System Modelling and Design
A Simple ATM
Beyond Specification

Revision: 1.2, May 16, 2006

Ken Robinson

May 17, 2006

mailto::k.robinson@unsw.edu.au

Contents

1 Objectives of this Lecture 2

2 A Simplistic Model of an ATM 2
   2.1 ATM0 ......................................................... 3
   2.2 Improving the Model ....................................... 5
   2.3 Bank Context machine ...................................... 5
   2.4 Password machine .......................................... 6
   2.5 ATM1 ......................................................... 6
   2.6 ATM ......................................................... 8
   2.7 Bank ......................................................... 9
   2.8 Login ......................................................... 10
   2.9 A Note on structure ........................................ 10

3 Refinement 10
   3.1 Implementation ............................................ 11
   3.2 Implementing the ATM machine ........................... 11
   3.3 ATMI ......................................................... 11
   3.4 ATMBankSys ................................................. 13
   3.5 ATMBankSysI ............................................... 13
   3.6 Password Encryption ........................................ 15
   3.7 Encryption ................................................... 15
   3.8 BankR ......................................................... 16
1 Objectives of this Lecture

- to introduce the concepts of refinement and implementation;
- to introduce detail through design (refinement & implementations);
- to illustrate model decomposition using multiple machines;
- to illustrate the use —now seen for the first time— of the results returned by operations;
- to illustrate all the above using a simple ATM example.

2 A Simplistic Model of an ATM

We want to produce a model of an ATM. The model will be kept reasonably simple, but also reasonably realistic.

Required ATM operations:

- an operation to insert the card and provide a password;
- an operation to withdraw money;
- an operation to deposit money;
- an operation to get the account balance.

Additionally, we will need an operation, that is really a bank operation, to open an account so that the above operations can be executed.

The initial attempt might be as shown in the ATM0 machine. This is likely to be the type of specification produced by someone familiar only with machine level development.
2.1 ATM0

A simplistic and incorrect model of an ATM

MACHINE ATM0
SETS
ACCOUNT ;
PASSWORD ;
RESPONSE = \{ OK , REFUSED \}

VARIABLES
accounts ,
password ,
balance ,
loggedin
INARIANT
accounts ⊆ ACCOUNT ∧
password ∈ accounts → PASSWORD ∧
balance ∈ accounts → ℤ ∧
loggedin ∈ seq ( accounts ) ∧ size ( loggedin ) ≤ 1
INITIALISATION
accounts , password := \{ \} , \{ \} ∥
balance , loggedin := \{ \} , []

OPERATIONS

response ←− InsertCard ( account , pass ) \equiv
PRE account ∈ ACCOUNT ∧ pass ∈ PASSWORD
THEN IF loggedin = [] ∧
account ∈ accounts ∧
pass = password ( account )
THEN response := OK ∥
loggedin := [ account ]
ELSE response := REFUSED
END
END ;

copyfile response _ _ WithDraw ( amount ) \equiv
PRE amount ∈ ℤ THEN
IF loggedin ≠ [] THEN
LET account BE account = first ( loggedin ) IN
**IF** amount ≤ balance (account) **THEN**

\[
\text{balance (account)} := \text{balance (account)} - \text{amount} \parallel
\]

\[
\text{bal} := \text{balance (account)} - \text{amount} \parallel
\]

\[
\text{response} := \text{OK}
\]

ELSE \quad \text{bal} ∈ \mathbb{N} \parallel \text{response} := \text{REFUSED}

**END**

**END**

ELSE \quad \text{bal} ∈ \mathbb{N} \parallel \text{response} := \text{REFUSED}

**END**

END ;

\[
\text{response} \leftarrow \text{Deposit (amount)} \quad \triangleq
\]

PRE amount ∈ \mathbb{N} **THEN**

**IF** loggedin ≠ [ ] **THEN**

**LET** account BE account = first (loggedin) **IN**

\[
\text{balance (account)} := \text{balance (account)} + \text{amount} \parallel
\]

\[
\text{response} := \text{OK}
\]

**END**

ELSE \quad \text{response} := \text{REFUSED}

**END**

END ;

\[
\text{response , bal} \leftarrow \text{Balance} \quad \triangleq
\]

**IF** loggedin ≠ [ ] **THEN**

**LET** account BE account = first (loggedin) **IN**

\[
\text{bal} := \text{balance (account)} \parallel
\]

\[
\text{response} := \text{OK}
\]

**END**

ELSE \quad bal ∈ \mathbb{N} \parallel

\[
\text{response} := \text{REFUSED}
\]

**END**

END ;

\[
\text{response , account} \leftarrow \text{OpenAccount (pass)} \quad \triangleq
\]

PRE pass ∈ \text{PASSWORD} **THEN**

**IF** accounts ≠ ACCOUNT

**THEN** ANY acc WHERE acc ∈ ACCOUNT − accounts

**THEN** accounts := accounts ∪ \{acc\} \parallel

\[
\text{balance (acc)} := 0 \parallel
\]

\[
\text{password (acc)} := \text{pass} \parallel
\]
\begin{verbatim}
account := acc ||
response := OK
END
ELSE account ∈ ACCOUNT ||
response := REFUSED
END
END;

WithDrawCard ≜ loggedin := []
END
\end{verbatim}

2.2 Improving the Model

This ATM0 model is seriously ill-conceived. It puts bank-like state inside the ATM. This is obviously wrong: ATMs have no banking knowledge, they are simply boxes in the wall that interact with a card user and communicate with a remote banking system.

