Coffee Vending Machine – Team 5

양한석
김은우
최동민
권기완

– Content –

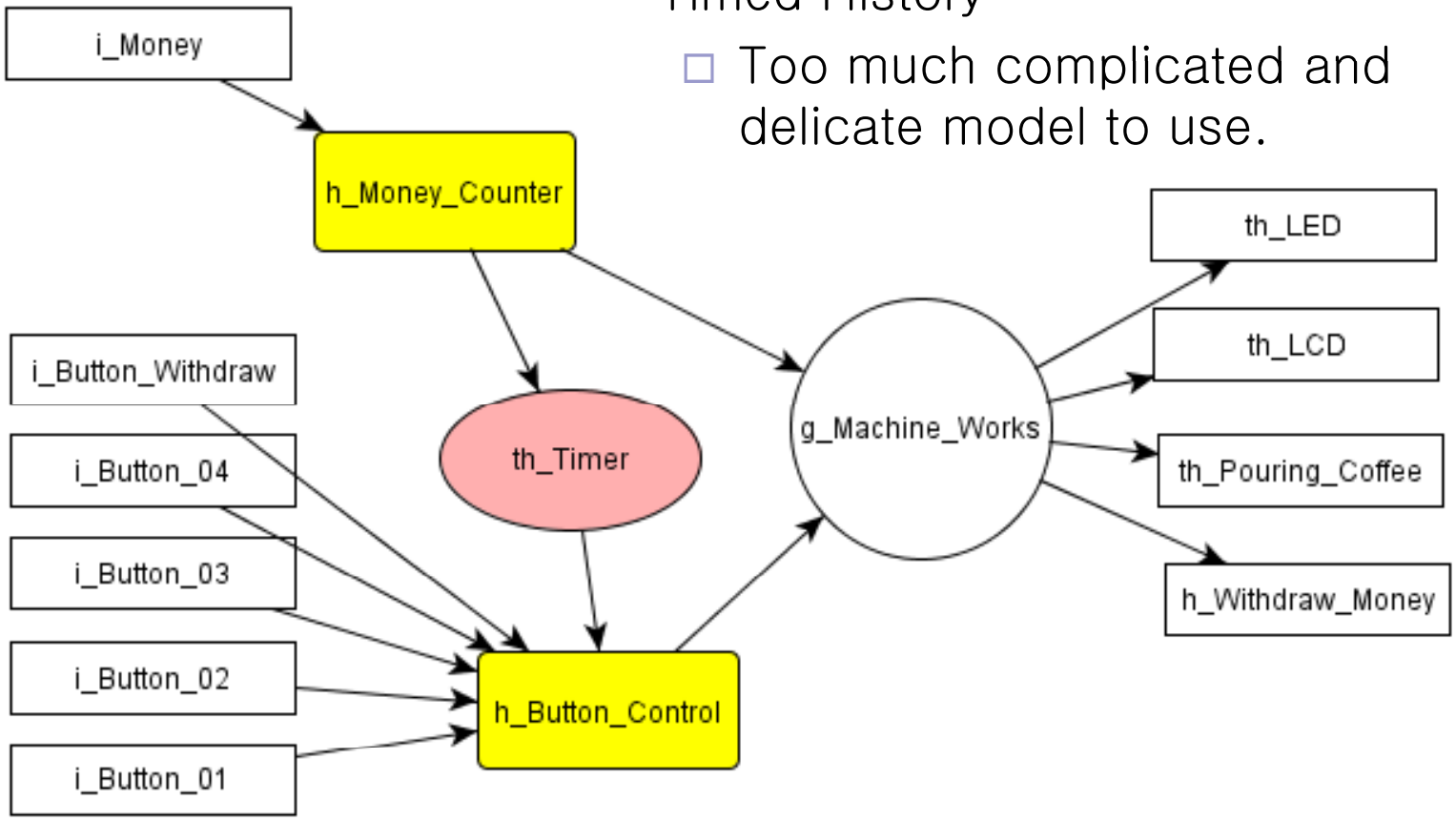
- Requirement
- Old Models
- Entire Logic
- Properties

Requirement

- Vending Machine take **max 1,000 won** and there is a assumption which the Overtaken money is automatically being rebated in the outside of this Model.
- If Money is insulted and Button is pushed at a time, **Button must be ignored.**
- There are 3 buttons and 2 buttons are for 2 kinds of Coffee. Another button is for Rebate. Here is a assumption these buttons' signals are serialized by outside of a model and **Only ONE button can be inputted at one time**.
- LEDs are turned on same time when Coffee is working.
- **Single step of job** in one cycle.

