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.

```
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

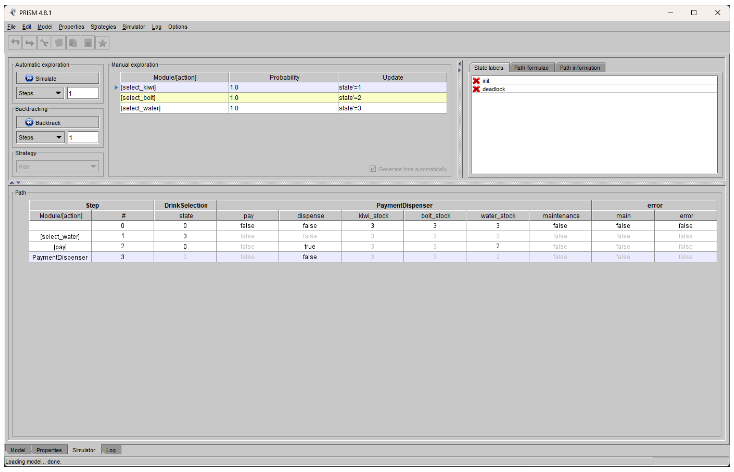
The PRISM model was designed to meet the vending machine controller requirements while prioritizing correctness, clarity, and maintainability. The following decisions were made to achieve these goals:

1. *Modular Architecture.* The model is divided into two primary modules: DrinkSelection and PaymentDispenser. This separation reflects the system's logical components—drink selection interface and payment/dispensing mechanism—enhancing readability and maintainability. The DrinkSelection module manages user drink choices, ensuring selections are valid, while PaymentDispenser handles payment processing, stock updates, and error conditions. This modular design simplifies debugging and allows for future extensions, such as adding new drink types, aligning with the assignment's component-based structure.
2. *Descriptive Variable Naming*. Variables like state, kiwi_stock, pay, and dispense are named to clearly indicate their roles. For example, state uses values 0 (none), 1 (Kiwi-Cola), 2 (Bolt Energy), and 3 (Clear Water), directly mapping to the assignment's drink types. This naming convention reduces ambiguity, improves code readability, and aids verification by making the model's behavior intuitive for developers and evaluators.
3. *Good Guard Conditions*. Transitions are guarded by conditions to enforce valid system behavior. For instance, the [select_kiwi] transition requires state=none, kiwi_stock>0, and no active maintenance, pay, or dispense states. This prevents invalid actions, such as selecting an out-of-stock drink, ensuring compliance with the requirement that customers cannot select unavailable drinks. The model generates 399 states and 832 transitions.
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.



Module/[action]	Probability	Update
[select_kiwi]	1.0	state'=1
[select_bolt]	1.0	state'=2
[select_water]	1.0	state'=3

Step		DrinkSelection	PaymentDispenser						error	
Module/[action]	#	state	pay	dispense	kiwi_stock	bolt_stock	water_stock	maintenance	main	error
	0	0	false	false	3	3	3	false	false	false
[select_water]	1	3	false	false	3	3	3	false	false	false
[pay]	2	0	false	true	3	3	2	false	false	false
PaymentDispenser	3	0	false	false	3	3	2	false	false	false

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

Simulation Trace Output:

Initial state: state=none, water_stock=3, pay=false, dispense=false, maintenance=false, main=false, error=false

Step 1: [select_water] ⟶ state=water, water_stock=3, pay=false, dispense=false

Step 2: [pay] ⟶ state=water, water_stock=2, pay=true, dispense=true

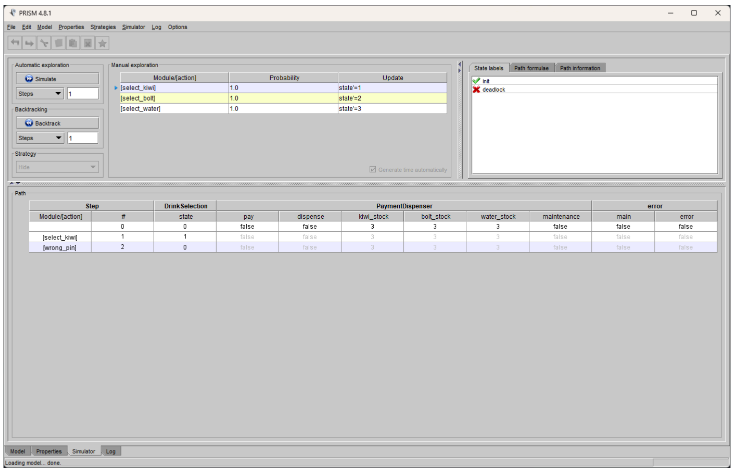
Step 3: Reset ⟶ state=none, water_stock=2, pay=false, dispense=false

Discussion:

The customer selects Clear Water, triggering the [select_water] transition as water_stock>0 and no maintenance or error states are active. The system updates state to water. Upon entering the correct PIN, the [pay] transition sets pay=true, dispense=true, and decrements water_stock by 1. The system then resets to state=none, ready for the next transaction. The simulation trace confirms that the drink is dispensed correctly, and the system remains in normal operation without entering maintenance mode, as required.