We will attempt to build a more realistic model that separates the ATM and the remote banking system.

First, we need to specify the context information that is common to both the ATM and the remote banking system. This is shown in BankContext and Password. It’s split into two machines because the account, service card and response modelling “belongs” to the banking system, but the modelling of passwords is global.

One way of thinking about how you should model a machine like an ATM is to imagine watching another person using an ATM, from a little distance. Your model should describe what you see, not what you might imagine is going on inside the machine.

ATM0 could never be the result of such an exercise. Nor could ATM1.

We are emphasising what rather than how, perhaps more than in any previous exercise.

2.3 Bank Context machine

\begin{verbatim}
MACHINE BankContext
SETS
  The set of account ids
  ACCOUNT ;
  The set of services cards
  SCARD ;
  The set of responses
  RESPONSE = { OK, REFUSED }
CONSTANTS
  A constant, injective function, that maps accounts to service cards
\end{verbatim}
GENSCARD

PROPERTIES

\[
\text{card}(\text{ACCOUNT}) = \text{card}(\text{SCARD}) \land \\
\text{GENSCARD} \in \text{ACCOUNT} \rightarrow \text{SCARD}
\]

OPERATIONS

The operations implement the \textit{GENSCARD} function and its inverse.

\[
\text{scard} \leftarrow \text{GenScard}(\text{account}) \quad \triangleq \\
\text{PRE} \quad \text{account} \in \text{ACCOUNT} \land \text{account} \in \text{dom}(\text{GENSCARD}) \\
\text{THEN} \quad \text{scard} := \text{GENSCARD}(\text{account}) \\
\text{END} ; \\
\text{account} \leftarrow \text{ExtractAccount}(\text{scard}) \quad \triangleq \\
\text{PRE} \quad \text{scard} \in \text{SCARD} \land \text{scard} \in \text{ran}(\text{GENSCARD}) \\
\text{THEN} \quad \text{account} := \text{GENSCARD}^{-1}(\text{scard}) \\
\text{END}
\]

END

We are now modelling a service card, distinct from the account. We assume that the service card can be represented by information that is generated from the account, and that the account can be extracted from the service card.

Looking ahead to secure implementation to numeric operations we use the finite \textit{SCALAR} type from the \textit{Scalar_TYPE} machine instead of \( \mathbb{N} \).

2.4 Password machine

MACHINE Password
SETS PASSWORD
END

2.5 ATM1

We show two stages in specifying the ATM. The first attempt, ATM1, is nearly what we are aiming for, but it contains modelling of the login management that is really nothing to do with the pure interface view of an ATM.

This machine attempts to model the observed behaviour of an ATM.

Note the extensive use of non-determinism.

MACHINE ATM1
SEES BankContext, Password, Scalar_TYPE
VARIABLES loggedin
INVARIANT \( \text{loggedin} \subseteq \text{ACCOUNT} \land \text{card}(\text{loggedin}) \leq 1 \)
INITIALISATION \( \text{loggedin} := \{\} \)
OPERATIONS

response ← InsertCard ( scard, pass ) ≜
PRE scard ∈ SCARD ∧ scard ∈ ran ( GENSCARD ) ∧
   pass ∈ PASSWORD THEN
   IF loggedin = {} THEN
     CHOICE
     response := OK ∥
     loggedin := { GENSCARD⁻¹ ( scard ) }
     OR
     response := REFUSED
     END
   ELSE response := REFUSED
   END
END ;

response, money, bal ← ATMWithDraw ( amount ) ≜
PRE amount ∈ SCALAR THEN
   IF loggedin ≠ {} THEN
     CHOICE
     response := OK ∥ money := amount
     OR
     response := REFUSED ∥ money := 0
     END
   ELSE response := REFUSED ∥ money := 0
   END ∥ bal ∈ SCALAR
END ;

response, bal ← ATMDeposit ( amount ) ≜
PRE amount ∈ SCALAR THEN
   IF loggedin ≠ {} THEN
     response ∈ RESPONSE
     END ∥ bal ∈ SCALAR
   ELSE response := REFUSED
   END ∥ bal ∈ SCALAR
END ;

response, bal ← ATMBalance ≜
BEGIN

7
\begin{verbatim}
IF loggedin \neq \{} THEN
    response \in \text{RESPONSE}
ELSE
    response := \text{REFUSED}
END \parallel bal \in \text{SCALAR}
END ;

\textbf{WithdrawCard} \triangleq \text{loggedin} := \{}
END

\section{ATM}

ATM is a stateless machine with nondeterministic operations.
This machine models the observed behaviour of an ATM. It does not attempt to model the reasons for
the behaviour.
Note the extensive use of non-determinism.

