Design Team Project Vending Machine Controller Verification

ENSE803 - Formal Specification and Design

Declan Ross (20108351)
Anson Huang (20120333)
Maahir 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; //Minimum 3 per assignment

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

    //Select any drink directly from the menu only if it's in stock
    [select_kiwi] state=none & kiwi_stock>0 & !maintenance & !pay & !dispense & !main
& !error -> (state'=kiwi);
    [select_bolt] state=none & bolt_stock>0 & !maintenance & !pay & !dispense & !main
& !error -> (state'=bolt);
    [select_water] state=none & water_stock>0 & !maintenance & !pay & !dispense & !
main & !error -> (state'=water);

    //Change selection if selected drink is out of stock
    [change_selection] state=kiwi & kiwi_stock=0 & !maintenance & !pay & !dispense
& !main & !error -> (state'=none);
    [change_selection] state=bolt & bolt_stock=0 & !maintenance & !pay & !dispense
& !main & !error -> (state'=none);
    [change_selection] state=water & water_stock=0 & !maintenance & !pay & !dispense
& !main & !error -> (state'=none);

    //Synchronize with payment, incorrect PIN, or error
    [pay] state>none & !maintenance & !main & !error -> (state'=none);
    [wrong_pin] state>none & !maintenance & !main & !error -> (state'=none);
    [error] state>none & !maintenance & !main & !error -> (state'=none);
endmodule

module PaymentDispenser
    pay: bool init false; //True when payment is initiated
    dispense: bool init false; //True when dispensing
    kiwi_stock: [0..max_stock] init max_stock;
    bolt_stock: [0..max_stock] init max_stock;
    water_stock: [0..max_stock] init max_stock;
    maintenance: bool init false;
```

```
//Process payment (correct PIN, sets dispense=true, reduces stock if available)
    [pay] state=kiwi & !maintenance & !dispense & !main & !error & kiwi_stock>0 ->
(pay'=false) & (dispense'=true) & (kiwi_stock'=kiwi_stock-1);
    [pay] state=bolt & !maintenance & !dispense & !main & !error & bolt_stock>0 ->
(pay'=false) & (dispense'=true) & (bolt_stock'=bolt_stock-1);
    [pay] state=water & !maintenance & !dispense & !main & !error & water_stock>0 ->
(pay'=false) & (dispense'=true) & (water_stock'=water_stock-1);
    [pay] state=kiwi & !maintenance & !dispense & !main & !error & kiwi_stock=0 ->
(pay'=false) & (dispense'=true);
    [pay] state=bolt & !maintenance & !dispense & !main & !error & bolt_stock=0 ->
(pay'=false) & (dispense'=true);
    [pay] state=water & !maintenance & !dispense & !main & !error & water_stock=0 ->
(pay'=false) & (dispense'=true);

    //Reset dispense flag after payment
    [] dispense=true & !maintenance & !main & !error -> (dispense'=false);

    //Incorrect PIN
    [wrong_pin] state>none & !pay & !dispense & !main & !error -> true;

    //Maintenance mode when all drinks are empty
    [check_stock] state=none & kiwi_stock=0 & bolt_stock=0 & water_stock=0 & !
maintenance & !main & !error -> (maintenance'=true);

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

    //Synchronize maintenance with error
    [error] state>none & !pay & !dispense & !main & !error -> (maintenance'=true);
endmodule

