A Rigorous Approach To An ATM System Using B

The University of Teesside

School of Computing and Mathematics

Middlesbrough

TS1 3BA

Endre Moen

BSc(Hons) Computer Science

Supervisor: Bill Stoddart

Second Reader: Steve Dunne

March 10, 2003

Abstract

This report will breifly talk about basic concepts of the B-Method using the BToolkit. Basic concepts of number-theory will be introduced to help to understand the specification, refinement and implemention of an exponential symmetric cipher. The cipher development will demonstrate the use of user-defined library functions to aid in its refinement and implementation. A detailed discussion will be devoted to prove the symmetry of the cipher. In the process, other proof obligations will be investigated. This will demonstrate use of the BToolProver to discharge both classical mathematical proofs and logic proofs for which the BToolprover is intended to deal with.

The second part of the project investigates a formal specification of an Automated Teller Machine (ATM) system. Different design patterns will be discussed. The purpose of this chapter is to find a a formal model of the ATM system which both allows for global invariants to be expressed and can be refined into an implementation. The chapter will start by discussing a airtraffic control system which has a flaw. This flaw is investigated to draw connections with problems found in developing the ATM system. A in implementation is given of the ATM system. If not formally developed, it at least demonstrates the capability for B to implement this system. Finally a buffer model of the system is given within a B machine.

ACKNOWLEDGE

I would like to take this opportunity to thank my supervisor, Mr Bill Stoddart, who has provided me with remarkable ideas and provided in-depth feedback on my ideas and for his patient and openness towards discussions at any time, and for sharing his knowledge of the B-Method with me.

Mr Frank Zeyda provided invaluable help with the cipher development, specification, refinement and implementation of the exponential cipher and provided ideas on the client-server architecture. He has motivated me though is enusiasm for his work.

I would finaly like to thank the technitians at The University of Teesside School of Computing and Mathematics for helping me to install and update the B-licence file.

Contents

Chapter 1

Introduction

1.1 Personal Background

I had expreience with formal methods prior to this project, and having undertaken the *formal methods1* and *formal methods2* modules I was confident I had sufficient knowledge to attempt a project in formal methods. However, this project required firm knowledge about the B-Method and some concepts about number theory. It examines proving internal consistency proof obligations and investigates doing a classical mathematical proof in the BToolkit. It also explores the feasibility of modelling a client-server architecture in B for the purpose of refining a specification to make use of the BToolkit's library machines for communicating over a TCP/IP connection.

I believe the project has provided me with skills in theoretical computer science which are applicable in a variety of areas of computer science, and will give me a better understanding of software engineering in general. The final year module *Formal Aspects of Computer Science* has been valuble in teaching the theory behind the B-Method. The B-Method is the underlying principle of the BToolkit.

This project attracted me because it allows me to make use of mathematical knowledge. I had previously completed an introductory and advanced course in calculus during my placement year at The University of Oslo, so I was keen to apply some of that knowledge to my final year project.

This development has two major parts, a cipher and an ATM development. I believe this project explores much of the capabilities of the BToolkit. It investigates the hypotheses proving mechanisms presented by the autoprover and the BToolprover, and investigates the machine structuring mechanisms.

Some of the models discussed are expressed in such a way that they can easily be refined into other problems like a buffered reader-writer. At the end of the ATM development we look at the ATM system as a transaction systems and we see how B can express the interactions of a transaction system. Something that has been explored in great detail by the mathematical specifiaction language CSP.

The project idea for the cipher development was given by Mr Frank Zeyda, and the idea for the network development came from Mr Bill Stoddart. The ATM system allows for both of these ideas to be put together.

1.2 Example Scenario of the ATM system

When a user enters an ATM he/she can:

- create an account with the bank. The ATM will then request for an account to be created and reply to the request which, in turn is presented to the user.
- deposit money to an account.
- withdraw money if there is sufficient funds on the account
- query the balance of an account
- query the existence of an account

For each action, the ATM requests a transaction with the bank, and the bank replies. The reply is in turn presented to the user

1.3 Overall Goals of the Project

The overall goals of the project were to:

1. specify an ATM machine with operations
   - request to create accounts
   - request for a deposit to an account
   - request for a withdrawal from an account
   - query the existence of an account
   - query the balance of an account
2. specify a bank machine which holds data to allow for the ATM functionallity.
3. define the networking that takes place when a transaction is requested.
4. implement an *ATM* and a *Bank* machine
5. specify, refine and implement an exponential, symmetric cipher
6. combine these two developments.

1.4 Structure of the Report

For the novice reader, a brief introduction about the B-Method will be presented. Introducing basic concepts about the anatomy of a B development, the software development life cycle in B, and an introduction to the ideas behind the B-method. This section will point to references for further reading. The reader is assumed to be familliar with the BToolkit and to have a working knowlege of formal methods. It is necessary to have knowledge about Abrial's Generalised Substituation Language (GSL) [3]. It should also be noted that this builds on the ideas from E. Dijkstra's Guarded Command Language [8].

The report will continue to talk about basic concepts of number theory in relation to cryptograpy. This will provide the basis for the next section about a cipher development. The final section will discuss an ATM system, proposing several models with the purpose of refining them into working code.

Chapter 2

The B-Method

2.1 Anatomy of a B Development

The B-Method was invented by J-R Abrial. This project uses the BToolkit from specification to code generation. The BToolkit is a development environment that uses the B-Method to produce software.

The B-Method is a development methodology that makes use of stepwise refinement to produce working code from a specification. The specification makes use of concepts from set theory to say something abstract about the behaviour of operations supported by the specifiaction. The B-Method is a way of programming by refinement using ideas from set theory. The first step is to specify what a program or a section of a program is supposed to do. How it is done is deferred. The BToolkit uses Abstract Machine Notation (AMN) which allows for specification, refinement and implementation, and permits for formal verification of the development, so that the internal consistency of a program can be proven to be correct.

2.1.1 Abstract Machines

An abstact machine is organised by headings, with the most important headings being:

- MACHINE states name of machine and can take parameters. There are two typs of parameters, variables that holds a value and variables that denote sets, these are in capical letters
- CONSTRAINTS place constraints on input parameters from the machine heading
- SETS declare any sets. It may or may not be enumerated
- CONSTANTS introduce constants
- PROPERTIS constrain constants and sets
- VARIABLES introduce variables
- INVARIANT give the variables type, and constraints
- INITIALISATION initialise the varibles
- OPERATIONS says what functionality the machien can provide. The operations may take in and out-put parameters. It may restrict these parametres, and modify variables.

2.1.2 Refinement

Refinement is the process of gradualy making the abstract machine specification into working code. Data refinement involves deciding how the data in a specification is to be represented in an implementation. The result of refinement is a combination of specification and implementation, eg a set may be refined into a sequence. This requires a linking invariant in the refinement to say that the range of the sequence is the same as the set from the specificaiton. Refinement introduces a new range of consistency proof obligations which must be met to say that a refinement refines a specification or another refinement.

2.1.3 Implementation

A B implementaition is a particular type of refinement which is detailed enough to be understood by the computer as instructions. An implementation has the same refinement relation to its machine or refinement as any other refinement, so it will have the same proof obligations.

These steps make up the development life cycle.

For futher reading consult [2] for an easy introduction or [3] for a detailed describtion. [3] will also give a describtion of GSL. For details on Guarded Command Language consult [5] and [12]. Good examples on developments by refinement is found in [4] with an ATM development called The B Bank (p 115).

2.2 The Theory behind the BToolkit

2.2.1 Weakest preconditions

Weakest preconditions say something about the state space of a program before and after execution of an operation. The weakest precondition for S to establish Q is written $[S]Q$.

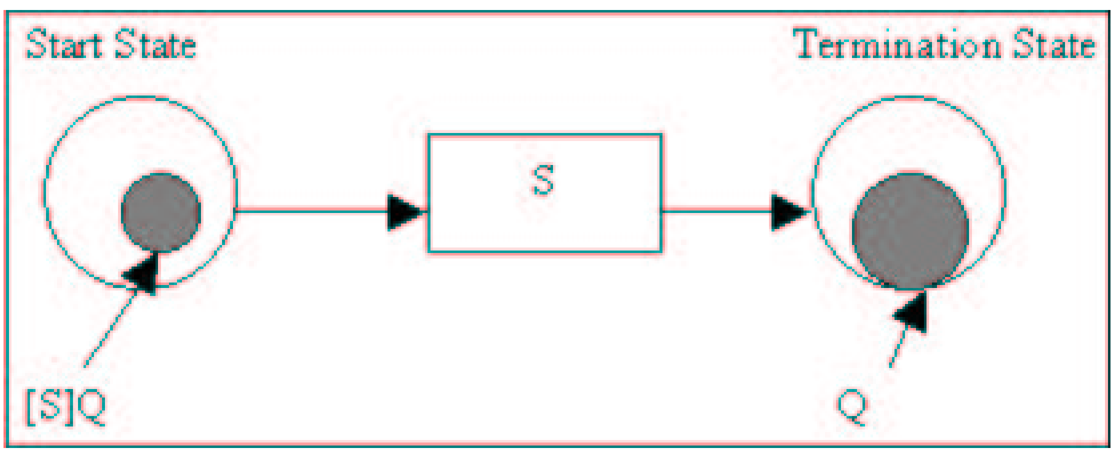


Figure 2.1: illustration of before and after state of program when executing an operation

AMN is a sugar-coated syntax for GSL and GSL builds on weakest precondition semantics described in E. Dijkstra's Guarded Command Language [5]. This language introduces ideas like *skip*. An operation that does nothing, sequential composition, ;, $a; b$. a executes then b executes. Guard, $\rightarrow$, $a \Rightarrow b$; $\not a \Rightarrow c$ can be understood as *if a then b else c*. Preconditions, —, $a|b$. b can only garantee to execute if a holds.

2.3 Abstract Machine Specification and Proof Obligations

Abstract machines follow some rules to produce consistency proof obligations. These proof obligations must be met to say that a specification is internaly consistent to avoid contradictions within a specification.

> "Proof of internal consistency of an Abstract Machine specification requires demonstration that its context may exist - the formal parameters, constants and variables - and that within this context of the machine, the initialisation establishes the invariant, and each machine operation maintains that invariant."

(The B-method and the B-Toolkit, p 6)

This is taken from a compendium from the module *Formal Methods2*. A machine specification gives rise to proof obligations. It is important to understand how proof obligations are created to develop a consistent specification.

MACHINE machine_name(x)

CONSTRAINTS P

CONSTANTS y

PROPERTIES Q

VARIABLES z

INVARIANT R

INITIALISATION T

OPERATIONS op_name = PRE L THEN S END;

END

Gives rise to the following proof obligations

$\exists x.P$

$P \Rightarrow \exists y.Q$

$(P \wedge Q) \Rightarrow \exists z.R$

$(P \wedge Q) \Rightarrow [T]R$

$(P \wedge Q \wedge R \wedge L) \Rightarrow [S]R$

Chapter 3

Cryptography

3.1 Introduction to Cryptography

The ATM needs to communicate with the bank. This communication should idealy be encrypted with a cipher. The cipher uses the RSA encryption mechanism and has the properties that it is symmetric and exponential. Let us say that M is any finite message and $m \in M$ where m is an instance of a message. Then the symmetric property implies:

$$encrypt(decrypt(m)) = m \tag{1}$$

$$decrypt(encrypt(m)) = m \tag{2}$$

The cipher also holds the property that it is exponential. The encrypt and decrypt functions are defined as:

$$encrypt(m) = m^e \; mod \; n \tag{3}$$

$$decrypt(m) = m^d \; mod \; n \tag{4}$$

where e, d, and n are integers such that

$$e \times d \; mod \; n \;=\; 1 \tag{5}$$

The RSA cipher is used in a public-key cryptosystem, the diagram illustrates how a secure connection can be established.

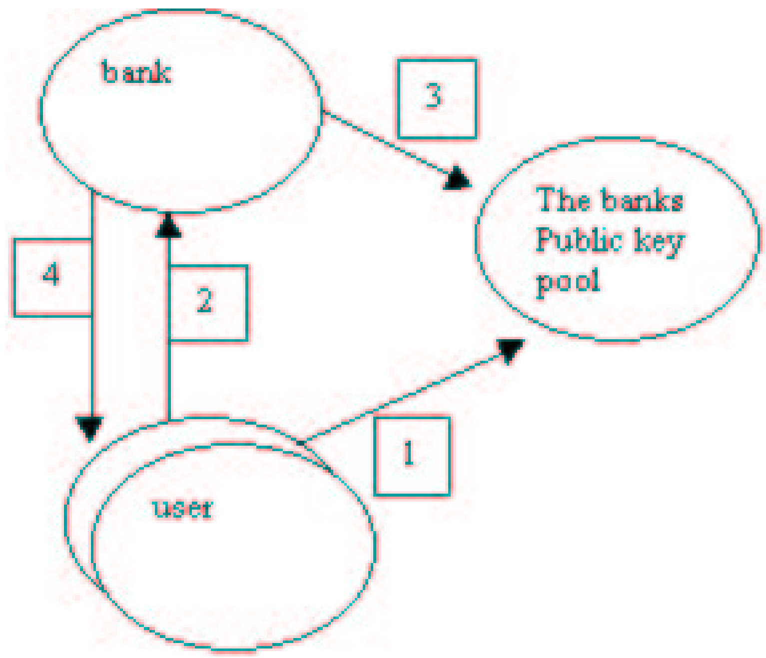


Figure 3.1: Key distribution in RSA

1. generate public and private key, send public key to public key directory.

2. send a message to the bank saying key available (not encrypted), (The user can also send an 'decrypted' message using its private key to authenticate that the user is who the user clames he/she is).

3. generate public and private key mapped to the particular user. Get users public key and send an encrypted message back using users public key. Then send the public key mapped to the user, to the public key pool

4. user decrypts message with its private key, and gets the bank's public key so it can send encrypted messages to the bank.

In a public key system the keys should be shifted frequently for increased security. RSA Inc posts competitions to decipher messages and to current date the best effort was made by a team of researchers which managed to break the cipher in seven months(!) using a supercomputer. This does not pose as big a threat as it looks like because the system can be used to agree on a common key for further use. This key is normaly changed each time a new connection is established. That means there is very little data for any crypto analysts to work on deciphering the RSA cipher.

A more important threat to todays internet banking is the future of quantum computers. They will be able to solv NP hard problems in polynominal time. The first quantum-algorithm found is one that can be used to decrypt just this system.

Remember that changing keys form time to time gives any cryptoanalysts (hackers) less information and more unreliable information as the cryptoanalysts can not know when the cipher key has been changed.

The RSA cipher is such that deducting the private key from the public is a NP hard problem. This means that as the number of digits used for the key are increased linearly, the time taken by today's computers increase exponentially, ciphers with keys of 512 digits are used giving numbers in the range of 0 .. $2^{512} - 1$ in twos compliment arithmetic.

3.1.1 The class P and NP

P is the class of algorithms that run in polynomial time. That is, they are deterministic. NP is the class of algorithms that run in exponential time. That is, they are non-deterministic. Visualise a binary tree, the more levels you add to the tree the time taken to visit each level is going to grow exponentialy.

$P \subset NP$ as any P problem can be described with a decision at each step of one (deterministic choice).

To describe why it is an NP hard problem to deduct the private key from the public key we need to understand that it is easy to generate large prime numbers but difficult to factor the product of two large prime numbers. We will now introduce some new ideas to better understand the RSA public key cryptosystem.

3.1.2 Modulus

$a\ mod\ n = b,\ n \neq 0$ has a solution such that $b \in [0, n-1]$. b is the remainder in integer division a/n, and $a = b + k \times n$ for some k, and k is the product(?) of integer division a/n.

Eg	$17\ mod\ 12 = 5$
remainder	$17/5 = 2$
integer division	$17/5 = 3$
We take the lower bound, which is written:	$\lfloor 17/5 \rfloor = 3$
So	$a = b + k \times n = 2 + 3 \times 5 = 17$

3.1.3 Inverse of modulus

Unlike ordinary arithmetic, modular arithmetic sometimes has inverses. We find inverses when solving equations of the form $a \times x\ mod\ n = b, n > 0, a \in 0..n-1, x \in 0..n-1$ This equation may have $0, 1$ or many solutions, hence inverses.

The greatest common divisor function defined recursively as $gcd(a, n) = gcd(n, a\ mod\ n)$ is used to determine the existence of inverses.

- If $gcd(a, n) = 1$ then, $a \times i\ mod\ n \neq a \times j\ mod\ n, 0 \leq i < j < n$
- If $gcd(a, n) = 1$ then a^{-1} exists such that $a^{-1} \in [0, n-1]$ and $a \times a^{-1}\ mod\ n = 1$

Eg. Find the inverse x given the equation $a \times x\ mod\ n = 1$
and $a = 3, n = 7$

1. $gcd(3, 7) = 1 \Rightarrow$ there exists an inverse.
2. $3 \times i\ mod\ 7 = \{0, 3, 6, 2, 5, 1, 4\}$ where $i = \{0, 1, 2, 3, 4, 5, 6\}$
   We see that $\{0, 3, 6, 2, 5, 1, 4\}$ is a permutation of $i = \{0, 1, 2, 3, 4, 5, 6\}$. This is not true if $gcd(a, n) \neq 1$.

   If $a = 3$ and $n = 6$ then $gcd(3, 6) = 3$ and we see that
   $3 \times i\ mod\ 6 = \{0, 3, 0, 3, 0, 3\}$ where $i = \{0, 1, 2, 3, 4, 5\}$ does not have an inverse.

Table 3.1: iteration of *extended gcd*

Iteration	a	b	a/b	x'	y'	d
1	50	35	1	-2	3	5
2	35	15	2	1	-2	5
3	15	5	3	0	1	5
4	5	0	–	1	0	

Table 3.2: Elaborate explanation of *extended gcd*

$gcd(50, 35)$	$i = 4, x' = 1, y' = 0$ $gcd(5, 0) = 5 \times 1 + 0 \times 0 = 5$
$gcd(35, 50\ mod\ 35) = gcd(35, 15)$	$i = 3, x' = 0, y' = 1 - 3 \times 0$ $gcd(15, 5) = 15 \times 0 + 5 \times 1 = 5$
$gcd(15, 35\ mod\ 15) = gcd(15, 5)$	$i = 2, x' = 1, y' = 0 - 2 \times 1$ $gcd(35, 15) = 1 \times 5 - 2 \times 15 = 5$
$gcd(5, 15\ mod\ 5) = gcd(5, 0) = 5$	$i = 1, x' = -2, y' = -1 - 1 \times (-2)$ $gcd(50, 35) = 50 \times -2 + 35 \times 3 = 5$

By seeing what value i holds when $3 \times i\ mod\ 7 = 1$ we find the inverse a^{-1}. We see that $i = x = a^{-1} = 5$.

To solve an equation of the form $a \times x\ mod\ n = b$ we need to use what we know about inverses. The complete set of recidues modulo 10 is:
$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} = r$.
The reduced set of recidues is $\{1, 3, 7, 9\}$. These elements hold the property that they are relatively prime to n, in this case $n = 10$. That is, $gcd(r_i, n) = 1$, where $i \in [0, n-1]$. Euler's phi function, $\varphi(n)$ gives the cardinality of the reduced set of residues. To solve a modular linear equation like $35x\ mod\ 50 = 10$ we first find $gcd(35, 50)$:

Compute left-hand side of table first, top down. Then use the result to compute right-hand side of table bottom up. Remember $\lfloor a/b \rfloor$ is integer division taking the lower boundary.

We have that $gcd(a,b) = ax' + by' = d$ x' and y' are computed as follows: $x'_i = y'_i + 1, y'_i = x'_i - 1 - \lfloor a_i/b_i \rfloor \times y'_i - 1$ x' and y' are obtained by:

$Extended - Euclid(a,\ b)$

- $if\ b = 0$

- -$return(a,\ 1,\ 0)$

- $(d',\ x',\ y') \leftarrow Extended - Euclid(b,\ a\ mod\ b)$

- $(d,\ x,\ y) \leftarrow d',\ y',\ x' - \lfloor a/b \rfloor \times y'$

- $return(d, x, y)$

- $a \times x\ mod\ n = b$ has d solutions where $d = gcd(a,b)$ or no solutions
- Let $d = gcd(a,n)$ and suppose $d = a \times x' + n \times y'$ for some integers x' and y'. If $d|b$ (d divides b) then the equation $a \times x\ \ mod\ \ n = b$ has one of its solutions x_0, $x_0 = x'(b/d)\ mod\ n$

so $x_0 = -2(10/5)\ mod\ 50 = -4\ mod\ 50 = 46$, and the rest of the solutions can be described with $x_i = x_0 + i(n/d)$ for $i = 1, 2, ..., d-1$

The solutions needs to be in the range $x_i \in [1, n]$ This gives us: $x_1 = 6, x_2 = 16, x_3 = 26, x_4 = 36, x_0 = 46$ This is indeed, d = 5, solutions.

We test the results x_0 and x_1:

$46 \times 35\ mod\ 50 = 1610\ mod\ 50 = 10$

$6 \times 35\ mod\ 50 = 210\ mod\ 50$

We know from:

- Euler's theorem states that $n > 1,\ p^{\varphi(n)}\ mod\ p = 1$
- Fermat's theorem states that if p is a prime which implies $gcd(a,p) = 1$ then $a^{p-1}\ mod\ p = 1$

Finding inverses in modular arithmetic can take very long time.

Table 3.3: modular exponensiation with and without two relative primes

3^i *mod* 7:	1	3	2	6	4	5	1	3	2	6
i	0	1	2	3	4	5	6	7	8	9

2^i *mod* 7:	1	2	4	1	2	4	1	2	4	1
i	0	1	2	3	4	5	6	7	8	9

See how a pattern is created with the value 2.

Chapter 4

The Cipher Development

4.1 Structure

The cipher development is composed of two machine specifications, *Cipher* and *Arithmetic*. *Arithmetic* specifies the exponent (*exp*) function (a^b), and then it is refined directly into an implementation. It is used within the cipher development (introduced in the refinement of *Cipher*).

Cipher is very abstract and only says that there exists operations to decrypt and encrypt a message, and that these functions together are symmetric. This machine may then be refined into any cipher like the Caesar shift cipher and the Vigen'ere cipher which make use of symmetric ciphers and a key, or the RSA cipher which have asymmetric ciphers to decrypt and encrypt.

This refinement refines the abstract cipher into an exponential cipher by using the *exp* function to define the exact encryption and decryption formulas. It can be refined into an RSA cipher with small changes to the refinement and a little more complications in the proof of its correctness. Where we have

$$e \times d \; mod \; n - 1 = 1 \quad (1)$$

in the exponential cipher we would have in

$$e \times d = 1 + k(p-1)(q-1) \tag{2}$$

in the RSA cipher. $\varphi(n) = (p-1)(q-1)$ where p and q are primes and $k = \lfloor a/n \rfloor$ (see section 4.3.1). (1) states that $e \times d$ must be relative prime to n, and (2) states that $(p-1)(q-1)$ must be relativ prime to n.

In the exponential cipher e and d are used to encrypt and decrypt. Whiles the RSA makes use of the product of p and q to encrypt and the prime numbers are used to decrypt. The security of the RSA cipher rests on the fact that multiplying two large prime numbers is a one way function. It is easy to find the product of two prime numbers but difficult (NP hard) to factor a large number to find the two primes.

The exponential cipher is then implemented. Interfaces are used to provide a non-Motif execution environment for the two implementations. See chapter 6 for more detail.

Arithmetic.mch	Cipher.mch
↓	↓
	ExponentialCipherRef.ref
	↓
ArithmeticI.imp	ExponentialCipherI.imp

4.2 The Arithmetic Machine development

Arithmetic functions as a library machine for the cipher refinement. The need for it may be argued. The ANSII C library already presents an *exp* function. This function is well tested and may prove to be more efficient than the one specified here. Unfortunately there are no mechanisms in the BToolkit to import C-library function. This may be something the B-Core can extend its toolkit do to in the future. The reason for doing it in this project is well justified though, without this constraint. The development is an exercise in implementing a specification using a b-loop construct. Loop constructs in B requires an INVARAINT heading which is a linking invariant to the refining specificaiton. It adds a level of difficulty to implementations in B.