\textbf{MACHINE} \textit{ATM}
\textbf{SEES} BankContext , Password , Scalar\_TYPE

\textbf{OPERATIONS}

\(\text{response} \leftarrow \text{InsertCard} (\text{scard} , \text{pass} ) \triangleq \)
\begin{align*}
\text{PRE} & \quad \text{scard} \in \text{SCARD} \land \text{scard} \in \text{ran} (\text{GENSCARD}) \land \\
& \quad \text{pass} \in \text{PASSWORD} \quad \text{THEN} \\
& \quad \text{response} \in \text{RESPONSE}
\end{align*}
END ;

\(\text{response} , \text{money} , \text{bal} \leftarrow \text{ATMWithdraw} (\text{amount} ) \triangleq \)
\begin{align*}
\text{PRE} & \quad \text{amount} \in \text{SCALAR} \quad \text{THEN} \\
& \quad \text{CHOICE}
\quad \text{response} := \text{OK} \parallel \text{money} := \text{amount}
\quad \text{OR}
\quad \text{response} := \text{REFUSED} \parallel \text{money} := 0
\end{align*}
\begin{align*}
& \quad \text{END} \parallel \text{bal} \in \text{SCALAR}
\end{align*}
END ;

\(\text{response} , \text{bal} \leftarrow \text{ATMDeposit} (\text{amount} ) \triangleq \)
\begin{align*}
\text{PRE} & \quad \text{amount} \in \text{SCALAR} \quad \text{THEN}
\quad \text{response} \in \text{RESPONSE} \parallel \text{bal} \in \text{SCALAR}
\end{align*}
END ;
\end{verbatim}
response , bal ← ATMBalance ≜
BEGIN
    response ∊ RESPONSE ∥ bal ∊ SCALAR
END ;

WithdrawCard ≜ skip
END

2.7 Bank

This machine provides a simple model of a remote banking system. The machine operations represent transactions and queries that can be requested by a client system.

MACHINE Bank ( maxaccount )
CONSTRAINTS maxaccount ∈ 1..MaxScalar
SEES BankContext , Password , Scalar_TYPE , Bool_TYPE

OPERATIONS

response , account , scard ← OpenAccount ( pass ) ≜
    PRE  pass ∊ PASSWORD THEN
    account ∈ ACCOUNT ∥ scard ∊ SCARD ∥
    response ∊ RESPONSE
END ;

response ← CheckPassword ( account , pass ) ≜
    PRE  account ∈ ACCOUNT ∧ pass ∊ PASSWORD THEN
    response ∊ RESPONSE
END ;

response , bal ← Withdraw ( account , amount ) ≜
    PRE  account ∈ ACCOUNT ∧ amount ∊ SCALAR THEN
    bal ∊ SCALAR ∥ response ∊ RESPONSE
END ;

response ← Deposit ( account , amount ) ≜
    PRE  account ∈ ACCOUNT ∧ amount ∊ SCALAR THEN
    response ∊ RESPONSE
END ;
response , bal ← Balance ( account ) ≜
\[ \text{PRE } \text{account } \in \text{ACCOUNT THEN} \]
\[ \text{bal } : \in \text{SCALAR } \parallel \text{response } : \in \text{RESPONSE} \]
END ;

\[ \text{response } ← \text{BackUp } ≜ \text{response } : \in \text{BOOL} ; \]
\[ \text{response } ← \text{Restore } ≜ \text{response } : \in \text{BOOL} \]
END

2.8 Login
The login machine is responsible for maintaining login status.

MACHINE Login
SEES BankContext , Bool_TYPE
VARIABLES loggedin
INVARIANT loggedin ⊆ ACCOUNT ∧ card (loggedin ) ≤ 1
INITIALISATION loggedin := {} 

OPERATIONS

login ( account ) ≜
\[ \text{PRE } \text{account } \in \text{ACCOUNT THEN} \]
\[ \text{loggedin } := \{ \text{account } \} \]
END ;

status ← isloggedin ≜
\[ \text{status } := \text{bool } ( \text{loggedin } \neq \{ \} ) ; \]
account ← getaccount ≜
\[ \text{PRE } \text{loggedin } \neq \{ \} \ \text{THEN} \]
\[ \text{account } : \in \text{loggedin} \]
END ;
logout ≜ loggedin := {}
END

2.9 A Note on structure
Notice that the ATM, Bank, Password and Login machines each SEES BankContext.
Further along the development, BankContext will be imported into the implementation of Bank.
BankContext and Password will be implemented independently.