Old Models

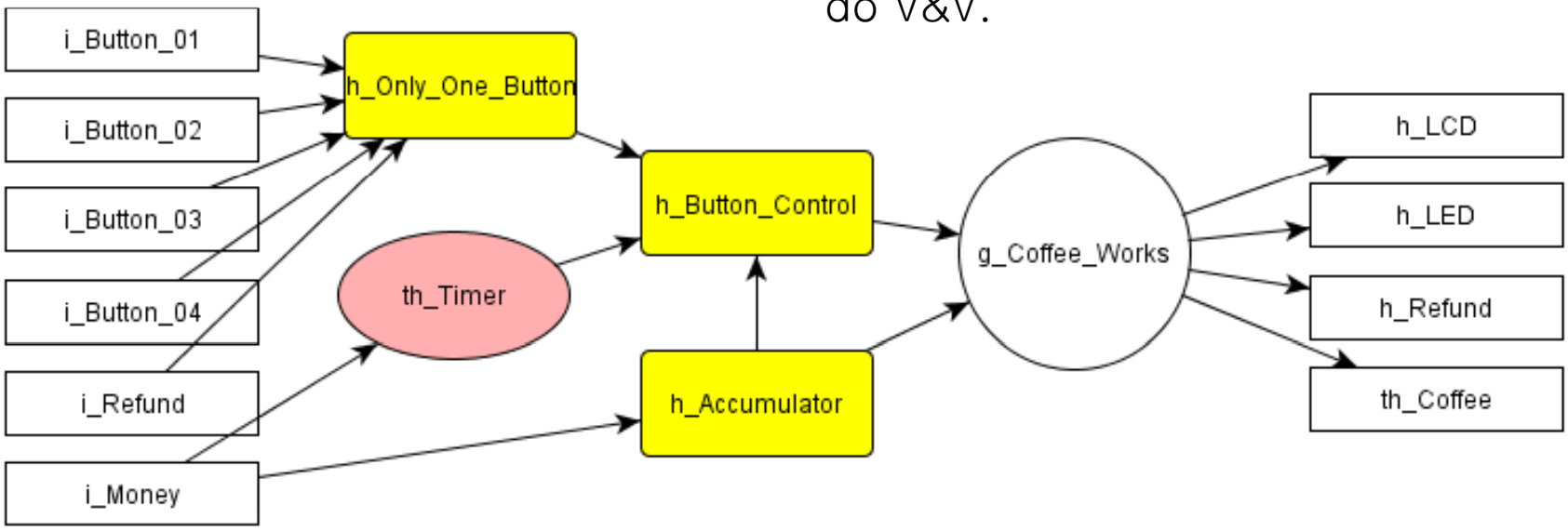
- V 0.1
- 5 Button Vending Machine with Timed History
  - Too much complicated and delicate model to use.



Old Models

- V 0.2

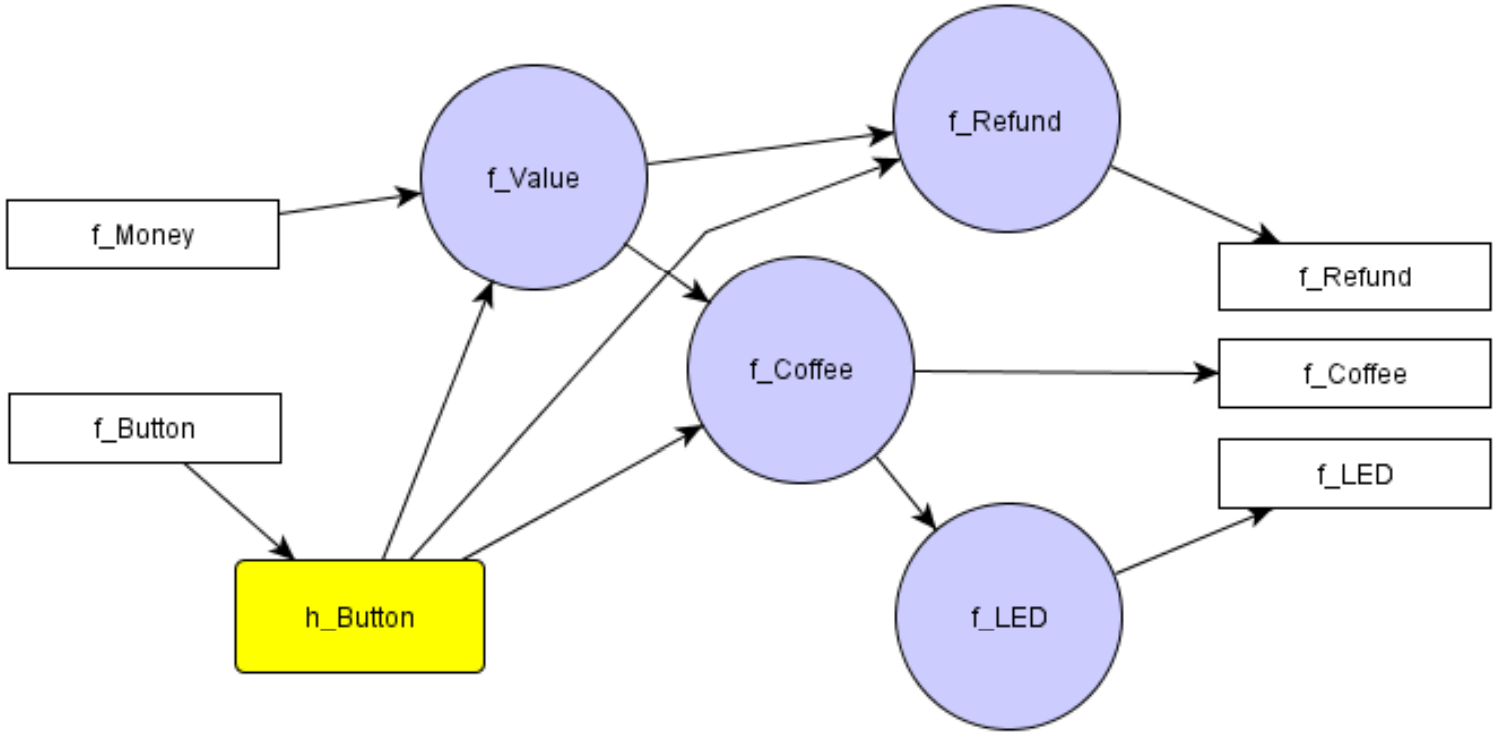
- Button Logic Enforced Model
  - Requirement is reduced and The node for Serializing Button signals is added
  - Too many Nodes in this model to do V&V.