The machine introduces a function in the CONSTRAINTS heading

$$exp \in \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N}$$

and it is a partial function because exp(0,0) is not defined. An operation *exp_op* calls this function and returns the value of exponentiation, not yet defined.
The specification gives rise to 2 proof obligations:

1. $\exists prime.(prime \subseteq \mathbb{N})$

   this is trivial. We might prove this by example by saying

   $\{3\} \in prime$

   but unfortunately the BToolProver has no way of deducting that

   $\{3\} \in prime \Rightarrow \exists prime.(prime \subseteq \mathbb{N})$

   so we have to add the trivial rule:

   $\exists p.(p \subseteq \mathbb{N})$

2. $\exists exp.(exp \in \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N} \wedge$

   $\forall(aa, bb).(aa \in \mathbb{N} \wedge bb \in \mathbb{N}_1 \Rightarrow$

   $exp(aa, bb) = exp(aa, bb-1) \times aa \wedge exp(bb, 0) = 1 \wedge exp(0, bb) = 0))$

   This is saying that there exists an *exp* function and that it is defined recursively as

   $$exp(aa, bb) = exp(aa, bb-1) \times aa$$

   This we know and it is given in the specification.

In general, to prove the existents of a function you can give an example of a value for the domain and range of the function.

The implementation is given by the b-loop construct:

OPERATIONS

$rr \longleftarrow$ **exp_op** (aa , bb) $\widehat{=}$

IF $aa \neq 0 \vee bb \neq 0$ **THEN**

VAR

ii ,

kk

IN

$ii := bb$ **;**

$kk := 1$ **;**

WHILE

$ii \neq 0$

DO

$kk := kk \times aa$ **;**

$ii := ii - 1$

INVARIANT

$ii \in \mathbb{N} \wedge$

$kk = exp\ (\ aa\ ,\ bb - ii\ )$

VARIANT

ii

END **;**

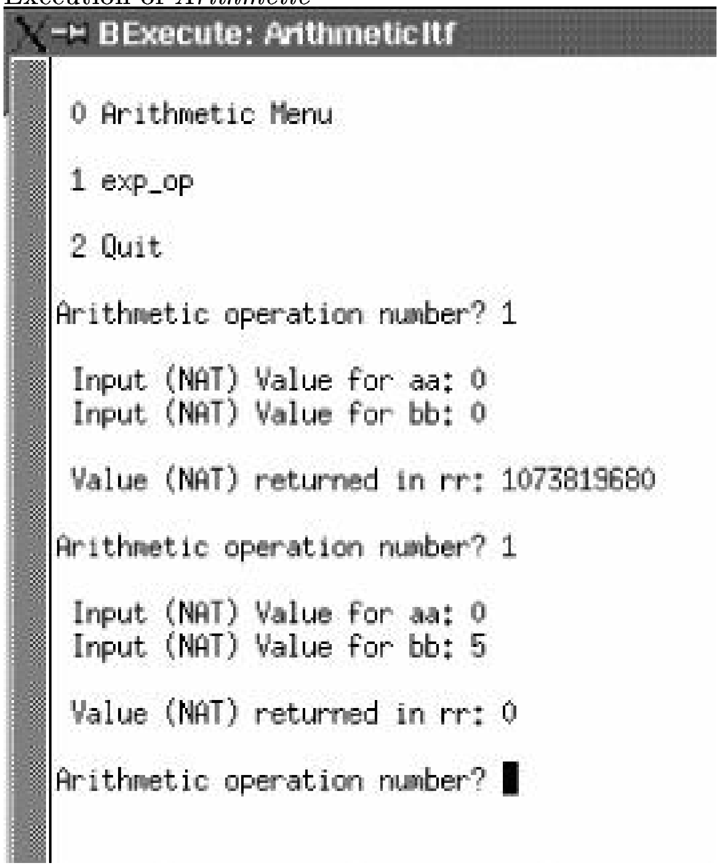
$rr := kk$

END

END

The invariant states that ii should never be less than 0, and that the current value of $kk = exp(aa, bb - ii)$. This is true as it corresponds to the quantification of exp given in the specification. It is the linking invariant.

Execution of *Arithmetic*



This implementation gives rise to 13 proof obligations. There are 6 proof obligations which are automatically discharged by the autoprover and can therefore be assumed to be trivial. We will have another look at the remaining 7 proof obligation and we will see that some of them also are trivial. There are 11 proof obligations related to the *exp_op* operation, 5 which have not been discharged by the BToolkit and which we will have a closer look at.

1. $cst(ArithmeticI\$1) \wedge$

   $ctx(ArithmeticI\$1) \wedge$

   $inv(ArithmeticI\$1) \wedge$

   $asn(ArithmeticI\$1) \wedge$

   $pre(exp\_op) \Rightarrow$

   $aa \neq 0 \ \wedge$

   $ii \in NAT \ \wedge$

   $kk \ = \ exp(aa,\ bb - ii) \ \wedge$

$ii \neq 0$

$\Rightarrow kk \times aa = exp(aa, bb - (ii - 1)))$

This can easily be proven by adding the substitution rule:

$$aa \in \mathbb{N} \Rightarrow (aa^b) \times aa == aa^{b+1}$$

This proof can be found in appendix A1, Proofs for Arithmetic.imp. Notice that the user defined theory introduces variable aa but uses a joker, b aswell in the rewrite rule. There is a significant difference between these two symboles because b can be anything.

2. ... $\wedge\ pre(exp\_op) \Rightarrow$

   $aa \neq 0 \Rightarrow$

   $1 \ = \ exp(aa, bb - bb)$

   This is is defined in the specification of exp. So a user defined theory $aa \in \mathbb{N}_1 \Rightarrow 1 = exp(aa, 0)$ is added.

3. ... $\wedge\ pre(exp\_op) \Rightarrow$

   $bb \neq 0\ \wedge$

   $ii \in \mathbb{N}\ \wedge$

   $kk \ = \ exp(aa,\ bb - ii)\ \wedge$

   $ii \neq 0 \Rightarrow$

   $kk \times aa \ = \ exp(aa, bb - (ii - 1))$

   It can be proven with what we already know and by using the user defined theory $(aa^b) \times aa == aa^{b+1}$, where $aa \in \mathbb{N}$.

4. ... $\wedge\ pre(exp\_op) \Rightarrow$

   $bb \neq 0 \Rightarrow$

   $1 = exp(aa,\ bb - bb)$

   This proof obligation can not be proven. It looks like we can use the user theory $aa \in \mathbb{N}_1 \Rightarrow 1 = exp(aa, 0)$, but we can not because that does not cover the case where $aa = 0$, and we can not change this because we can not violate the property that $exp(0, 0)$ does not exist. If $aa \neq 0$ then we can say

   $bb \in \mathbb{N}_1 \Rightarrow exp(aa, bb - bb) = 1$

   but if $aa = 0$ then we must say $bb \in \mathbb{N}_1 \Rightarrow exp(aa, bb - bb) = 0$

5. ... $\wedge\ pre(exp\_op) \Rightarrow$

   $aa = 0 \wedge$

   $bb = 0 \wedge$

   $rrZ = exp(aa,\ bb)$

   This can neither be proven. There is no value for $exp(0,0)$, but the BToolkit insists that there must be some value for it.

There are 2 proof obligations related to the context of the implementation. These are the same as for *Arithmetic* and have been discussed above.

4.3 Proof of the Exponential Cipher Refinement

The final cipher development is the result of many other specifications that have failed or have been dead ends because they have not resulted in the right proof obligation. For example in the first attempt of specifying a cipher machine, *encrypt_op* and *decrypt_op* was defined inside the operations clause and functions encrypt and decrypt where defined in the invariant:

MACHINE *CipherB* (*ee* , *dd* , *nn*)

CONSTRAINTS

$ee \in \mathbb{N} \wedge$

$dd \in \mathbb{N} \wedge$

$nn \in \mathbb{N}_1 \wedge$

$ee \times dd \;\mathsf{mod}\; nn - 1 = 1$

VARIABLES

encrypt , *decrypt*

INVARIANT

$encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$

$decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$

$\forall\, mm \,.\, (\, mm \in 0 \,..\, nn - 1 \Rightarrow encrypt\,(\, decrypt\,(\, mm\,)\,) = mm\,) \wedge$

$\forall\, mm \,.\, (\, mm \in 0 \,..\, nn - 1 \Rightarrow decrypt\,(\, encrypt\,(\, mm\,)\,) = mm\,)$

OPERATIONS

$rr \longleftarrow$ **encrypt_op** $(\, mm\,) \quad \widehat{=}$

PRE

$mm \in 0 \,..\, nn - 1$

THEN

$rr := encrypt\,(\, mm\,)$

END

END

This generates proof obligation

$$(P \wedge Q \wedge R \wedge L) \Rightarrow [S]R$$

for the operation *encrypt_op*. With this development we wishe to prove the $encrypt(decrypt(mm)) = mm$. That is, the cipher is symmetric.

The first proof obligation of the internal consistency of a machine is generated by
$\exists\, x.P$
and it says if P are the constraints on the parameter of the machine, then there should be some values of the parameter x that meet this constraints. *Cipher* has constraints:
CONSTRAINTS

$ee \in \mathbb{N} \wedge$

$dd \in \mathbb{N} \wedge$

$nn \in \mathbb{N}_1 \wedge$

$ee \times dd\ mod\ nn - 1 = 1$

and the proof obligation generated says

$$\exists(ee, dd, nn).(ee \in \mathbb{N} \wedge dd \in \mathbb{N} \wedge nn \in \mathbb{N}_1 \wedge ee \times dd \ mod \ nn - 1 = 1)$$

This can be proven by example given:

$$ee = 5, dd = 5, nn = 5 \Rightarrow (ee \in \mathbb{N} \wedge dd \in \mathbb{N} \wedge nn \in \mathbb{N}_1 \wedge ee \times dd \ mod \ nn - 1 = 1)$$

If we look at the proof obligation for PROPERTIES

$P \Rightarrow \exists\, y.Q$

It says that there are constants y that meet the PROPERTIES clause Q provided they meet the constraints P. This gives the proof obligation we are looking for.

In the final cipher specification, functions *encrypt* and *decrypt* are defined under the CONSTANTS clause and then the operations *encrypt_op* and *decrypt_op* simply returns the result of calling these fuctions. This generates proof obligation *encrypt(decrypt(mm))* $=$ *mm*. It is a common way of defining operations in B, and has also been applied in the arithmetic machine to define exp. It results in proof obligation

$P \Rightarrow \exists\, y.Q$

and is an easier proof than $\exists\, x.P$

The exponential cipher was introduced in last section. Here we will provide a proof of its correctness. It is worth noticing that *binhyp* is a way of introduing variables that only appares on the rigth hand of a substitution rule.

Choose n, e, d so that n is prime, $gcd(n, e) = 1$, and $e \times d \ mod \ \varphi(n) = 1$

$$E(m) = m^e \ mod \ n, m \in [0, n - 1]$$

$$D(m) = m^d \ mod \ n$$

Then we must show that $D(E(m)) = m$

After applying the DED (deduction) rule we have the following expression in the BToolkit

$\exists(encrypt, decrypt).(encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge \forall mm.(mm : 0..nn - 1 \Rightarrow encrypt(decrypt(mm)) = mm) \wedge \forall.mm.(mm \in 0..nn - 1 \Rightarrow decrypt(encrypt(mm) = mm))$

Then we apply a rewrite rule to substitute *encrypt(mm)* and *decrypt(mm)* with their respectiv functions. So one rewrite rule say:

$binhyp(n \in \mathbb{N}_1) \wedge$

$binhyp(e \in \mathbb{N}) \Rightarrow$

$encrypt(x) == exp(x, e)\ mod\ n$

$$
\begin{aligned}
\text{So } D(E(m)) &= (m^e\ mod\ n)^d\ mod\ n \\
&= (m^e)^d\ mod\ n \\
&= m^{e \times d}\ mod\ n
\end{aligned}
$$

from $e \times d\ mod\ \varphi(n) = 1$ we know that $e \times d = K \times \varphi(n) + 1$

for some K from the definition of modulus.

The BToolUserheory.5 states:

$e \in \mathbb{N}$

$d \in \mathbb{N}$

$binhyp(n \in \mathbb{N}_1)$

$e \times d\ mod\ n - 1 = 1$

$k \in \mathbb{N}_1$

$\Rightarrow$

$e \times d == k \times (n-1) + 1$

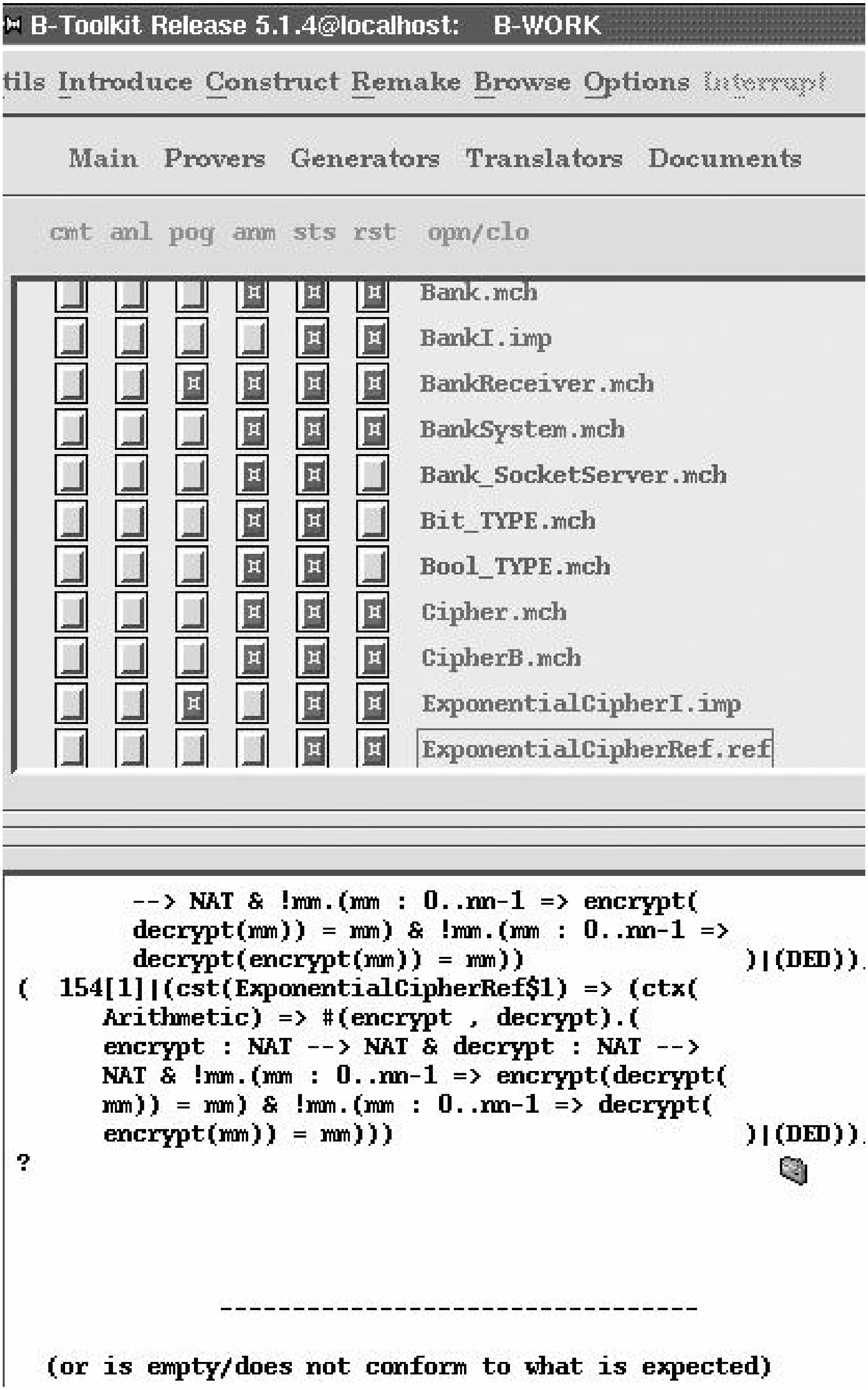
$$
\begin{aligned}
&= m^{k \times \varphi(n)+1}\ mod\ n \\
&= m^{k \times \varphi(n)} \times m\ mod\ n \\
&= (m^{k \times \varphi(n)}\ mod\ n) \times m\ mod\ n \\
&= ((m^{\varphi(n)}\ mod\ n)^k) \times m\ mod\ n
\end{aligned}
$$

Eulers theorem states that $a^{\varphi(n)}\ mod\ n = 1$

$$
\begin{aligned}
&= 1^k \times m\ mod\ n \\
&= m\ mod\ n
\end{aligned}
$$

$m \in [0, n-1] \Rightarrow m\ mod\ n = m$

The complete proof print can be found in Appendix A1, Proofs for ExponentialCipherRef.ref, together with all the added rewrite rules and the order in which they have been applied. Note that the second part of the symmetry proof has been galantly skiped over by adding a simple substitution rule. This is because the proof file became so big that the BToolkit was not able to generate a marked up latex file from it as can be seen from the picture.


B-Toolkit Release 5.1.4@localhost: B-WORK
tils Introduce Construct Remake Browse Options Interrupt
Main Provers Generators Translators Documents
cmt anl pog anm sts rst opn/clo
Bank.mch
BankI.imp
BankReceiver.mch
BankSystem.mch
Bank_SocketServer.mch
Bit_TYPE.mch
Bool_TYPE.mch
Cipher.mch
CipherB.mch
ExponentialCipherI.imp
ExponentialCipherRef.ref
--> NAT & !mm.(mm : 0..nn-1 => encrypt(
decrypt(mm)) = mm) & !mm.(mm : 0..nn-1 =>
decrypt(encrypt(mm)) = mm)))|(DED)).
(154[1]|(cst(ExponentialCipherRef$1) => (ctx(
Arithmetic) => #(encrypt , decrypt).(
encrypt : NAT --> NAT & decrypt : NAT -->
NAT & !mm.(mm : 0..nn-1 => encrypt(decrypt(
mm)) = mm) & !mm.(mm : 0..nn-1 => decrypt(
encrypt(mm)) = mm))))|(DED)).
?

(or is empty/does not conform to what is expected)

4.3.1 Some Issues About the BToolUserTheories and the exponential cipher proof

In the Appendix A1, Proofs for ExponentialCipherRef.ref, BToolUserTheory.5 is defined as:

$$e \in \mathbb{N} \land d \in \mathbb{N} \land binhyp(n \in \mathbb{N}_1) \land k \in \mathbb{N}_1 \Rightarrow e \times d == k \times (n-1) + 1$$

The variable k is introduced with very little restriction. There is a particular k we are looking for but we do not care what value it holds as it has no influence on the proof, and k is defined as $k = \lfloor a/n \rfloor$. We can not say $\exists k.(k \in \mathbb{N}_1)$ bacause there is a particular k. The way of introduceing k in this rule is not very formal.

Because k is not added to the hypotesis of the 'cipher proof', we have to add BToolUserTheory.6 which state $k \in \mathbb{N}_1$ and BToolUserTheory.8 which is a consequence of the prevoius rule, $k \in \mathbb{N}_1 \Rightarrow k \in \mathbb{N}$. Had k been added to the hypotesis, at least the last rule would have been 'explored' by the hypotesis environment. Is there a way of adding variables to the hypotesis of a proof?

BToolUserTheory11 is long-winded but very nessesery to complete the proof,

$$n \in \mathbb{N}_1 \Rightarrow a \times b \ mod \ n == a \times \ mod \ n \times b \ mod \ n \ mod \ n$$

There should be a way of expressing BToolUserTheory17 more precise. It uses a joker mm to substitute $mm \ mod \ nn = m$. It would be more precise to state:

$$n \in \mathbb{N}_1 \land a \in 0..n-1 \Rightarrow a \ mod \ n == a$$

but there are no facts about mm(!) in the hypothesis so the rule does not 'fire' .

BToolUserTheory18 is there because we have not shown that there exists function *encrypt* and *decrypt*. This bit of the proof is a shortcut and not formal. The proof consists of two parts,

1. prove $encrypt(decrypt(mm)) = mm$
2. prove $decrypt(encrypt(mm)) = mm$

Obiously, these two parts involve the same steps to be proven. For the reasons described at the end of last sections it has been added to make the proof shorter.

4.4 Benefits of Completing Proof Obligations

The purpose of completing proof obligations is to say that the specification models something and is consistent. In proving the consistency the specifier might discover errors in the specification. One such flaw that arose in *Cipher* is described here. The PROPERTIES heading initially said:

$\forall mm.(mm \in \mathbb{N} \Rightarrow D(E(m)) = m \;\wedge\; E(D(m)) = m)$

this is too strong and should say:

$\forall mm.(mm \in 0..nn-1 \Rightarrow D(E(m)) = m \;\wedge\; E(D(m)) = m)$

This shows that completing a proof can either make you change the specification because it is not correct or you find that it does not specify what you intend it to specify.

Chapter 5

The ATM Development

5.1 Structure

This chapter will discuss ideas of how to specify networking in B. It will review some awkward restrictions implied by the BToolkit which are too strong. It will discuss the problems it has resulted in, how it has been dealt with, how some models generalise to other problems found in computer science. Finally a discussion is offered as to why the client-server architecture is not suited for implementation in B.

The reasons for attempting to model something as uncertain as network communication are many. This development searches to find a model that both allows the specifier to state some global invariants about a client-server transaction system and at the same time provide opportunities to refine the model. Finding a model which both allows for a global invariant to be proven and which can be refined into working code would be a step forward in modelling transaction systems in B.

One of the challenges when specifying network communication is that the specification can become too abstract as a result of the uncertainty caused by the network layer or in an attempt in stating some global invariants about the system. What happens to the the customer's balance if the ATM crashes just before the customer gets his or her money? What happens with the customer's balance if the network goes down just after a customer has deposited money? There should be invariants in the development to say something about this, and in particular that "no money is lost over the network layer". How can this be achived in B, and can such a specification be implemented?

5.2 What the Application Should do

The attempts to specifying an ATM system has one common component, the *Bank* provides functionality to create an account, withdraw and deposit money, query if an account exists, and query the balance of an account. These operations are basic and they are only there to illustrate the network communication. In a more realistic system, access restrictions would be imposed on the query balance, withdrawal, and deposit operations. There would be operations; log on and log off and an operation to delete accounts amongst others.

In developments where the ATM has been specified seperately, the specification has been very simple. The ATM should idealy only present information to its user.

5.3 First Approach

The first step was to learn about how to construct specifications. In [1] there is an airspace specification which makes use of some of the structuring mechanisms provided by the BToolkit with the two most important; SEES and INCLUDES.

There was a subtle error in this development that gave a clue to a major obstacle which required a different mind set when developing the ATM system. From previous experience of Java programming, B can be viewed in some way similar to an object oriented system. Each machine specification can be viewed as an object, and a machine including another machine can be seen as an object instantiating another object. Of course that is almost as far as the similarity goes. The B INCLUDES construct demands exclusive access and INCLUDES is used in specification. Java is merely a programming language.

The airtraffic development discussed in [1] introduces machines: *Aircraft* which holds a set of aircrafts, *Controller* relates a controller to an aircraft, *Airspace* defines flight regions like city, military and airport zones and provides operation to move aircrafts between zones. Finally *ATCSystem* defines the operation *hand_over_aircraft* needed when an aircraft changes zone. The aircraft must be handed over to a new controller and the two zones the aircraft flights from and to must be updated.

ATCSystem promotes an operation from *Controller* which it does not extend but makes use of in *hand_over_aircraft*. First of all this is an error, second it can not be solved by extending *Controller*. If the machine had extended *Controller*, there would be a circular structure where *ATCSystem* includes/extends *Airspace* and *Controller*, (*ATCSystem* $\rightarrow$ (*Airspace*, *Controller*)) and *Airspace* includes *Controller*, (*Airspace* $\rightarrow$ *Controller*). Hence there would be two 'instances' of *Controller* in *ATCSystem* because INCLUDES is defined as:

> "INCLUDES: exclusive access; sets and constants of the included machine are visible in the including; variables of the included machine are visible in the invariant...;operations of the included can be used in the operations of the including;..."
> "Extends: as for INCLUDES, ...". If the operations is to be used in the extending machine they must be promoted."

([1], p 39)

The other constructs like SEES and USES are defined to take shared access but do not allow operations to be called.

The solution to the problem is to move the call to operation *add_aircraft* from *Airspace* to *ATCSystem*, which will both add a new aircraft to a new airspace and assign the aircraft to a new controller. Then the INCLUDES link between *Airspace* and *Controller* can be broken. A new INCLUDES link between *ATCSystem* and *Controller* must be made. Therefore in *ATCSystem* we have to redefine both operation *add_aircraft* from *Airspace* machine and *transfer_aircraft* from the *Controller* machine. The modified specification can be found in Appendix B, the original specification can be found in [1] (p 44).