3 Refinement

Refinement is the name given to design in B Method (B). In general refinement is conducted as follows:
1. The old variables are replaced by new variables.

2. Part of the invariant, called the refinement relation, relates the new and old variables.

3. The operations in the refinement machine are modified so that the new operations simulate the old operations under the refinement relation.

The idea of refinement is that any user who is satisfied with the behaviour of an operation, must be satisfied with the behaviour of the refinement of that operation.

Notice that refinement is not equivalence. Outside the precondition of an operation, the refined operation can do anything. Refinement can also discard non-determinism.

Refinement allows an operations to be strengthened, but not weakened.

### 3.1 Implementation

*Implementation* is a form of refinement, subject to the following extra constraints:

1. the implementation machine can have no variables;

2. the implementation machine should import machines and use the variables and operations of the imported machines to implement the machine being refined;

3. *full hiding* of the variables of the imported machines is enforced, *ie* the variables of the imported machines can be neither referenced or modified directly within operations. All references and modifications within operations must be achieved through the use of operations.

### 3.2 Implementing the ATM machine

We now demonstrate that we can implement the ATM machine by importing the Bank and Login machines. The implementation is a programming exercise in which we are limited to using the operations of the imported machines. Notice that we are allowed to use sequential composition.

### 3.3 ATMI

```plaintext
IMPLEMENTATION ATMI
REFINES ATM
SEES Password, Bool_TYPE
IMPORTS
    BankContext,
    Bank (maxaccount),
    Login

OPERATIONS

response ← InsertCard (scard, pass) ≡
    VAR bb, res, account IN
```
$bb \leftarrow \text{isloggedin}$;

\textbf{IF} $bb = FALSE$ \textbf{THEN}

$account \leftarrow \text{ExtractAccount} \left( \text{scard} \right)$;

$res \leftarrow \text{CheckPassword} \left( \text{account}, \text{pass} \right)$;

\textbf{IF} $res = OK$ \textbf{THEN}

$\text{login} \left( \text{account} \right)$

\textbf{END}$;

$response := res$

\textbf{ELSE}$

$response := \text{REFUSED}$

\textbf{END}$

\textbf{END} $response := \text{REFUSED}$$

\textbf{END}$

\textbf{END};

\begin{small}
\textbf{response}, \text{money}, \text{bal} \leftarrow \text{ATMWithDraw} \left( \text{amount} \right)$

\textbf{VAR} $bb, res, acc$ IN

$response := \text{REFUSED}; \text{money} := 0; \text{bal} := 0$

$bb \leftarrow \text{isloggedin};$

\textbf{IF} $bb = TRUE$ \textbf{THEN}

$acc \leftarrow \text{getaccount};$

$res, bal \leftarrow \text{WithDraw} \left( \text{acc}, \text{amount} \right)$;

\textbf{IF} $res = OK$ \textbf{THEN}

$\text{money} := \text{amount}$

\textbf{END}$;$

$response := res$

\textbf{END}$

\textbf{END}$

\textbf{END};$

\textbf{END} $response, \text{bal} \leftarrow \text{ATMDeposit} \left( \text{amount} \right)$

\textbf{VAR} $bb, res, acc$ IN

$bb \leftarrow \text{isloggedin};$

\textbf{IF} $bb = TRUE$ \textbf{THEN}

$acc \leftarrow \text{getaccount};$

$response \leftarrow \text{Deposit} \left( \text{acc}, \text{amount} \right)$;

$res, bal \leftarrow \text{Balance} \left( \text{acc} \right)$;

\textbf{ELSE}$

$\text{bal} := 0; response := \text{REFUSED}$

\textbf{END}$

\textbf{END}$

\textbf{END};$

\textbf{END};$

\textbf{END}$ $response, \text{bal} \leftarrow \text{ATMBalance}$

\textbf{VAR} $bb, acc$ IN

\end{small}
bb ← isloggedin;
IF \( bb = \) TRUE THEN
acc ← getaccount;
response, bal ← Balance (acc)
ELSE
bal := 0; response := REFUSED
END
END;

WithDrawCard ≡ logout
DEFINITIONS maxaccount ≡ 100
END

3.4 ATMBankSys

In order to exercise the ATM machine it is necessary to be able to create accounts and service cards, so we extend the ATM machine by adding a clone of the OpenAccount operation.

MACHINE ATMBankSys
SEES BankContext, Password, Scalar_TYPE, Bool_TYPE
EXTENDS ATM

OPERATIONS
response, account, scard ← OpenAccount (pass) ≡
PRE pass ∈ PASSWORD THEN
account ∈ ACCOUNT || scard ∈ SCARD || response ∈ RESPONSE
END;
response ← BackUp ≡ response ∈ BOOL;
response ← Restore ≡ response ∈ BOOL
END

3.5 ATMBankSysI

And we implement the ATMBankSys machine.

