```
MODULE dao
EXTENDS TLC, Integers, Sequences
CONSTANT BALANCE, AMOUNT

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

define {
  SafeWithdrawal ≜
      ∨ bankBalance = BALANCE ∧ malloryBalance = 0
      ∨ bankBalance = BALANCE ∧ malloryBalance = AMOUNT
      ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT }

procedure BankWithdraw( amount ) {
  CheckBalance:  check if Mallory has sufficient balance
    if ( bankBalance < amount ) return;
  DispenseAmount:  dispense Mallory the amount
    call MallorySendMoney(amount);
  FinishWithdraw:  update Mallory's bankBalance
    bankBalance := bankBalance − 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 16 col 24 changed to amount_
CONSTANT defaultInitValue
VARIABLES attack, bankBalance, malloryBalance, pc, stack

 define statement
SafeWithdrawal ≜
    ∨ bankBalance = BALANCE ∧ malloryBalance = 0
    ∨ bankBalance = BALANCE ∧ malloryBalance = AMOUNT
    ∨ bankBalance = BALANCE − AMOUNT ∧ malloryBalance = AMOUNT
```

```
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] = "DispenseAmount"]
                              ∧ UNCHANGED ⟨stack, amount_⟩
                     ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance,
                                  amount⟩

DispenseAmount(self) ≜ ∧ pc[self] = "DispenseAmount"
                       ∧ ∧ amount' = [amount EXCEPT ![self] = amount_[self]]
                         ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦ "MallorySendMoney",
                                                              pc ↦ "FinishWithdraw",
                                                              amount ↦ amount[self]]⟩
                                                            ∘ stack[self]]]
                       ∧ pc' = [pc EXCEPT ![self] = "Receive"]
                       ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance,
                                    amount_⟩

FinishWithdraw(self) ≜ ∧ pc[self] = "FinishWithdraw"
                       ∧ bankBalance' = bankBalance − amount_[self]
                       ∧ 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, malloryBalance, amount⟩

BankWithdraw(self) ≜ CheckBalance(self) ∨ DispenseAmount(self)
                     ∨ FinishWithdraw(self)
```

```
96  Receive(self) ≜  ∧ pc[self] = "Receive"
97                   ∧ malloryBalance' = malloryBalance + amount[self]
98                   ∧ IF attack > 0
99                         THEN ∧ attack' = attack − 1
100                              ∧ ∧ amount_' = [amount_ EXCEPT ![self] = amount[self]]
101                                ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦ "BankWithdraw",
102                                                                    pc        ↦ "FC",
103                                                                    amount_   ↦ amount_[self]]⟩
104                                                                  ∘ stack[self]]
105                              ∧ pc' = [pc EXCEPT ![self] = "CheckBalance"]
106                         ELSE ∧ pc' = [pc EXCEPT ![self] = "FC"]
107                              ∧ UNCHANGED ⟨attack, stack, amount_⟩
108                   ∧ UNCHANGED ⟨bankBalance, amount⟩

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

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

118 Transact ≜  ∧ pc["blockchain"] = "Transact"
119             ∧ ∧ amount_' = [amount_ EXCEPT !["blockchain"] = AMOUNT]
120               ∧ stack' = [stack EXCEPT !["blockchain"] = ⟨[procedure ↦ "BankWithdraw",
121                                                            pc        ↦ "Done",
122                                                            amount_   ↦ amount_["blockchain"]]⟩
123                                                          ∘ stack["blockchain"]]
124             ∧ pc' = [pc EXCEPT !["blockchain"] = "CheckBalance"]
125             ∧ UNCHANGED ⟨attack, bankBalance, malloryBalance, amount⟩

127 blockchain ≜ Transact

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

135 Spec ≜  ∧ Init ∧ □[Next]_vars
136        ∧ ∧ WF_vars(blockchain)
137          ∧ WF_vars(BankWithdraw("blockchain"))
138          ∧ WF_vars(MallorySendMoney("blockchain"))

140 Termination ≜ ◇(∀ self ∈ ProcSet : pc[self] = "Done")

142 END TRANSLATION
```

144

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

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

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

after Bob got money, but before it was subtracted from *Alice*'s account, the $SafeWithdrawal1$ broke. So I need to relax this.

Yes, TLA found the double-spending!!!
$\land\ accountAlice = 12$
$\land\ accountBob = 10$
$\land\ accountTotal = 12$
$\land\ amount = [blockchain \mapsto 5]$
$\land\ amount\_ = [blockchain \mapsto 5]$
$\land\ pc = [blockchain \mapsto$ "CheckBalance"$]$
$\land\ stack = [blockchain \mapsto \langle[pc \mapsto$ "FinishAlice2", $amount\_ \mapsto 5$, $procedure \mapsto$ "withdrawFromAlice"$], [pc \mapsto$ "FinishAlice2", $amount\_ \mapsto 5$, $procedure \mapsto$ "withdrawFromAlice"$], [pc \mapsto$ "Done", $amount\_ \mapsto defaultInitValue$, $procedure \mapsto$ "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 $withdraw(uint\ amount)\{$
$client = msg.sender;$
$if(balance[client] \geq amount)\{$
$if(client.call.sendMoney(amount))\{balance[client] = amount;$
$\}\}\}$

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