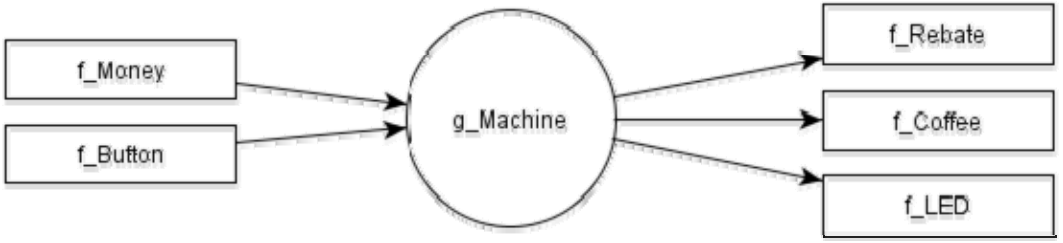
Old Models

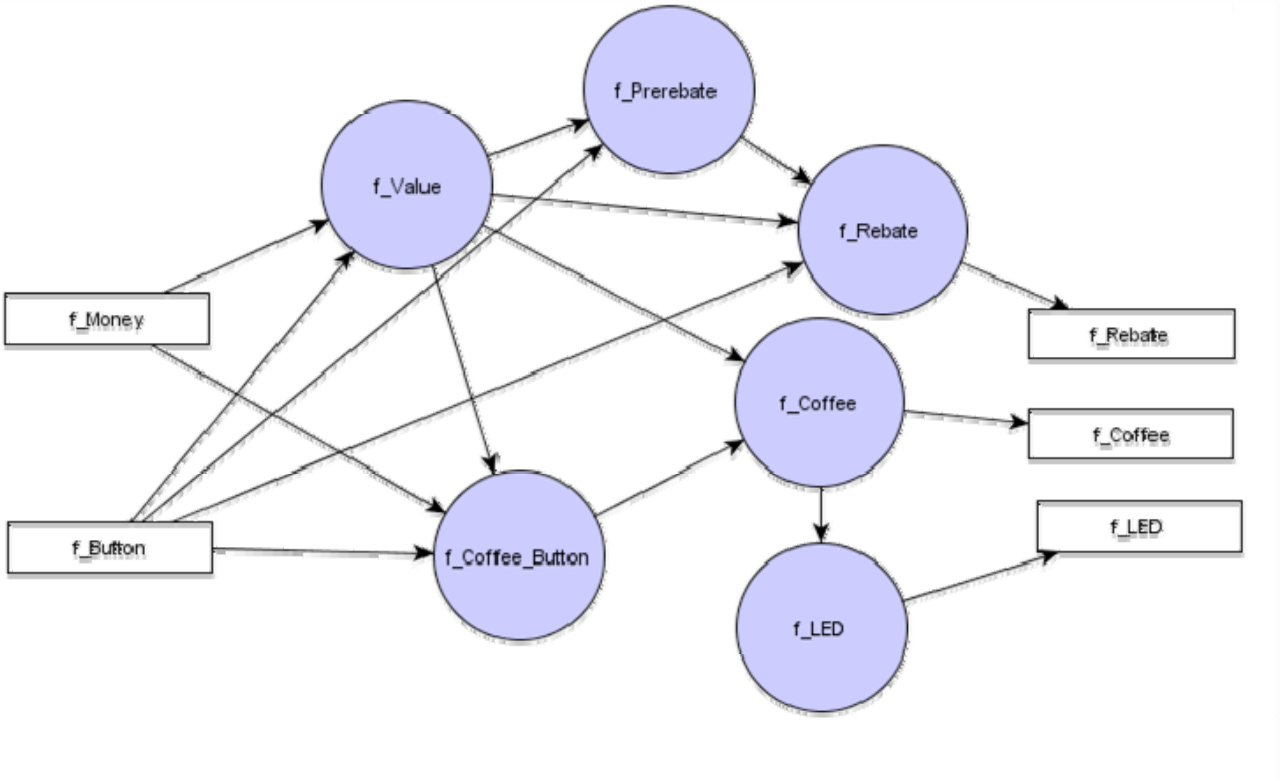
- V 1.0

- The Simple model by Reduced Requirement
  - Similar with current model, but still needs to fix, cause of improper f_Refund and f_Value.