IMPLEMENTATION ATMBankSysI
REFINES ATMBankSys
SEES Password, Bool_TYPE
IMPORTS
BankContext,
Bank (maxaccount),
Login
PROMOTES
OpenAccount, BackUp, Restore

OPERATIONS

\[ \text{response} \leftarrow \text{InsertCard}(\text{scard, pass}) \]

\[
\begin{align*}
\text{VAR} & \quad \text{bb, res, account IN} \\
\text{bb} & \leftarrow \text{isloggedin} ; \\
\text{IF} & \quad \text{bb} = \text{FALSE} \text{ THEN} \\
& \quad \text{account} \leftarrow \text{ExtractAccount}(\text{scard}) ; \\
& \quad \text{res} \leftarrow \text{CheckPassword}(\text{account, pass}) ; \\
& \quad \text{IF} \quad \text{res} = \text{OK} \text{ THEN} \\
& \quad \text{login}(\text{account}) \\
& \quad \text{END} ; \\
& \quad \text{response} := \text{res} \\
\text{ELSE} & \quad \text{response} := \text{REFUSED} \\
\text{END} \\
\text{END} ;
\end{align*}
\]

\[ \text{response, money, bal} \leftarrow \text{ATMWithDraw}(\text{amount}) \]

\[
\begin{align*}
\text{VAR} & \quad \text{bb, res, acc IN} \\
\text{response} & := \text{REFUSED} ; \text{money} := 0 ; \text{bal} := 0 ; \\
\text{bb} & \leftarrow \text{isloggedin} ; \\
\text{IF} & \quad \text{bb} = \text{TRUE} \text{ THEN} \\
& \quad \text{acc} \leftarrow \text{getaccount} ; \\
& \quad \text{res, bal} \leftarrow \text{WithDraw}(\text{acc, amount}) ; \\
& \quad \text{IF} \quad \text{res} = \text{OK} \text{ THEN} \\
& \quad \text{money} := \text{amount} \\
& \quad \text{END} ; \\
& \quad \text{response} := \text{res} \\
\text{END} \\
\text{END} ;
\end{align*}
\]

\[ \text{response, bal} \leftarrow \text{ATMDeposit}(\text{amount}) \]

\[
\begin{align*}
\text{VAR} & \quad \text{bb, res, acc IN} \\
\text{bb} & \leftarrow \text{isloggedin} ; \\
\text{IF} & \quad \text{bb} = \text{TRUE} \text{ THEN} \\
& \quad \text{acc} \leftarrow \text{getaccount} ; \\
& \quad \text{response} \leftarrow \text{Deposit}(\text{acc, amount}) ;
\end{align*}
\]
res, bal ← Balance (acc)
ELSE bal := 0; response := REFUSED
END
END;

response, bal ← ATMBalance
VAR bb, acc IN
bb ← isloggedin;
IF bb = TRUE THEN
acc ← getaccount;
response, bal ← Balance (acc)
ELSE bal := 0; response := REFUSED
END
END;

WithDrawCard ≜ logout
DEFINITIONS maxaccount ≜ 100
END

3.6 Password Encryption

In BankR, the refinement of Bank, we model the mapping from account to password with a function

\( \text{accounts} \rightarrow \text{PASSWORD} \).

Looking ahead to implementation, we recognise that it would be unwise to implement a mapping from account to a plaintext password. It would be more secure to encrypt the password. To provide facilities for this we introduce a new machine Encryption.

We also specify the operation CheckPassword as comparing encrypted passwords, rather than comparing plain passwords. Notice that we need to “think ahead” on this issue: if we specified the operation as comparing plain passwords, we could not later decide to implement the operation using comparison of encrypted passwords as this is weaker than comparing plain passwords and is hence not a refinement.

3.7 Encryption

MACHINE Encryption
SEES Password
SETS CRYPT
CONSTANTS ENCRYPT
PROPERTIES ENCRYPT ∈ PASSWORD → CRYPT

OPERATIONS

\( \text{encrypted} \leftarrow \text{Encrypt} (\text{password}) \)
PRE  \( \text{password} \in \text{PASSWORD} \)
THEN  \( \text{encrypted} := \text{ENCRYPT} \left( \text{password} \right) \)
END

3.8 BankR

BankR is a refinement of Bank. This machine adds variables to resolve the non-determinism in the operations of Bank.