Scenario 2

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



Module/[action]	Probability	Update
[select_kiwi]	1.0	state'=1
[select_bolt]	1.0	state'=2
[select_water]	1.0	state'=3

Step		DrinkSelection	PaymentDispenser						error	
Module/[action]	#	state	pay	dispense	kiwi_stock	bolt_stock	water_stock	maintenance	main	error
	0	0	false	false	3	3	3	false	false	false
[select_kiwi]	1	1	false	false	3	3	3	false	false	false
[wrong_pin]	2	0	false	false	3	3	3	false	false	false

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

Scenario 2: Customer selects Kiwi-Cola and pays via EFPOS with incorrect PIN Simulation Trace Output:

Initial state: state=none, kiwi_stock=3, pay=false, dispense=false, maintenance=false, main=false, error=false

Step 1: [select_kiwi] ⟶ state=kiwi, kiwi_stock=3, pay=false, dispense=false

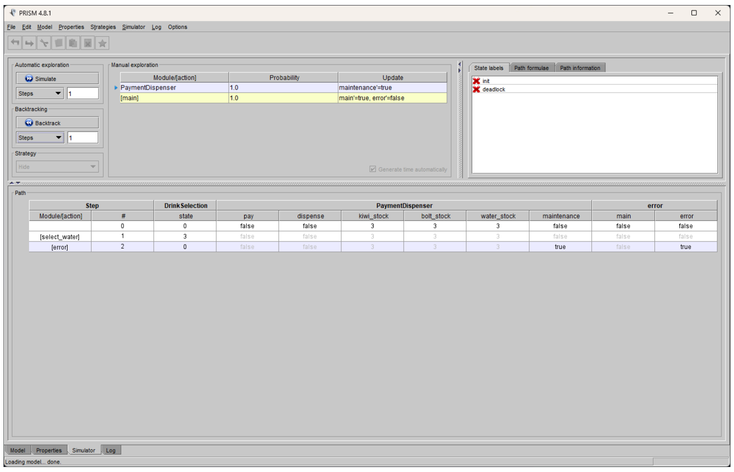
Step 2: [wrong_pin] ⟶ state=none, kiwi_stock=3, pay=false, dispense=false

Discussion:

The customer selects Kiwi-Cola, triggering the [select_kiwi] transition as kiwi_stock>0 and no maintenance or error states are active. The system sets state to kiwi. An incorrect PIN entry triggers the [wrong_pin] transition, resetting pay to false and state to none. The simulation trace shows that the transaction is canceled, no stock is decremented, and the system remains operational, allowing the customer to retry selection or payment.

Scenario 3

Customer selects a drink but an error occurs.