module error
    main : bool init false;
    error : bool init false;
    [error] !main & !error -> (error'=true);
    [main] error -> (main'=true) & (error'=false);
endmodule
```

Design Decisions

1. *Modular Structure.* The model is designed modularly, with components logically divided into separate modules: *DrinkSelection, Payment Dispenser,* and *Error.* 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.
3. *Well-Defined Transitions.* Transisitons between states are clearly defined, with conditions specified for each transition. This ensures that the model accurately reflects the intended behavior of the vending machine, such as allowing drink selection only when stock is available, or permanently staying in maintenance mode.
4. *Intentional Error.* You can intentionally cause an error by selecting error after selecting a drink. The error transition leads to a maintenance mode.
5. *PaymentDispenser.* This module handles the payment and dispensing logic. It includes transitions for successful payments, incorrect PIN entries, and error handling. The payment process is designed to ensure that drinks are dispensed only when payment is successful and stock is available.

Scenarios

Scenario 1

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

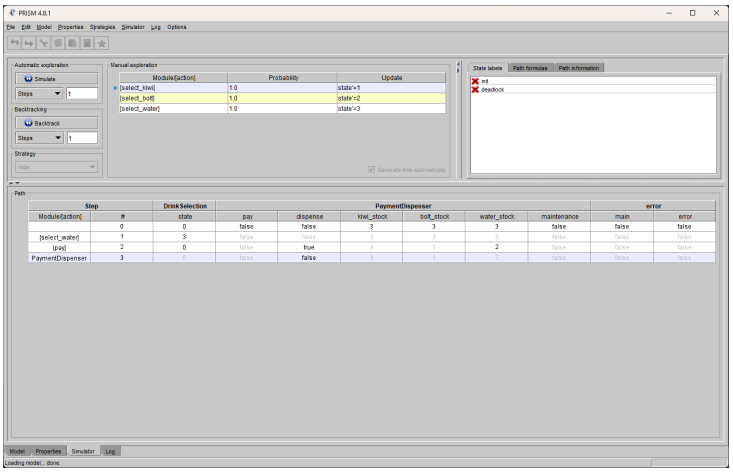

Figure 1: Simulation trace output: Customer selects Clear Water and pays via EFPOS with correct pin

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

Customer selects Kiwi-Cola and pays via EFPOS, with incorrect pin.

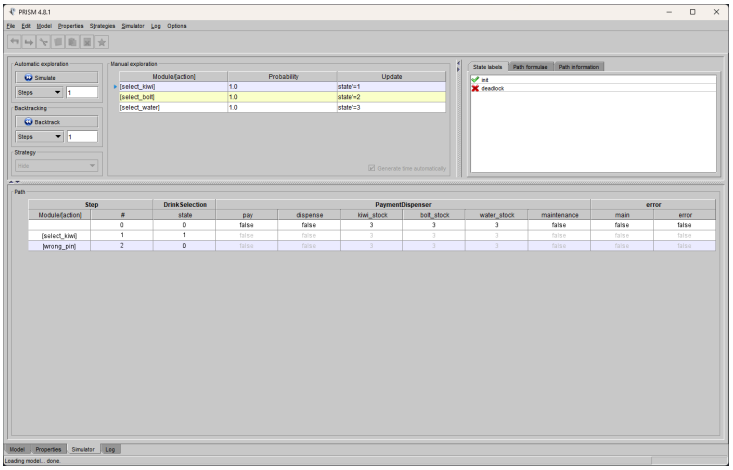