REFINEMENT BankR
REFINES Bank
SEES BankContext, Password, Encryption, Scalar_TYPE, Bool_TYPE
VARIABLES accounts, password, balance

INVARIANT
\( \text{accounts} \subseteq \text{ACCOUNT} \land \ \\
\text{card} \left( \text{accounts} \right) \leq \text{maxaccount} \land \ \\
\text{password} \in \text{accounts} \rightarrow \text{PASSWORD} \land \ \\
\text{balance} \in \text{accounts} \rightarrow \text{SCALAR} \)

INITIALISATION accounts, password, balance := \{ \}, \{ \}, \{ \}

OPERATIONS
\( \text{response, account, scard } \leftarrow \text{OpenAccount} \left( \text{pass} \right) \triangleq \)

\begin{align*}
\text{CHOICE} \\
\text{IF } \text{card} \left( \text{accounts} \right) \neq \text{maxaccount} \\
\text{THEN } \text{ANY acc WHERE acc } \in \text{ACCOUNT} - \text{accounts} \\
\text{THEN } \text{accounts} := \text{accounts} \cup \{ \text{acc} \} \land \ \\
& \text{balance} \left( \text{acc} \right) := 0 \land \ \\
& \text{password} \left( \text{acc} \right) := \text{pass} \land \ \\
& \text{account} := \text{acc} \land \ \\
& \text{scard} := \text{GENSCARD} \left( \text{acc} \right) \land \ \\
& \text{response} := \text{OK} \land \ \\
\text{END} \\
\text{ELSE } \text{account } \in \text{ACCOUNT} \land \text{scard } \in \text{SCARD} \land \ \\
& \text{response} := \text{REFUSED} \land \ \\
\text{END} \\
\end{align*}

\text{OR}
\( \text{account } \in \text{ACCOUNT} \land \text{scard } \in \text{SCARD} \land \ \\
& \text{response} := \text{REFUSED} \land \ \\
\text{END} ; \)
response ← CheckPassword ( account , pass ) ≜
  IF account ∈ accounts ∧
      ENCRYPT ( password ( account ) ) = ENCRYPT ( pass )
  THEN response := OK
  ELSE response := REFUSED
END ;

response , bal ← WithDraw ( account , amount ) ≜
  IF account ∈ accounts ∧ amount ≤ balance ( account )
  THEN balance ( account ) := balance ( account ) − amount ||
      bal := balance ( account ) − amount ||
      response := OK
  ELSE bal ∈ SCALAR || response := REFUSED
END ;
response ← Deposit ( account , amount ) ≜
  IF account ∈ accounts ∧
      balance ( account ) + amount ≤ MaxScalar THEN
      balance ( account ) := balance ( account ) + amount ||
      response := OK
  ELSE response := REFUSED
END ;

response , bal ← Balance ( account ) ≜
  IF account ∈ accounts THEN
      bal := balance ( account ) ||
      response := OK
  ELSE bal ∈ SCALAR ||
      response := REFUSED
END ;

response ← BackUp ≜
BEGIN response ∈ BOOL END ;
response ← Restore ≜
BEGIN
  ANY acc , passw , bal
  WHERE acc ⊆ ACCOUNT ∧
      card ( acc ) ≤ maxaccount ∧
      passw ∈ acc → PASSWORD ∧
      bal ∈ acc → SCALAR
  THEN accounts , password , balance := acc , passw , bal
3.9 Implementing Bank

To implement the Bank machine we will use the B-Toolkit’s base generator capability.

The database definition shown in AccountDB base consists of an accounts database called accountsdb containing records with mandatory fields passwordf and balancef.

The B-Toolkit takes the base definition and generates the machine AccountDB, which is imported into the implementation BankRI.

Notice that having given service cards the possibility of having an identity different from accounts, we implement service cards as identical to accounts. We can do this without loss of generality.

3.10 AccountDB base

SYSTEM AccountDB
SUPPORTS BankRI
IS

BASE
  accountsdb

MANDATORY
  passwordf ∈ CRYPT ;
  balancef ∈ N

END

END

3.11 AccountDB

MACHINE AccountDBCtx
SETS
  accountsdb_ABSOBJ
DEFINITIONS
  AccountDB_ABSOBJ <-> accountsdb_ABSOBJ ∪ N;
  AccountDB_ABSOBJ_SETS <-> accountsdb_ABSOBJ ∪ N;
  AccountDB_SETS <-> N
END

MACHINE AccountDB (max_accountsdb : CRYPT)
CONSTRAINTS
  max_accountsdb ∈ 1 . . 2147483646
SEES
  AccountDBCtx, Bool.TYPE, Scalar.TYPE
VARIABLES
  accountsdb , locate_accountsdb , passwordf , balancef
INVARIANT
  accountsdb ⊆ accountsdb_ABSOBJ ∧
  card (accountsdb) ≤ max_accountsdb ∧
  locate_accountsdb ∈ 1 . . card (accountsdb) →→ accountsdb ∧
passwordf ∈ accountsdb → CRYPT ∧ balancef ∈ accountsdb → SCALAR

INITIALISATION
accountsdb := {} || locate_accountsdb := {} || passwordf := {} || balancef := {}

