Design Team Project Vending Machine Controller Verification

ENSE803 - Formal Specification and Design

Declan Ross (20108351)
Anson Huang (20120333)
Mahiir Hussain Shaik (21154501)

PRISM Model

Here is our PRISM model for the project, it was created and initially written in VSCode, then tested in the PRISM model checker.

```
mdp

//Constants for drink types and stock
const int none = 0;
const int kiwi = 1;
const int bolt = 2;
const int water = 3;
const int max_stock = 3;

module DrinkSelection
    state: [0..3] init none; //0=none, 1=Kiwi-Cola, 2=Bolt Energy, 3=Clear Water

    //Select drink if stock available and not in maintenance mode
    [select_kiwi] state=none & kiwi_stock>0 & !maintenance -> (state'=kiwi);
    [select_bolt] state=none & bolt_stock>0 & !maintenance -> (state'=bolt);
    [select_water] state=none & water_stock>0 & !maintenance -> (state'=water);

    //Change selection before payment
    [change_selection] state>none & !maintenance & !pay -> (state'=none);

    //Synchronize with payment and reset after dispensing
    [pay] state>none & pay & !maintenance -> (state'=none);

    //Reset on incorrect PIN
    [reset] state>none & payment_error & !maintenance -> (state'=none);
endmodule

module EFPOSPayment
    pay: bool init false; //True when payment is successful
    payment_error: bool init false; //True when incorrect PIN entered

    //Start payment after selection
    [start_payment] state>none & !pay & !payment_error & !maintenance -> (pay'=true);

    //Incorrect PIN
    [wrong_pin] state>none & !pay & !payment_error & !maintenance ->
(payment_error'=true);

    //Reset after incorrect PIN
    [reset] payment_error & !maintenance -> (payment_error'=false) & (pay'=false);

    //Reset after dispensing
    [dispense] pay & !maintenance -> (pay'=false);
endmodule
```

```
module Dispenser
    kiwi_stock: [0..max_stock] init max_stock;
    bolt_stock: [0..max_stock] init max_stock;
    water_stock: [0..max_stock] init max_stock;
    error: bool init false;
    maintenance: bool init false;
    dispense: bool init false;

    //Dispense drink if paid and stock available
    [dispense] pay & state=kiwi & kiwi_stock>0 & !maintenance ->
(kiwi_stock'=kiwi_stock-1) & (dispense'=true);
    [dispense] pay & state=bolt & bolt_stock>0 & !maintenance ->
(bolt_stock'=bolt_stock-1) & (dispense'=true);
    [dispense] pay & state=water & water_stock>0 & !maintenance ->
(water_stock'=water_stock-1) & (dispense'=true);

    //Reset dispense flag
    [reset_dispense] dispense & !maintenance -> (dispense'=false);

    //Simulate error event
    [error_event] !error & !maintenance -> 0.99:(error'=false) + 0.01:(error'=true);

  //Error event
  [error_event] error & !maintenance -> (maintenance'=true);

    //Maintenance mode when all drinks are empty
    [check_stock] kiwi_stock=0 & bolt_stock=0 & water_stock=0 & !maintenance ->
(maintenance'=true);

    //Stay in maintenance mode
    [] maintenance -> (maintenance'=true);
endmodule
```

Design Decisions

1. *Modular Structure.* The model is designed modularly, with components logically divided into separate modules: *DrinkSelection, EFPOSPayment,* and *Dispenser.* This separation of concerns improves readability and maintainability, as each module has a well-defined purpose.
2. *Clear and Concise Variable Naming.* Descriptive and consistent variable names such as *state*, *kiwi_stock*, and *pay* are used throughout the model. This improves code clarity and helps users understand the model's behavior at a glance.
3. *Well-Defined Transitions.* Transitions are defined with precise guard conditions to ensure that actions only occur under valid circumstances. For instance, the dispense transition only triggers when payment has been successfully made and a valid drink selection has been made.
4. *Error Modeling with Probabilistic Transitions.* We introduced a probabilistic transition for simulating internal machine faults ([error_event] with 0.01 probability). This models real-world uncertainty and allows for analysis of system reliability using probabilistic model checking.

Scenarios

Here are our scenarios.

Scenario 1

Customer selects Clear Water and pays via EFPOS, with correct pin.