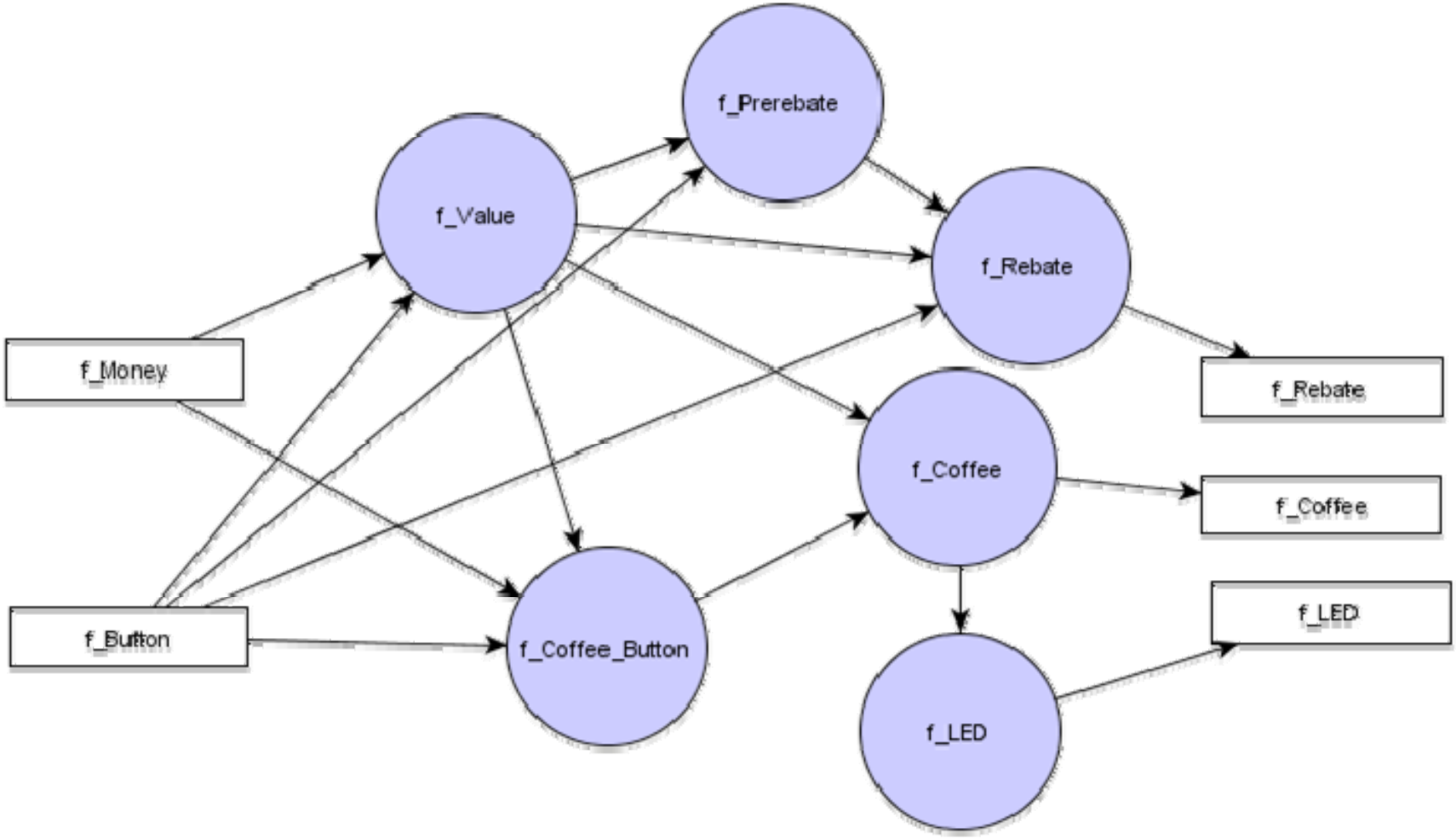


Entire Logic V 1.2

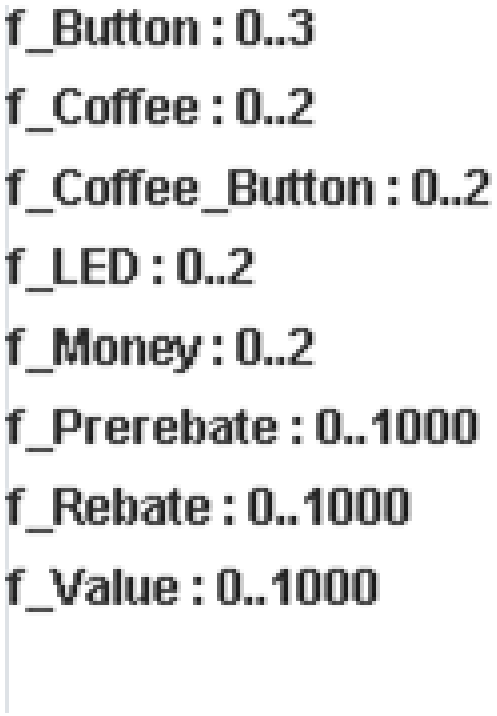




Entire Logic



Entire Logic



- f_Button : 3 Button signals. 1, 2 signals means each 50 won and 100won as Coffee Value. 3 signal is for Rebate. 0 for Non-input signal.
- f_Coffee : Actual Coffee Vending Part. 2 kinds of coffee.