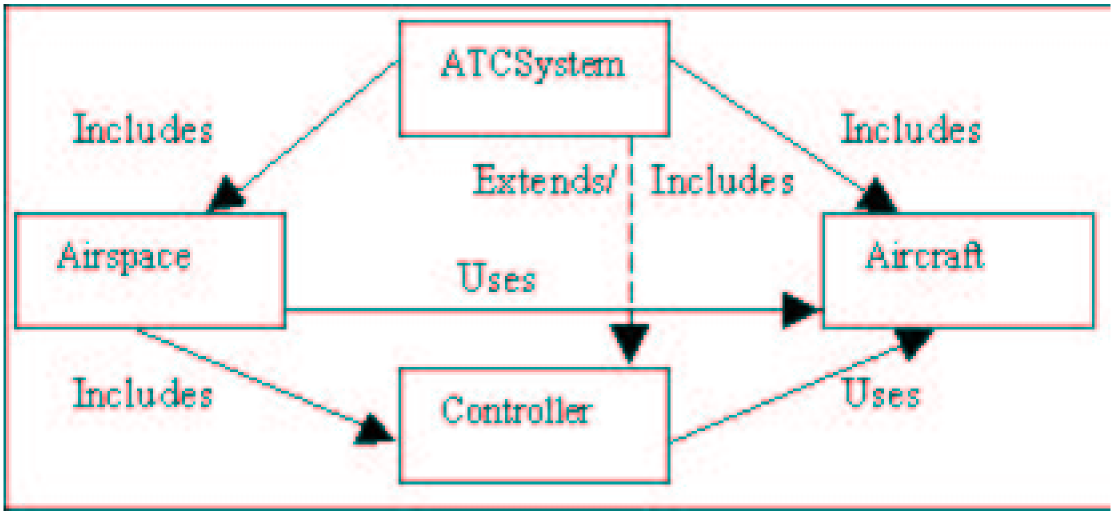


Figure 5.1: Aircraft control system from [1]

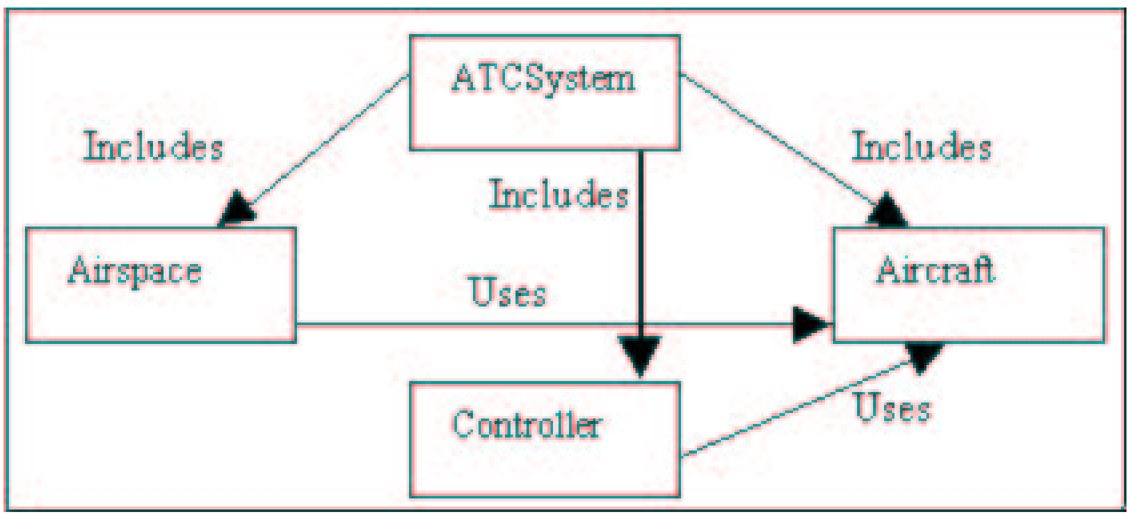


Figure 5.2: modified system

5.4 A Client-Server Protocol

The initial attempt was made with a client and server machine and a protocol specification to connect them. The first issues were to resolve how does the *ATM* machine send messages to the server, and where does the event; receiving messages happen? Does it happen in *ATM* and *Bank* or in a global protocol machine?

The first problem encountered was that of including machines to allow a two way communication. The first specification resulted in a circular include construct just like that found in the airtraffic control system

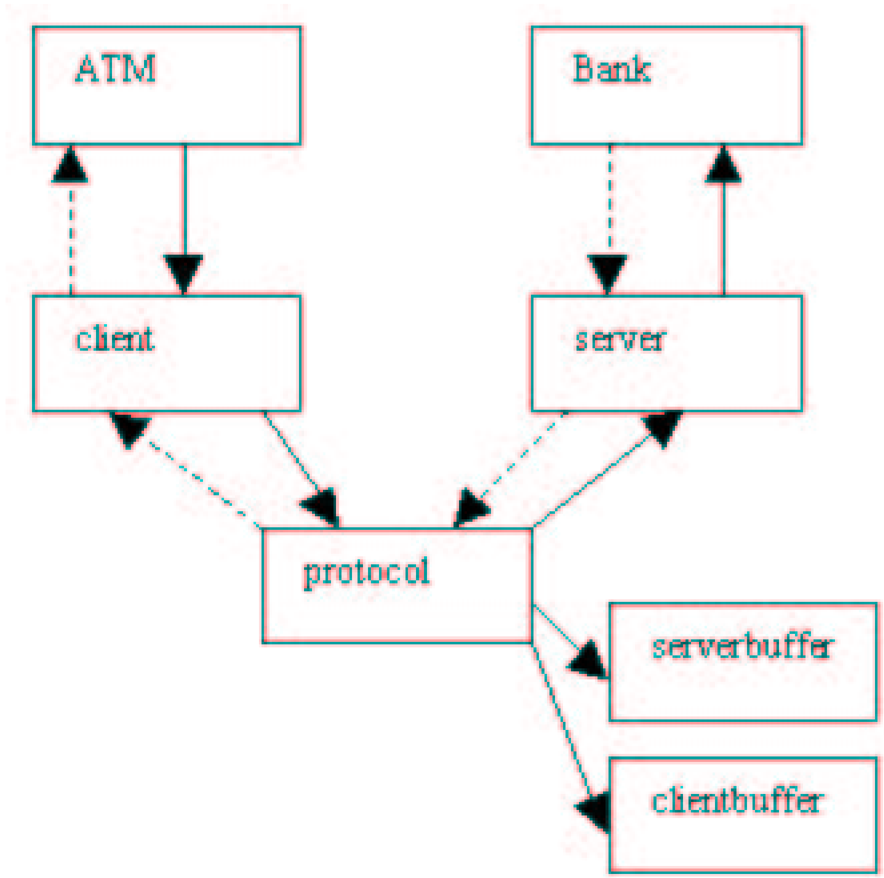


Figure 5.3: Protocol model for establishing communication, all the arrows denote INCLUDES

When a user logs on to the ATM it needs to send messages to bank. So the message must go through the client. The client calls operations on the protocol machine. First the client request a connection with the server, then it may request data which the server will forward to the bank, so the server must again call operations on the bank.

So far the message has only gone one way. If the bank is to reply to the message, it must see operations on the server, hence include/extend server. The server must call operations on protocol and so forth. This is not allowed, so this approach can only model one-way communication. It is unfortunate bacause the model has a good prospect of being refined and global invarinats may be stated in the protocol machine.

Throughout, these machines are defined as follows:

- *ATM* and *Bank* specifies the business logic for the ATM development
- *Client* and *Server* establish and maintain a connection. It can also be extended to preserve session ids for each user loggin on to the ATM. The *Server* can also be extended to allow a "protocol" call mechanism to the *Bank* operations as discussed in section 5.6, Networked Bank
- The *Protocol* has a system view. It holds two buffers, *clientbuffer* and *serverbuffer*. These buffers model data travelling on the network.

The next sections discusses different approaches to solving this problem and making the communication two-way.

Appendix C.1.1 shows how a connection between the server and client can be established. This builds on the realtime communication protocol from [1] (p. 135).

Appendix C.1.2 Shows how the communication protocol can be extended to allow the *ATM* and the *Bank* to communicate.

Server INCLUDES *Bank* and *ATM* INCLUDES *Protocol*. Both *Protocol* and *ATM* can be animated to show system functionality.

There is a problem with this approach in that each operation in *Bank* and *ATM* must be dupplicated in *Protocol* and in *Server*, and if the *ATM* requests an operation from *Bank*, it is going to get the reply via a return value from the opertion in *Bank* when more realisticly it should be stored in a buffer.

Two important answers to consider are:

1. How do we refine the development into an implementation?
2. Can *Protocol* be implemented, and what functionality would it have?

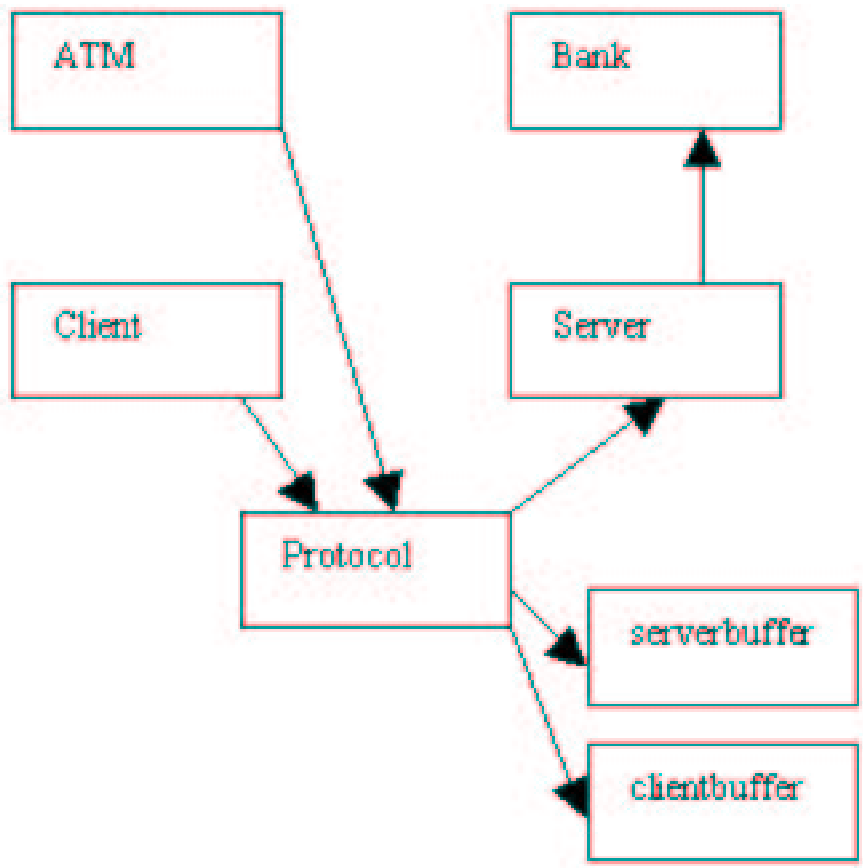


Figure 5.4: A new model with extensions to the ATM system

5.5 Receiver, Sender and Readers-Writers Problem

This development has a *SecureNetwork* machine which holds data in a buffer, and two other machines, *Receiver* and *Sender*, which communicate with each other by calling operations *receive* and *send* on *SecureNetwork*. These two machines may be refined into an ATM system, and the implementation can make use of the *SocketServer* and *SocketClient* library machines. See Appendix C.2

This models the interaction between a client and a server with a buffer held by *SecureNetwork*. It can provide the basis for an ATM system that allows for refinement of the *ATM* and *Bank* into two seperate implementations. Can this model refine a buffer specification where both

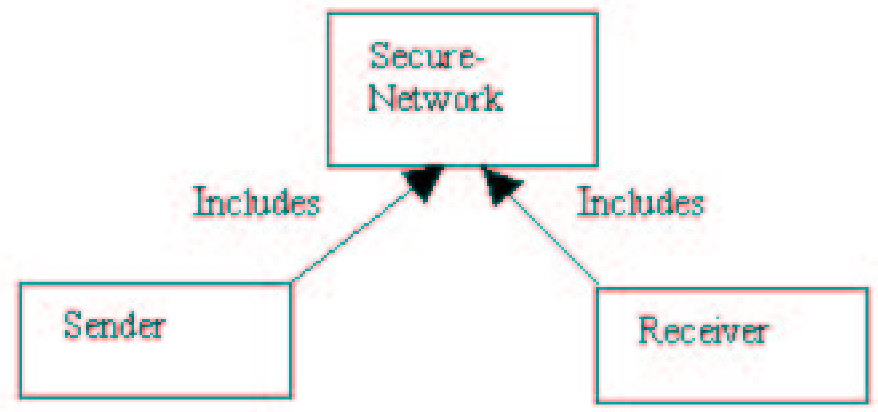


Figure 5.5: secure Network model

the *ATM* and the *Bank* is specified in the same machine? A solution to this has been added to the last section of this chapter and the resulting problems will be discussed.

SecureNetwork holds a sequence of messages. When anyone sends a message, it is added to the end of the message sequence and when someone calls *receive* they are given the first message in the sequence.

To distinguish the recipient of a message from other recipients, the *SecureNetwork* may be refined such that the message consists of a head and a tail. The head identifies to whom the message is intended for, and the tail consists of the data. The data must be stored in such a way that anyone can enquire the heading of a message without "popping" it of the fifo (first in, first out) sequence. See the data link layer of the OSI model [10] (p 43). This is the same strategy applied in the LAN (Local Area Network) protocol. So this model may be extended by making the receiver the *Bank* and by making the sender the *ATM*.

This model may also be seen as modelling the buffered readers-writers problem like the ones found in database systems or operating systems. Then the two operations *receive* and *send* in *SecureNetwork* should be refined into semafores.

The first approach to this model was made with use of preconditions instead of guards. The operations in SecureNetwork would then look something like this:

mm $\leftarrow$ receive =

PRE

$card(msg) > 0$

THEN

mm := first(msg)

END

For a *receiver* specification to make use of this operation it would first have to enquire the *SecureNetwork* if it had received any messages or it would have to be notified by the *sender* that it had sent a message. The first approach is tedious, having to call two operations to receive a message. The second approach requires a lot of work from the sender and in addition the sender may not know all the receivers, just as a datafield in C may not know who is pointing at it, consequently the problem of garbage collection.

A fictitious model (with fictitious I mean a model with cicular includes constructs) was made where the *Sender* would include *SecureNetwork* and *Receiver*. *Receiver* included *SecureNetwork* and whenever *Sender* wrote to the buffer in *SecureNetwork* it would set a flag in *Receiver* to say that it had written to the buffer.

5.6 Networked Bank

This is an attempt on only a specificating the server. It builds on the idea that there is not much to say about the operations of ATM as ATMs only receive and send data to the server and therefore does not maintain any data. So hence a specification of the client would mostly use non-deterministic substitution (:∈) to say something about the return value. This is a diversion on the route to find a model that both satisfies global invariants and can be refiend. The purpose is to produce two executable B implementations.

The *networkedBank* provides an interface for calling the operations on the *Bank*. Each operation in *Bank* is specified in *NetworkedBank* such that it can be called from any *ATM* using a protocol. When *ATM* calls an operation on *NetworkedBank* it sends a number. This number is stored in a sequence, *creq* for call-request, and each number in *creq* is associated with an operation in *Bank*. Another sequence *ff* holds the parameter value(s) to be passed to the operation in *Bank*.

MACHINE *NetworkedBank*

SEES *Bool_TYPE*

INCLUDES *Bank*

VARIABLES

ff , $creq$

INVARIANT

$ff \in \mathsf{seq} (\mathbb{N}) \wedge$

$creq \in \mathsf{seq} (\mathbb{N}) \wedge$

$\mathsf{size} (ff) \leq maxParams$

...

END

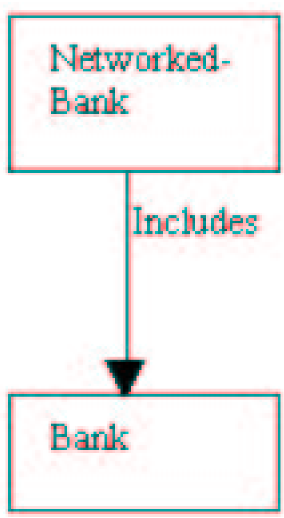


Figure 5.6: Wraped bank model

Then there is an operation, *check_clientreq*, which takes the last value in *creq* and compares it with the numbers associated with operations in *Bank*. If a match is found, the operation is called. The parameter values are taken from the *ff* sequence. *check_clientreq* does not check if the in-parameters from *ff* are correctly associated with the right operation calls. It is assumed to be correctly associated with the*ATM_x* operation (x are operations like *create_account deposit* and any other operation found in *Bank* and which has been 'promoted' to *NetworkedBank*). *NetworkedBank* pushes in-parameter(s) onto *ff*, and *check_clientreq* pops each element off *ff*. Though, this is not a proof obligation from the machine, the specification should have been expressed in a way to state that each parameter is correctly associated with an operation call on *Bank* even if the parameters are of the correct type. This is a weakness. How may the specification constraine this? Doing the type checking in *check_clientreq* would be useless as many of the operations in *Bank* take the same type. Hence checking for type does not guarantee an operation call with the right parameter(s). Also type checking in *NetworkedBank* will generate proof obligation in any machine which makes use of its operation (*ATM)* without sufficient knowledge to discharge them in thouse machines.

$num\ ,\ ok\ ,\ op \longleftarrow$ **check_clientreq** $\widehat{=}$

  IF $\mathsf{card}\ (\ creq\ ) > 0$ **THEN**

    VAR $aa\ ,\ bb$ **IN**

      IF $\mathsf{last}\ (\ creq\ ) = 1$ **THEN**

        $creq := \mathsf{front}\ (\ creq\ )\ \ \|$

        $num\ ,\ ok \longleftarrow create\_account\ \ \|$

        $op := 1$

      ELSE **IF** $\mathsf{last}\ (\ creq\ ) = 2$ **THEN**

          $creq := \mathsf{front}\ (\ creq\ )\ \ \|$

          $aa := ff\ (\ \mathsf{card}\ (\ ff\ ) - 1\ )\ \ \|$

          $bb := ff\ (\ \mathsf{card}\ (\ ff\ )\ )\ \ \|$

          $ff := ff \uparrow \mathsf{card}\ (\ ff\ ) - 2\ \ \|$

          $ok\ ,\ num \longleftarrow deposit\ (\ aa\ ,\ bb\ )\ \ \|$

          $op := 2$

        ...

The return values are passed back through the chain of communication created when the *ATM* calls an operation. This allows for a two-way communication at the same time as creating a top-down include-construct.

In order to prove the correctness of the specification, the operations in one of *Bank*'s precondition has been changed to a guard.

create_account =

PRE accountNumber $\neq$ ACCOUNTS THEN

...

has been changed to

create_account =

IF accountNumber /= ACCOUNTS THEN

...

The reason for mentioning this is that a similar change would have to be made on the operations *withdraw*, *isaccount*, and *getbalance* to completely remove all unproved proof obligations in *NetworkedBank*. Alternatively this type checking could be forwarded to *NetworkedBank*, which again would be generating proof obligations in ATM. An *ATM* machine should not know about all the types and variables in *Bank* and consequently there are no possibility of proving it withing *ATM*.

5.6.1 parallel operations, relational image and sequence concatenation

In the *ATM_x* operations where in-parameters are required they must be stored somehow. In *ATM_deposit* the following was first given:

...

$ff := ff(a) \leftarrow acc \ \ \|$

$ff := ff(b) \leftarrow amount \ \ \|$

and ff is a sequence. This is legal if $a \neq b$, put does not parse in B. This is too strong, but we know that

$ff(a) := x$

$ff(b) := y$

is equivelant to

$ff := ff \lhd\!\!\!- \{a \mapsto x, b \mapsto x\}$

This does the same as the parallel substitution and is allowed in B. It can be used as long as the relational override preserves the sequence constraint that $dom(ff) = 1..size(ff)$ but what if $a = size(ff) + 1$ and $b = size(ff) + 2$?

The only rule in the BToolkit that applies to this expression is InSequenceX.20

$binhyp(f \in seq(S)) \wedge$

$dom(g) \subset dom(f) \wedge$

$ran(g) \subset S$

$\Rightarrow$

$f \lhd\!\!\!- g \in seq(S)$

Applying this rule to $ff \lhd\!\!\!- \{a \mapsto x, b \mapsto y\} \in seq(ff)$

where $a = card(ff) + 1$ and $b = card(ff) + 2$ gives

$H \vdash ff \lhd\!\!\!- \{card(ff) + 1 \mapsto x, card(ff) + 2 \mapsto y\} \in seq(\mathbb{N})$

– $H \vdash ff \in seq(\mathbb{N})$

– $H \vdash dom(\{card(ff) + 1 \mapsto x, card(ff) + 2 \mapsto y\}) \in seq(\mathbb{N})$

...

It can not bee proven that

$dom(\{card(ff) + 1 \mapsto x, card(ff) + 2 \mapsto y\}) \in seq(\mathbb{N})$

The proof rule InSequenceX.20 is true, but too strong for this proof obligation as the hypotheses

$dom(g) \subset dom(f)$

is false for this relational override and it does not allow a sequence to grow.

Definition of relational over-ride:

"if $R_0 \in s \leftrightarrow T$ and $R_1 \in s \leftrightarrow T$ are two relations between s and t, then the relational over-ride $R_0 \mathbin{\lhd\!\!-} R_1$ is the relation R_0 with certain relationships replaced by those in R_1. This amounts to the relation R_1 together with any reminding relationships that R_0 has which are outside the **domain of** R_1."

([2], p 79)

Therefore $ff := ff \mathbin{\lhd\!\!-} \{card(ff) + 1 \mapsto acc, card(ff) + 2 \mapsto amount\}$
should prove, and the proof obligation
$card(ff) + 1 \in 1..size(ff)$ should not be needed.

The same thing can be said much simpler though,
$ff := ff \frown [\, x, y \,]$
and expresses the same thing and there exists a rule about sequence concatinationthat the autoprover can fire.

The attempt on the proof has generated from operation *ATM_deposit* in *NetworkedBank*.

5.7 An Implementation of the ATM System

The *ATM* talks with the server side by a protocol where each operation is referred to by a number just like the networked bank devlopment. The protocol is such that if the operation requires parametres, then they are sent after the operation-number, The ordering of the parametres being sent is important.

On the server side there are two layers. The *ATM* talks to *ServeBank*. *ServeBank* is a layer that handles communication with the client. When a request is received, it is sent to an operation in *Bank*. The *ServeBank* knows about the protocol and hence it knows which operation to call on *Bank*, and *Bank* holds the business logic. It completes the operation and returns a result. *ServeBank* sends the result back to the client. There is always a response to a request. The specification and implementation can be found in Appendix C.4.