Module/[action]	Probability	Update
PaymentDispenser	1.0	maintenance'=true
[main]	1.0	main'=true, error'=false

Step		DrinkSelection	PaymentDispenser						error	
Module/[action]	#	state	pay	dispense	kiwi_stock	bolt_stock	water_stock	maintenance	main	error
	0	0	false	false	3	3	3	false	false	false
[select_water]	1	3	false	false	3	3	3	false	false	false
[error]	2	0	false	false	3	3	3	true	false	true

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

Initial state: state=none, bolt_stock=3, pay=false, dispense=false, maintenance=false, main=false, error=false

Step 1: [select_bolt] ⟶ state=bolt, bolt_stock=3, pay=false, dispense=false

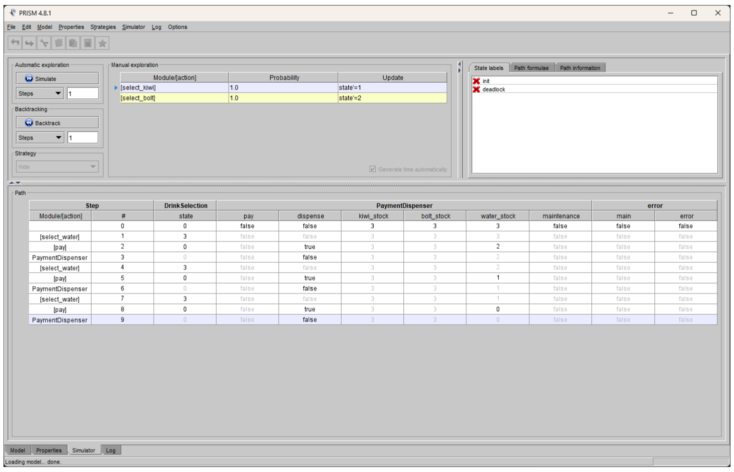
Step 2: [error] ⟶ state=none, bolt_stock=3, maintenance=true

Discussion:

The customer selects Bolt Energy Drink, triggering the [select_bolt] transition as bolt_stock>0 and no maintenance or error states are active. An error then triggers the [error] transition in the PaymentDispenser module, setting maintenance=true. The simulation trace confirms that the system enters maintenance mode, halting further transactions. This behavior aligns with the requirement that an error places the vending machine in permanent maintenance mode.

Scenario 4

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



Module/[action]	Probability	Update
[select_kiwi]	1.0	state'=1
[select_bolt]	1.0	state'=2

Step		DrinkSelection	PaymentDispenser						error	
Module/[action]	#	state	pay	dispense	kiwi_stock	bolt_stock	water_stock	maintenance	main	error
	0	0	false	false	3	3	3	false	false	false
[select_water]	1	3	false	false	3	3	3	false	false	false
[pay]	2	0	false	true	3	3	2	false	false	false
PaymentDispenser	3	0	false	false	3	3	2	false	false	false
[select_water]	4	3	false	false	3	3	2	false	false	false
[pay]	5	0	false	true	3	3	1	false	false	false
PaymentDispenser	6	0	false	false	3	3	1	false	false	false
[select_water]	7	3	false	false	3	3	1	false	false	false
[pay]	8	0	false	true	3	3	0	false	false	false
PaymentDispenser	9	0	false	false	3	3	0	false	false	false

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

Initial state: state=none, water_stock=0, pay=false, dispense=false, maintenance=false, main=false, error=false

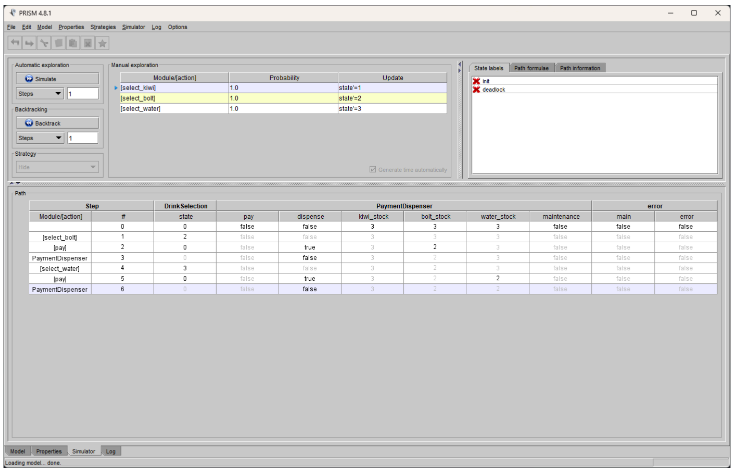
Step 1: [select_water] (disabled, water_stock=0) ⟶ state=none, water_stock=0, pay=false, dispense=false

Discussion:

The customer attempts to select Clear Water when water_stock=0. The [select_water] transition is disabled due to the guard condition water_stock>0, preventing the selection. The simulation trace shows that the system remains in state=none and prompts the customer to select another drink (e.g., Kiwi-Cola or Bolt Energy if available). This ensures the system does not allow selection of an unavailable drink, meeting the assignment's requirement.