- f_Coffee_Button : f_Value could be 0, even enough condition for vending coffee by logic. This is to effect recovering Value for f_Button and f_Coffee. Also ignoring Buttons when Money is insulted.

Entire Logic

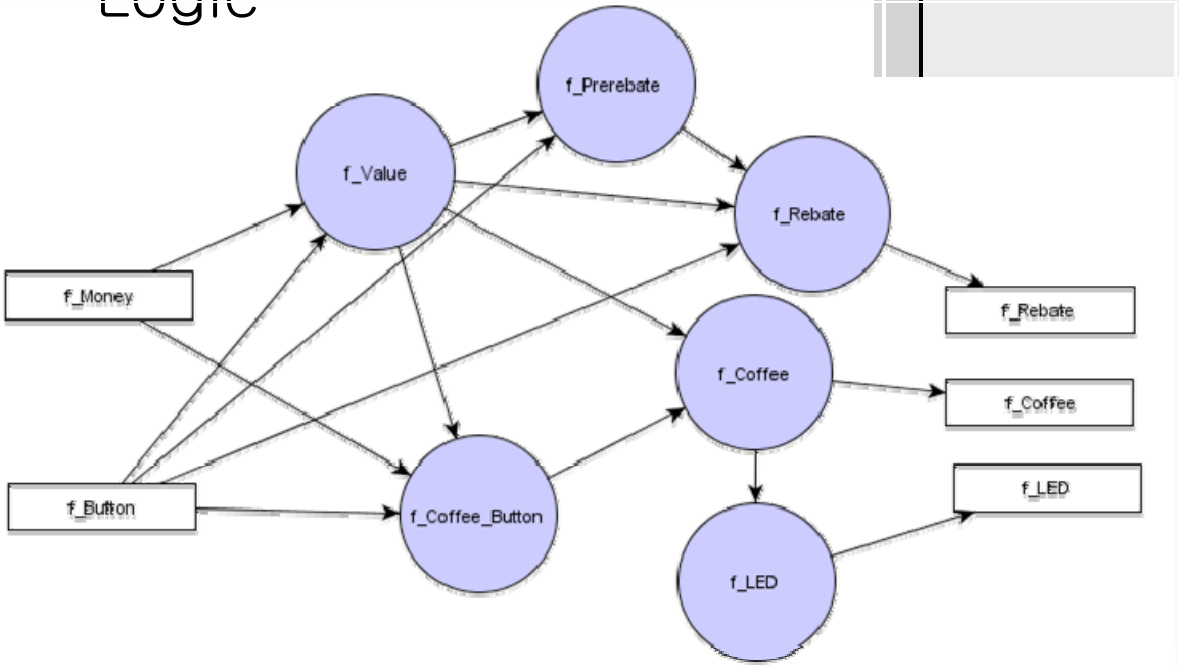
f_Button : 0..3
f_Coffee : 0..2
f_Coffee_Button : 0..2
f_LED : 0..2
f_Money : 0..2
f_Prerebate : 0..1000
f_Rebate : 0..1000
f_Value : 0..1000