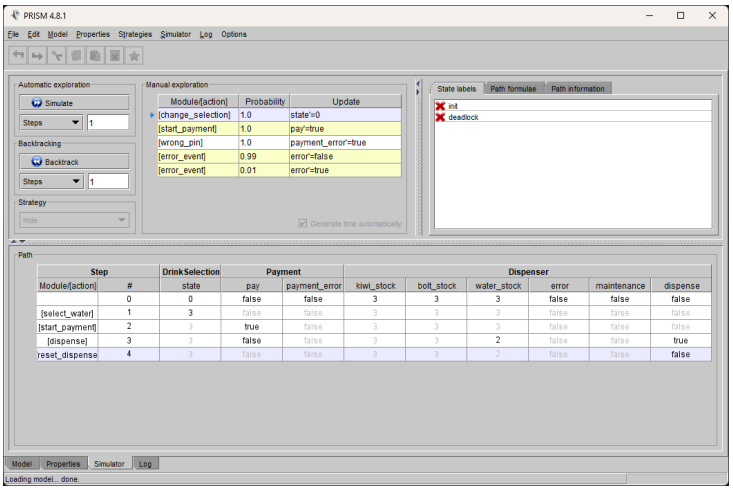

Figure 1: Simulation trace output: Customer selects Clear Water and pays via EFPOS with correct pin

In this scenario, the customer successfully selects Clear Water and pays via EFPOS with the correct pin. The simulation trace shows that the drink is dispensed correctly, and the stock of Clear Water is reduced by one. The system remains in a normal operational state without entering maintenance mode.

Scenario 2

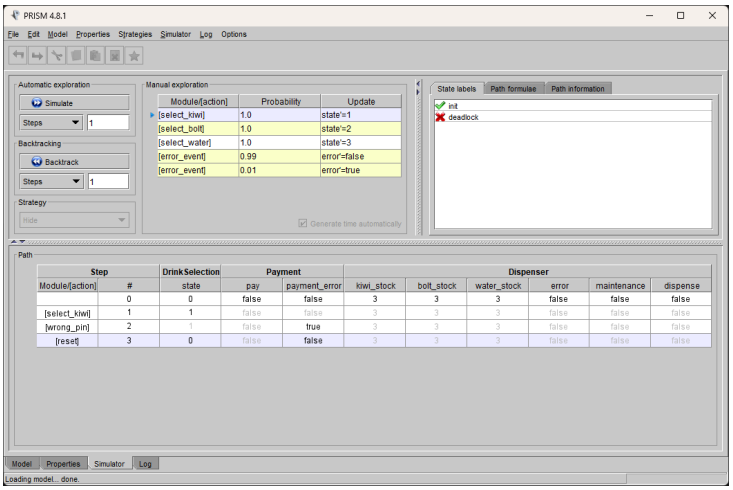

Figure 2: Simulation trace output: Customer selects Kiwi-Cola and pays via EFPOS with incorrect pin

In this scenario, the customer attempts to select Kiwi-Cola and pay via EFPOS but enters an incorrect pin. The simulation trace shows that the payment fails, and the system resets the payment status. The customer can then reattempt the selection or payment without entering maintenance mode.

Scenario 3

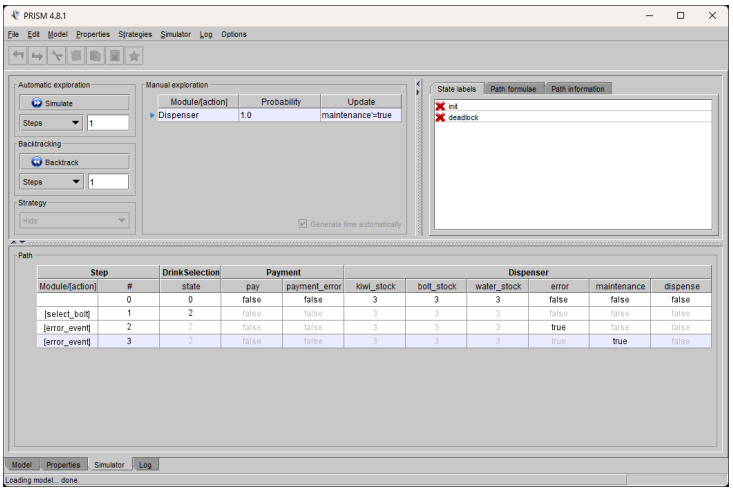

Figure 3: Simulation trace output: Customer selects a drink but an error occurs

In this scenario, the customer selects a drink, but an error occurs during the process. The simulation trace shows that the system enters maintenance mode due to the error. The customer cannot complete the transaction until the error is resolved, ensuring that the system remains in a safe state.