This layering structure is both neat and very much forced on by the BToolkit. Think what would happen if the *Bank* machine both handelled requests from the client and executed the business logic. Then the request-operation would have to call other operations *within the same machine.*

BExecute: ATMItf

```
    `  0   0   0   1   1 '
    `requesting 'create_account'
_in_buf (2):
    `  1   1 '
encrypted account number is:
1
Your new account has Account Number: 1 1
 Value (NAT) returned in accNr: 1
 Value (BOOL) returned in rep: TRUE

ATM operation number? 3

ATM_WRITE (5 including 1st four length bytes):
    `  0   0   0   1   1 '
    `requesting 'create_account'
_in_buf (2):
    `  4   1 '
encrypted account number is:
4
Your new account has Account Number: 1 4
 Value (NAT) returned in accNr: 1
 Value (BOOL) returned in rep: TRUE

ATM operation number?
```

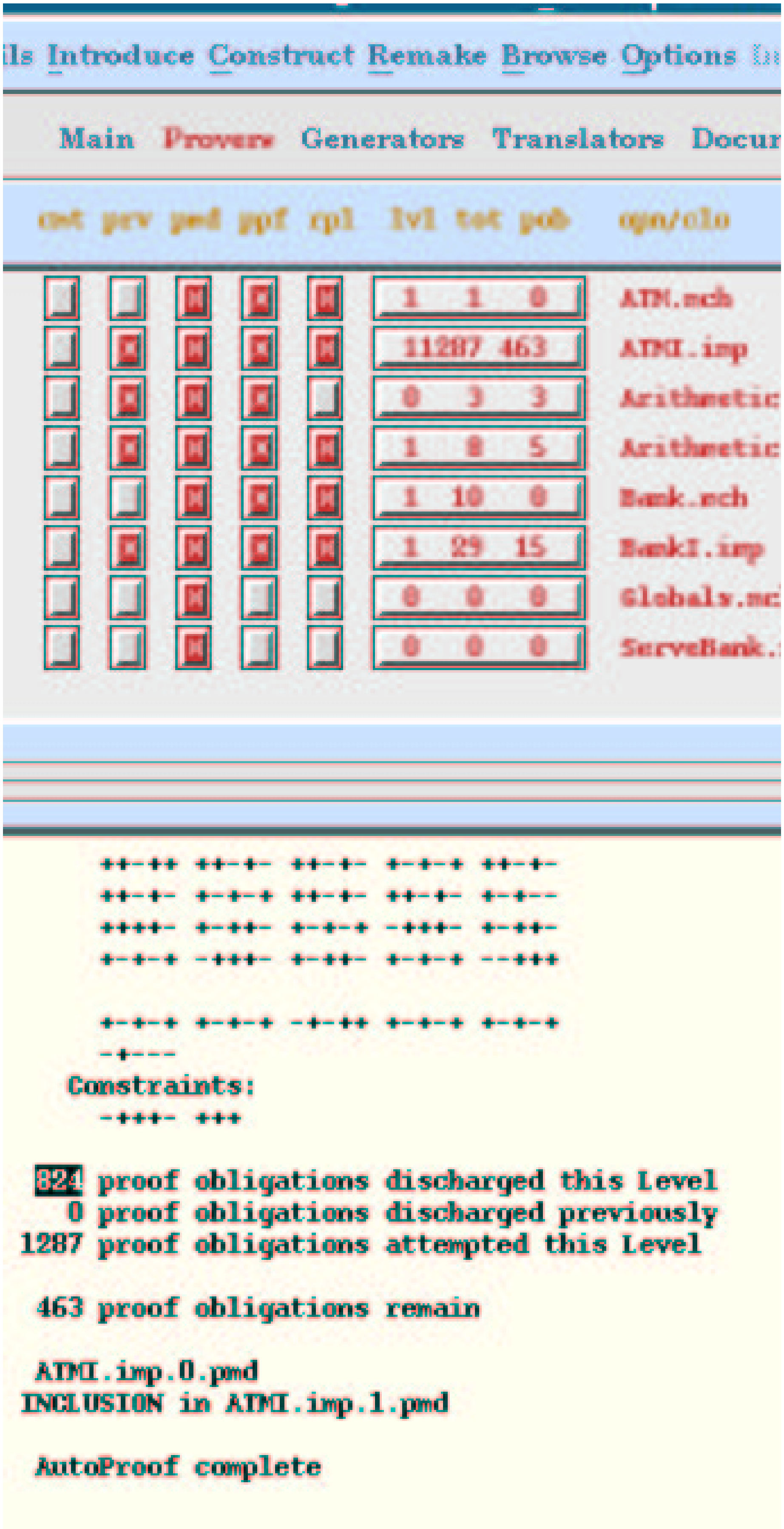
BExecute: ServeBankItf

```
Going to write tok
Bank_WRITE (6 including 1st four length bytes):
    `  0   0   0   2   1   1 '
have put str
AND returning values
value of OPO

WAITING HERE...
_in_buf (1):
    `  1 '
First tok is1

INSIDE create_account
calling create acc.
account number is:2
Going to write tok
Bank_WRITE (6 including 1st four length bytes):
    `  0   0   0   2   4   1 '
have put str
AND returning values
value of OPO

WAITING HERE...
```

The *listenForUser* operation in *ServeBank* is not specficied at all. It states *skip*. The implementation of *listenForUser* is very much similar to *check_clientreq* in *NetworkedBank* but is not expressed such that it is suited for refinement of this implementation. The specification does not store parameters in a buffer, and more importantly; the implementation uses an indefinite loop to check for client requests. How can this be expressed in a B specification, and how can it be expressed in the BToolkit? Formally the loop construct is wrong, but there is nothing wrong with having it in the implementation. When the client is shut down, so does the server. The termination of the server is dependent on the client.

The encryption mechanisms used in *ServeBank*'s *create_account* has not been specified. In the implementation the *create_account* operation encrypts the account number before sending it back to the ATM which decrypts it. The normal procedure in a public-key crypto system is that the public key is sent when a connection is established but since B does not allow for run-time 'instantiation' of machines. The keys have been presented in the *Globals* machine. It is there only to demonstate linking the two developments.

The implementation gave a suprising amount of proof obligation, 1287 proof obligations for *ATMI* when *ATM* only had one! It must be said that some of the proof obligations are of the form: ... $\Rightarrow$ "n" $\in$ STRING, and is trivial.


Introduce Construct Remake Browse Options
Main Provers Generators Translators
lvl tot pob
1 1 0 ATM.mch
1287 463 ATMI.imp
0 3 3 Arithmetic
1 8 5 Arithmetic
1 10 0 Bank.mch
1 29 15 BankI.imp
0 0 0 Globals.m
0 0 0 ServeBank.
++-++ ++-+- ++-+- +-+-+ ++-+-
++-+- +-+-+ ++-+- ++-+- +-+--
++++- +-++- +-+-+ -+++- +-++-
+-+-+ -+++- +-++- +-+-+ --+++
+-+-+ +-+-+ -+-++ +-+-+ +-+-+
-+---
Constraints:
-+++- +++
824 proof obligations discharged this Level
0 proof obligations discharged previously
1287 proof obligations attempted this Level
463 proof obligations remain
ATMI.imp.0.pmd
INCLUSION in ATMI.imp.1.pmd
AutoProof complete

5.8 A Buffer Model

This specification models the ATM system, and says something about transactions. It is refered to as a model rather than a specification as it has no prospect of being refined into an implementation. A transaction finds place when a customer deposits or withdraws money. A transaction must be done in such a way that no money is lost. (See Appendix C.5)

When a transaction takes place, the amount to withdraw or deposit is stored at the *ATM* in the set *ATMData*. When the request has gone through the network and **correctly** entered the *Bank* and the *Bank* has updated *accountbalance*, the transaction id is added to *confirm_withdraw/deposit*. This is a set which is intented to tell the ATM that a transaction has been completed. It is a handshake. The model has four sets which hold information about account balances; *ATMData*, *networkData*, *accountBalance* and *totalBalance*. *networkData* is unreliable and may disappere at any time. *ATMData* holds data about a transaction until the transaction is found in the *confirm_deposit/withdraw* set. *totalBalance* is a model-answere set. It always knows what is in the system. It is there so the model can say if money is lost over the network. This is given by the invariant:
$\forall(aa).(aa \in accountNumber \Rightarrow totalBalance(aa) = accountBalance(aa) + ATMData(aa))$

The buffer model defines actions that must take place when a customer keys into an ATM and wishes to deposit or withdraw money. It describes a one-to-one connection between the ATM and the bank. This prevents the model from having to include session ids to distinguish customers and their (account $\mapsto$ transaction) relationship. It is still possible to have many transactions, but it will require many *BankSystem*-models.

The buffer model assumes that the *ATM* is reliable even though this might not be. What if the electricity goes just after a customer has entered money in the ATM for deposit?

In an implementation, the *request_deposit* operation will be **started** by the *ATM*. The model assumes that the *ATM* knows about all account numbers, but in an implementation it is highly likely that the *ATM* will have to request a query to check that the account number entered matches an acount number in the *Bank*. In a refinement, the query should check that a given PIN (Personal Identification Number) matches the account number for which a transaction has been requested. This model does not care about identifying that the customer is who he/she says he/she is. To do this we would have to have an $accountNumber \rightarrow pinid$ relationship aswell. It is important that the *ATM* does not store any information related to the balance of customers.

Another illustration is that the model does not care how data verification messages are communicated between the *ATM* and the *Bank*, and between the *ATM* and the customer, is given by the *ATM_withdraw* operation. The operation chooses a zz such that $zz \in \mathbb{N}$ and $zz \leq accountBalance(yy)$, how does the ATM get that information and what happens to those zz where $zz > accountBalance(yy)$? Will the customer be informed that it has requested to overdraw the limit of the balance? The model does not care about this. It can be refined later. The check that amount to withdraw is less than or equal to balance can also be placed at *request_withdraw* but this would not allow customers to make mistakes.

The final question of this development is to find out if the model can be refined into two specifications. The *ATM* and the *Bank* and still presever a global invaraints.

The model can also be slightly altered to say something about a buffered-reader and a buffered-writer found in typical operating systems.

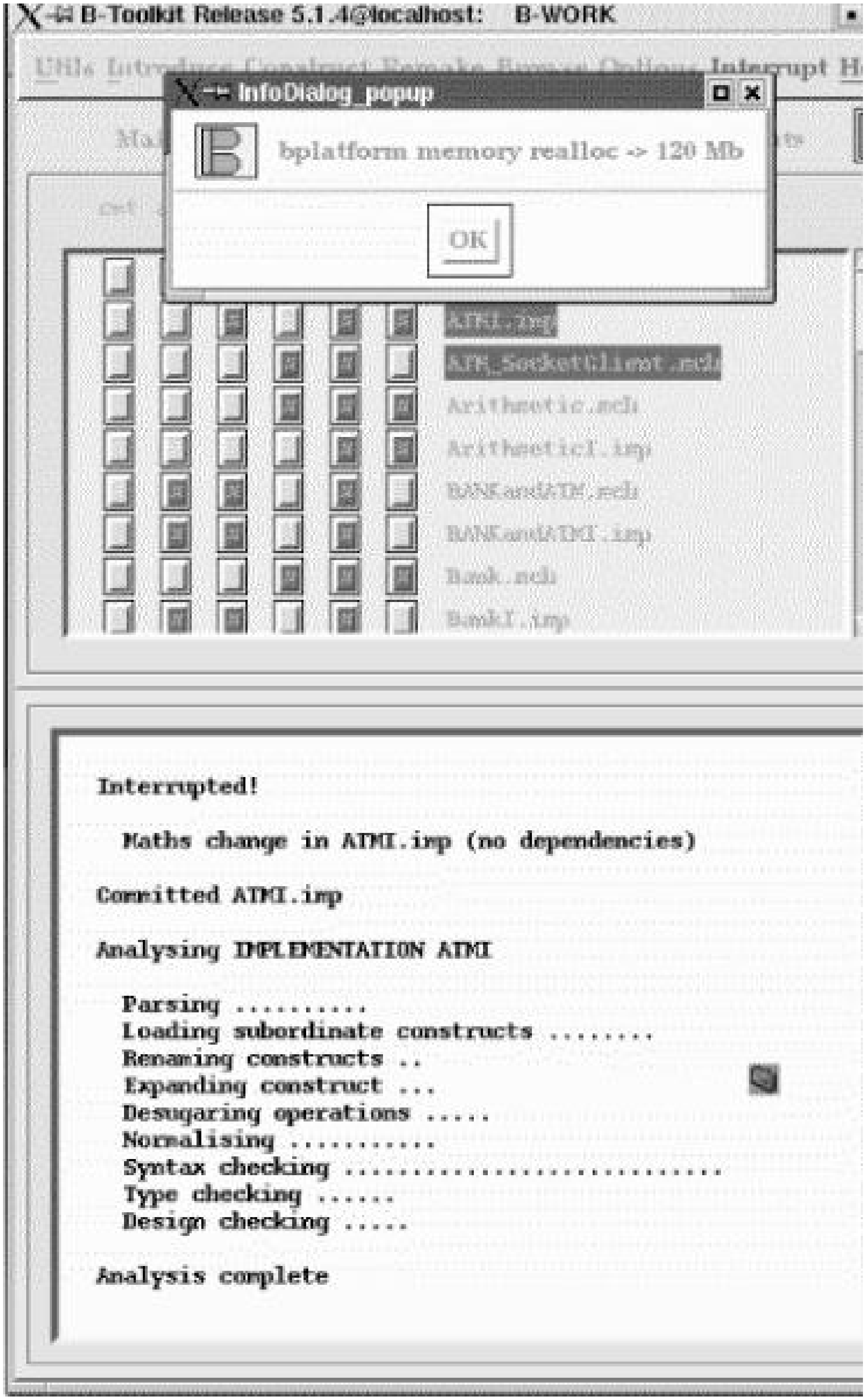
5.9 Bringing it all together

The *Protocol* development has shown that it is possible to state global invaraints about an ATM system. The development does not easily allow for two way communication, and it is not well suited for refinement. What happens with the *Protocol* machine in a possible implementation. The model does not say anything about where data is stored before and after transmission on the network, unlike the last development discussed.

The secure network development has shown that there exists a design pattern in B that will accurately specify the reading and writing to a common buffer by two seperate machines. Unfortunatly there is no way anything global can be stated about this system. Try and add a *Global* machine that both sees the *ATM* and the *Bank*. Such a development can be found on the floppy disk. It is development *7_newBuffer*. It was started in great hope that it would refine the Buffer model using the design pattern from the sender-receiver model, but does not jet analyse.

The *networkedBank* development has shown that any server-side development needs a wraper layer ontop of the business logic, and it has done it with a protocol for calling operations on the *Bank*

The implementation of the ATM system has made use of the idea of *networkedBank*. It shows that client-server programs are difficult to fully prove as the *ATM* implementation generates more than 1200 proof obligations. and *ServeBank* is poorly specified, but there is reason to believe that the implementation *ServeBankI* generates more than 2000 (!) proof obligations. When attempting to generete proof obligations for this implementation, the BToolkit went on for three hours making use of more than 500 MB of memory, before it finally crashed Red hat Linux. In contrast, generating proof obligations for the ATM implementation took about 120 MB of memory and took the toolkit about one hour.



An attempt to refine this system into an implementation should make the buffer model a starting point. This model should then be re-specified using either the Protocol structure or the sender-receiver structure.

This chapter has discussed a range of models and specifications which either allows the specifier to state some global invariants about the system, eg the buffer specification and the protocol development, and it has provided a model that allows for future refinements, ei the *SecureNetwork* development. But it has not been able to find a model that allows for both of these two functionalities at the same time. The *SecureNetwork* specification does not give a global view of communication. Neither the protocol development nor the buffer specification can easily be refined into an *ATM* and a *Bank* machine.

5.10 An attempt on Refining the Buffer specification

Provided on the floppy disk, as mentioned in last section, is an attempt on specifying the ATM system with *ATM*, *Bank*, *BankSystem* and a *Globals* machine. This model avoids the problem of INCLUDES by letting *ATM* and *Bank* see *Globals*. *Globals* states the invariant that no money is lost over the network layer. Hence it must hold the set *totalBalance*. *ATM* and *Bank* attempt to update *totalBalance* but is not allowed as variables of machines can only be modiefied by operations of that machine, which is obvosly needed to generated consistency proof obligations.

Chapter 6

Using the B-Toolkit

6.1 Executing Implementations; Creating Interfaces

An implementation may be executed by adding an interface of the machine specification. For this project non-motif interfaces have been choosen. This option can be made by choosing Options - interface - non-motif. Then the interface is created by choosing Introduce - new - Interfaec of Implemented Machine and choosing the specification. The development must be converted to C code, compiled and linked. This is done in the generators environment. The code is executed in the translators environment.

The interface has this format:
INTERFACE Arithmetic
OPERATIONS
exp_op
< list of operations >
END

For the implementation of the ATM system it is nessesary to set the option -DTEST_FLAG under Options - Translateors
Compilors menu, and choose the C Compilor/Flags line.

Chapter 7

Conclusion

7.1 Topics Covered by this Project

This project has covered three major topics.

- Number theory and cryptology
- Formal proofs applied to the BToolkit
- The B-Method and in particular an investigation into models using machine composing constructs.

In that order.

7.2 What to have in Mind when Writing a Specification

Writing specifications is very different to programming. Writing specifications require you to think carefully about what you are specifying. During this project I have found that I repetedly keept changing specifications to better model what I intend it to specify. When I program, I find myself often mindlessly hacking away at a problem because of some bug. When writing specifications you have to think "What do I want to say? Do these invariants express this in the best possible way?". Very often there is no yes-no answer to this. I have used tools to help you obtain an answer though. This is the animator functionality in the BToolkit and most importantly the provers environment.

When writing specifications two things should be in mind

- What functionality shall it present?
- What invariants need it preserve?

These two do not nessesarily conjoin. Maybe the invariant can be expressed but not proven or the invaraint cannot be expressed at a level. Finding the right place to state an invariant is well illustrated in the airtraffic control system discussed in [1], where *ATCSystem* expresses the invariant "all aircraft in a given airspace are controlled by the controller assigned to that airspace", expressed by
$\forall(acft, as).(acft \in aircraft \wedge as \in airspaces \wedge acft \in occupied_by(as) \Rightarrow acft \in controls(assigned(as)))$
in *ATCSystem* because neither *Aircraft* nor *Controller* has sufficient knowledge to express this.

7.3 Why is B a Difficult Language to Learn?

Firstly, anyone using B should have a firm knowledge of set theory and formal methods. Secondly, the the B syntax is not easily understood coming from a classical programming background. The B syntax has many small oddities. Eg. "-0 is not element in $\mathbb{N}$ and will give you a parse error, but it surely is defined as 0. The BToolkit also gives poor feedback on errors, eg. to show special rules of B is: operation:
status $\Leftarrow$ ATM_create_account =
$creq := creq \leftarrow 1$
It parses and analysises fine

status $\Leftarrow$ ATM_create_account =
$creq := creq \leftarrow 1 \;\;\|$
$status := TRUE$

does not. It should be modified to:

status $\Leftarrow$ ATM_create_account =

BEGIN $creq := creq \leftarrow 1$ $\|$ status := TRUE END

7.4 Final thoughts

This project has shown refinement of algorithms into code, proof of classical mathematics and logic in the BToolkit and that a development can be formaly proven. In the ATM development different patterns to express global invariants have been investigated. With the aditional goal of still allowing for refinement.

From the ATM development I would not recommend the B-Method for specifying and implementing transaction and concurrency systems. If we look at the components of the ATM system, we have two distinct systems, the ATM and the bank. They communicate, but in an implemention, where would we be able constrain that no money is lost over the network layer? There are other formalisems that have delved into this area, one is CSP.

I would say that the B-Method is suited for algorithm refinement, and for applications which manage and manipulate large amounts of data. The effort it takes to develop applications in B should make you carefully decide if the system is so safety-critical that it should use formal methods to verify its correctness. In some industries, formal methods is mandatory, like in the development of key-cards.

References

The following papers have been consulted during the project:

- [1] K. Lano, H. Haughton Specification in B An Introduction using the B Toolkit, Imperial College Press,1996. ISBN: 1-86094-008-0
- [2] S. Schneider the b-method an introduction, Palgrave, 2001. ISBN: 0-333-79284-X
- [3] J-R Abrial the b-book, assigning programs to meanings, Cambridge University Press, 1996. ISBN: 0-521-49619-5
- [4] E. Sekerinski, K. Sere program development by refinement case studies using the B method, Springer, 1999. ISBN: 1852330538
- [5] E. Dijkstra A disipline of Programming, Prentice Hall 1976, ISBN:
- [6] J. Seberry, J. Pieprzyk Cryptography an introdution to computer security, Prentice Hall, 1989. ISBN: 0-7248-0274-6
- [7] T. H. Corman, C. E Leiserson, R. L Rivest Introduction to Algorithms, The MIT Press, 1999. ISBN: 0-262-53091-0
- [8] F. Zeyda Investigation of the B Method, Diploma arbeit, University of Teesside, 2002
- [9] S. Schneider Concurrent and Rea-time Systems the CSP approach, Wiley, 1999
- [10] B. Forouzan Date Communications and Networking, McGraw-Hill, 1998. ISBN: 0-07-282294-5

- [11] Silberschatz, Galvin Operating System Concepts, Addison Wesley, 1997. ISBN: 0-201-59113-8
- [12] S. Dunne A Theory of Generalised Substitutaions. In Springer, editor, *ZB2002: Formal Specification and Develoment in Z and B*, Lecture Notes in Computer Science 2272, 2002
- [13] B-Core(UK) B-Toolkit User's Manual. Technical report, 1997

Appendix A

Cipher Development

MACHINE *Arithmetic*

CONSTANTS

$prime$, exp

PROPERTIES

$exp \in \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N} \wedge$

$\forall (aa , bb) . (aa \in \mathbb{N} \wedge bb \in \mathbb{N}_1 \Rightarrow exp (aa , bb) = exp (aa , bb - 1) \times aa \wedge$

$exp (aa , 1) = aa \wedge$

$exp (bb , 0) = 1 \wedge$

$exp (0 , bb) = 0) \wedge$

$prime \subseteq \mathbb{N}$

OPERATIONS

$rr \longleftarrow$ **exp_op** $(aa , bb) \quad \widehat{=}$

PRE

$aa \in \mathbb{N} \wedge$

```
        bb ∈ ℕ
    THEN
        rr := exp ( aa , bb )
    END

END


IMPLEMENTATION   ArithmeticI

REFINES

  Arithmetic

OPERATIONS

  rr ⟵ exp_op ( aa , bb )  ≙
    IF   aa ≠ 0 ∨ bb ≠ 0   THEN
      VAR
        ii ,
        kk
      IN
        ii := bb ;
        kk := 1 ;
        WHILE
          ii ≠ 0
        DO
          kk := kk × aa ;
          ii := ii − 1
        INVARIANT
```

$ii \in \mathbb{N} \wedge$

$kk = exp\ (\ aa\ ,\ bb - ii\ )$

VARIANT

ii

END ;

$rr := kk$

END

END

Cross-references

exp	*Arithmetic*	CONSTANTS	67

END

Cross-references for ArithmeticI

Arithmetic		MACHINE	68
exp	*Arithmetic*	CONSTANTS	67

MACHINE *Cipher* (*ee* , *dd* , *nn*)

CONSTRAINTS

$ee \in \mathbb{N} \wedge$

$dd \in \mathbb{N} \wedge$

$nn \in \mathbb{N}_1 \wedge$

$ee \times dd\ \mathsf{mod}\ (\ nn - 1\ ) = 1$

SEES

Arithmetic

CONSTANTS

encrypt , *decrypt*

PROPERTIES

$encrypt \in \mathbb{N} \to \mathbb{N} \wedge$

$decrypt \in \mathbb{N} \to \mathbb{N} \wedge$

$\forall\, mm \,.\, (mm \in \mathit{0} \,..\, nn - \mathit{1} \Rightarrow encrypt (decrypt (mm)) = mm) \wedge$

$\forall\, mm \,.\, (mm \in \mathit{0} \,..\, nn - \mathit{1} \Rightarrow decrypt (encrypt (mm)) = mm)$

OPERATIONS

$rr \longleftarrow$ **encrypt_op** (mm) $\widehat{=}$

PRE

$mm \in \mathit{0} \,..\, nn - \mathit{1}$

THEN

$rr := encrypt (mm)$

END ;

$rr \longleftarrow$ **decrypt_op** (mm) $\widehat{=}$

PRE

$mm \in \mathit{0} \,..\, nn - \mathit{1}$

THEN

$rr := decrypt (mm)$

END

END

Cross-references for Cipher

Arithmetic	MACHINE	68

REFINEMENT *ExponentialCipherRef*

REFINES

Cipher

SEES

Arithmetic

OPERATIONS

$rr \longleftarrow$ **encrypt_op** (mm) $\widehat{=}$

$rr := exp\ (\ mm\ ,\ ee\ )\ \mathsf{mod}\ nn$ **;**

Cross-references

exp	*Arithmetic*	CONSTANTS	67

$rr \longleftarrow$ **decrypt_op** (mm) $\widehat{=}$

$rr := exp\ (\ mm\ ,\ dd\ )\ \mathsf{mod}\ nn$

Cross-references

exp	*Arithmetic*	CONSTANTS	67

END

Cross-references for ExponentialCipherRef

Arithmetic	MACHINE	68

Cipher		MACHINE	159
exp	*Arithmetic*	CONSTANTS	67

IMPLEMENTATION *ExponentialCipherI*

REFINES

ExponentialCipherRef

IMPORTS

Arithmetic

OPERATIONS

$rr \longleftarrow$ **encrypt_op** (mm) $\widehat{=}$

VAR *tmp* **IN**

$tmp \longleftarrow exp\_op$ (mm , ee) **;**

$rr := tmp \ \mathsf{mod}\ nn$

END **;**

Cross-references

exp_op	*Arithmetic*	OPERATIONS	68

$rr \longleftarrow$ **decrypt_op** (mm) $\widehat{=}$

VAR *tmp* **IN**

$tmp \longleftarrow exp\_op$ (mm , dd) **;**

$rr := tmp \ \mathsf{mod}\ nn$

END

Cross-references

exp_op	*Arithmetic*	OPERATIONS	68

END

Cross-references for ExponentialCipherI

Arithmetic		MACHINE	68
ExponentialCipherRef		REFINEMENT	68
exp_op	*Arithmetic*	OPERATIONS	68

A.1 Proof Prints

Proofs (Level 0) for Arithmetic.mch

Context.1

1	$cst\ (\ Arithmetic\ )$	HYP
2	$\exists\ prime\ .\ (\ prime \subseteq \mathbb{N}\ )$	BToolUsersTheory.1
3	QED	DED