- f_LED : these show the Coffee machin is working with proper value of Coffee.
- f_Money : 50 and 100 won is numbered from this node.
- f_Prerebate : only for Rebate function, this node save the f_Value before one cycle, just like f_Value_t0.
- f_Value : the most Important part of this model. Calculating Money and Buttons' signals.

Properties

- [Dead Freeness]
  - Result of the Entire Logic

Property	Result
(AG (EX 1))	true



Properties [Auto-Generated]

- [Completeness]
  - Such as Coverage Check Property in SMV
  - SPEC AG (in-_init_ -> AX ! in-_init_)

Property	Result
(AG (\in-_init_ ->(AX (~\in-_init_))))	true

Properties [Auto-Generated]

- [Consistency]
  - Such as Non-determinisim(disjointness) check properties
  - SPEC AG !(0 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s1-taken)

```
--Non-determinisim(disjointness) check properties
SPEC AG !( 0
 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s1-taken
 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s2-taken
 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s3-taken
 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s4-taken
 || FROM-_init_-TO-s0-taken & FROM-_init_-TO-s5-taken
 || FROM-_init_-TO-s1-taken & FROM-_init_-TO-s2-taken
 || FROM-_init_-TO-s1-taken & FROM-_init_-TO-s3-taken
 || FROM-_init_-TO-s1-taken & FROM-_init_-TO-s4-taken
 || FROM-_init_-TO-s1-taken & FROM-_init_-TO-s5-taken
 || FROM-_init_-TO-s2-taken & FROM-_init_-TO-s3-taken
 || FROM-_init_-TO-s2-taken & FROM-_init_-TO-s4-taken
 || FROM-_init_-TO-s2-taken & FROM-_init_-TO-s5-taken
 || FROM-_init_-TO-s3-taken & FROM-_init_-TO-s4-taken
 || FROM-_init_-TO-s3-taken & FROM-_init_-TO-s5-taken
 || FROM-_init_-TO-s4-taken & FROM-_init_-TO-s5-taken)
```

Property	Result
(AG (~((((((((((((((\FROM-_init_-TO-s0-taken &\FROM-_init_-TO-s1-take	true

<< from f_Value's auto-generated SMV >>

Properties [f_Value]

Conditions	1	2	3	4	5	6	7	8
f_Money=0 & f_Value_t0=0	T							F
f_Money=0 & f_Value_t0<0 & f_Value_t0>0		T						F
f_Money=1			T					F
f_Money=2				T				F
f_Money=0 & f_Button=1 & f_Value_t0>=50					T			F
f_Money=0 & f_Button=2 & f_Value_t0>=100						T		F
f_Money=0 & f_Button=3							T	F
Action	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
f_Value := 0	0						0	0
f_Value := f_Value_t0		0						
f_Value := f_Value_t0 + 50			0					
f_Value := f_Value_t0 + 100				0				
f_Value := f_Value_t0 - 50					0			
f_Value := f_Value_t0 - 100						0		

Properties [f_Value]

- When 100 won is inputted, 100 won should be saved.
- SPEC AG ((f_Money=2 & f_Button=0)->(AX f_Value >= 100))

Property	Result
(AG (~((((((((((((((((\FROM-_init_-TO-s0-taken &\FROM-_init_-TO-s1-take	true
(AG (\in-_init_ ->(AX (~\in-_init_))))	true
(AG (((f_Money=2)&(f_Button=0)) ->(AX (f_Value>=100))))	true

Properties [f_Value]

- Last value (f_Value_t0) has not enough range for this Model's assumption.
  Solve this problem by fix SMV code.

```
-- SMV Input for f_Value

MODULE m_f_Value(f_Money, f_Button, cycle, sec)
VAR
 f_Value : 0..1000;
 f_Value_t0 : 0..900;
 -- inputs
```

Properties [f_Coffee]

Conditions	1	2	3	4
f_Value >= 0 & f_Coffee_Button = 1	T	F	F	F
f_Value >= 0 & f_Coffee_Button = 2	F	T	F	F
f_Coffee_Button = 0	F	F	T	F
Action	**1**	**2**	**3**	**4**
f_Coffee := f_Coffee_Button	0	0		
f_Coffee := 0			0	0