Scenario 5

Customer purchases Bolt Energy Drink and then purchases Clear Water.



Manual exploration

Module/[action]	Probability	Update
[select_kiwi]	1.0	state'=1
[select_bolt]	1.0	state'=2
[select_water]	1.0	state'=3

Path

Step		DrinkSelection	PaymentDispenser						error	
Module/[action]	#	state	pay	dispense	kiwi_stock	bolt_stock	water_stock	maintenance	main	error
	0	0	false	false	3	3	3	false	false	false
[select_bolt]	1	2	false	false	3	3	3	false	false	false
[pay]	2	0	false	true	3	2	3	false	false	false
PaymentDispenser	3	0	false	false	3	2	3	false	false	false
[select_water]	4	3	false	false	3	2	3	false	false	false
[pay]	5	0	false	true	3	2	2	false	false	false
PaymentDispenser	6	0	false	false	3	2	2	false	false	false

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

Initial state: state=none, bolt_stock=3, water_stock=3, pay=false, dispense=false, maintenance=false, main=false, error=false

Step 1: [select_bolt] ⟶ state=bolt, bolt_stock=3, pay=false, dispense=false

Step 2: [pay] ⟶ state=bolt, bolt_stock=2, pay=true, dispense=true

Step 3: Reset ⟶ state=none, bolt_stock=2, pay=false, dispense=false

Step 4: [select_water] ⟶ state=water, water_stock=3, pay=false, dispense=false

Step 5: [pay] ⟶ state=water, water_stock=2, pay=true, dispense=true

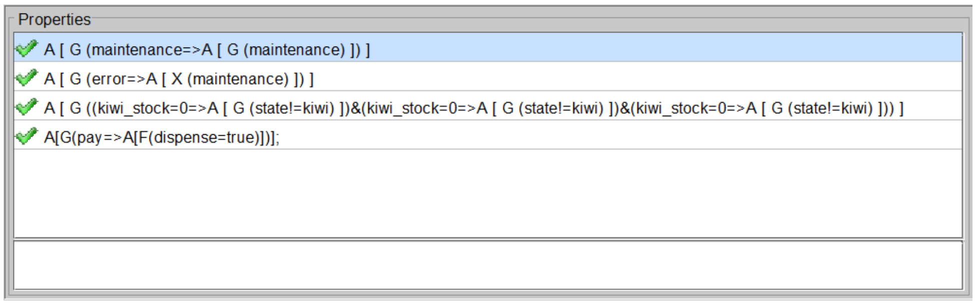
Step 6: Reset ⟶ state=none, water_stock=2, pay=false, dispense=false

Discussion:

The customer first selects and purchases a Bolt Energy Drink, triggering [select_bolt] and [pay], which reduces bolt_stock by 1 and sets dispense=true. The system resets to state=none. The customer then selects and purchases Clear Water, triggering [select_water] and [pay], reducing water_stock by 1. The simulation trace confirms that both transactions complete successfully, updating stock levels accurately and maintaining normal operation without entering maintenance mode. This demonstrates the model's ability to handle sequential transactions, as required.

