Author: Chanpech Hoeng
Course: CS4710
Date: Dec 8 2023

Online Payment System

Functions:
Define at least four functions that are reused in your facts/assertions.
1.

```
// Function to see the funds that were spent
fun fundsSpentWhen[a: Account, f: Funds, s: States]: set Funds { }

// Function to see that a receiver account should have a receiving relationship with a sender
account
fun receive1EqualSender2[a: Account]: set Account { }

// Function to see all accounts that were able to successfully send funds
fun successfulSenders[fi: FinancialInstitution, s: States]: set Account { }

// Function to see all accounts that had their funds processed by the financial institution
fun processedAccounts[fi: FinancialInstitution, s: States]: set Account { }
```

Assertions:

a. Specify at least four assertions that you expect to be true.

- //assert noExcessFunds { }
- // Assert that all accounts in processed state are also in successfulSenders
  assert processedAccountsInSuccessfulSenders {}
- // Assert that all accounts that have sent funds are also in successfulSenders
  assert sendersInSuccessfulSenders { }
- // Assert that all successfully received accounts are also in receive1EqualSender2

b. Did assertions fail? How did you fix it?
Yes all of them initially fail. Some the ways in which I fix them are:

- Made no excess funds into fact. This makes much more sense because at no time should there be excess funds. Thus it makes complete to turn this into a global fact.

c. Argue why you should not specify your assertions as facts.

Some reasons why I left out some of the assertions not as fact is because the constraints will render the model to not generate any instances.

Predicates

A. Define at least four predicates.

//If there are accounts that share the same funds then one of the accounts must have initiated send.
pred sameFundsImpliesSendReceive[a1, a2: Account] {}
//An account can send but it does not have an incoming receive from other account
pred sendWithoutReceive[a1, a2: Account] { }
// Capture how funds are initially moved from sender to financialInst
pred moveFunds[a: Account, f: Funds, fi: FinancialInstitution] { }
// Capture how funds are initially placed under processing in financial institution
pred processFunds[a: Account, f: Funds, fi: FinancialInstitution] { }

B. Dynamic Predicates: Specify at least four predicates that capture dynamic changes in the structure of your alloy model.

// Capture how funds are initially moved from sender to financial institution
pred moveFunds[a: Account, f: Funds, fi: FinancialInstitution] {}
// Capture how funds are initially placed under processing in financial institution
pred processFunds[a: Account, f: Funds, fi: FinancialInstitution] { }
// Capture how funds are transferred from financial institution to destination
pred transferFunds[fi: FinancialInstitution, s: States] {
}
// Capture how receiver sends a confirmation message back to the account
pred confirmFunds[a: Account, f: Funds, s: States] { }

Scope-based Model Checking

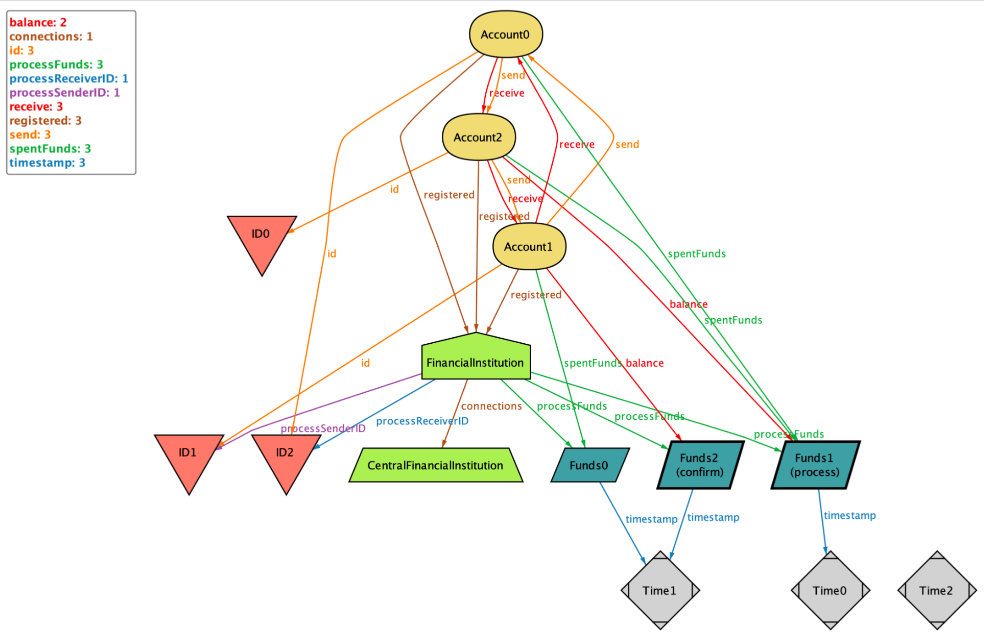
a. Increase your scope incrementally for all the assertions and predicates you have specified in Items 2 and 3 until it takes more than 5 minutes to check some assertion/predicate. (20 points)