OPERATIONS
rep ←− save_AccountDB b = BEGIN
rep := BOOL END ;

rep ←− restore_AccountDB b = BEGIN
.accountsdbx, locate_accountsdbx, passwordfx, balancefx
WHERE
.accountsdbx ⊆ accountsdb ABSOBJ ∧
.card (accountsdbx) ≤ max_accountsdb ∧
.locate_accountsdbx ∈ 1 . . card (accountsdbx) ↦−→ accountsdb ∧
.passwordfx ∈ accountsdbx → CRYPT ∧
.balancefx ∈ accountsdbx → SCALAR
THEN
.accountsdb := accountsdbx || locate_accountsdb := locate_accountsdbx ||
.passwordf := passwordfx || balancef := balancefx
END ||
rep := BOOL END ;

rep, Base_accountsdb ←− make_accountsdb ( Val_passwordf, Val_balancef ) b = BEGIN

Base_accountsdb := accountsdb ABSOBJ − accountsdb ∧
.locate_accountsdb := locate_accountsdbx \{ Base_accountsdbx \} ||
.passwordf := Val_passwordf ||
.balancef := Val_balancef ||
.locate_accountsdb := locate_accountsdbx \{ locate_accountsdbx \} ||
rep := TRUE
END

OR

ANY Base_accountsdb WHERE
Base_accountsdb ∈ accountsdb ABSOBJ
THEN
Base_accountsdb := Base_accountsdbx ||
rep := FALSE
END

END ;

rep ←− vld_accountsdb ( Base_accountsdb ) b = BEGIN
Base_accountsdb ∈ accountsdb ABSOBJ THEN
rep := boolean ( Base_accountsdb ∈ accountsdb ) END ;

Base_accountsdb ←− any_accountsdb b = BEGIN Base_accountsdb := accountsdb ABSOBJ END ;
rep ←− eq_password ( Base_accountsdb, Elom_CRYPT ) b = BEGIN
Base_accountsdb ∈ accountsdb ∧ Elom_CRYPT ∈ CRYPT THEN
rep := boolean ( passwordf ( Base_accountsdb ) = Elom_CRYPT ) END ;

END ;

nat ←− vld_balancef ( Base_accountsdb ) b = BEGIN
Base_accountsdb ∈ accountsdb THEN
nat := balancef ( Base_accountsdb ) END ;

mod_balancef ( Base_accountsdb, nat ) b = BEGIN
Base_accountsdb ∈ accountsdb ∧ nat ∈ SCALAR THEN
balancef ( Base_accountsdb ) := nat
END

END

3.12 BankRI

IMPLEMENTATION BankRI
REFINES BankR
SEES
  BankContext, Password, Encryption, AccountDBCtx,
  Scalar_TYPE, Scalar_TYPE_Ops, Bool_TYPE
IMPORTS AccountDB (maxaccount, CRYPT)

PROPERTIES
  ACCOUNT = accountsdb_ABSOBJ ∧
  SCARD = ACCOUNT
INVAR
  accounts = accountsdb ∧
  ( password ; ENCRYPT ) = passwordf ∧
  balance = balancef

OPERATIONS

response, account, scard ← OpenAccount (pass) ≜
  VAR bb, db, encpass IN
  account ← any_accountsdb ; scard ← any_accountsdb ;
  response := REFUSED ;
  encpass ← Encrypt (pass) ;
  bb, db ← make_accountsdb (encpass, 0) ;
  IF bb = TRUE THEN
    account := db ; scard := db ; response := OK
  END
END ;

response ← CheckPassword (account, pass) ≜
  VAR bb, encpass IN
  response := REFUSED ;
  bb ← vld_accountsdb (account) ;
  IF bb = TRUE THEN
    encpass ← Encrypt (pass) ;
    bb ← eql_passwordf (account, encpass) ;
    IF bb = TRUE THEN
      response := OK
    END
  END
END ;

response, bal ← WithDraw (account, amount) ≜
  VAR bb, balv IN
response := REFUSED ; bal := 0 ;
bb ←− vld_accountsdb ( account ) ;

IF \( bb = TRUE \) THEN
  balv ←− val_balancef ( account ) ;
  IF amount ≤ balv THEN
    balv ←− SUB ( balv , amount ) ;
    mod_balancef ( account , balv ) ;
    bal := balv ;
    response := balv ;
  END
END

response ←− Deposit ( account , amount ) \( \triangleq \)

VAR \( bb , bal \) IN
response := REFUSED ;
bb ←− vld_accountsdb ( account ) ;

IF \( bb = TRUE \) THEN
  bal ←− val_balancef ( account ) ;
  IF bal ≤ MaxScalar − amount THEN
    bal ←− ADD ( bal , amount ) ;
    mod_balancef ( account , bal ) ;
    response := OK
  END
END

response , bal ←− Balance ( account ) \( \triangleq \)

VAR \( bb \) IN
bal := 0 ; response := REFUSED ;
bb ←− vld_accountsdb ( account ) ;

IF \( bb = TRUE \) THEN
  bal ←− val_balancef ( account ) ;
  response := OK
END

END

response ←− BackUp \( \triangleq \) response ←− save_AccountDB ;
response ←− Restore \( \triangleq \)
response ←− restore_AccountDB

END
3.13 Implementing Login

We implement Login by importing two instances of the renameable variable machine, *Rename_Vvar*, from the standard library. These two machines implement a Boolean variable and an ACCOUNT variable, respectively, and provide a refinement of the singleton or empty set of ACCOUNT.