Scenario 4

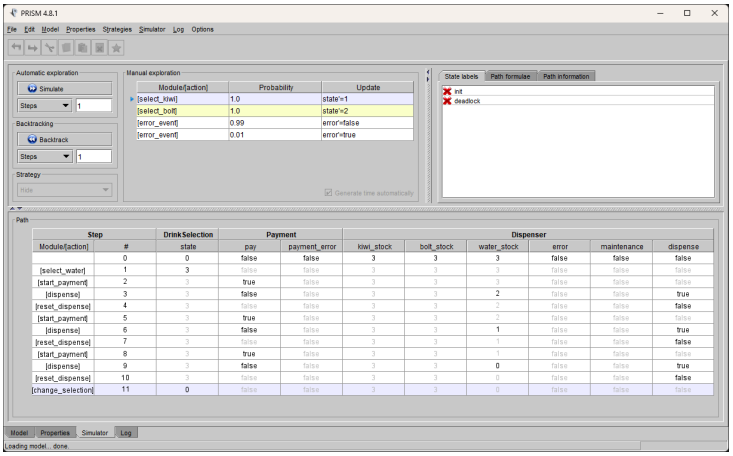

Figure 4: Simulation trace output: Customer selects Clear Water but no drinks available

Customer selects a Clear Water but there are no drinks of this kind available.
In this scenario, the customer attempts to select Clear Water, but there are no drinks available. The simulation trace shows that the system does not allow the selection of an unavailable drink, and the customer is prompted to make a different selection. This ensures that the vending machine does not dispense an empty or unavailable drink.

Scenario 5

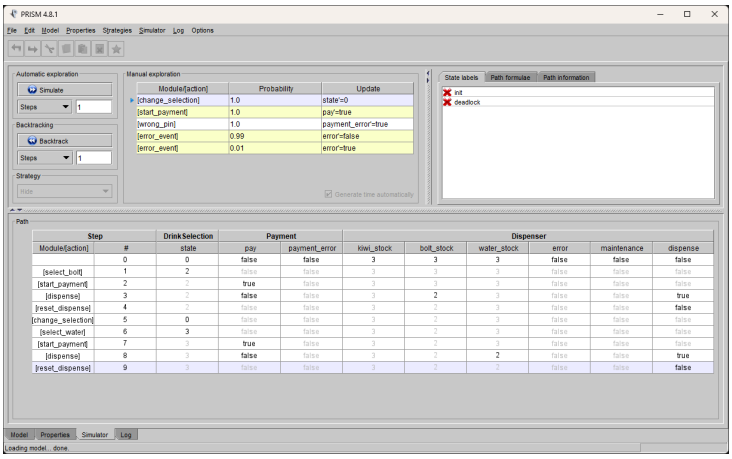

Figure 5: Simulation trace output: Customer purchases Bolt Energy Drink and then purchases Clear Water

Simulation trace output for each scenario with discussion In this scenario, the customer successfully purchases a Bolt Energy Drink and then proceeds to purchase Clear Water. The simulation trace shows that the system correctly updates the stock levels for both drinks and dispenses them as expected. The system remains in a normal operational state without entering maintenance mode, demonstrating its ability to handle multiple transactions sequentially.

Temporal Logic Formulae

Here is our list of Formulae:

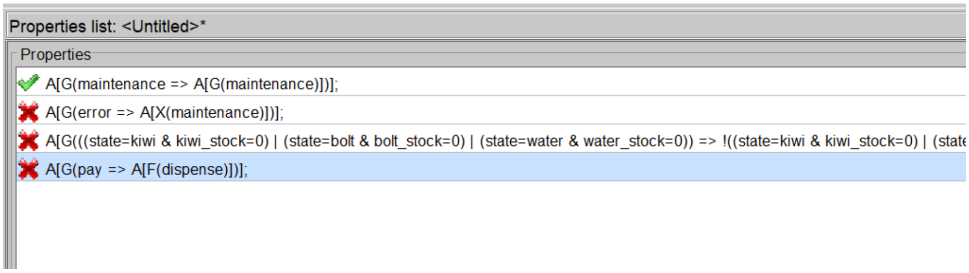

Figure 6: Temporal Logic Formulae for Vending Machine Controller Verification

1. Once in, the vending machine never leaves maintenance mode:

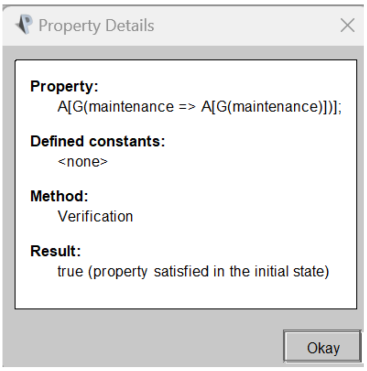

Figure 7: Maintenance Mode

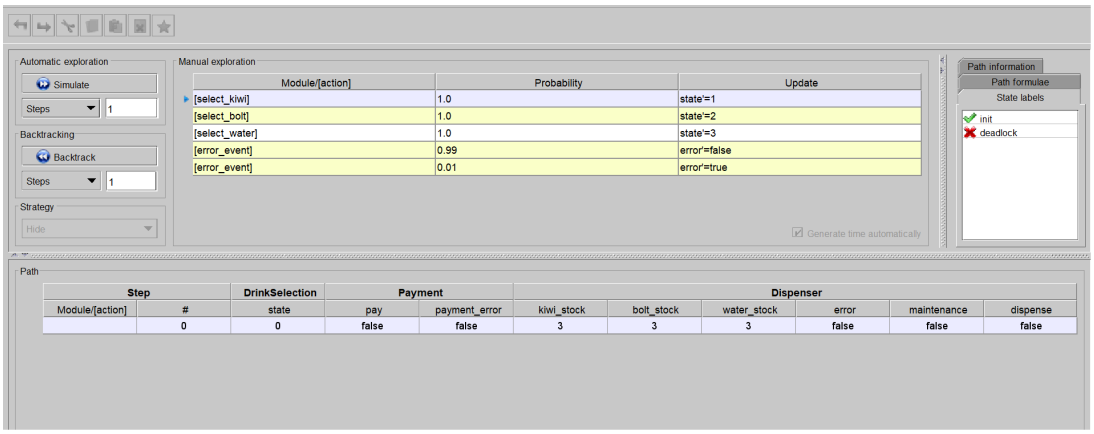

Figure 8: Maintenance mode, simulator tab

2. If an error occurs, then maintenance mode occurs in the next state:

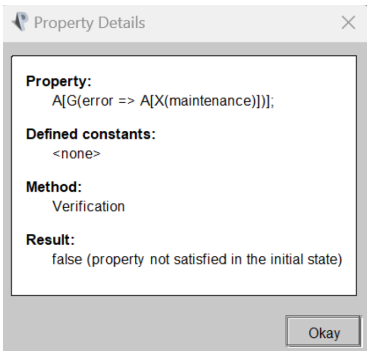

Figure 9: Error Handling

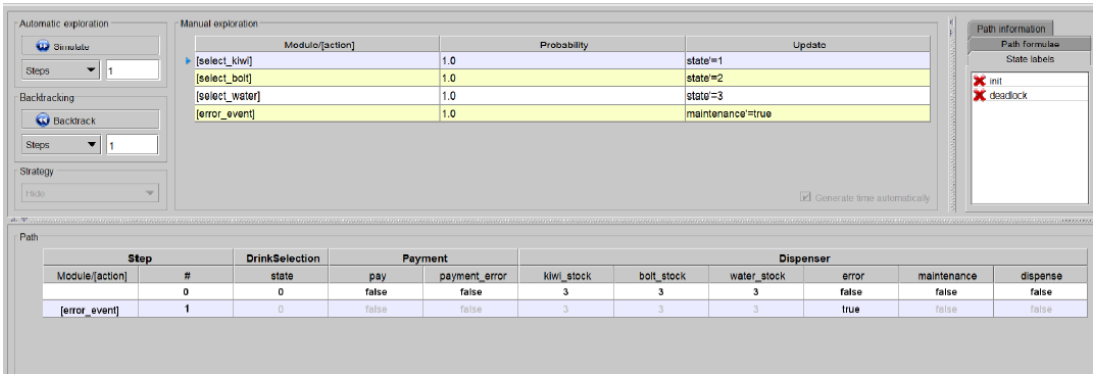