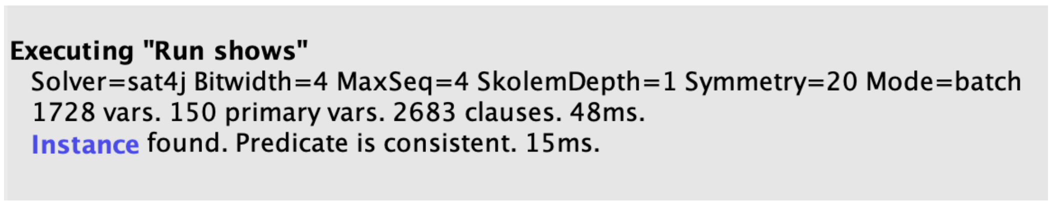
1. 1st Increment

In the first increment, I have added the util of ordering to dynamically help capture the state of the transaction process. However, I haven’t fully implemented a function or facts to enforce the relationship that should occur. By the end of the implementation, the model should dynamically capture the following:

1. The state should capture the account sending its funds to the financial institution with the designated recipient.
2. The state should capture the financial institution that receives the request and place this transaction as processing/pending.
3. The state should capture the financial institution transferring the funds to the designated recipient.
4. State should capture the recipient sending a confirmation message that it has received.

Initially the model was able to capture about 1728 at 15ms.





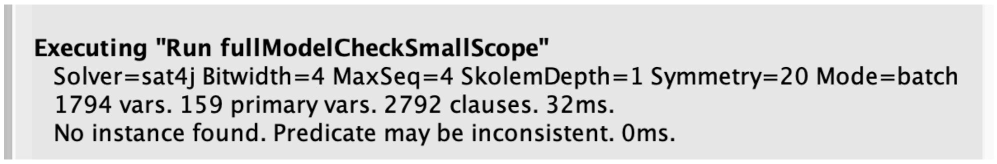
```
Executing "Run shows"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
  1728 vars. 150 primary vars. 2683 clauses. 48ms.
  Instance found. Predicate is consistent. 15ms.
```

I didn’t understand the scaling well enough to get my model’s assertion and predicates to scales past 5 minutes.

b. Is there a case where your model holds for small scopes but fails for larger ones? (20 points)

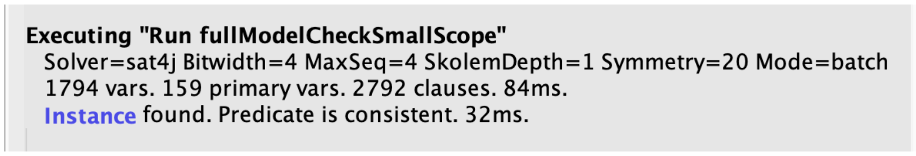
Yes, there was one instance where the run works for only small scopes but not for longer scopes as in no instances were generated.

If you increase the number of financial institutions in the run to be greater than 2. The run would no longer work.



```
Executing "Run fullModelCheckSmallScope"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
  1794 vars. 159 primary vars. 2792 clauses. 32ms.
  No instance found. Predicate may be inconsistent. 0ms.
```

However, for smaller scopes such as below 3, it was able to generate some instances.



```
Executing "Run fullModelCheckSmallScope"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
  1794 vars. 159 primary vars. 2792 clauses. 84ms.
  Instance found. Predicate is consistent. 32ms.
```

Tables that were generated:

this/FinancialInstitution	connections	processSenderID	processReceiverID	processFunds
FinancialInstitution^0	CentralFinancialInstitution^0	ID^1	ID^2	Funds^0
				Funds^1
				Funds^2
FinancialInstitution^1	CentralFinancialInstitution^0	ID^0	ID^1	

this/Funds	timestamp
Funds^0	time/Time^2
Funds^1	time/Time^1
Funds^2	time/Time^2

this/Account	id	balance	send	receive	registered	spentFunds
Account^0	ID^2	Funds^1	Account^2	Account^2	FinancialInstitution^1	Funds^1
		Funds^2				
Account^1	ID^1	Funds^0			FinancialInstitution^0	Funds^0
		Funds^2				
Account^2	ID^0	Funds^0		Account^0	FinancialInstitution^0	Funds^1
		Funds^1				