**IMPLEMENTATION** Login

**REFINES** Login

**SEES** BankContext, Bool_TYPE

**IMPORTS** loggedin_Vvar (ACCOUNT), islogged_Vvar (BOOL)

**INVARIANT**

(loggedin ≠ {}) ⇒ islogged_Vvar = TRUE ∨
(loggedin = {}) ⇒ islogged_Vvar = FALSE ∨
(islogged_Vvar = TRUE ⇒ {loggedin_Vvar} = loggedin)

**INITIALISATION** islogged_STO_VAR (FALSE)

**OPERATIONS**

login (account) ≡

BEGIN
    loggedin_STO_VAR (account);
    islogged_STO_VAR (TRUE)
END;

status ← islogged

status ← islogged_VAL_VAR;
account ← getaccount

account ← loggedin_VAL_VAR;
logout ≡

islogged_STO_VAR (FALSE)

END

3.14 Implementing Encryption

Having set the scene for encryption, we will implement encryption by making the encrypted password the same as the password. Clearly, we can do this without loss of generality as we could use any other encryption we wish.

**IMPLEMENTATION** Encryption

**REFINES** Encryption

**SEES** Password

**PROPERTIES**

CRYPT = PASSWORD ∧

ENCRYPT = id (PASSWORD)

**OPERATIONS**
\[ \text{encrypted} \leftarrow \text{Encrypt}(\text{password}) = \]  
\[ \text{encrypted} := \text{password} \]

\section*{4 APPENDIX}

The following frames show the instantiation of the machines \textit{loggedin\_Vvar} and \textit{islogged\_Vvar.mch}.

\section*{5loggedin\_Vvar}

\textbf{MACHINE} \textit{loggedin\_Vvar (VALUE)}

\textbf{SEES} \textit{Bool\_TYPE}

\textbf{VARIABLES} \textit{loggedin\_Vvar}

\textbf{INVARIANT} \textit{loggedin\_Vvar} \text{\in VALUE}

\textbf{INITIALISATION} \textit{loggedin\_Vvar} := \text{VALUE}

\textbf{OPERATIONS}

\[ \text{vv} \leftarrow \text{loggedin\_VAL\_VAR} = \]
\[ \text{BEGIN} \]
\[ \text{vv} := \text{loggedin\_Vvar} \]
\[ \text{END} ; \]

\[ \text{loggedin\_STO\_VAR (vv)} = \]
\[ \text{PRE} \]
\[ \text{vv} \in \text{VALUE} \]
\[ \text{THEN} \]
\[ \text{loggedin\_Vvar} := \text{vv} \]
\[ \text{END} ; \]

\[ \text{bb} \leftarrow \text{loggedin\_EQL\_VAR (vv)} = \]
\[ \text{PRE} \]
\[ \text{vv} \in \text{VALUE} \]
\[ \text{THEN} \]
\[ \text{bb} := \text{bool} (\text{loggedin\_Vvar} = \text{vv}) \]
\[ \text{END} ; \]

\[ \text{bb} \leftarrow \text{loggedin\_NEQ\_VAR (vv)} = \]
\[ \text{PRE} \]
\[ \text{vv} \in \text{VALUE} \]
\[ \text{THEN} \]
\[ \text{bb} := \text{bool} (\text{loggedin\_Vvar} \neq \text{vv}) \]
\[ \text{END} ; \]

\[ \text{loggedin\_SAV\_VAR} = \]
\[ \text{BEGIN \hspace{0.67cm}} \text{skip} \hspace{0.67cm} \text{END} ; \]

\[ \text{loggedin\_RST\_VAR} = \]
BEGIN

loggedin_Vvar :∈ VALUE

END;
loggedin_SAVN_VAR ≜
BEGIN  skip  END;
loggedin_RSTN_VAR ≜
BEGIN

loggedin_Vvar :∈ VALUE

END

END

6  islogged_Vvar

MACHINE  islogged_Vvar ( VALUE )
SEES  Bool_TYPE
VARIABLES  islogged_Vvar
INVARIANT  islogged_Vvar ∈ VALUE
INITIALISATION  islogged_Vvar :∈ VALUE

OPERATIONS

vv ←−  islogged.VAL_VAR ≜
BEGIN
    vv := islogged_Vvar
END;

islogged.STO_VAR ( vv ) ≜
PRE
    vv ∈ VALUE
THEN
    islogged_Vvar := vv
END;

bb ←−  islogged.EQL_VAR ( vv ) ≜
PRE
    vv ∈ VALUE
THEN
    bb := bool ( islogged_Vvar = vv )
END;

bb ←−  islogged.NEQ_VAR ( vv ) ≜
PRE
    vv ∈ VALUE
THEN
    bb := bool ( islogged_Vvar ≠ vv )
END;