Figure 10: Error Handling mode, simulator tab

3. A customer may not select an unavailable drink *AG (soda empty -> AF soda not selected):*

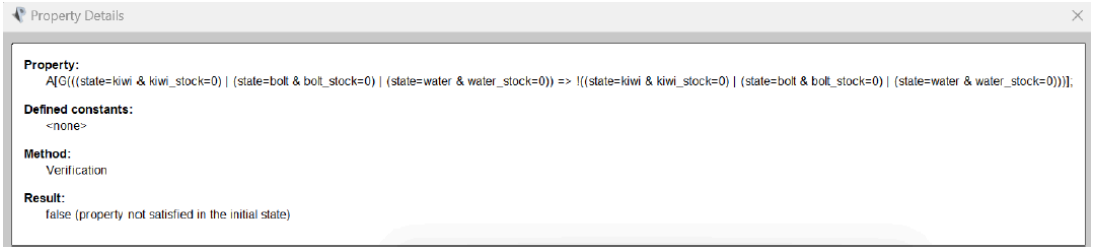

Figure 11: Unavailable Drink Selection

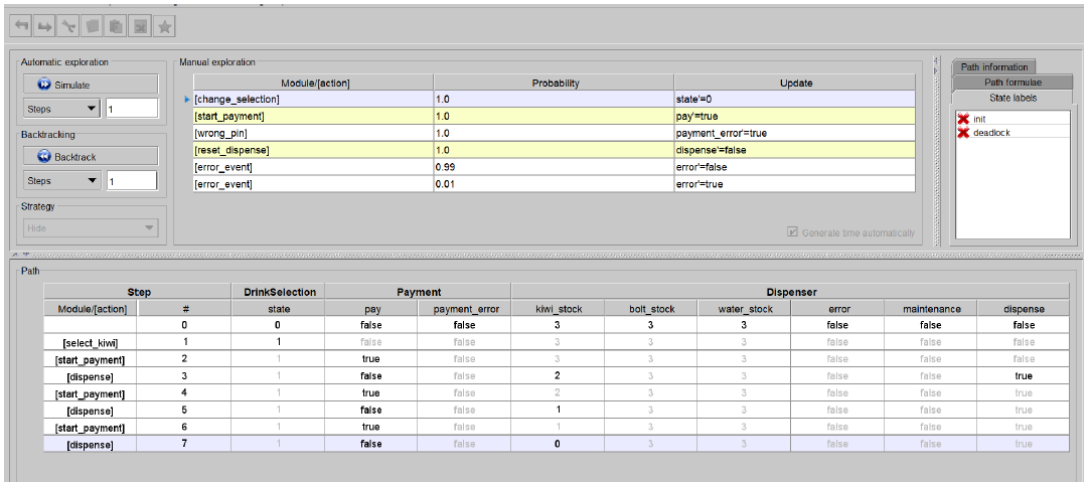

Figure 12: Unavailable Drink Selection, simulator tab

4. When a customer pays for a drink, it is dispensed *AG (pay -> AF dispense):*

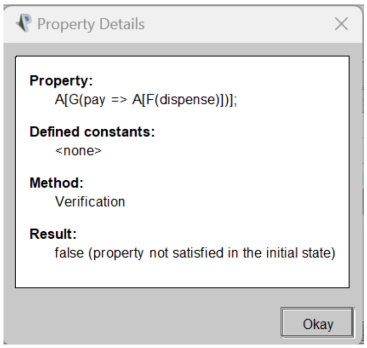

Figure 13: Drink Dispensing

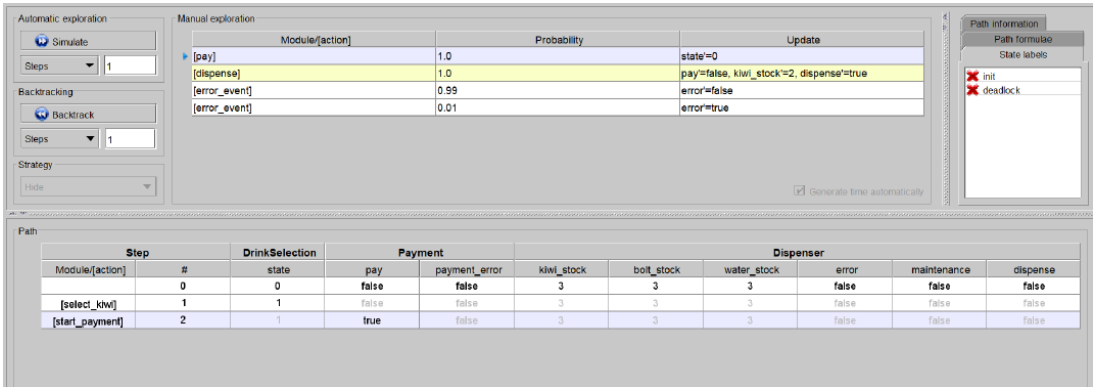

Figure 14: Drink Dispensing, simulator tab