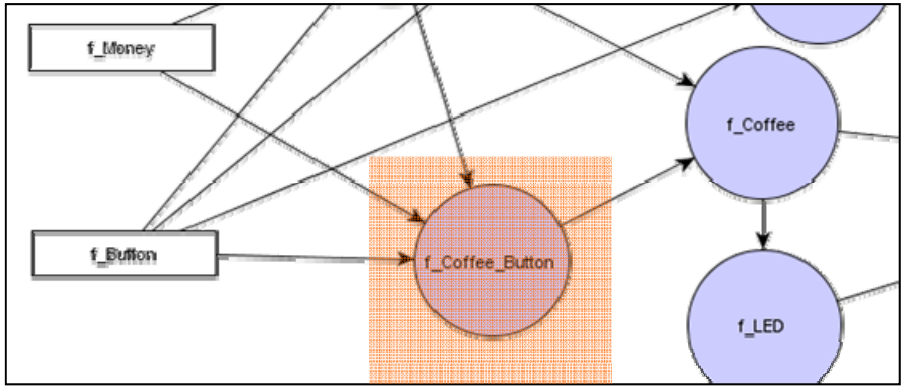
Properties [f_Coffee]

- There is no Coffee with no money
- SPEC AG ((f_Money =0 & f_Value =0 & f_Button =1)->(AX f_Coffee = 0))

Property	Result
(AG ((((f_Money.f_Money=0)&(((f_Value.f_Value=0)&(f_Value.f_Value_t0=0	true

Properties [f_Coffee]

- **f_Value** could calculate money to zero in case. **f_Coffee_Button** is added to control button input for effecting Recovering Value.



Conditions	1	2	3	4
!f_Button=0 & f_Value = 0 & f_Money = 0	T			F
!f_Button=0 & !f_Money=0		T		F
f_Button=0 & !f_Money=0			T	F
Action	**1**	**2**	**3**	**4**
f_Coffee_Button := 0	0	0	0	
f_Coffee_Button := f_Button				0

Properties [f_Coffee]

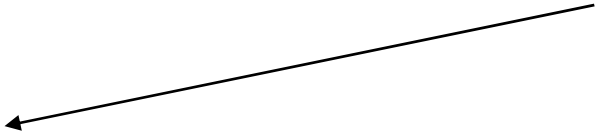
- There is Coffee with enough money
- SPEC AG ((f_Value =100 & f_Button =1)->(AX f_Coffee = 1))

Property	Result
(AG (((((f_Value.f_Value=0)&(f_Value.f_Value_t0=0))&(f_Value.STATE=0))	true

- Money and Buttons can not be input in this model, had to try two steps.
- Also, the error message about range condition
  - vector expression appears where scalar expression is required.

Properties [f_Coffee]

- There is Coffee with enough money
- Step 1
  - SPEC AG (f_Money=1 & f_Button=2) -> (EX f.Value.f_Value = 50)
- Step 2
  - SPEC AG (f_Value.f_Value=50 & f_Button.f_Button=2) -> (AX f_Coffee = 2)



Property	Result
(AG ((f_Money.f_Money=1)&(f_Button.f_Button=2))) ->(EX (f_Value.f_Val	true
(AG ((f_Value.f_Value=50)&(f_Button.f_Button=2))) ->(AX ((f_Coffee.f_	true

Properties [f_Coffee_Button]

- If Money and Button signals are input at one time, should ignore the Button.
- f_Coffee_Button
  - SPEC AG ((f_Money=1 & f_Button=2)->(AX f_Coffee_Button=0))

Property	Result
(AG (~(\FROM-_init_-TO-s0-taken &\FROM-_init_-TO-s1-taken)))	true
(AG (\in-_init_ ->(AX (~\in-_init_))))	true
(AG (((f_Money=1)&(f_Button=2)) ->(AX (f_Coffee_Button=0))))	true

Properties [f_Coffee_Button]

- If Money and Button signals are input at one time, should ignore the Button.
- Entire Logic
  - SPEC AG ((f_Money.f_Money=1 & f_Button.f_Button=2)->(AX f_Coffee_Button.f_Coffee_Button=0))

Property	Result
(AG (((f_Money.f_Money=1)&(f_Button.f_Button=2)) ->(AX (f_Coffee_Butto	true

- Without dot(.), Same Property with Opposite Result.