Laws (Level 0) for Arithmetic.mch

BToolUsersTheory.1

$\exists\ p\ .\ (\ p \subseteq \mathbb{N}\ )$

Proofs (Level 1) for ArithmeticI.imp

<u>*exp_op.2*</u>

1	$cst\ (\ ArithmeticI_1\ )$	HYP
2	$ctx\ (\ ArithmeticI_1\ )$	HYP
3	$inv\ (\ ArithmeticI_1\ )$	HYP
4	$asn\ (\ ArithmeticI_1\ )$	HYP
5	$pre\ (\ exp\_op\ )$	HYP
6	$aa \in \mathbb{N}$	5, HypExp.1
7	$\neg\ (\ aa = 0\ )$	HYP
8	$ii \in \mathbb{N}$	HYP
9	$kk = exp\ (\ aa\ ,\ bb - ii\ )$	HYP
10	$\neg\ (\ ii = 0\ )$	HYP
11	$exp\ (\ aa\ ,\ bb - ii + 1\ ) = exp\ (\ aa\ ,\ bb - ii + 1\ )$	EQL
12	$exp\ (\ aa\ ,\ bb - ii + 1\ ) = exp\ (\ aa\ ,\ bb - (\ ii - 1\ )\ )$	11, Law.1
13	$exp\ (\ aa\ ,\ bb - ii\ ) \times aa = exp\ (\ aa\ ,\ bb - (\ ii - 1\ )\ )$	6, 12, BToolUsersTheory.1
14	$kk \times aa = exp\ (\ aa\ ,\ bb - (\ ii - 1\ )\ )$	13, 9
15	QED	DED

<u>*exp_op.5*</u>

1	$cst\ (\ ArithmeticI_1\ )$	HYP
2	$ctx\ (\ ArithmeticI_1\ )$	HYP
3	$inv\ (\ ArithmeticI_1\ )$	HYP
4	$asn\ (\ ArithmeticI_1\ )$	HYP
5	$pre\ (\ exp\_op\ )$	HYP
6	$bb \in \mathbb{N}$	5, HypExp.2
7	$aa \in \mathbb{N}$	5, HypExp.1

8	$\neg\ (\ aa = \mathit{0}\ )$	HYP
9	$aa \in \mathbb{N}_1$	8, 7, Law.2
10	$exp\ (\ aa\ ,\ \mathit{0}\ ) = exp\ (\ aa\ ,\ \mathit{0}\ )$	EQL
11	$exp\ (\ aa\ ,\ \mathit{0}\ ) = exp\ (\ aa\ ,\ bb - bb\ )$	6, 10, Law.3
12	$\mathit{1} = exp\ (\ aa\ ,\ bb - bb\ )$	9, 11, BToolUsersTheory.2
13	*QED*	DED

<u>*exp_op.7*</u>

1	$cst\ (\ ArithmeticI_{\ 1}\ )$	HYP
2	$ctx\ (\ ArithmeticI_{\ 1}\ )$	HYP
3	$inv\ (\ ArithmeticI_{\ 1}\ )$	HYP
4	$asn\ (\ ArithmeticI_{\ 1}\ )$	HYP
5	$pre\ (\ exp\_op\ )$	HYP
6	$aa \in \mathbb{N}$	5, HypExp.1
7	$\neg\ (\ bb = \mathit{0}\ )$	HYP
8	$ii \in \mathbb{N}$	HYP
9	$kk = exp\ (\ aa\ ,\ bb - ii\ )$	HYP
10	$\neg\ (\ ii = \mathit{0}\ )$	HYP
11	$exp\ (\ aa\ ,\ bb - ii + \mathit{1}\ ) = exp\ (\ aa\ ,\ bb - ii + \mathit{1}\ )$	EQL
12	$exp\ (\ aa\ ,\ bb - ii + \mathit{1}\ ) = exp\ (\ aa\ ,\ bb - (\ ii - \mathit{1}\ )\ )$	11, Law.1
13	$exp\ (\ aa\ ,\ bb - ii\ ) \times aa = exp\ (\ aa\ ,\ bb - (\ ii - \mathit{1}\ )\ )$	6, 12, BToolUsersTheory.1
14	$kk \times aa = exp\ (\ aa\ ,\ bb - (\ ii - \mathit{1}\ )\ )$	13, 9
15	*QED*	DED

Laws (Level 1) for ArithmeticI.imp

HypExp.1

$pre\ (\ exp\_op\ )$

$\Rightarrow$

$aa \in \mathbb{N}$

HypExp.2

$pre\ (\ exp\_op\ )$

$\Rightarrow$

$bb \in \mathbb{N}$

BToolUsersTheory.1

$aa \in \mathbb{N}$

$\Rightarrow$

$exp\ (\ aa\ ,\ b\ ) \times aa \ \ \widehat{=}\ \ exp\ (\ aa\ ,\ b + 1\ )$

BToolUsersTheory.2

$aa \in \mathbb{N}_1$

$\Rightarrow$

$1 \ \ \widehat{=}\ \ exp\ (\ aa\ ,\ 0\ )$

Law.1 (RewriteAlgebra2X.8)

$c \in 0\ ..\ 2147483647$

$\Rightarrow$

$a - (\ b - c\ ) \ \ \widehat{=}\ \ a - b + c$

Law.2 (FwdInNat1X.17)

$\neg\ (\ n = 0\ ) \wedge n \in \mathbb{N}$

$\Rightarrow$

$n \in \mathbb{N}_1$

Law.3 (RewriteNat0X.2)

$a \in \mathbb{N}$

$\Rightarrow$

$a - a \ \widehat{=} \ 0$

Proofs (Level 0) for Cipher.mch

Constraints.1

1	$1 = 1$	EQL
2	$1 - 0 = 1$	1, ARI
3	$1 - 0 \times 2 = 1$	2, ARI
4	$1 - 1 / 2 \times 2 = 1$	3, ARI
5	$1 \ \mathsf{mod}\ 2 = 1$	4, Law.1
6	$1 \times 1 \ \mathsf{mod}\ 2 = 1$	5, ARI
7	$[\ ee := 1\ ]\ (\ ee \times 1 \ \mathsf{mod}\ 2 = 1\ )$	6, SUB
8	$\exists\ ee\ .\ (\ ee \in \mathbb{N} \land ee \times 1 \ \mathsf{mod}\ 2 = 1\ )$	7, Law.2
9	$\exists\ ee\ .\ [\ dd := 1\ ]\ (\ ee \in \mathbb{N} \land ee \times dd \ \mathsf{mod}\ 2 = 1\ )$	8, SUB
10	$\exists\ dd\ ,\ ee\ .\ ($	
	$dd \in \mathbb{N} \land$	
	$ee \in \mathbb{N} \land$	
	$ee \times dd \ \mathsf{mod}\ 2 = 1\ )$	9, Law.3
11	$\exists\ dd\ ,\ ee\ .\ ($	
	$dd \in \mathbb{N} \land$	
	$ee \in \mathbb{N} \land$	
	$ee \times dd \ \mathsf{mod}\ (\ 3 - 1\ ) = 1\ )$	10, ARI

12 $\exists\ dd\ ,\ ee\ .\ ($

$[\ nn := 3\ ]\ ($

$dd \in \mathbb{N} \wedge$

$ee \in \mathbb{N} \wedge$

$ee \times dd\ \ \mathsf{mod}\ (\ nn - 1\ ) = 1\ )$ 11, SUB

13 $\exists\ dd\ ,\ nn\ ,\ ee\ .\ ($

$dd \in \mathbb{N} \wedge$

$ee \in \mathbb{N} \wedge$

$nn \in \mathbb{N}_1\ \wedge$

$ee \times dd\ \ \mathsf{mod}\ (\ nn - 1\ ) = 1\ )$ 12, BToolUsersTheory.1

Laws (Level 0) for Cipher.mch

BToolUsersTheory.1

bsearch $(\ (\ a \in \mathbb{N}_1\ )\ ,\ b\ ,\ c\ ) \wedge$ bsearch $(\ a\ ,\ A\ ,\ B\ ) \wedge \exists\ B\ .\ [\ a := 3\ ]\ c$

$\Rightarrow$

$\exists\ A\ .\ b$

Law.1 (RewritePredicate0X.1)

$a \in 0\ ..\ 2147483647 \wedge b \in 0\ ..\ 2147483647 \wedge b > 0$

$\Rightarrow$

$a\ \ \mathsf{mod}\ b\ \ \widehat{=}\ \ a - a\ /\ b \times b$

Law.2 (Exist1X.15)

bsearch $(\ (\ a \in \mathbb{N}\ )\ ,\ b\ ,\ c\ ) \wedge [\ a := 1\ ]\ c$

$\Rightarrow$

$\exists\ a\ .\ b$

Law.3 (Exist1X.16)

bsearch $(\ (\ a \in \mathbb{N}\ )\ ,\ b\ ,\ c\ ) \wedge$ bsearch $(\ a\ ,\ A\ ,\ B\ ) \wedge \exists\ B\ .\ [\ a := 1\ ]\ c$

$\Rightarrow$

$\exists\ A\ .\ b$

Proofs (Level 0) for ExponentialCipherRef.ref

Context.1

1	$cst\ (\ ExponentialCipherRef\ _1\ )$	HYP
2	$nn \in \mathbb{N}_1$	1, HypExp.9
3	$ee \in \mathbb{N}$	1, HypExp.8
4	$dd \in \mathbb{N}$	1, HypExp.7
5	$ee \times dd\ \ \mathsf{mod}\ (\ nn - \mathit{1}\ ) = \mathit{1}$	1, HypExp.6
6	$nn \in \mathbb{N}$	2, Law.1
7	$ctx\ (\ Arithmetic\ )$	HYP
8	$k \in \mathbb{N}_1$	BToolUsersTheory.6
9	$k \in \mathbb{N}$	8, BToolUsersTheory.8
10	$\mathit{1} \in \mathbb{N}$	Law.2
11	$\mathit{1} \leq nn$	2, Law.3
12	$nn - \mathit{1} \in \mathbb{N}$	6, 10, 11, Law.4
13	$k \times (\ nn - \mathit{1}\ ) \in \mathbb{N}$	9, 12, Law.5
14	$\mathit{0} < \mathit{1}$	ARI
15	$\mathit{1} \in \mathbb{N}_1$	10, 14, Law.6
16	$\exists\ encrypt\ ,\ decrypt\ .\ (\ encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge \forall\ mm\ .\ (\ mm \in \mathit{0}\ ..\ nn - \mathit{1} \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	BToolUsersTheory.18

17 $\exists\ encrypt\ ,\ decrypt\ .\ ($

$encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$true\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt$ 16, Law.7

$(\ mm\ )\ ) = mm\ )\ )$

18 $\exists\ encrypt\ ,\ decrypt\ .\ ($

$encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$\forall\ mm\ .\ true\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt$ 17, Law.8

$(\ mm\ )\ ) = mm\ )\ )$

19 $\exists\ encrypt\ ,\ decrypt\ .\ ($

$encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow true\ )\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt$ 18, Law.9

$(\ mm\ )\ ) = mm\ )\ )$

20 $\exists\ encrypt\ ,\ decrypt\ .\ ($

$encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow mm = mm\ )\ \wedge$

$\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt$ 19, Law.10

$(\ mm\ )\ ) = mm\ )\ )$

21 $\exists\, encrypt\,,\, decrypt\,.\,($
$encrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$decrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow mm \ \mathsf{mod}\ nn = mm\,) \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow decrypt\,(\, encrypt\,(\, mm\,)\,) = mm\,)\,)$ 2, 20, BToolUsersTheory.17

22 $\exists\, encrypt\,,\, decrypt\,.\,($
$encrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$decrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow 1 \times mm \ \mathsf{mod}\ nn = mm\,) \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow decrypt\,(\, encrypt\,(\, mm\,)\,) = mm\,)\,)$ 21, Law.11

23 $\exists\, encrypt\,,\, decrypt\,.\,($
$encrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$decrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow 1 \ \mathsf{mod}\ nn \times mm \ \mathsf{mod}\ nn = mm\,) \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow decrypt\,(\, encrypt\,(\, mm\,)\,) = mm\,)\,)$ 2, 15, 22, BToolUsersTheory.14

24 $\exists\, encrypt\,,\, decrypt\,.\,($
$encrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$decrypt \in \mathbb{N} \to \mathbb{N} \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow 1 \ \mathsf{mod}\ nn \times mm \ \mathsf{mod}\ nn \ \mathsf{mod}\ nn = mm\,) \wedge$
$\forall\, mm\,.\,(\, mm \in 0\,..\,nn - 1 \Rightarrow decrypt\,(\, encrypt\,(\, mm\,)\,) = mm\,)\,)$ 2, 23, BToolUsersTheory.16

25	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ 1\ ,\ k\ )\ \mathsf{mod}\ nn \times mm\ \ \mathsf{mod}\ nn\ \ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	8, 24, BToolUsersTheory.15
26	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ exp\ (\ mm\ ,\ nn - 1\ )\ \ \mathsf{mod}\ nn\ ,\ k\ )\ \ \mathsf{mod}\ nn \times mm\ \ \mathsf{mod}\ nn\ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	2, 25, BToolUsersTheory.13
27	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ exp\ (\ mm\ ,\ nn - 1\ )\ ,\ k\ )\ \ \mathsf{mod}\ nn \times mm\ \ \mathsf{mod}\ nn\ \ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	12, 9, 2, 26, BToolUsersTheory.12

28	$\exists\, encrypt , decrypt . ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow exp (exp (mm , nn - \mathit{1}) , k) \times mm \ \mathsf{mod}\ nn = mm) \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow decrypt (encrypt (mm)) = mm))$	2, 27, BToolUsersTheory.11
29	$\exists\, encrypt , decrypt . ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow exp (mm , (nn - \mathit{1}) \times k) \times mm \ \mathsf{mod}\ nn = mm) \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow decrypt (encrypt (mm)) = mm))$	12, 9, 28, BToolUsersTheory.10
30	$\exists\, encrypt , decrypt . ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow exp (mm , k \times (nn - \mathit{1})) \times mm \ \mathsf{mod}\ nn = mm) \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow decrypt (encrypt (mm)) = mm))$	9, 12, 29, BToolUsersTheory.9
31	$\exists\, encrypt , decrypt . ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow exp (mm , k \times (nn - \mathit{1}) + \mathit{1}) \ \mathsf{mod}\ nn = mm) \wedge$ $\forall\, mm . (mm \in \mathit{0} .. nn - \mathit{1} \Rightarrow decrypt (encrypt (mm)) = mm))$	13, 30, BToolUsersTheory.7

32	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ mm\ ,\ dd \times ee\ )\ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	4, 3, 2, 5, 8, 31, BToolUsersTheory.5
33	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ exp\ (\ mm\ ,\ dd\ )\ ,\ ee\ )\ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	4, 3, 32, BToolUsersTheory.4
34	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow exp\ (\ exp\ (\ mm\ ,\ dd\ )\ \mathsf{mod}\ nn\ ,\ ee\ )\ \mathsf{mod}\ nn = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	3, 2, 33, BToolUsersTheory.3
35	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N}\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow encrypt\ (\ exp\ (\ mm\ ,\ dd\ )\ \mathsf{mod}\ nn\ ) = mm\ )\ \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	2, 3, 34, BToolUsersTheory.1

36	$\exists\ encrypt\ ,\ decrypt\ .\ ($ $encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow encrypt\ (\ decrypt\ (\ mm\ )\ ) = mm\ ) \wedge$ $\forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$	2, 4, 35, BToolUsers-Theory.2
37	*QED*	DED

Laws (Level 0) for ExponentialCipherRef.ref

<u>HypExp.9</u>

$cst\ (\ ExponentialCipherRef_{\ 1}\ )$

$\Rightarrow$

$nn \in \mathbb{N}_1$

<u>HypExp.8</u>

$cst\ (\ ExponentialCipherRef_{\ 1}\ )$

$\Rightarrow$

$ee \in \mathbb{N}$

<u>HypExp.7</u>

$cst\ (\ ExponentialCipherRef_{\ 1}\ )$

$\Rightarrow$

$dd \in \mathbb{N}$

<u>HypExp.6</u>

$cst\ (\ ExponentialCipherRef_{\ 1}\ )$

$\Rightarrow$

$ee \times dd\ \ \mathsf{mod}\ (\ nn - 1\ ) = 1$

<u>BToolUsersTheory.6</u>

$k \in \mathbb{N}_1$

<u>BToolUsersTheory.8</u>

$k \in \mathbb{N}_1$

$\Rightarrow$

$k \in \mathbb{N}$

<u>BToolUsersTheory.18</u>

$\exists\ (\ encrypt\ ,\ decrypt\ )\ .\ (\ encrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge decrypt \in \mathbb{N} \rightarrow \mathbb{N} \wedge \forall\ mm\ .\ (\ mm \in 0\ ..\ nn - 1 \Rightarrow decrypt\ (\ encrypt\ (\ mm\ )\ ) = mm\ )\ )$

<u>BToolUsersTheory.17</u>

$n \in \mathbb{N}_1$

$\Rightarrow$

$mm\ \mathsf{mod}\ n\ \ \widehat{=}\ \ mm$

<u>BToolUsersTheory.14</u>

$n \in \mathbb{N}_1\ \wedge a \in \mathbb{N}_1$

$\Rightarrow$

$a\ \mathsf{mod}\ n \times b\ \mathsf{mod}\ n\ \ \widehat{=}\ \ a \times b\ \mathsf{mod}\ n$

<u>BToolUsersTheory.16</u>

$n \in \mathbb{N}_1$

$\Rightarrow$

$a\ \mathsf{mod}\ n\ \mathsf{mod}\ n\ \ \widehat{=}\ \ a\ \mathsf{mod}\ n$

<u>BToolUsersTheory.15</u>

$a \in \mathbb{N}_1$

$\Rightarrow$

$exp\ (\ 1\ ,\ a\ )\ \ \widehat{=}\ \ 1$

BToolUsersTheory.13

$n \in \mathbb{N}_1$

$\Rightarrow$

$exp\ (\ a\ ,\ n - 1\ )\ \ \mathsf{mod}\ n\ \ \widehat{=}\ \ 1$

BToolUsersTheory.12

$b \in \mathbb{N} \wedge c \in \mathbb{N} \wedge n \in \mathbb{N}_1$

$\Rightarrow$

$exp\ (\ exp\ (\ a\ ,\ b\ )\ ,\ c\ )\ \ \mathsf{mod}\ n\ \ \widehat{=}\ \ exp\ (\ exp\ (\ a\ ,\ b\ )\ \ \mathsf{mod}\ n\ ,\ c\ )\ \ \mathsf{mod}\ n$

BToolUsersTheory.11

$n \in \mathbb{N}_1$

$\Rightarrow$

$a \times b\ \ \mathsf{mod}\ n\ \ \widehat{=}\ \ a\ \ \mathsf{mod}\ n \times b\ \ \mathsf{mod}\ n\ \ \mathsf{mod}\ n$

BToolUsersTheory.10

$b \in \mathbb{N} \wedge c \in \mathbb{N}$

$\Rightarrow$

$exp\ (\ a\ ,\ b \times c\ )\ \ \widehat{=}\ \ exp\ (\ exp\ (\ a\ ,\ b\ )\ ,\ c\ )$

BToolUsersTheory.9

$a \in \mathbb{N} \wedge b \in \mathbb{N}$

$\Rightarrow$

$a \times b\ \ \widehat{=}\ \ b \times a$

BToolUsersTheory.7

$b \in \mathbb{N}$

$\Rightarrow$

$exp\ (\ a\ ,\ b + 1\ )\ \ \widehat{=}\ \ exp\ (\ a\ ,\ b\ ) \times a$

BToolUsersTheory.5

$e \in \mathbb{N} \wedge d \in \mathbb{N} \wedge n \in \mathbb{N}_1 \wedge d \times e \ \mathsf{mod}\ (n - 1) = 1 \wedge k \in \mathbb{N}_1$

$\Rightarrow$

$e \times d \ \widehat{=}\ k \times (n - 1) + 1$

<u>BToolUsersTheory.4</u>

$b \in \mathbb{N} \wedge c \in \mathbb{N}$

$\Rightarrow$

$exp (exp (a , b) , c) \ \widehat{=}\ exp (a , b \times c)$

<u>BToolUsersTheory.3</u>

$b \in \mathbb{N} \wedge n \in \mathbb{N}_1$

$\Rightarrow$

$exp (a \ \mathsf{mod}\ n , b) \ \mathsf{mod}\ n \ \widehat{=}\ exp (a , b) \ \mathsf{mod}\ n$

<u>BToolUsersTheory.1</u>

$n \in \mathbb{N}_1 \wedge e \in \mathbb{N}$

$\Rightarrow$

$encrypt (x) \ \widehat{=}\ exp (x , e) \ \mathsf{mod}\ n$

<u>BToolUsersTheory.2</u>

$n \in \mathbb{N}_1 \wedge dd \in \mathbb{N}$

$\Rightarrow$

$decrypt (x) \ \widehat{=}\ exp (x , dd) \ \mathsf{mod}\ n$

<u>Law.1 (FwdInNat1X.19)</u>

$n \in \mathbb{N}_1$

$\Rightarrow$

$n \in \mathbb{N}$

<u>Law.2 (InNatX.25)</u>

$n \in 0 \ .. \ 2147483647$

$\Rightarrow$

$n \in \mathbb{N}$

Law.3 (LessThanOrEqualX.74)

$x \in \mathbb{N}_1$

$\Rightarrow$

$1 \leq x$

Law.4 (InNatX.20)

$n \in \mathbb{N} \wedge p \in \mathbb{N} \wedge p \leq n$

$\Rightarrow$

$n - p \in \mathbb{N}$

Law.5 (InNatX.23)

$n \in \mathbb{N} \wedge p \in \mathbb{N}$

$\Rightarrow$

$n \times p \in \mathbb{N}$

Law.6 (InNatX.10)

$n \in \mathbb{N} \wedge 0 < n$

$\Rightarrow$

$n \in \mathbb{N}_1$

Law.7 (RewriteNonHypLogic2X.3)

$(a \wedge true) \;\; \widehat{=} \;\; a$

Law.8 (RewriteToFalseOrTrueX.14)

$\forall a \; . \; true \;\; \widehat{=} \;\; true$

Law.9 (RewriteToFalseOrTrueX.15)

$(a \Rightarrow true) \;\; \widehat{=} \;\; true$

Law.10 (RewritePredicate1X.9)

$a = a \;\; \widehat{=} \;\; true$

Law.11 (RewriteNat0X.6)

$1 \times n \;\; \widehat{=} \;\; n$

Appendix B

Airtraffic Control System

B.1 New Airtraffic Control System

MACHINE *Aircraft* (*maxaircraft*)

SETS

$AIRCRAFT$; $STRINGS$

PROPERTIES

$\mathsf{card}\ (\ AIRCRAFT\ ) = maxaircraft$

VARIABLES

$aircraft$, $flight\_id$

INVARIANT

$maxaircraft \in \mathbb{N}_1 \quad \wedge$

$aircraft \subseteq AIRCRAFT \wedge$

$flight\_id \in aircraft \to STRINGS$

INITIALISATION

$aircraft\ , flight\_id := \{\}\ , \{\}$

OPERATIONS

$aa \longleftarrow$ **create_aircraft** (fid) $\widehat{=}$

PRE

$fid \in STRINGS \wedge$

$aircraft \neq AIRCRAFT$

THEN

ANY oo

WHERE $oo \in AIRCRAFT - aircraft$

THEN

$aa := oo \ \|$

$aircraft := aircraft \cup \{ \ oo \ \} \ \|$

$flight\_id \ (\ oo \ ) := fid$

END

END **;**

$fid \longleftarrow$ **id_of** (aa) $\widehat{=}$

PRE

$aa \in aircraft$

THEN

$fid := flight\_id \ (\ aa \ )$

END

END

MACHINE $Airspace \ (\ maxairspace \ )$

USES

$Aircraft \ , \ Controller$

SETS

$AIRSPACE$

PROPERTIES

$\mathsf{card} \ (\ AIRSPACE \ ) = maxairspace$

VARIABLES

$airspaces \ , \ maxholding \ , \ occupied\_by \ , \ assigned \ ,$

$airport\_zones \ , \ military\_zones \ , \ city\_regions$

INVARIANT

$airspaces \subseteq AIRSPACE \ \wedge$

$airport\_zones \subseteq airspaces \ \wedge$

$military\_zones \subseteq airspaces \ \wedge$

$city\_regions \subseteq airspaces \ \wedge$

$airport\_zones \cup military\_zones \cup city\_regions = airspaces \ \wedge$

$airport\_zones \cap military\_zones = \{\} \ \wedge$

$airport\_zones \cap city\_regions = \{\} \ \wedge$

$military\_zones \cap city\_regions = \{\} \ \wedge$

$maxholding \in airspaces \rightarrow \mathbb{N} \ \wedge$

$occupied\_by \in airspaces \rightarrow \mathbb{F} \ (\ aircraft \ ) \ \wedge$

$assigned \in airspaces \rightarrowtail controllers \ \wedge$

$\forall \ (\ as1 \ , \ as2 \ ) \ . \ (\ as1 \in airspaces \wedge as2 \in airspaces \ \wedge$