Figure 2: Simulation trace output: Customer selects Kiwi-Cola and pays via EFPOS with incorrect pin

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

Customer selects a drink but an error occurs.

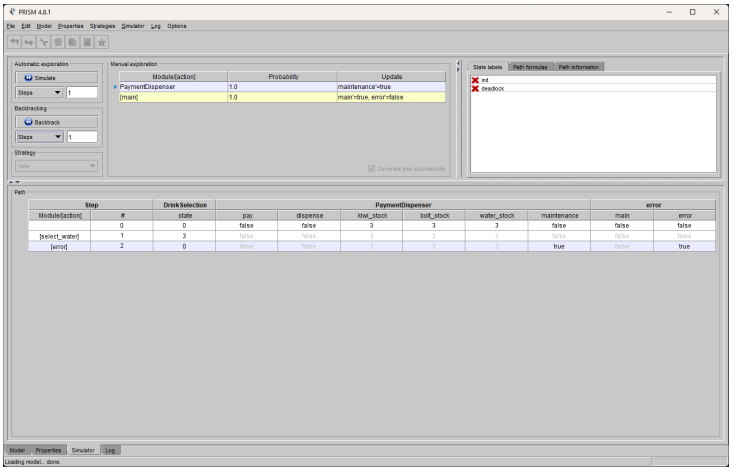

Figure 3: Simulation trace output: Customer selects a drink but an error occurs

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

Customer selects a Clear Water but there are no drinks of this kind available.

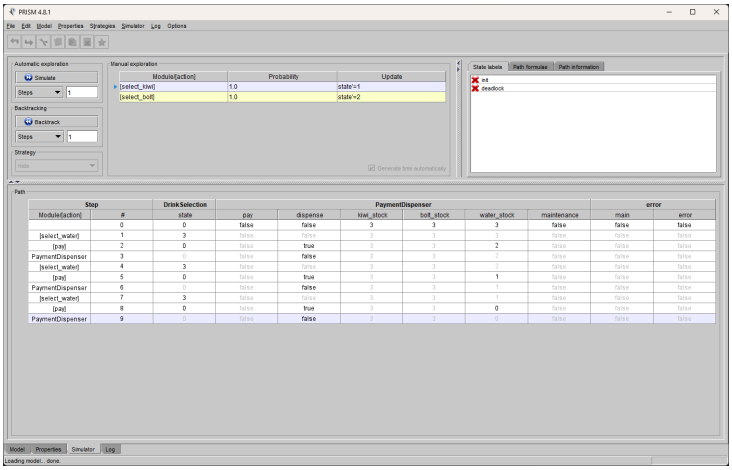

Figure 4: Simulation trace output: Customer selects Clear Water but no drinks available

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

Customer purchases Bolt Energy Drink and then purchases Clear Water.

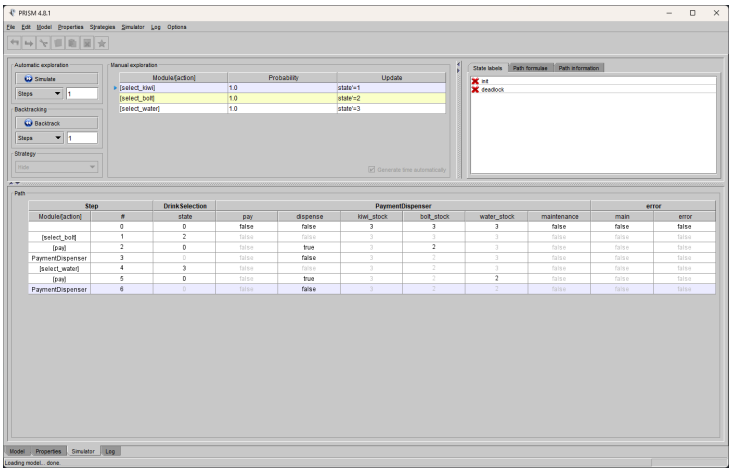

Figure 5: Simulation trace output: Customer purchases Bolt Energy Drink and then purchases Clear Water

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:

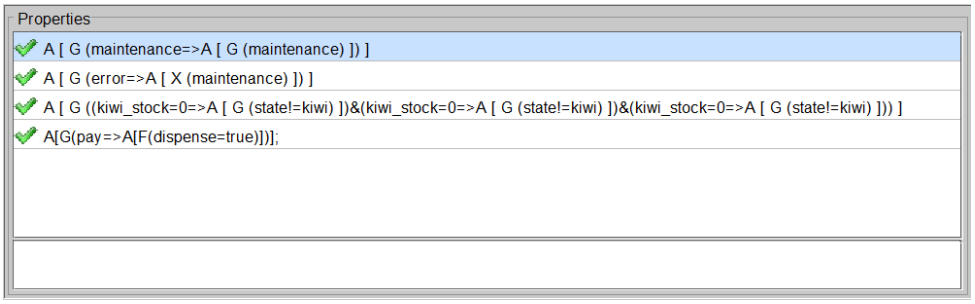

Figure 6: Temporal Logic Formulae for Vending Machine Controller Verification

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

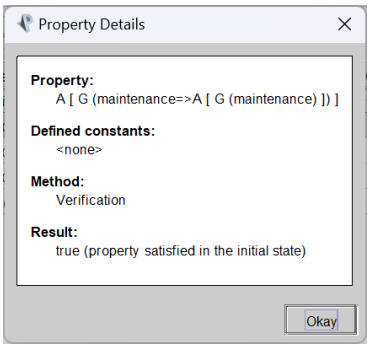

Figure 7: Maintenance Mode

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

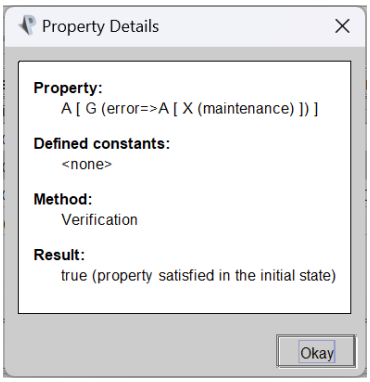

Figure 8: Error Handling

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

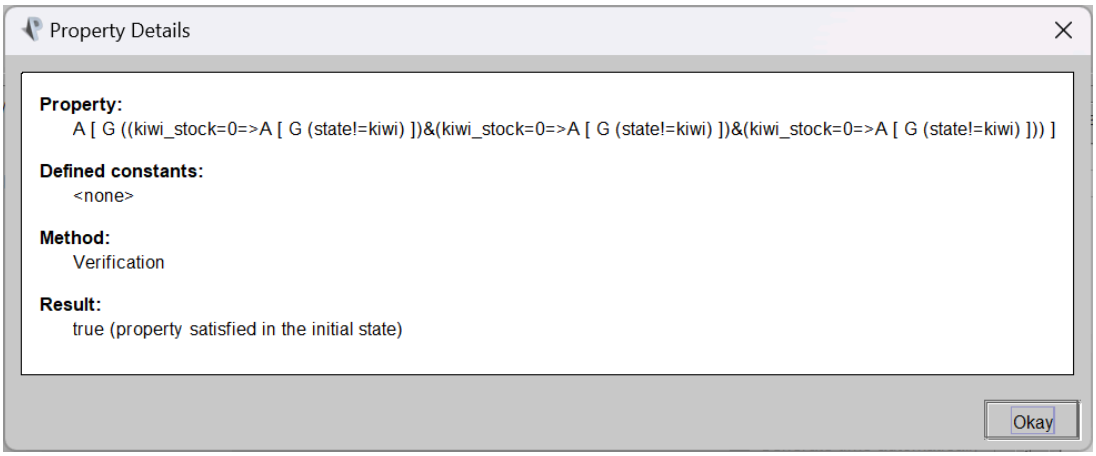

Figure 9: Unavailable Drink Selection

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

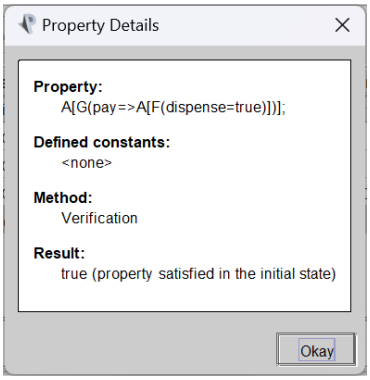

Figure 10: Drink Dispensing