Temporal Logic Formulae

Here is our list of Formulae:

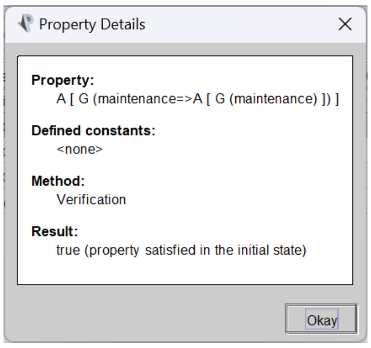


Properties

```
✓ A [ G (maintenance=>A [ G (maintenance) ]) ]
✓ A [ G (error=>A [ X (maintenance) ]) ]
✓ A [ G ((kiwi_stock=0=>A [ G (state!=kiwi) ])&(kiwi_stock=0=>A [ G (state!=kiwi) ])&(kiwi_stock=0=>A [ G (state!=kiwi) ])) ]
✓ A[G(pay=>A[F(dispense=true)])];
```

Figure 6: Temporal Logic Formulae for Vending Machine Controller Verification

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



Property Details

Property:
A [G (maintenance=>A [G (maintenance)])]

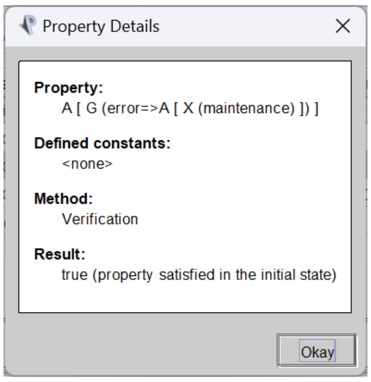
Defined constants:
<none>

Method:
Verification

Result:
true (property satisfied in the initial state)

Figure 7: Maintenance Mode

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



Property Details

Property:
A [G (error=>A [X (maintenance)])]

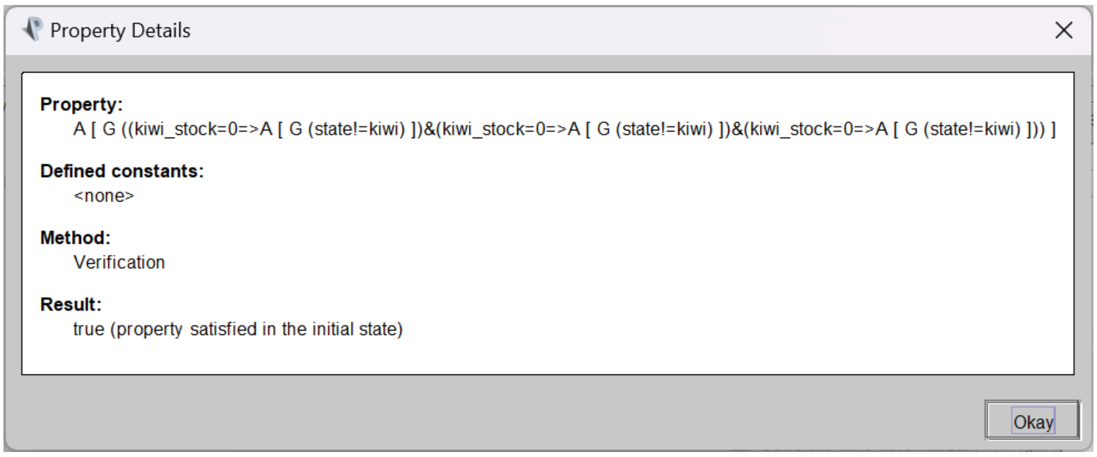
Defined constants:
<none>

Method:
Verification

Result:
true (property satisfied in the initial state)

Figure 8: Error Handling

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



Property:
A [G ((kiwi_stock=0=>A [G (state!=kiwi)])&(kiwi_stock=0=>A [G (state!=kiwi)])&(kiwi_stock=0=>A [G (state!=kiwi)]))]

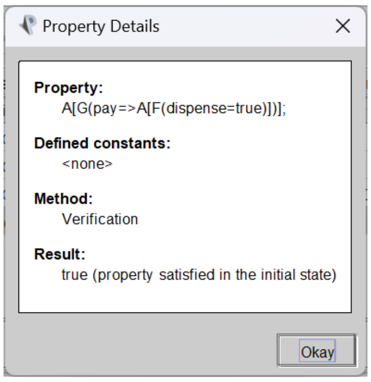
Defined constants:
<none>

Method:
Verification

Result:
true (property satisfied in the initial state)

Figure 9: Unavailable Drink Selection

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



Property:
A[G(pay=>A[F(dispense=true)])];

Defined constants:
<none>

Method:
Verification

Result:
true (property satisfied in the initial state)

Figure 10: Drink Dispensing