$as1 \neq as2 \Rightarrow$

$occupied\_by \ (\ as1 \ ) \cap occupied\_by \ (\ as2 \ ) = \{\} \ ) \ \wedge$

$\forall\ as\ .\ (\ as \in airspaces \Rightarrow \mathsf{card}\ (\ occupied\_by\ (\ as\ )\ ) \leq maxholding\ (\ as\ )\ )$

Cross-references

aircraft	*Aircraft*	VARIABLES	96
controllers	*Controller*	VARIABLES	92

INITIALISATION

$airspaces := \{\}\ \ \|\ \ maxholding := \{\}\ \ \|\ \ occupied\_by := \{\}\ \ \|$

$assigned := \{\}\ \ \|\ \ city\_regions := \{\}\ \ \|\ \ military\_zones := \{\}\ \ \|$

$airport\_zones := \{\}$

OPERATIONS

$as \longleftarrow$ **create_airspace** $(\ maxacft\ ,\ cont\ )\ \ \widehat{=}$

PRE

$maxacft \in \mathbb{N}\ \wedge$

$cont \in controllers\ \wedge$

$airspaces \neq AIRSPACE$

THEN

ANY oo

WHERE $oo \in AIRSPACE - airspaces$

THEN

$as := oo\ \ \|$

$maxholding\ (\ oo\ ) := maxacft\ \ \|$

$occupied\_by\ (\ oo\ ) := \{\}\ \ \|$

$assigned\ (\ oo\ ) := cont\ \ \|$

$airspaces := airspaces \cup \{\ oo\ \}\ \ \|$

$airport\_zones := airport\_zones \cup \{\ oo\ \}$

END

END ;

Cross-references

controllers	*Controller*	VARIABLES	92

do_aircraft_arrives (*as* , *acft*) $\widehat{=}$

PRE

$acft \in aircraft \land$

$as \in airspaces \land$

$acft \notin \mathsf{union}\ (\ \mathsf{ran}\ (\ occupied\_by\ )\ ) \land$

$acft \notin \mathsf{union}\ (\ \mathsf{ran}\ (\ controls\ )\ ) \land$

$\mathsf{card}\ (\ occupied\_by\ (\ as\ )\ ) < maxholding\ (\ as\ )$

THEN

$occupied\_by\ (\ as\ ) := occupied\_by\ (\ as\ ) \cap \{\ acft\ \}$

END ;

Cross-references

aircraft	*Aircraft*	VARIABLES	96
controls	*Controller*	VARIABLES	96

transfer_aircraft (*as1* , *as2* , *acft*) $\widehat{=}$

PRE

$as1 \in airspaces \land as2 \in airspaces \land acft \in aircraft \land$

$as1 \neq as2 \land$

$acft \in occupied\_by\ (\ as1\ )$

THEN

$occupied\_by := occupied\_by \mathbin{\triangleright\!\!\!>} \{\ as1 \mapsto occupied\_by\ (\ as1\ ) - \{\ acft\ \}\ ,$

$as2 \mapsto (\ occupied\_by\ (\ as2\ ) \cup \{\ acft\ \}\ )\ \}$

END

Cross-references

aircraft	*Aircraft*	VARIABLES	96

END

Cross-references for Airspace

aircraft	*Aircraft*	VARIABLES	96
Aircraft		MACHINE	95
Controller		MACHINE	95
controllers	*Controller*	VARIABLES	92
controls	*Controller*	VARIABLES	96

MACHINE *Controller* (*maxcontroller*)

USES

Aircraft

SETS

CONTROLLER

PROPERTIES

$\mathsf{card}\ (\ CONTROLLER\ ) = maxcontroller$

VARIABLES

controllers , *controls*

INVARIANT

$controllers \subseteq CONTROLLER \wedge$

$controls \in controllers \rightarrow \mathbb{F}\ (\ aircraft\ )\ \wedge$

$\forall\ (\ cc1\ ,\ cc2\ )\ .\ (\ cc1 \in controllers\ \wedge$

$cc2 \in controllers \wedge cc1 \neq cc2 \Rightarrow$

$controls\ (\ cc1\ ) \cap controls\ (\ cc2\ ) = \{\}\ )$

Cross-references

aircraft	*Aircraft*	VARIABLES	96

INITIALISATION

$controllers := \{\}\ \parallel\ controls := \{\}$

OPERATIONS

$cc \longleftarrow$ **create_controller** $\widehat{=}$

PRE

$controllers \neq CONTROLLER$

THEN

ANY oo

WHERE

$oo \in CONTROLLER - controllers$

THEN

$controllers := controllers \cup \{\ oo\ \}\ \parallel$

$cc := oo\ \parallel$

$controls\ (\ oo\ ) := \{\}$

END

END **;**

hand_over $(\ cc1\ ,\ cc2\ ,\ acft\ )\ \widehat{=}$

PRE

$cc1 \in controllers \wedge cc2 \in controllers \wedge$

$cc1 \neq cc2 \wedge$

$acft \in aircraft \wedge$

$acft \in controls (cc1)$

THEN

$controls := controls \lhd\!\!- \{ cc1 \mapsto controls (cc1) - \{ acft \} ,$

$cc2 \mapsto (controls (cc2) \cup \{ acft \}) \}$

END ;

Cross-references

aircraft	*Aircraft*	VARIABLES	96

add_aircraft (cc , $acft$) $\widehat{=}$

PRE

$cc \in controllers \wedge$

$acft \in aircraft \wedge$

$\neg (acft \in \mathsf{union} (\mathsf{ran} (controls)))$

THEN

$controls (cc) := controls (cc) \cap \{ acft \}$

END

Cross-references

aircraft	*Aircraft*	VARIABLES	96

END

Cross-references for Controller

aircraft	*Aircraft*	VARIABLES	96

MACHINE $ATCSystem\ (\ maxairspace\ ,\ maxaircraft\ )$

EXTENDS

$Aircraft\ (\ maxaircraft\ )$

INCLUDES

$Airspace\ (\ maxairspace\ )\ ,\ Controller\ (\ maxairspace \times 2\ )$

PROMOTES

$create\_controller\ ,\ create\_airspace$

INVARIANT

$\forall\ (\ acft\ ,\ as\ )\ .\ (\ acft \in aircraft \wedge as \in airspaces \wedge$

$acft \in occupied\_by\ (\ as\ ) \Rightarrow$

$acft \in controls\ (\ assigned\ (\ as\ )\ )\ )$

Cross-references

OPERATIONS

hand_over_aircraft $(\ as1\_\ ,\ as2\_\ ,\ acft\_\ )\ \ \widehat{=}$

PRE

$as1\_ \in airspaces \wedge as2\_ \in airspaces \wedge$

$acft\_ \in aircraft \wedge$

$as1\_ \neq as2\_ \wedge$

$acft\_ \in occupied\_by (as1\_) \wedge$

$\mathsf{card} (occupied\_by (as2\_)) < maxholding (as2\_)$

THEN

$hand\_over (assigned (as1\_) , assigned (as2\_) , acft\_) \parallel$

$transfer\_aircraft (as1\_ , as2\_ , acft\_)$

END ;

Cross-references

aircraft	*Aircraft*	VARIABLES	96
airspaces	*Airspace*	VARIABLES	96
assigned	*Airspace*	VARIABLES	97
hand_over	*Controller*	OPERATIONS	96
maxholding	*Airspace*	VARIABLES	96
occupied_by	*Airspace*	VARIABLES	96
transfer_aircraft	*Airspace*	OPERATIONS	96

aircraft_arrives $(as , acft) \quad \widehat{=}$

PRE

$acft \in aircraft \wedge$

$as \in airspaces \wedge$

$acft \notin \mathsf{union} (\mathsf{ran} (occupied\_by)) \wedge$

$acft \notin \mathsf{union} (\mathsf{ran} (controls)) \wedge$

$\mathsf{card} (occupied\_by (as)) < maxholding (as)$

THEN

$do\_aircraft\_arrives (as , acft) \parallel$

add_aircraft (*assigned* (*as*) , *acft*)

END

Cross-references

add_aircraft	*Controller*	OPERATIONS	97
aircraft	*Aircraft*	VARIABLES	96
airspaces	*Airspace*	VARIABLES	96
assigned	*Airspace*	VARIABLES	97
controls	*Controller*	VARIABLES	96
do_aircraft_arrives	*Airspace*	OPERATIONS	96
maxholding	*Airspace*	VARIABLES	96
occupied_by	*Airspace*	VARIABLES	96

END

Cross-references for ATCSystem

add_aircraft	*Controller*	OPERATIONS	97
aircraft	*Aircraft*	VARIABLES	96
Aircraft		MACHINE	95
Airspace		MACHINE	95
airspaces	*Airspace*	VARIABLES	96
assigned	*Airspace*	VARIABLES	97
Controller		MACHINE	95
controls	*Controller*	VARIABLES	96
create_airspace	*Airspace*	OPERATIONS	95
create_controller	*Controller*	OPERATIONS	95
do_aircraft_arrives	*Airspace*	OPERATIONS	96
hand_over	*Controller*	OPERATIONS	96
maxholding	*Airspace*	VARIABLES	96
occupied_by	*Airspace*	VARIABLES	96

Appendix C

ATM Development

C.1 A Client-Server Protocol

C.1.1 Communication Protocol

MACHINE *Protocol*

SEES

$Bool\_TYPE$, $CommsDefs$

INCLUDES

$Client$, $Server$,

$client$. $Buffer$ ($MESSAGES$, $buffsize$) ,

$server$. $Buffer$ ($MESSAGES$, $buffsize$)

VARIABLES

p_state , $report$

INVARIANT

$p\_state \in STATE \wedge$

$report \in BOOL$

Cross-references

BOOL	*Bool_TYPE*	SETS	159
STATE	*CommsDefs*	SETS	127

INITIALISATION

$p\_state\ ,\ report := idle\ ,\ FALSE$

OPERATIONS

pinit $\widehat{=}$

BEGIN

$p\_state := idle\ \ \|$

$client\ .\ binit\ \ \|$

$server\ .\ binit$

END ;

Cross-references

binit	*Buffer*	OPERATIONS	111

$ok\_protocol\ ,\ ok\_client\ ,\ ok\_buffer \longleftarrow$ **pcon_request** $\widehat{=}$

IF $p\_state = idle$ **THEN**

$p\_state := conpending\ \ \|$

$ok\_protocol := TRUE\ \ \|$

$ok\_client \longleftarrow client\_con\_request\ \ \|$

$ok\_buffer \longleftarrow client\ .\ add\ (\ con\ )$

ELSE $ok\_protocol := FALSE$

END **;**

Cross-references

add	*Buffer*	OPERATIONS	112
client_con_request	*Client*	OPERATIONS	111

$ok\_protocol \ , \ ok\_server \ , \ ok\_buffer \longleftarrow$ **pcon_in** $\widehat{=}$

IF $p\_state = conpending$ **THEN**

$p\_state := conrequest \ \|$

$ok\_protocol := TRUE \ \|$

$ok\_server \longleftarrow conin \ \|$

$ok\_buffer \longleftarrow client \ . \ remove$

ELSE $ok\_protocol := FALSE$

END **;**

Cross-references

conin	*Server*	OPERATIONS	111
remove	*Buffer*	OPERATIONS	113

$ok\_protocol \ , \ ok\_server \ , \ ok\_buffer \longleftarrow$ **pcon_report** $\widehat{=}$

IF $p\_state = conrequest$ **THEN**

$p\_state := have\_report \ \|$

$report \longleftarrow server\_con\_report \ \|$

$ok\_protocol := report \ \|$

$ok\_server := report \ \|$

IF $report = TRUE$ **THEN**

$ok\_buffer \longleftarrow server \ . \ add \ (\ true \ )$

ELSE $ok\_buffer \longleftarrow server \ . \ add \ (\ false \ )$

END

ELSE $ok\_protocol := FALSE$

END **;**

Cross-references

add	*Buffer*	OPERATIONS	112
server_con_report	*Server*	OPERATIONS	112

$ok\_protocol$, ok_client , $ok\_buffer \longleftarrow$ **pcon_give_report** $\widehat{=}$

BEGIN

$ok\_client \longleftarrow client\_con\_report (report)$ $\|$

IF $p\_state = have\_report$ **THEN**

IF $report = TRUE$ **THEN**

$p\_state := connected$

ELSE

$sinit$ $\|$

$p\_state := idle$

END $\|$

$ok\_protocol := TRUE$ $\|$

$ok\_buffer \longleftarrow server . remove$

ELSE $ok\_protocol := FALSE$

END

END **;**

Cross-references

client_con_report	*Client*	OPERATIONS	112
remove	*Buffer*	OPERATIONS	113
sinit	*Server*	OPERATIONS	112

$ps \longleftarrow$ **prot_state** $\widehat{=}$

$ps := p\_state$

END

Cross-references for Protocol

MACHINE *Server*

SEES

Bool_TYPE , *CommsDefs*

VARIABLES

sstate

INVARIANT

$sstate \in STATE$

Cross-references

STATE	*CommsDefs*	SETS	127

INITIALISATION

$sstate := idle$

OPERATIONS

sinit $\widehat{=}$

$sstate := idle$ **;**

$ok \longleftarrow$ **conin** $\widehat{=}$

IF $sstate = idle$ **THEN**

$sstate := conrequest$ $\|$

$ok := TRUE$

ELSE $ok := FALSE$

END **;**

$msg \longleftarrow$ **server_con_report** $\widehat{=}$

IF $sstate = conrequest$ **THEN**

ANY pp

WHERE $pp \in BOOL$ **THEN**

IF $pp = TRUE$ **THEN**

$sstate := connected$ $\|$

$msg := TRUE$

ELSE

$sstate := idle \quad \|$

$msg := FALSE$

END

END

END

Cross-references

BOOL	*Bool_TYPE*	SETS	159

END

Cross-references for Server

BOOL	*Bool_TYPE*	SETS	159
Bool_TYPE		MACHINE	166
CommsDefs		MACHINE	127
STATE	*CommsDefs*	SETS	127

MACHINE *Client*

SEES

Bool_TYPE , *CommsDefs*

VARIABLES

clientstate

INVARIANT

$clientstate \in STATE$

Cross-references

STATE	*CommsDefs*	SETS	127

INITIALISATION

$clientstate := idle$

OPERATIONS

cinit $\widehat{=}$

$clientstate := idle$ **;**

$ok \longleftarrow$ **client_con_request** $\widehat{=}$

IF $clientstate = idle$ **THEN**

$clientstate := conpending$ $\|$

$ok := TRUE$

ELSE $ok := FALSE$

END **;**

$ok \longleftarrow$ **client_con_report** ($response \in BOOL$) $\widehat{=}$

IF $clientstate = conpending$ **THEN**

IF $response = TRUE$ **THEN**

$clientstate := connected$

ELSE $clientstate := idle$

END $\|$

$ok := TRUE$

ELSE $ok := FALSE$

END

Cross-references

BOOL	*Bool_TYPE*	SETS	159

END

Cross-references for Client

BOOL	*Bool_TYPE*	SETS	159
Bool_TYPE		MACHINE	166
CommsDefs		MACHINE	127
STATE	*CommsDefs*	SETS	127

MACHINE *Buffer* (*ITEM* , *maxsize*)

SEES

Bool_TYPE

VARIABLES

contents

INVARIANT

$contents \in \mathsf{seq}\ (\ ITEM\ ) \wedge$

$\mathsf{size}\ (\ contents\ ) \leq maxsize$

INITIALISATION

$contents := [\ ]$

OPERATIONS

binit $\widehat{=}$

BEGIN

```
        contents := [ ]
    END ;

ok ⟵ add ( ee ) ≙
    PRE
        ee ∈ ITEM
    THEN
        IF size ( contents ) < maxsize
        THEN
            contents := contents ← ee ||
            ok := TRUE
        ELSE
            ok := FALSE
        END
    END ;

ok , oo ⟵ initial ≙
    IF contents ≠ [ ]
    THEN
        oo := first ( contents ) ||
        ok := TRUE
    ELSE
        oo :∈ ITEM ||
        ok := FALSE
    END ;

ok ⟵ remove ≙
    IF contents ≠ [ ]
    THEN
        contents := tail ( contents ) ||
```

ok := *TRUE*

ELSE

ok := *FALSE*

END

END

Cross-references for Buffer

Bool_TYPE	MACHINE	166

MACHINE *CommsDefs*

SETS

STATE = { *idle* , *conpending* , *conrequest* , *have_report* , *connected* } **;**

MESSAGES = { *con* , *true* , *false* }

CONSTANTS

buffsize

PROPERTIES

buffsize = *100*

END

C.1.2 Extension of the Protocol

MACHINE

Protocol

SEES

Bool_TYPE , *CommsDefs* , *Globals*

INCLUDES

Client , *Server* ,

client . *Buffer* (*MESSAGES* , *buffsize*) ,

server . *Buffer* (*MESSAGES* , *buffsize*)

VARIABLES

p_state , *report*

INVARIANT

$p\_state \in STATE \land$

$report \in BOOL$

Cross-references

BOOL	*Bool_TYPE*	SETS	159
STATE	*CommsDefs*	SETS	127

INITIALISATION

$p\_state , report := idle , FALSE$

OPERATIONS

pinit $\widehat{=}$

BEGIN

$p\_state := idle \;\|$

$client . binit \;\|$

$server \ . \ binit$

END ;

Cross-references

binit	*Buffer*	OPERATIONS	111

$ok\_protocol \ , \ ok\_client \ , \ ok\_buffer \longleftarrow$ **pcon_request** $\widehat{=}$

IF $p\_state = idle$ **THEN**

$p\_state := conpending \ \|$

$ok\_protocol := TRUE \ \|$

$ok\_client \longleftarrow client\_con\_request \ \|$

$ok\_buffer \longleftarrow client \ . \ add \ (\ con \ )$

ELSE $ok\_protocol := FALSE$

END ;

Cross-references

add	*Buffer*	OPERATIONS	112
client_con_request	*Client*	OPERATIONS	111

$ok\_protocol \ , \ ok\_server \ , \ ok\_buffer \longleftarrow$ **pcon_in** $\widehat{=}$

IF $p\_state = conpending$ **THEN**

$p\_state := conrequest \ \|$

$ok\_protocol := TRUE \ \|$

$ok\_server \longleftarrow conin \ \|$

$ok\_buffer \longleftarrow client \ . \ remove$

ELSE $ok\_protocol := FALSE$

END ;

Cross-references

conin	*Server*	OPERATIONS	111

remove	*Buffer*	OPERATIONS	113

$ok\_protocol$, ok_server , $ok\_buffer \longleftarrow$ **pcon_report** $\widehat{=}$

IF $p\_state = conrequest$ **THEN**

$p\_state := have\_report$ $\parallel$

$report \longleftarrow server\_con\_report$ $\parallel$

$ok\_protocol := report$ $\parallel$

$ok\_server := report$ $\parallel$

IF $report = TRUE$ **THEN**

$ok\_buffer \longleftarrow server . add (true)$

ELSE $ok\_buffer \longleftarrow server . add (false)$

END

ELSE $ok\_protocol := FALSE$

END ;

Cross-references

add	*Buffer*	OPERATIONS	112
server_con_report	*Server*	OPERATIONS	112

$ok\_protocol$, ok_client , $ok\_buffer \longleftarrow$ **pcon_give_report** $\widehat{=}$

BEGIN

$ok\_client \longleftarrow client\_con\_report (report)$ $\parallel$

IF $p\_state = have\_report$ **THEN**

IF $report = TRUE$ **THEN**

$p\_state := connected$

ELSE

$sinit$ $\parallel$

$p\_state := idle$

END $\parallel$

$ok\_protocol := TRUE \ \ \|$

$ok\_buffer \longleftarrow server\ .\ remove$

ELSE $ok\_protocol := FALSE$

END

END ;

Cross-references

client_con_report	*Client*	OPERATIONS	112
remove	*Buffer*	OPERATIONS	113
sinit	*Server*	OPERATIONS	112

$ps \longleftarrow$ **prot_state** $\widehat{=}$

$ps := p\_state$;

$aa \longleftarrow$ **pcreate_acc** $\widehat{=}$

PRE

$p\_state = connected$

THEN

$aa \longleftarrow screate\_acc$

END ;

Cross-references

screate_acc	*Server*	OPERATIONS	113

$new \longleftarrow$ **pdeposit** $(acc \in \mathbb{N} \wedge amount \in \mathbb{N})$ $\widehat{=}$

PRE

$p\_state = connected$

THEN

$new \longleftarrow sdeposit\ (\ acc\ ,\ amount\ )$

END ;

Cross-references

sdeposit	*Server*	OPERATIONS	113

$ans \longleftarrow$ **pwithdraw** $(\; acc \in \mathbb{N}_1 \;\; \wedge \; amount \in \mathbb{N}_1 \;\;) \quad \widehat{=}$

PRE

$p\_state = connected$

THEN

$ans \longleftarrow swithdraw \; (\; acc \; , \; amount \;)$

END ;

Cross-references

swithdraw	*Server*	OPERATIONS	114

$rr \longleftarrow$ **pisaccount** $(\; acc \in \mathbb{N} \;) \quad \widehat{=}$

PRE

$p\_state = connected$

THEN

$rr \longleftarrow sisaccount \; (\; acc \;)$

END ;

Cross-references

sisaccount	*Server*	OPERATIONS	114

$rr \longleftarrow$ **query_pstate** $\quad \widehat{=} \quad rr := p\_state$

END

Cross-references for Protocol

add	*Buffer*	OPERATIONS	112

MACHINE

Server

SEES

Bool_TYPE , *CommsDefs* , *Globals*

INCLUDES

Bank (*10*)

VARIABLES

sstate

INVARIANT

$sstate \in STATE$

Cross-references

STATE	*CommsDefs*	SETS	127

INITIALISATION

$sstate := idle$

OPERATIONS

sinit $\widehat{=}$

$sstate := idle$ **;**

$ok \longleftarrow$ **conin** $\widehat{=}$

IF $sstate = idle$ **THEN**

$sstate := conrequest$ $\|$

$ok := TRUE$

ELSE $ok := FALSE$

END **;**

$msg \longleftarrow$ **server_con_report** $\widehat{=}$

IF $sstate = conrequest$ **THEN**

ANY pp

WHERE $pp \in BOOL$ **THEN**

IF $pp = TRUE$ **THEN**

$sstate := connected \ \|$

$msg := TRUE$

ELSE

$sstate := idle \ \|$

$msg := FALSE$

END

END

END ;

Cross-references

$BOOL$	$Bool\_TYPE$	SETS	159

$ans \longleftarrow$ **screate_acc** $\widehat{=}$

PRE

$sstate = connected$

THEN

$ans \longleftarrow create\_acc$

END ;

Cross-references

$create\_acc$	$Bank$	OPERATIONS	117

$new \longleftarrow$ **sdeposit** $(acc \in \mathbb{N} \wedge amount \in \mathbb{N})$ $\widehat{=}$

PRE

$sstate = connected$

THEN

$new \longleftarrow deposit (acc , amount)$

END ;

Cross-references

deposit	*Bank*	OPERATIONS	161

$ans \longleftarrow$ **swithdraw** $(\; acc \; , \; amount \;) \quad \widehat{=}$

PRE

$acc \in \mathbb{N} \wedge amount \in \mathbb{N} \wedge$

$sstate = connected$

THEN

$ans \longleftarrow withdraw \; (\; acc \; , \; amount \;)$

END ;

Cross-references

withdraw	*Bank*	OPERATIONS	162

$rr \longleftarrow$ **sisaccount** $(\; acc \in \mathbb{N} \;) \quad \widehat{=}$

PRE

$sstate = connected$

THEN

$rr \longleftarrow isaccount \; (\; acc \;)$

END

Cross-references

isaccount	*Bank*	OPERATIONS	162

END

Cross-references for Server

Bank		MACHINE	159
BOOL	*Bool_TYPE*	SETS	159

Bool_TYPE		MACHINE	166
CommsDefs		MACHINE	127
create_acc	*Bank*	OPERATIONS	117
deposit	*Bank*	OPERATIONS	161
Globals		MACHINE	171
isaccount	*Bank*	OPERATIONS	162
STATE	*CommsDefs*	SETS	127
withdraw	*Bank*	OPERATIONS	162

MACHINE

Client

SEES

Bool_TYPE , *CommsDefs*

VARIABLES

clientstate

INVARIANT

$clientstate \in STATE$

Cross-references

STATE	*CommsDefs*	SETS	127

INITIALISATION

$clientstate := idle$

OPERATIONS

cinit $\widehat{=}$

$clientstate := idle$ **;**

$ok \longleftarrow$ **client_con_request** $\widehat{=}$

IF $clientstate = idle$ **THEN**

$clientstate := conpending \;\|$

$ok := TRUE$

ELSE $ok := FALSE$

END **;**

$ok \longleftarrow$ **client_con_report** $(\, response \in BOOL \,)$ $\widehat{=}$

IF $clientstate = conpending$ **THEN**

IF $response = TRUE$ **THEN**

$clientstate := connected$

ELSE $clientstate := idle$

END $\|$

$ok := TRUE$

ELSE $ok := FALSE$

END **;**

Cross-references

BOOL	*Bool_TYPE*	SETS	159

$state \longleftarrow$ **query_cstate** $\widehat{=}$ $state := clientstate$

END

Cross-references for Client

BOOL	*Bool_TYPE*	SETS	159

Bool_TYPE		MACHINE	166
CommsDefs		MACHINE	127
STATE	*CommsDefs*	SETS	127

The Buffer machine is the same as from previous section.

