```
──────────────────────── MODULE dao2 ────────────────────────
EXTENDS TLC, Integers, Sequences
CONSTANT BALANCE, AMOUNT

 --algorithm Doa_Attack{
variable attack = 3,
  bankBalance = BALANCE,
  malloryBalance = 0 ;

define {
  SafeWithdrawal ≜
      ∨ bankBalance = BALANCE ∧ malloryBalance = 0
      ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = 0
      ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT
  Invariant ≜ bankBalance + malloryBalance ≤ BALANCE
  EndState ≜ ◇(bankBalance + malloryBalance = BALANCE
                    ∧ bankBalance ≤ BALANCE − AMOUNT) }

procedure BankWithdraw( amount ) {
  CheckBalance:  check if Mallory has sufficient balance
    if ( bankBalance < amount ) return ;
  UpdateBalance:  update Mallory's bankBalance
    bankBalance := bankBalance − amount ;
  DispenseAmount:  dispense Mallory the amount
    call MallorySendMoney(amount) ;
  return ;  }

procedure MallorySendMoney( amount ) {
  Receive:
    malloryBalance := malloryBalance + amount ;
    if ( attack > 0 ) {
      attack := attack − 1 ;  avoid infinite stack; don't run out of gas
      call BankWithdraw(amount) ;  } ;  cheating! doublecalling withdraw
FC: return ;  }

fair process ( blockchain = "blockchain" ) {
  Transact:  Mallory calls Bank to withdraw AMOUNT from her bankBalance
    call BankWithdraw(AMOUNT) ;  }

 }
 BEGIN TRANSLATION
 Parameter amount of procedure BankWithdraw at line 19 col 24 changed to amount_
CONSTANT defaultInitValue
VARIABLES attack, bankBalance, malloryBalance, pc, stack

 define statement
SafeWithdrawal ≜
```

```
      ∨ bankBalance = BALANCE ∧ malloryBalance = 0
      ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = 0
      ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT
Invariant ≜ bankBalance + malloryBalance ≤ BALANCE
EndState ≜
     ◇(bankBalance ≤ BALANCE − AMOUNT ∧ bankBalance + malloryBalance = BALANCE)

VARIABLES amount_, amount

vars ≜ ⟨attack, bankBalance, malloryBalance, pc, stack, amount_, amount⟩

ProcSet ≜ {"blockchain"}

Init ≜  Global variables
          ∧ attack = 3
          ∧ bankBalance = BALANCE
          ∧ malloryBalance = 0
           Procedure BankWithdraw
          ∧ amount_ = [self ∈ ProcSet ↦ defaultInitValue]
           Procedure MallorySendMoney
          ∧ amount = [self ∈ ProcSet ↦ defaultInitValue]
          ∧ stack = [self ∈ ProcSet ↦ ⟨⟩]
          ∧ pc = [self ∈ ProcSet ↦ "Transact"]

CheckBalance(self) ≜ ∧ pc[self] = "CheckBalance"
                     ∧ IF bankBalance < amount_[self]
                           THEN ∧ pc' = [pc EXCEPT ![self] = Head(stack[self]).pc]
                                ∧ amount_' = [amount_ EXCEPT ![self] = Head(stack[self]).amount_]
                                ∧ stack' = [stack EXCEPT ![self] = Tail(stack[self])]
                           ELSE ∧ pc' = [pc EXCEPT ![self] = "UpdateBalance"]
                                ∧ UNCHANGED ⟨stack, amount_⟩
                     ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance,
                                  amount⟩

UpdateBalance(self) ≜ ∧ pc[self] = "UpdateBalance"
                      ∧ bankBalance' = bankBalance − amount_[self]
                      ∧ pc' = [pc EXCEPT ![self] = "DispenseAmount"]
                      ∧ UNCHANGED ⟨attack, malloryBalance, stack, amount_,
                                   amount⟩

DispenseAmount(self) ≜ ∧ pc[self] = "DispenseAmount"
                       ∧ ∧ amount' = [amount EXCEPT ![self] = amount_[self]]
                         ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦  "MallorySendMoney",
                                                              pc        ↦ Head(stack[self]).pc,
                                                              amount    ↦  amount[self]]⟩
                                                              ∘ Tail(stack[self])]
                       ∧ pc' = [pc EXCEPT ![self] = "Receive"]
```

```
                              ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance,
                                           amount_⟩

BankWithdraw(self) ≜ CheckBalance(self) ∨ UpdateBalance(self)
                        ∨ DispenseAmount(self)

Receive(self) ≜ ∧ pc[self] = "Receive"
                ∧ malloryBalance' = malloryBalance + amount[self]
                ∧ IF attack > 0
                     THEN ∧ attack' = attack − 1
                          ∧ ∧ amount_' = [amount_ EXCEPT ![self] = amount[self]]
                            ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦  "BankWithdraw",
                                                                 pc        ↦ "FC",
                                                                 amount_   ↦  amount_[self]]⟩
                                                                 ∘ stack[self]]
                          ∧ pc' = [pc EXCEPT ![self] = "CheckBalance"]
                     ELSE ∧ pc' = [pc EXCEPT ![self] = "FC"]
                          ∧ UNCHANGED ⟨attack, stack, amount_⟩
                ∧ UNCHANGED ⟨bankBalance, amount⟩

FC(self) ≜ ∧ pc[self] = "FC"
           ∧ pc' = [pc EXCEPT ![self] = Head(stack[self]).pc]
           ∧ amount' = [amount EXCEPT ![self] = Head(stack[self]).amount]
           ∧ stack' = [stack EXCEPT ![self] = Tail(stack[self])]
           ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance, amount_⟩

MallorySendMoney(self) ≜ Receive(self) ∨ FC(self)

Transact ≜ ∧ pc["blockchain"] = "Transact"
           ∧ ∧ amount_' = [amount_ EXCEPT !["blockchain"] = AMOUNT]
             ∧ stack' = [stack EXCEPT !["blockchain"] = ⟨[procedure ↦  "BankWithdraw",
                                                        pc        ↦ "Done",
                                                        amount_   ↦  amount_["blockchain"]]⟩
                                                        ∘ stack["blockchain"]]
           ∧ pc' = [pc EXCEPT !["blockchain"] = "CheckBalance"]
           ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance, amount⟩

blockchain ≜ Transact

Next ≜ blockchain
          ∨ (∃ self ∈ ProcSet :  ∨ BankWithdraw(self)
                                 ∨ MallorySendMoney(self))
          ∨  Disjunct to prevent deadlock on termination
             ((∀ self ∈ ProcSet : pc[self] = "Done") ∧ UNCHANGED vars)

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ ∧ WF_vars(blockchain)
```

