

COMPUTING SCIENCE

Modelling and Refinement of the MONDEX Electronic Purse in VDM

Zoe Andrews, Jeremy Bryans, John Fitzgerald, John Hughes, Richard Payne, Ken Pierce and Steve Riddle

TECHNICAL REPORT SERIES

Modelling and Refinement of the MONDEX Electronic Purse in VDM

Z. Andrews, J. Bryans, J. Fitzgerald, J. Hughes, R. Payne, K. Pierce, S. Riddle

Abstract

We present the modelling and refinement of the Mondex Electronic Purse "challenge problem". Our approach uses the well-established Vienna Development Method. Abstract and concrete models are presented, with a refinement step that addresses error detection and recovery in the implementation. We exercise the range of verification and validation techniques associated with VDM (animation, testing and proof.) The study suggests future developments in machine-assisted proof to support the development of VDM models and their associated refinements. We compare the models and proof developed in this way with the approaches taken in several other contemporary studies of the Mondex system.

Bibliographical details

ANDREWS, Z., BRYANS, J., FITZGERALD, J., HUGHES, J., PAYNE, R., PIERCE, K., RIDDLE, S.

Modelling and Refinement of the MONDEX Electronic Purse in VDM
[By] Z. Andrews, J. Bryans, J. Fitzgerald, J. Hughes, R. Payne, K. Pierce, S. Riddle
Newcastle upon Tyne: Newcastle University: Computing Science, 2011.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1308)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1308

Abstract

We present the modelling and refinement of the Mondex Electronic Purse "challenge problem". Our approach uses the well-established Vienna Development Method. Abstract and concrete models are presented, with a refinement step that addresses error detection and recovery in the implementation. We exercise the range of verification and validation techniques associated with VDM (animation, testing and proof.) The study suggests future developments in machine-assisted proof to support the development of VDM models and their associated refinements. We compare the models and proof developed in this way with the approaches taken in several other contemporary studies of the Mondex system.

About the authors

Zoe obtained her BSc in Computer Science & Statistics at the University of Newcastle upon Tyne in 2004. After spending the Summer working as a General Duties Assistant for DIRC, she went on to take an MSc in Process Analytics and Quality Technology, also at Newcastle. After completing her MSc she worked as an RA for two months in the area of Complex Systems. Zoe spent two years working as an RA on the ReSIST network of excellence and was responsible for work on developing metadata-based descriptions of resilience mechanisms and providing formal support for decision making over such mechanisms. Zoe is now a full time PhD student under the supervision of Dr. John Fitzgerald. She is investigating ways in which stochastic reasoning can be combined with logical reasoning for the specification and analysis of fault-tolerant systems.

Jeremy received his BSc in Mathematics and Computer Science from Reading University in 1993, and his PhD in 1997, also from Reading University. He has worked in a number of university departments, including Royal Holloway, Kent and Stirling, and has been at Newcastle since December 2002. His research is in the security of information within large computer-based systems. A particular area of current interest is access control the development and maintenance of access control policies within dynamic coalitions. In the past at Newcastle he has worked on including DIRC (the Interdisciplinary Research Collaboration on Dependability) and GOLD (Grid Oriented Lifecycle Development) He is currently employed on the User Friendly Grid Security project and TrAmS (Trustworthy Ambient Systems). He is part of the RESIST network, and a member of RESIST's working group on Verification.

John Fitzgerald is Reader in Computing Science and Deputy Director of the Centre for Software Reliability at Newcastle University. He is the Newcastle Principal Investigator in the DESTTECS project, in which he leads work on methods for exploring design spaces for dependable embedded systems in terms of co-simulation. He is Chairman of Formal Methods Europe.

John Hughes received his PhD from Newcastle University in 2011.

Richard received his BSc (Hons) in Computing Science from Newcastle University in 2005. He has returned to Newcastle University to undertake a PhD under the supervision of Dr. John Fitzgerald, as part of the DIRC project, in the field of Verifiable Resilience in Architectural Reconfiguration. As part of his PhD, Richard provided a basis for the formal verification of policies defined using a reconfiguration policy language (RPL) for the governance of resilient component-based systems. Richard worked as an RA on the Ministry of Defence funded SSEI project and was involved in the 'Interface Contracts for Architectural Specification and Assessment' sub task, investigating the use of contract-based interface specification in system-of-system architectural models. Richard is now working on the COMPASS project, on the use of model-based techniques for developing and maintaining systems-of-systems.

Ken received his BSc (Hons) in Computer Science (Software Engineering) from Newcastle University in 2005. Ken studied for a PhD under the supervision of Prof. Cliff Jones as part of the EPSRC "Splitting (Software) Atoms Safely" project. His thesis, titled "Enhancing the Usability of Rely-Guarantee Conditions for Atomicity Refinement", was published in December 2009. Ken is currently working on the DESTTECS project (destecs.org). The project is a consortium of research groups and companies working on the challenge of developing fault-tolerant embedded systems. Specifically, the aim is to explore collaborative modelling and simulation in the

design of embedded systems. The Newcastle team is principally concerned with methodology and how fault-tolerance can be incorporated into these collaborative, multi-disciplinary models.