MACHINE *ATM*

SEES

CommsDefs , *Bool_TYPE* , *Globals*

INCLUDES

Protocol

VARIABLES

atm_state

INVARIANT

$atm\_state \in STATE$

Cross-references

STATE	*CommsDefs*	SETS	127

INITIALISATION

$atm\_state := idle$

OPERATIONS

get_protocol_state $\widehat{=}$

$atm\_state \longleftarrow query\_pstate$ **;**

Cross-references

query_pstate	*Protocol*	OPERATIONS	122

$ans \longleftarrow$ **deposit** $(\; acc \in \mathbb{N} \wedge amount \in \mathbb{N}_1 \;)$ $\widehat{=}$

IF $atm\_state = connected$ **THEN**

$ans \longleftarrow pwithdraw \; (\; acc \; , \; amount \;)$

END **;**

Cross-references

pwithdraw	*Protocol*	OPERATIONS	122

$ans \longleftarrow$ **withdraw** $(\; acc \in \mathbb{N} \wedge amount \in \mathbb{N}_1 \;)$ $\widehat{=}$

IF $atm\_state = connected$ **THEN**

$ans \longleftarrow pwithdraw \; (\; acc \; , \; amount \;)$

END **;**

Cross-references

pwithdraw	*Protocol*	OPERATIONS	122

$aa \longleftarrow$ **create_acc** $\widehat{=}$

IF $atm\_state = connected$ **THEN**

$aa \longleftarrow pcreate\_acc$

END **;**

Cross-references

pcreate_acc	*Protocol*	OPERATIONS	122

$rr \longleftarrow$ **isaccount** ($acc \in \mathbb{N}$) $\widehat{=}$

IF $atm\_state = connected$ **THEN**

$rr \longleftarrow pisaccount$ (acc)

END

Cross-references

pisaccount	*Protocol*	OPERATIONS	123

END

Cross-references for ATM

Bool_TYPE		MACHINE	166
CommsDefs		MACHINE	127
Globals		MACHINE	171
pcreate_acc	*Protocol*	OPERATIONS	122
pisaccount	*Protocol*	OPERATIONS	123
Protocol		MACHINE	121
pwithdraw	*Protocol*	OPERATIONS	122
query_pstate	*Protocol*	OPERATIONS	122
STATE	*CommsDefs*	SETS	127

MACHINE *Bank* (*maxAccounts*)

CONSTRAINTS

$maxAccounts \in 1 \;..\; 200000$

SEES

Globals

VARIABLES

$account$, $accountNumber$

INVARIANT

$accountNumber \subseteq ACCOUNTS \wedge$

$account \in accountNumber \rightarrowtail \mathbb{N}$

INITIALISATION

$accountNumber$, $account := \{\}$, $\{\}$

OPERATIONS

$aa \longleftarrow$ **create_acc** $\widehat{=}$

ANY acc

WHERE $acc \in ACCOUNTS - accountNumber$

THEN

$aa := acc$ $\|$

$accountNumber := accountNumber \cup \{ acc \}$ $\|$

$account (acc) := 0$

END ;

$ans \longleftarrow$ **deposit** $(acc \in accountNumber \wedge amount \in \mathbb{N}_1)$ $\widehat{=}$

BEGIN

$account (acc) := account (acc) + amount$ $\|$

$ans := account (acc) + amount$

END ;

$ans \longleftarrow$ **withdraw** $(acc \in accountNumber \wedge amount \in \mathbb{N}_1)$ $\widehat{=}$

IF $amount \leq account (acc)$ **THEN**

$account (acc) := account (acc) - amount$ $\|$

$ans := yes$

ELSE $ans := no$

END **;**

$rr \longleftarrow$ **isaccount** ($acc \in \mathbb{N}$) $\widehat{=}$

IF $acc \in accountNumber$ **THEN**

$rr := yes$

ELSE $rr := no$

END

DEFINITIONS

$ACCOUNTS \;\widehat{=}\; 1 \,..\, maxAccounts$

END

Cross-references for Bank

Globals	MACHINE	171

MACHINE *Bank* (*maxAccounts*)

CONSTRAINTS

$maxAccounts \in 1 \,..\, 200000$

SEES

Globals

VARIABLES

account , *accountNumber*

INVARIANT

$accountNumber \subseteq ACCOUNTS \land$

$account \in accountNumber \rightarrowtail \mathbb{N}$

INITIALISATION

$accountNumber \ , \ account := \{\} \ , \ \{\}$

OPERATIONS

$aa \longleftarrow$ **create_acc** $\widehat{=}$

ANY acc

WHERE $acc \in ACCOUNTS - accountNumber$

THEN

$aa := acc \ \ \|$

$accountNumber := accountNumber \cup \{ \ acc \ \} \ \ \|$

$account \ (\ acc \ ) := 0$

END **;**

$ans \longleftarrow$ **deposit** $(\ acc \in accountNumber \land amount \in \mathbb{N}_1 \ )$ $\widehat{=}$

BEGIN

$account \ (\ acc \ ) := account \ (\ acc \ ) + amount \ \ \|$

$ans := account \ (\ acc \ ) + amount$

END **;**

$ans \longleftarrow$ **withdraw** $(\ acc \in accountNumber \land amount \in \mathbb{N}_1 \ )$ $\widehat{=}$

IF $amount \leq account \ (\ acc \ )$ **THEN**

$account \ (\ acc \ ) := account \ (\ acc \ ) - amount \ \ \|$

$ans := yes$

ELSE $ans := no$

END **;**

$rr \longleftarrow$ **isaccount** $(\ acc \in \mathbb{N} \ )$ $\widehat{=}$

IF $acc \in accountNumber$ **THEN**

$rr := yes$

ELSE $rr := no$

END

DEFINITIONS

$ACCOUNTS \;\widehat{=}\; 1 \;..\; maxAccounts$

END

Cross-references for Bank

Globals	MACHINE	171

MACHINE *CommsDefs*

SETS

$STATE = \{ \; idle \; , \; conpending \; , \; conrequest \; , \; have\_report \; , \; connected \; \}$ **;**

$MESSAGES = \{ \; con \; , \; true \; , \; false \; \}$

CONSTANTS

$buffsize$

PROPERTIES

$buffsize = 100$

END

MACHINE *Globals* (*maxAccounts*)

SETS

$ACCOUNTNR$ **;**

$YESNO = \{ \ yes \ , \ no \ \}$

END

C.2 Receiver, Sender and SecureNetwork

MACHINE

$SecureNetwork$

USES

$Bool\_TYPE$

SETS

$MESSAGE$

VARIABLES

msg

INVARIANT

$msg \in \mathsf{seq} \ (\ MESSAGE \ ) \ \wedge$

$\mathsf{card} \ (\ msg \ ) \leq maxSize$

INITIALISATION

$msg := \{\}$

OPERATIONS

$mm \ , \ ok \longleftarrow$ **receive** $\widehat{=}$

BEGIN

IF $\mathsf{card}\ (\ msg\ ) > 0$ **THEN**

$mm := \mathsf{first}\ (\ msg\ )\ \ \|$

$ok := TRUE$

ELSE $ok := FALSE$

END

END **;**

$ok \longleftarrow$ **send** $(\ mm \in MESSAGE\ )\ \ \widehat{=}$

IF $\mathsf{card}\ (\ msg\ ) < maxSize$ **THEN**

$msg := msg \leftarrow mm\ \ \|$

$ok := TRUE$

ELSE $ok := FALSE$

END

DEFINITIONS

$maxSize\ \ \widehat{=}\ \ 100$

END

Cross-references for SecureNetwork

Bool_TYPE	MACHINE	166

MACHINE *Sender*

USES *Bool_TYPE*

INCLUDES *SecureNetwork*

VARIABLES

inval

INVARIANT

$inval \in MESSAGE$

Cross-references

MESSAGE	*SecureNetwork*	SETS	132

OPERATIONS

$rep \longleftarrow$ **do_send** $\widehat{=}$

BEGIN

ANY bb

WHERE $bb \in MESSAGE$

THEN

$rep \longleftarrow send\ (\ bb\ )$

END

END **;**

Cross-references

MESSAGE	*SecureNetwork*	SETS	132
send	*SecureNetwork*	OPERATIONS	132

$rep \longleftarrow$ **can_do_receive** $\widehat{=}$

$inval\ ,\ rep \longleftarrow receive$

Cross-references

receive	*SecureNetwork*	OPERATIONS	132

END

Cross-references for Sender

Bool_TYPE		MACHINE	166
MESSAGE	*SecureNetwork*	SETS	132
receive	*SecureNetwork*	OPERATIONS	132
SecureNetwork		MACHINE	131
send	*SecureNetwork*	OPERATIONS	132

MACHINE *Receiver*

USES *Bool_TYPE*

INCLUDES *SecureNetwork*

VARIABLES

inval

INVARIANT

$inval \in MESSAGE$

Cross-references

MESSAGE	*SecureNetwork*	SETS	132

OPERATIONS

$rep \longleftarrow$ **do_receive** $\widehat{=}$

$inval$, $rep \longleftarrow receive$ **;**

Cross-references

receive	*SecureNetwork*	OPERATIONS	132

$rep \longleftarrow$ **can_do_send** $\widehat{=}$

BEGIN

ANY bb

WHERE $bb \in MESSAGE$

THEN

$rep \longleftarrow send\ (\ bb\ )$

END

END

Cross-references

MESSAGE	*SecureNetwork*	SETS	132
send	*SecureNetwork*	OPERATIONS	132

END

Cross-references for Receiver

Bool_TYPE		MACHINE	166
MESSAGE	*SecureNetwork*	SETS	132
receive	*SecureNetwork*	OPERATIONS	132
SecureNetwork		MACHINE	131
send	*SecureNetwork*	OPERATIONS	132

C.3 Networked Bank

MACHINE *NetworkedBank*

SEES *Bool_TYPE*

INCLUDES *Bank*

VARIABLES

ff , $creq$

INVARIANT

$ff \in \mathsf{seq}\ (\mathbb{N}) \wedge$

$creq \in \mathsf{seq}\ (\mathbb{N}) \wedge$

$\mathsf{size}\ (ff) \leq maxParams$

INITIALISATION

$creq\ , ff := [\,]\ , [\,]$

OPERATIONS

$status \longleftarrow$ **ATM_create_account** $\widehat{=}$

BEGIN

$creq := creq \leftarrow 1 \ \ \|$

$status := TRUE$

END ;

$status \longleftarrow$ **ATM_deposit** (acc , $amount$) $\widehat{=}$

PRE

$acc \in accountNumber \wedge$

$amount \in \mathbb{N}_1$

THEN

IF $\mathsf{size}\ (\ \mathit{ff}\ ) \leq \mathit{maxParams} - 2 \wedge \mathsf{size}\ (\ \mathit{ff}\ ) < \mathit{maxReq}$ **THEN**

$\mathit{creq} := \mathit{creq} \leftarrow 2 \ \ \|$

$\mathit{ff} := \mathit{ff} \lhd\!\!\!+ \{\ \mathsf{card}\ (\ \mathit{ff}\ ) + 1 \mapsto \mathit{acc}\ ,\ \mathsf{card}\ (\ \mathit{ff}\ ) + 2 \mapsto \mathit{amount}\ \} \ \ \|$

$\mathit{status} := \mathit{TRUE}$

ELSE

$\mathit{status} := \mathit{FALSE}$

END

END ;

Cross-references

accountNumber	*Bank*	VARIABLES	166

$\mathit{status} \longleftarrow$ **ATM_withdraw** $(\ \mathit{acc}\ ,\ \mathit{amount}\ ) \ \ \widehat{=}$

PRE

$\mathit{acc} \in \mathit{accountNumber} \wedge$

$\mathit{amount} \in \mathbb{N}_1$

THEN

IF $\mathsf{size}\ (\ \mathit{ff}\ ) \leq \mathit{maxParams} - 2 \wedge \mathsf{size}\ (\ \mathit{ff}\ ) < \mathit{maxReq}$ **THEN**

$\mathit{creq} := \mathit{creq} \leftarrow 3 \ \ \|$

$\mathit{ff} := \mathit{ff} \frown [\ \mathit{acc}\ ,\ \mathit{amount}\ ] \ \ \|$

$\mathit{status} := \mathit{TRUE}$

ELSE $\mathit{status} := \mathit{FALSE}$

END

END ;

Cross-references

accountNumber	*Bank*	VARIABLES	166

$status \longleftarrow$ **ATM_isaccount** (acc) $\widehat{=}$

PRE

$acc \in accountNumber$

THEN

IF $\mathsf{size}\ (\ ff\ ) \leq maxParams - 2 \land \mathsf{size}\ (\ ff\ ) < maxReq$ **THEN**

$creq := creq \leftarrow 4 \ \|$

$ff := ff \frown [\ acc\ ] \ \|$

$status := TRUE$

ELSE $status := FALSE$

END

END ;

Cross-references

accountNumber	*Bank*	VARIABLES	166

$status \longleftarrow$ **ATM_getbalance** (acc) $\widehat{=}$

PRE

$acc \in accountNumber$

THEN

IF $\mathsf{size}\ (\ ff\ ) \leq maxParams - 2 \land \mathsf{size}\ (\ ff\ ) < maxReq$ **THEN**

$creq := creq \leftarrow 5 \ \|$

$ff := ff \frown [\ acc\ ] \ \|$

$status := TRUE$

ELSE $status := FALSE$

END

END ;

Cross-references

accountNumber	*Bank*	VARIABLES	166

$num\ ,\ ok\ ,\ op \longleftarrow$ **check_clientreq** $\widehat{=}$

IF $\mathsf{card}\ (\ creq\ ) > 0$ **THEN**

VAR $aa\ ,\ bb$ **IN**

IF $\mathsf{last}\ (\ creq\ ) = 1$ **THEN**

$creq := \mathsf{front}\ (\ creq\ )\ \ \|$

$num\ ,\ ok \longleftarrow create\_account\ \ \|$

$op := 1$

ELSE **IF** $\mathsf{last}\ (\ creq\ ) = 2$ **THEN**

$creq := \mathsf{front}\ (\ creq\ )\ \ \|$

$aa := ff\ (\ \mathsf{card}\ (\ ff\ ) - 1\ )\ \ \|$

$bb := ff\ (\ \mathsf{card}\ (\ ff\ )\ )\ \ \|$

$ff := ff \uparrow \mathsf{card}\ (\ ff\ ) - 2\ \ \|$

$ok\ ,\ num \longleftarrow deposit\ (\ aa\ ,\ bb\ )\ \ \|$

$op := 2$

ELSE **IF** $\mathsf{last}\ (\ creq\ ) = 3$ **THEN**

$creq := \mathsf{front}\ (\ creq\ )\ \ \|$

$aa := ff\ (\ \mathsf{card}\ (\ ff\ ) - 1\ )\ \ \|$

$bb := ff\ (\ \mathsf{card}\ (\ ff\ )\ )\ \ \|$

$ff := ff \uparrow \mathsf{card}\ (\ ff\ ) - 2\ \ \|$

$ok \longleftarrow withdraw\ (\ aa\ ,\ bb\ )\ \ \|$

$num := 0\ \ \|$

$op := 3$

ELSE **IF** $\mathsf{last}\ (\ creq\ ) = 4$ **THEN**

$creq := \mathsf{front}\ (\ creq\ )\ \ \|$

$aa := ff\ (\ \mathsf{card}\ (\ ff\ )\ )\ \ \|$

$ff := \mathsf{tail}\ (\ ff\ )\ \ \|$

$ok \longleftarrow isaccount\ (\ aa\ )\ \ \|$

$num := 0 \;\; \|$

$op := 4$

ELSE **IF** $\mathsf{last}\ (\ creq\ ) = 5$ **THEN**

$creq := \mathsf{front}\ (\ creq\ ) \;\; \|$

$aa := ff\ (\ \mathsf{card}\ (\ ff\ )\ ) \;\; \|$

$ff := \mathsf{tail}\ (\ ff\ ) \;\; \|$

$num \longleftarrow getbalance\ (\ aa\ ) \;\; \|$

$ok := FALSE \;\; \|$

$op := 5$

END

END

END

END

END

END

END

Cross-references

create_account	*Bank*	OPERATIONS	161
deposit	*Bank*	OPERATIONS	161
getbalance	*Bank*	OPERATIONS	163
isaccount	*Bank*	OPERATIONS	162
withdraw	*Bank*	OPERATIONS	162

DEFINITIONS

$maxParams \;\widehat{=}\; 100$ **;**

$maxReq \;\widehat{=}\; 33$

END

Cross-references for NetworkedBank

MACHINE

Bank

SEES

Bool_TYPE

INCLUDES

Globals

VARIABLES

account , *accountNumber*

INVARIANT

$accountNumber \subseteq ACCOUNTS \wedge$

$account \in accountNumber \rightarrow \mathbb{N} \wedge$

$\mathsf{dom}\ (\ account\ ) = accountNumber$

Cross-references

ACCOUNTS	*Globals*	DEFINITIONS	172

INITIALISATION

RES

OPERATIONS

$accNr$, $rep \longleftarrow$ **create_account** $\widehat{=}$

IF $accountNumber \neq ACCOUNTS$ **THEN**

$rep := TRUE \parallel$

ANY acc

WHERE

$acc \in ACCOUNTS - accountNumber$

THEN

$accNr := acc \parallel$

$accountNumber := accountNumber \cup \{ acc \} \parallel$

$account (acc) := 0$

END

ELSE $rep := FALSE \parallel$

$accNr := 0$

END **;**

Cross-references

ACCOUNTS	*Globals*	DEFINITIONS	172

ok , $newBalance \longleftarrow$ **deposit** (acc , $amount$) $\widehat{=}$

PRE

$acc \in accountNumber \wedge$

$amount \in \mathbb{N}_1$

THEN

IF $account\ (\ acc\ ) < MAXINT - amount$ **THEN**

$account\ (\ acc\ ) := account\ (\ acc\ ) + amount\ \ \|$

$newBalance := account\ (\ acc\ ) + amount\ \ \|$

$ok := TRUE$

ELSE

$ok := FALSE\ \ \|$

$newBalance := 0$

END

END **;**

Cross-references

MAXINT	*Globals*	DEFINITIONS	172

$ok \longleftarrow$ **withdraw** $(\ acc\ ,\ amount\ )\ \ \widehat{=}$

PRE

$acc \in accountNumber \wedge$

$amount \in \mathbb{N}$

THEN

IF

$amount \leq account\ (\ acc\ )$

THEN

$account\ (\ acc\ ) := account\ (\ acc\ ) - amount\ \ \|$

$ok := TRUE$

ELSE

$ok := FALSE$

END

END ;

$rr \longleftarrow$ **isaccount** (acc) $\widehat{=}$

PRE

$acc \in ACCOUNTS$

THEN

IF $acc \in accountNumber$

THEN

$rr := TRUE$

ELSE

$rr := FALSE$

END

END ;

Cross-references

$ACCOUNTS$	*Globals*	DEFINITIONS	172

$bal \longleftarrow$ **getbalance** (acc) $\widehat{=}$

PRE

$acc \in accountNumber$

THEN

$bal := account (acc)$

END

DEFINITIONS

$RES \ \widehat{=}\ accountNumber , account := \{\} , \{\}$

END

Cross-references for Bank

ACCOUNTS	*Globals*	DEFINITIONS	172
Bool_TYPE		MACHINE	166
Globals		MACHINE	171
MAXINT	*Globals*	DEFINITIONS	172

MACHINE *Globals*

CONSTANTS

maxAccounts

PROPERTIES

$maxAccounts = 6$

DEFINITIONS

$ACCOUNTS \ \widehat{=} \ 1 \,..\, maxAccounts$;

$MAXINT \ \widehat{=} \ 5000$;

$ee\_ser \ \widehat{=} \ 5$;

$dd\_ser \ \widehat{=} \ 17$;

$nn \ \widehat{=} \ 7$;

$ee\_cli \ \widehat{=} \ 9$;

$dd\_cli \ \widehat{=} \ 11$

END

C.4 An Implementation of the ATM System

C.4.1 The Specification

MACHINE

$Bank$

SEES

$Bool\_TYPE$

INCLUDES

$Globals$

VARIABLES

$account$, $accountNumber$

INVARIANT

$accountNumber \subseteq ACCOUNTS \land$

$account \in accountNumber \rightarrow \mathbb{N} \land$

$\mathsf{dom}\ (\ account\ ) = accountNumber$

Cross-references

$ACCOUNTS$	$Globals$	DEFINITIONS	172

INITIALISATION

RES

OPERATIONS

$accNr$, $rep \longleftarrow$ **create_account** $\widehat{=}$

IF $accountNumber \neq ACCOUNTS$ **THEN**

$rep := TRUE$ $\|$

ANY acc

WHERE

$acc \in ACCOUNTS - accountNumber$

THEN

$accNr := acc \ \|$

$accountNumber := accountNumber \cup \{ acc \} \ \|$

$account (acc) := 0$

END

ELSE $rep := FALSE \ \|$

$accNr := 0$

END ;

Cross-references

$ACCOUNTS$	*Globals*	DEFINITIONS	172

$ok , newBalance \longleftarrow$ **deposit** $(acc , amount) \ \widehat{=}$

PRE

$acc \in accountNumber \wedge$

$amount \in \mathbb{N}_1$

THEN

IF $account (acc) < MAXINT - amount$ **THEN**

$account (acc) := account (acc) + amount \ \|$

$newBalance := account (acc) + amount \ \|$

$ok := TRUE$

ELSE

$ok := FALSE \ \|$

$newBalance := 0$

END

END ;

Cross-references

MAXINT	*Globals*	DEFINITIONS	172

$ok \longleftarrow$ **withdraw** $(\ acc \ , \ amount \ ) \quad \widehat{=}$

PRE

$acc \in accountNumber \ \wedge$

$amount \in \mathbb{N}$

THEN

IF

$amount \leq account \ (\ acc \ )$

THEN

$account \ (\ acc \ ) := account \ (\ acc \ ) - amount \ \ \|$

$ok := TRUE$

ELSE

$ok := FALSE$

END

END ;

$rr \longleftarrow$ **isaccount** $(\ acc \ ) \quad \widehat{=}$

PRE

$acc \in ACCOUNTS$

THEN

IF $acc \in accountNumber$

THEN

$rr := TRUE$

ELSE

$rr := FALSE$

END

END ;

Cross-references

ACCOUNTS	*Globals*	DEFINITIONS	172

$bal \longleftarrow$ **getbalance** (acc) $\widehat{=}$

PRE

$acc \in accountNumber$

THEN

$bal := account\ (\ acc\ )$

END

DEFINITIONS

$RES \ \widehat{=}\ accountNumber\ ,\ account := \{\}\ ,\ \{\}$

END

Cross-references for Bank

ACCOUNTS	*Globals*	DEFINITIONS	172
Bool_TYPE		MACHINE	166
Globals		MACHINE	171
MAXINT	*Globals*	DEFINITIONS	172

MACHINE *ServeBank*

SEES

Bool_TYPE

OPERATIONS

$ok \longleftarrow$ **startServer** $\widehat{=}$ $ok :\in BOOL$;

Cross-references

BOOL	*Bool_TYPE*	SETS	159

listenForUser $\widehat{=}$ skip

END

Cross-references for ServeBank

BOOL	*Bool_TYPE*	SETS	159
Bool_TYPE		MACHINE	166

MACHINE *Globals*

CONSTANTS

maxAccounts

PROPERTIES

$maxAccounts = 6$

DEFINITIONS

$ACCOUNTS \;\widehat{=}\; 1 \,..\, maxAccounts$;

$MAXINT \;\widehat{=}\; 5000$;

$ee\_ser \;\widehat{=}\; 5$;

$dd\_ser \;\widehat{=}\; 17$;