$\land \mathrm{WF}_{vars}(BankWithdraw(\text{"blockchain"}))$
$\land \mathrm{WF}_{vars}(MallorySendMoney(\text{"blockchain"}))$

$Termination \triangleq \Diamond(\forall self \in ProcSet : pc[self] = \text{"Done"})$

END TRANSLATION

$SafeWithdrawal1 \triangleq$
$\lor accountAlice = BALANCE \land accountBob = 0$
$\lor accountAlice = BALANCE - AMOUNT \land accountBob = AMOUNT$

This was too restrictive, because updating of both *Alice*'s and Bob's accounts do not happen atomically.

$\land accountAlice = 12$
$\land accountBob = 5$
$\land accountTotal = 12$
$\land amount = [blockchain \mapsto 5]$
$\land amount\_ = [blockchain \mapsto 5]$
$\land pc = [blockchain \mapsto \text{"CheckBalance"}]$
$\land stack = [blockchain \mapsto \langle[pc \mapsto \text{"FinishAlice2"}, amount\_ \mapsto 5, procedure \mapsto \text{"withdrawFromAlice"}], [pc \mapsto \text{"Done"}, amount\_ \mapsto defaultInitValue, procedure \mapsto \text{"withdrawFromAlice"}]\rangle]$

after Bob got money, but before it was subtracted from *Alice*'s account, the *SafeWithdrawal*1 broke. So I need to relax this.

Yes, TLA found the double-spending!!!

$\land accountAlice = 12$
$\land accountBob = 10$
$\land accountTotal = 12$
$\land amount = [blockchain \mapsto 5]$
$\land amount\_ = [blockchain \mapsto 5]$
$\land pc = [blockchain \mapsto \text{"CheckBalance"}]$
$\land stack = [blockchain \mapsto \langle[pc \mapsto \text{"FinishAlice2"}, amount\_ \mapsto 5, procedure \mapsto \text{"withdrawFromAlice"}], [pc \mapsto \text{"FinishAlice2"}, amount\_ \mapsto 5, procedure \mapsto \text{"withdrawFromAlice"}], [pc \mapsto \text{"Done"}, amount\_ \mapsto defaultInitValue, procedure \mapsto \text{"withdrawFromAlice"}]\rangle]$

Bob's account got 10! Double withdrawal. Even if I make *Alice*'s account subtraction line 25 come before *sendMoney*, I would have the same double withdrawal problem!

* assert $accountAlice \geq BALANCE - AMOUNT$;

```
function withdraw(uint amount){
client = msg.sender;
if(balance[client] ≥ amount}{
if(client.call.sendMoney(amount)){balance[client] = amount;
}}}
```

```
function sendMoney(unit amount){
victim = msg.sender;
balance + = amount;
victim.withdraw(amount)
}
```