> X((beverage_out == 3)))                                          0.00   1.00       0.11
 7    G({((state == 3))} |-> X((beverage_out == 1)))                                          0.00   1.00       0.10
 8    G({((c == 15640) && (coin_in == 100))} |-> X((beverage_out == 3)))                      0.00   0.94       0.00
 9    G({((c == 15730) && (coin_in == 100))} |-> X((beverage_out == 3)))                      0.00   0.94       0.00
 10   G({(button_in == 1) ##1 (coin_in == 200) ##1 (button_in == 1)} |-> X((beverage_out == 1)))  0.00   0.93       0.00
```

Figure 1: Assertions with the full set of coins

However, the trace with the subset of coins allows the same properties to be asserted more often, therefore they're placed on top of the table.



```
[INFO] 23:28:38 - Message: Found 300 assertions
Extracting unique assertions... 92 discarded
[====================================================================] 100% 0s
[INFO] 23:28:38 - Message: Filtered 193 assertions
 N    Assertion (Context : default)                                                          final  causality  frequency

 0    G({((state == 3))} |-> X((beverage_out == 1)))                                          1.00   1.00       1.00
 1    G({((state == 4))} |-> X((beverage_out == 3)))                                          1.00   1.00       1.00
 2    G({(state == 5) ##1 (change_out == 10)} |-> X((change_out == 10)))                      0.00   0.99       0.04
 3    G({(change_out == 0) ##1 (change_out == 10)} |-> X((change_out == 10)))                 0.00   0.99       0.04
 4    G({((button_in == 3) && (change_out == 20))} |-> X((change_out == 20)))                 0.00   1.00       0.01
 5    G({(coin_in == 50) ##1 (change_out == 20)} |-> X((change_out == 20)))                   0.00   1.00       0.01
 6    G({(button_in == 0) ##1 (change_out == 20)} |-> X((change_out == 20)))                  0.00   1.00       0.01
 7    G({((button_in >= 0) && (button_in <= 1)) ##1 (change_out == 20)} |-> X((change_out == 20)))  0.00   1.00       0.01
 8    G({(coin_in == 0) ##2 (change_out == 20)} |-> X((change_out == 20)))                    0.00   1.00       0.01
 9    G({((coin_in >= 0) && (coin_in <= 10)) ##2 (change_out == 20)} |-> X((change_out == 20)))   0.00   1.00       0.01
 10   G({((coin_in >= 0) && (coin_in <= 20)) ##2 (change_out == 20)} |-> X((change_out == 20)))   0.00   1.00       0.01
 11   G({(button_in == 1) ##2 (change_out == 20)} |-> X((change_out == 20)))                  0.00   1.00       0.01
 12   G({((button_in == 3) && (change_out == 10))} |-> X((change_out == 10)))                 0.00   0.99       0.01
 13   G({(coin_in == 10) ##1 (change_out == 10)} |-> X((change_out == 10)))                   0.00   0.99       0.01
 14   G({(coin_in == 20) ##2 (change_out == 10)} |-> X((change_out == 10)))                   0.00   0.99       0.01
```

Figure 2: Assertions without 1 and 2 € coins