$nn \;\widehat{=}\; 7$;

$ee\_cli \;\widehat{=}\; 9$;

$dd\_cli \ \widehat{=} \ 11$

END

C.4.2 The Implementation

IMPLEMENTATION *BankI*

REFINES *Bank*

SEES *Bool_TYPE*

IMPORTS

accNr_Narr (*MAXINT* , *maxAccounts*) ,

count_Nvar (*maxAccounts*)

INITIALISATION

count_STO_NVAR (*0*)

Cross-references

count_STO_NVAR	*count_Nvar*	OPERATIONS	149

OPERATIONS

accNr , *rep* $\longleftarrow$ **create_account** $\widehat{=}$

VAR *ii* , *ok* **IN**

ii $\longleftarrow$ *count_VAL_NVAR* **;**

$ii := ii + 1$ **;**

IF $ii \leq maxAccounts$ **THEN**

accNr_STO_NARR (*ii* , *0*) **;**

```
        count_STO_NVAR ( ii ) ;
        accNr := ii ;
        rep := TRUE
      ELSE
        accNr := 0 ;
        rep := FALSE
      END
    END ;
```

Cross-references

accNr_STO_NARR	*accNr_Narr*	OPERATIONS	150
count_STO_NVAR	*count_Nvar*	OPERATIONS	149
count_VAL_NVAR	*count_Nvar*	OPERATIONS	151
maxAccounts	*Bank*	CONSTANTS	171

```
ok , newBalance <-- deposit ( acc , amount ) ≙
  BEGIN
    VAR ii IN
      ii <-- count_VAL_NVAR ;
      IF acc ≤ ii ∧ acc ≤ maxAccounts ∧ amount > 0 THEN
        VAR curBal IN
          curBal <-- accNr_VAL_NARR ( acc ) ;
          newBalance := curBal + amount ;
          accNr_STO_NARR ( acc , newBalance ) ;
          ok := TRUE
        END
      ELSE
        ok := FALSE ;
        newBalance := 0
```

END

END

END ;

Cross-references

accNr_STO_NARR	*accNr_Narr*	OPERATIONS	150
accNr_VAL_NARR	*accNr_Narr*	OPERATIONS	151
count_VAL_NVAR	*count_Nvar*	OPERATIONS	151
maxAccounts	*Bank*	CONSTANTS	171

$ok \longleftarrow$ **withdraw** (acc , $amount$) $\widehat{=}$

BEGIN

VAR ii **IN**

$ii \longleftarrow count\_VAL\_NVAR$;

IF $acc \leq ii \wedge acc \leq maxAccounts \wedge amount > 0$ **THEN**

VAR $curBal$ **IN**

$curBal \longleftarrow accNr\_VAL\_NARR$ (acc) ;

IF $curBal - amount \geq 0$ **THEN**

$accNr\_STO\_NARR$ (acc , $curBal - amount$) ;

$ok := TRUE$

END

END

ELSE

$ok := FALSE$

END

END

END ;

Cross-references

accNr_STO_NARR	*accNr_Narr*	OPERATIONS	150
accNr_VAL_NARR	*accNr_Narr*	OPERATIONS	151
count_VAL_NVAR	*count_Nvar*	OPERATIONS	151
maxAccounts	*Bank*	CONSTANTS	171

$rr \longleftarrow$ **isaccount** (acc) $\widehat{=}$

```
BEGIN
  VAR ii IN
    ii <-- count_VAL_NVAR ;
    IF acc <= ii THEN
      rr := TRUE
    ELSE
      rr := FALSE
    END
  END
END ;
```

Cross-references

count_VAL_NVAR	*count_Nvar*	OPERATIONS	151

$bal \longleftarrow$ **getbalance** (acc) $\widehat{=}$

```
BEGIN
  VAR ii IN
    ii <-- count_VAL_NVAR ;
    IF acc <= ii THEN
      bal <-- accNr_VAL_NARR ( acc )
    ELSE
      bal := 0
    END
  END
```

END

Cross-references

accNr_VAL_NARR	*accNr_Narr*	OPERATIONS	151
count_VAL_NVAR	*count_Nvar*	OPERATIONS	151

END

Cross-references for BankI

accNr_Narr	*accNr_Narr*	VARIABLES	148
accNr_STO_NARR	*accNr_Narr*	OPERATIONS	150
accNr_VAL_NARR	*accNr_Narr*	OPERATIONS	151
Bank		MACHINE	159
Bool_TYPE		MACHINE	166
count_Nvar	*count_Nvar*	VARIABLES	148
count_STO_NVAR	*count_Nvar*	OPERATIONS	149
count_VAL_NVAR	*count_Nvar*	OPERATIONS	151
maxAccounts	*Bank*	CONSTANTS	171
MAXINT	*Bank*	DEFINITIONS	172

IMPLEMENTATION *ServeBankI*

REFINES *ServeBank*

SEES

basic_io ,

file_dump ,

String_TYPE ,

Scalar_TYPE ,

Bool_TYPE

IMPORTS

Bank ,

Bank_SocketServer ($SCALAR \cup BOOL$, *10* , *10*) ,

Cipher (*ee_ser* , *dd_ser* , *nn*) ,

publicDecrypt_Nvar (*MAXINT*)

INITIALISATION

publicDecrypt_STO_NVAR (*0*)

Cross-references

publicDecrypt_STO_NVAR	*publicDecrypt_Nvar*	OPERATIONS	159

OPERATIONS

ok ⟵ **startServer** ≙

BEGIN

VAR *xx* **IN**

PUT_STR (" `Server running.` ") ;

FLSH ;

xx ⟵ *Bank_INIT* (" `banklock` " , *3200* , " `bankbuff` ") ;

ok ⟵ *Bank_ACCEPT* ;

PUT_STR (" `connection Established.` ")

END

END ;

Cross-references

```
listenForUser  ≙
  BEGIN
    VAR  rep , pp , op  IN
      WHILE  1 = 1  DO
        PUT_STR ( " WAITING HERE...   " ) ;
        rep , pp ⟵ Bank_READ ;
        op ⟵ Bank_GET_TOK ( 1 ) ;
        PUT_STR ( " First tok is " ) ;
        PUT_NAT ( op ) ;
        PUT_STR ( "  " ) ;
        IF  op = 1  THEN
          PUT_STR ( " INSIDE create_account  " ) ;
          VAR  ret_a , ret_b , crypt_ret  IN
            ret_a , ret_b ⟵ create_account ;
            PUT_STR ( " calling create acc.   " ) ;
            PUT_STR ( " account number is: " ) ;
            PUT_NAT ( ret_a ) ;
            PUT_STR ( "  " ) ;
            crypt_ret ⟵ encrypt_op ( ret_a ) ;
            Bank_PUT_TOK ( crypt_ret , 1 ) ;
            Bank_PUT_TOK ( ret_b , 1 ) ;
            PUT_STR ( " Going to write tok  " ) ;
            rep ⟵ Bank_WRITE ;
            PUT_STR ( " have put str  " )
```

```
    END
END ;
IF op = 2 THEN
    PUT_STR ( " INSIDE deposit " ) ;
    VAR in_a , in_b , ret_a , ret_b IN
        in_a <-- Bank_GET_TOK ( 1 ) ;
        in_b <-- Bank_GET_TOK ( 1 ) ;
        ret_a , ret_b <-- deposit ( in_a , in_b ) ;
        PUT_STR ( " deposit to accNr:  " ) ;
        PUT_NAT ( in_a ) ;
        PUT_STR ( " amount: " ) ;
        PUT_NAT ( in_b ) ;
        PUT_STR ( " " ) ;
        Bank_PUT_TOK ( ret_a , 1 ) ;
        Bank_PUT_TOK ( ret_b , 1 ) ;
        rep <-- Bank_WRITE
    END
END ;
IF op = 3 THEN
    PUT_STR ( " INSIDE withdraw " ) ;
    VAR in_a , in_b , ret_a IN
        in_a <-- Bank_GET_TOK ( 1 ) ;
        in_b <-- Bank_GET_TOK ( 1 ) ;
        ret_a <-- withdraw ( in_a , in_b ) ;
        PUT_STR ( " withdraw from accNr:  " ) ;
        PUT_NAT ( in_a ) ;
        PUT_STR ( " amount:  " ) ;
        PUT_NAT ( in_b ) ;
```

```
            PUT_STR ( " " ) ;
            Bank_PUT_TOK ( ret_a , 1 ) ;
            rep <-- Bank_WRITE
        END
    END ;
    IF op = 4 THEN
        PUT_STR ( " INSIDE isaccount " ) ;
        VAR in_a , ret_a IN
            in_a <-- Bank_GET_TOK ( 1 ) ;
            ret_a <-- isaccount ( in_a ) ;
            PUT_STR ( " isaccount with accNr:  " ) ;
            PUT_NAT ( in_a ) ;
            PUT_STR ( " " ) ;
            Bank_PUT_TOK ( ret_a , 1 ) ;
            rep <-- Bank_WRITE
        END
    END ;
    IF op = 5 THEN
        PUT_STR ( " INSIDE getbalance " ) ;
        VAR in_a , ret_a IN
            in_a <-- Bank_GET_TOK ( 1 ) ;
            ret_a <-- getbalance ( in_a ) ;
            PUT_STR ( " balance for accNr:  " ) ;
            PUT_NAT ( in_a ) ;
            PUT_STR ( " " ) ;
            Bank_PUT_TOK ( ret_a , 1 ) ;
            rep <-- Bank_WRITE
        END
```

```
            END ;
            IF rep = FALSE THEN
              PUT_STR ( " could not send value " )
            ELSE
              PUT_STR ( " AND returning values " )
            END ;
            op := 0 ;
            PUT_STR ( " value of OP " ) ;
            PUT_NAT ( op ) ;
            PUT_STR ( " " )
          INVARIANT
            1 = 1
          VARIANT
            1
          END
        END
      END
```

Cross-references

Bank_GET_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_PUT_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_READ	*Bank_SocketServer*	OPERATIONS	160
Bank_WRITE	*Bank_SocketServer*	OPERATIONS	163
create_account	*Bank*	OPERATIONS	161
deposit	*Bank*	OPERATIONS	161
encrypt_op	*Cipher*	OPERATIONS	161
getbalance	*Bank*	OPERATIONS	163
isaccount	*Bank*	OPERATIONS	162
PUT_NAT	*basic_io*	OPERATIONS	163

PUT_STR	*basic_io*	OPERATIONS	163
withdraw	*Bank*	OPERATIONS	162

END

Cross-references for ServeBankI

Bank		MACHINE	159
Bank_ACCEPT	*Bank_SocketServer*	OPERATIONS	160
Bank_GET_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_INIT	*Bank_SocketServer*	OPERATIONS	160
Bank_PUT_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_READ	*Bank_SocketServer*	OPERATIONS	160
Bank_SocketServer		MACHINE	159
Bank_WRITE	*Bank_SocketServer*	OPERATIONS	163
basic_io		MACHINE	159
BOOL	*Bool_TYPE*	SETS	159
Bool_TYPE		MACHINE	166
Cipher		MACHINE	159
create_account	*Bank*	OPERATIONS	161
dd_ser	*Bank*	DEFINITIONS	172
deposit	*Bank*	OPERATIONS	161
ee_ser	*Bank*	DEFINITIONS	172
encrypt_op	*Cipher*	OPERATIONS	161
file_dump		MACHINE	159
FLSH	*basic_io*	OPERATIONS	160
getbalance	*Bank*	OPERATIONS	163
isaccount	*Bank*	OPERATIONS	162
MAXINT	*Bank*	DEFINITIONS	172
nn	*Bank*	DEFINITIONS	172
publicDecrypt_Nvar	*publicDecrypt_Nvar*	VARIABLES	159

IMPLEMENTATION *ServeBankI*

REFINES *ServeBank*

SEES

basic_io ,

file_dump ,

String_TYPE ,

Scalar_TYPE ,

Bool_TYPE

IMPORTS

Bank ,

$Bank\_SocketServer\ (\ SCALAR \cup BOOL\ ,\ 10\ ,\ 10\ )$,

Cipher (*ee_ser* , *dd_ser* , *nn*) ,

publicDecrypt_Nvar (*MAXINT*)

INITIALISATION

publicDecrypt_STO_NVAR (*0*)

Cross-references

publicDecrypt_STO_NVAR	*publicDecrypt_Nvar*	OPERATIONS	159

OPERATIONS

$ok \longleftarrow$ **startServer** $\widehat{=}$

BEGIN

VAR xx **IN**

PUT_STR (" `Server running.` ") ;

FLSH ;

$xx \longleftarrow$ *Bank_INIT* (" `banklock` " , *3200* , " `bankbuff` ") ;

$ok \longleftarrow$ *Bank_ACCEPT* ;

PUT_STR (" `connection Established.` ")

END

END ;

Cross-references

Bank_ACCEPT	*Bank_SocketServer*	OPERATIONS	160
Bank_INIT	*Bank_SocketServer*	OPERATIONS	160
FLSH	*basic_io*	OPERATIONS	160
PUT_STR	*basic_io*	OPERATIONS	163

listenForUser $\widehat{=}$

BEGIN

VAR rep , pp , op **IN**

WHILE $1 = 1$ **DO**

PUT_STR (" `WAITING HERE...` ") ;

rep , $pp \longleftarrow$ *Bank_READ* ;

```
op <-- Bank_GET_TOK ( 1 ) ;
PUT_STR ( "First tok is" ) ;
PUT_NAT ( op ) ;
PUT_STR ( " " ) ;
IF  op = 1  THEN
   PUT_STR ( "INSIDE create_account  " ) ;
   VAR  ret_a , ret_b , crypt_ret  IN
      ret_a , ret_b <-- create_account ;
      PUT_STR ( "calling create acc.   " ) ;
      PUT_STR ( "account number is: " ) ;
      PUT_NAT ( ret_a ) ;
      PUT_STR ( " " ) ;
      crypt_ret <-- encrypt_op ( ret_a ) ;
      Bank_PUT_TOK ( crypt_ret , 1 ) ;
      Bank_PUT_TOK ( ret_b , 1 ) ;
      PUT_STR ( "Going to write tok  " ) ;
      rep <-- Bank_WRITE ;
      PUT_STR ( "have put str  " )
   END
END  ;
IF  op = 2  THEN
   PUT_STR ( "INSIDE deposit  " ) ;
   VAR  in_a , in_b , ret_a , ret_b  IN
      in_a <-- Bank_GET_TOK ( 1 ) ;
      in_b <-- Bank_GET_TOK ( 1 ) ;
      ret_a , ret_b <-- deposit ( in_a , in_b ) ;
      PUT_STR ( "deposit to accNr:   " ) ;
      PUT_NAT ( in_a ) ;
```

```
        PUT_STR ( " amount: " ) ;
        PUT_NAT ( in_b ) ;
        PUT_STR ( " " ) ;
        Bank_PUT_TOK ( ret_a , 1 ) ;
        Bank_PUT_TOK ( ret_b , 1 ) ;
        rep ⟵ Bank_WRITE
    END
END ;
IF op = 3 THEN
    PUT_STR ( " INSIDE withdraw " ) ;
    VAR in_a , in_b , ret_a IN
        in_a ⟵ Bank_GET_TOK ( 1 ) ;
        in_b ⟵ Bank_GET_TOK ( 1 ) ;
        ret_a ⟵ withdraw ( in_a , in_b ) ;
        PUT_STR ( " withdraw from accNr:   " ) ;
        PUT_NAT ( in_a ) ;
        PUT_STR ( " amount:   " ) ;
        PUT_NAT ( in_b ) ;
        PUT_STR ( " " ) ;
        Bank_PUT_TOK ( ret_a , 1 ) ;
        rep ⟵ Bank_WRITE
    END
END ;
IF op = 4 THEN
    PUT_STR ( " INSIDE isaccount " ) ;
    VAR in_a , ret_a IN
        in_a ⟵ Bank_GET_TOK ( 1 ) ;
        ret_a ⟵ isaccount ( in_a ) ;
```

```
        PUT_STR ( " isaccount with accNr:   " ) ;
        PUT_NAT ( in_a ) ;
        PUT_STR ( " " ) ;
        Bank_PUT_TOK ( ret_a , 1 ) ;
        rep <-- Bank_WRITE
    END
END ;
IF  op = 5  THEN
    PUT_STR ( " INSIDE getbalance  " ) ;
    VAR  in_a , ret_a  IN
        in_a <-- Bank_GET_TOK ( 1 ) ;
        ret_a <-- getbalance ( in_a ) ;
        PUT_STR ( " balance for accNr:   " ) ;
        PUT_NAT ( in_a ) ;
        PUT_STR ( " " ) ;
        Bank_PUT_TOK ( ret_a , 1 ) ;
        rep <-- Bank_WRITE
    END
END ;
IF  rep = FALSE  THEN
    PUT_STR ( " could not send value  " )
ELSE
    PUT_STR ( " AND returning values  " )
END ;
op := 0 ;
PUT_STR ( " value of OP " ) ;
PUT_NAT ( op ) ;
PUT_STR ( " " )
```

INVARIANT

1 = 1

VARIANT

1

END

END

END

Cross-references

Bank_GET_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_PUT_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_READ	*Bank_SocketServer*	OPERATIONS	160
Bank_WRITE	*Bank_SocketServer*	OPERATIONS	163
create_account	*Bank*	OPERATIONS	161
deposit	*Bank*	OPERATIONS	161
encrypt_op	*Cipher*	OPERATIONS	161
getbalance	*Bank*	OPERATIONS	163
isaccount	*Bank*	OPERATIONS	162
PUT_NAT	*basic_io*	OPERATIONS	163
PUT_STR	*basic_io*	OPERATIONS	163
withdraw	*Bank*	OPERATIONS	162

END

Cross-references for ServeBankI

Bank		MACHINE	159
Bank_ACCEPT	*Bank_SocketServer*	OPERATIONS	160
Bank_GET_TOK	*Bank_SocketServer*	OPERATIONS	163
Bank_INIT	*Bank_SocketServer*	OPERATIONS	160

C.5 Buffer Model

MACHINE *bankSystem3*

SEES

Bool_TYPE

INCLUDES

Globals

VARIABLES

accountNumber ,

accountBalance , *totalBalance* ,

ATMData , *networkData* ,

transactionId ,

req_deposit ,

req_withdraw ,

confirm_deposit , *confirm_withdrawal*

INVARIANT

$accountNumber \subseteq ACCOUNTS \land$

$accountBalance \in accountNumber \rightarrow \mathbb{N} \land$

$totalBalance \in accountNumber \rightarrow \mathbb{N} \land$

$ATMData \in accountNumber \rightarrow \mathbb{N} \land$

$networkData \in accountNumber \rightarrow \mathbb{N} \land$

$transactionId \subseteq 1 \;..\; 100 \land$

$req\_deposit \in transactionId \rightarrow accountNumber \times \mathbb{N} \land$

$req\_withdraw \in transactionId \rightarrow accountNumber \times \mathbb{N} \land$

$confirm\_transaction \subseteq transactionId \land$

$\forall\ aa\ .\ (\ aa \in accountNumber \Rightarrow$

$totalBalance\ (\ aa\ ) = accountBalance\ (\ aa\ ) + ATMData\ (\ aa\ )\ )$

Cross-references

ACCOUNTS	*Globals*	DEFINITIONS	172

INITIALISATION

$accountNumber$,

$accountBalance$, $totalBalance$,

$ATMData$, $networkData$,

$transactionId$,

$req\_deposit$, $req\_withdraw$,

$confirm\_deposit$, $confirm\_withdrawal := \{\} , \{\} , \{\} , \{\} , \{\} , \{\} , \{\} , \{\} , \{\} , \{\}$

OPERATIONS

request_deposit $(\ acc \in accountNumber \wedge amount \in \mathbb{N}\ )\ \ \widehat{=}$

BEGIN

ANY xx

WHERE

$xx \in transactionId\ \wedge$

$xx \notin \mathsf{dom}\ (\ req\_deposit\ ) \wedge$

$xx \notin confirm\_transaction$

THEN

$req\_deposit := req\_deposit \cup \{\ xx \mapsto (\ acc \mapsto amount\ )\ \}$

END

END **;**

request_withdraw $(\ acc \in accountNumber \wedge amount \in \mathbb{N}\ )\ \ \widehat{=}$

BEGIN

ANY xx

WHERE

$xx \in transactionId \wedge$

$xx \notin \mathsf{dom}\ (\ req\_withdraw\ ) \wedge$

$xx \notin confirm\_transaction$

THEN

$req\_withdraw := req\_withdraw \cup \{\ xx \mapsto (\ acc \mapsto amount\ )\ \}$

END

END ;

ATM_deposit $\widehat{=}$

BEGIN

ANY $xx\ ,\ yy\ ,\ zz$

WHERE

$xx \in transactionId \wedge$

$yy \in accountNumber \wedge$

$zz \in \mathbb{N} \wedge$

$xx \notin confirm\_transaction \wedge$

$yy \mapsto zz \notin ATMData \wedge$

$yy \mapsto zz \notin networkData \wedge$

$xx \mapsto (\ yy \mapsto zz\ ) \in req\_deposit$

THEN

$networkData\ (\ yy\ ) := zz\ \ \|$

$ATMData\ (\ yy\ ) := zz\ \ \|$

$totalBalance\ (\ yy\ ) := totalBalance\ (\ yy\ ) + zz$

END

END ;

Bank_deposit $\widehat{=}$

BEGIN

ANY xx , yy , zz

WHERE

$xx \in transactionId \wedge$

$yy \in accountNumber \wedge$

$zz \in \mathbb{N} \wedge$

$xx \notin confirm\_transaction \wedge$

$xx \mapsto yy \in ATMData \wedge$

$yy \mapsto zz \in networkData \wedge$

$xx \mapsto (yy \mapsto zz) \in req\_deposit \wedge$

$ATMData (yy) = networkData (yy)$

THEN

$networkData := networkData - \{ yy \mapsto zz \} \;\|$

$ATMData := ATMData - \{ yy \mapsto zz \} \;\|$

$accountBalance (yy) := accountBalance (yy) + zz \;\|$

$confirm\_transaction := confirm\_transaction \cup \{ xx \}$

END

END ;

ATM_withdraw $\widehat{=}$

BEGIN

ANY xx , yy , zz

WHERE

$xx \in transactionId \wedge$

$yy \in accountNumber \wedge$

$zz \in \mathbb{N} \wedge$

$xx \notin confirm\_transaction \wedge$

$xx \mapsto yy \notin ATMData \wedge$

$yy \mapsto zz \notin networkData \land$

$xx \mapsto (yy \mapsto zz) \in req\_withdraw \land$

$zz \leq accountBalance (yy)$

THEN

$networkData (yy) := zz \parallel$

$ATMData (yy) := zz \parallel$

$totalBalance (yy) := totalBalance (yy) - zz$

END

END ;

Bank_withdraw $\widehat{=}$

BEGIN

ANY xx , yy , zz

WHERE

$xx \in transactionId \land$

$yy \in accountNumber \land$

$zz \in \mathbb{N} \land$

$xx \notin confirm\_transaction \land$

$xx \mapsto (yy \mapsto zz) \in req\_withdraw \land$

$yy \mapsto zz \in networkData \land$

$zz \leq accountBalance (yy) \land$

$ATMData (yy) = networkData (yy)$

THEN

$networkData := networkData - \{ yy \mapsto zz \} \parallel$

$ATMData := ATMData - \{ yy \mapsto zz \} \parallel$

$accountBalance (yy) := accountBalance (yy) - zz \parallel$

$confirm\_transaction := confirm\_transaction \cup \{ xx \}$

END

END ;

network_goes_down $\widehat{=}$

$networkData := \{\}$ **;**

corrupt_network $\widehat{=}$

BEGIN

ANY xx , vv **WHERE**

$xx \in accountNumber \wedge$

$xx \in \mathsf{dom}\ (networkData) \wedge$

$vv \in \mathbb{N}$

THEN

$networkData (xx) := networkData (xx) - vv$

END

END

END

Cross-references for bankSystem3

ACCOUNTS	*Globals*	DEFINITIONS	172
Bool_TYPE		MACHINE	166
Globals		MACHINE	171

MACHINE *Globals*

CONSTANTS

$maxAccounts$

PROPERTIES

$maxAccounts = 6$

DEFINITIONS

$ACCOUNTS \ \widehat{=} \ 1 \,..\, maxAccounts$ **;**

$MAXINT \ \widehat{=} \ 5000$ **;**

$ee\_ser \ \widehat{=} \ 5$ **;**

$dd\_ser \ \widehat{=} \ 17$ **;**

$nn \ \widehat{=} \ 7$ **;**

$ee\_cli \ \widehat{=} \ 9$ **;**

$dd\_cli \ \widehat{=} \ 11$

END

Appendix D

Time Schedule