Properties [Ref:f_Coffee]

Property	Result
((AG ((((f_Value.f_Value=0)&(f_Value.f_Value_t0=0))&(f_Value.STATE=1))	true

- without dot
  - SPEC AG (f_Value=50 & f_Button=2) -> (AX f_Coffee = 2)

Property	Result
(AG ((f_Money.f_Money=1)&(f_Button.f_Button=2))) ->(EX (f_Value.f_Val	true
(AG ((f_Value.f_Value=50)&(f_Button.f_Button=2))) ->(AX ((f_Coffee.f_	true

- with dot
  - SPEC AG (f_Value.f_Value=50 & f_Button.f_Button=2) -> (AX f_Coffee = 2)

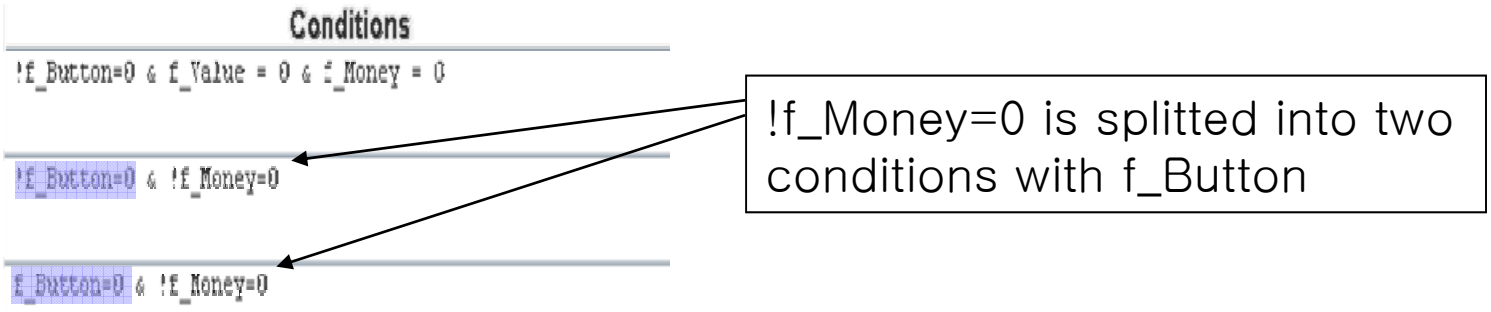
Properties [f_Coffee_Button]

- Leaking Condition
  - This Condition should not show the result “true”.
  - Basic Idea of this Condition is that f_Button and f_Money could not be together.

SPEC AG ((f_Value=50 & f_Button=2)->(AX f_Coffee_Button=0))

Property	Result
(AG (((((f_Value.f_Value=0)&(f_Value.f_Value_t0=0))&(f_Value.STATE=1))	true

Properties [f_Coffee_Button]



Property	Result
(AG (((f_Value=50)&(f_Button=2)) ->(AX (f_Coffee_Button=0))))	false

Property	Result
(AG (((f_Value=50)&(f_Button=2)) ->(EF (f_Coffee_Button=0))))	true

Property	Result
(AG (((f_Value=50)&(f_Button=2)) ->(EG (f_Coffee_Button=0))))	false

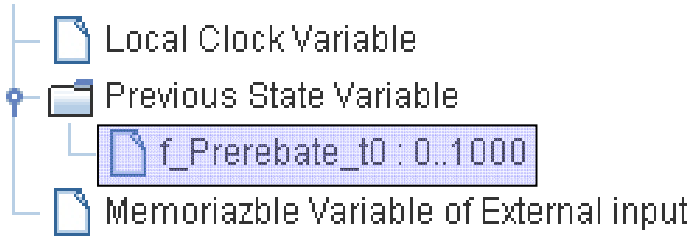
Properties [f_Rebate]

- If Rebate button is pushed, rebate proper money.
- SPEC AG ((f_Value=100 & f_Button=3)->(AX f_Prerebate=100))

Property	Result
(AG (~(\FROM-_init_-TO-s0-taken &\FROM-_init_-TO-s1-taken)))	true
(AG (\in-_init_ ->(AX (~\in-_init_))))	true
(AG (((f_Value=100)&(f_Button=3)) ->(AX (f_Prerebate=100))))	true

Properties [f_Prerebate]

- Using as f_Valut_t0.



Conditions	1	2	3
f_Button>3 & f_Button<3 & f_Value > 0	T		F
f_Value = 0 & f_Button = 3		T	F
Action	**1**	**2**	**3**
f_Prerebate := f_Value	0		0
f_Prerebate := f_Prerebate_t0		0	

Properties [f_LED]

- Working with f_Coffee
- SPEC AG ((f_Value=100 & f_Button=1)->(AX f_Coffee = f_LED))

Property	Result
(AG (((((f_Value.f_Value=0)&(f_Value.f_Value_t0=0))&(f_Value.STATE=0))	true

- The End
  - Q&A