Dr Steve Riddle is a lecturer in Computing Science. His research contributions have included novel safety analysis techniques (INCO-COPERNICUS ISAT), component contracts and protective wrapper architectures (EPSRC project DOTS) and resilience (FP6 NoE ReSIST). He is a Theme Lead in the SSEI project and leads work in dependable dynamic reconfiguration in that project. Beside these projects his research interests include requirements volatility, evidence-based argumentation and risk management.

Suggested keywords

Mondex
Specification
Refinement
Proof
VDM

Modelling and Refinement of the MONDEX Electronic Purse in VDM

Zoe Andrews, Jeremy Bryans, John Fitzgerald, John Hughes,
Richard Payne, Ken Pierce and Steve Riddle
School of Computing Science, Newcastle University, UK

December 21, 2011

Abstract

We present the modelling and refinement of the Mondex Electronic Purse “challenge problem”. Our approach uses the well-established Vienna Development Method. Abstract and concrete models are presented, with a refinement step that addresses error detection and recovery in the implementation. We exercise the range of verification and validation techniques associated with VDM (animation, testing and proof.) The study suggests future developments in machine-assisted proof to support the development of VDM models and their associated refinements. We compare the models and proof developed in this way with the approaches taken in several other contemporary studies of the Mondex system.

1 Introduction

The work reported here was carried out as a contribution to the initial Grand Challenge 6¹ problem – that of verifying the specification of an electronic purse system called *Mondex*. In [SCW00] the initial formal development and verification of the system using the Z specification language was carried out, in which a number of properties were verified. Some time after the original verification was carried out a number of teams decided to redo the Mondex specification using a variety of methods and technologies, in order to assess the current state of automated formal verification technology. This was as part of the Verification Grand Challenge [WB07]. The work reported now was carried out some time after that initial effort using the Vienna Development Method (VDM) in order to assess the level of maturity of verification and validation support within VDM.

The Vienna Development Method (VDM) contains a set of validation techniques for modelling and analysing systems including testing, animation and proof. This paper reports on our work and results using each of these techniques.

The Mondex system provides a substitute for cash in which electronic cards store a balance as part of their state and have the ability to take part in transactions by sending money to, and receiving money from, other cards. The key properties of the Mondex system are:

- no money may be created, so a purse balance cannot increase without a decrease in another purse balance, and
- all money must be accounted for within the system.

As in [SCW00], we developed two models of the system, an abstract and a concrete one. Both the abstract models were developed in the object-oriented VDM++ dialect. A conversion of the models to the (non object-oriented) VDM-SL dialect was performed in order to show refinement.

¹<http://www.ukcrc.org.uk/grand-challenge/index.cfm>

The bulk of our effort in this task was consumed by the proof work which was undertaken. To show that the concrete model is a valid implementation of the abstract one, a number of proof obligations must be discharged. A *retrieve* function from concrete worlds to abstract ones is given, which must be shown to be both *total* — every concrete world must be retrieved to an abstract one — and *adequate* — for any abstract world, there is a corresponding concrete world such that applying the retrieve function to the concrete world returns the abstract world. Further, the operations defined must be shown to respect the retrieve function. We chose a representative operation and demonstrated for it that the retrieve function was respected. As a learning exercise for (some of) those involved, all these proof obligations were discharged manually.

We developed tests to validate a number of properties for the abstract and concrete models. In this we made use of the tool support provided for VDM, and so developing and executing these tests consumed much less effort. The model was also animated, and a user-interface to the animation was developed.

The paper is structured as follows. In Section 2 we discuss the VDM validation technologies used. In Section 3 and Section 4 we explain the abstract and concrete models, and in Section 5 we describe the work carried out in testing and animating the model. Section 6 is a discussion of work involved in carrying the proof task and Section 6.5 briefly describes the team and estimates the effort required for the various tasks. Finally, Section 7 compares our work to other contemporary approaches and we draw conclusions in Section 8. Specifications and proofs are given in Appendices A and I.

2 The Vienna Development Method

Our contribution to the Mondex challenge problem uses the formal specification language VDM [Jon90]. VDM is a mature model-based specification language that supports the construction of abstract system models. The specifications in the Mondex case study are developed in the object-oriented dialect of the language VDM++ [FLM⁺05]. VDM++ is supported by a well-established industry strength toolset VDMTools² [FLS08], and more recently by the open source toolset, Overture³ [LFW⁺10]. To carry out the proofs, these specifications are translated into the “flat” or non-object-oriented dialect, known as VDM-SL [Jon89].

2.1 Modelling and Analysis

A model in VDM++ is composed of a set of abstract class specifications. Each class defines a set of data types, instance variables and operations. Data types are either simple types such as *nat* or *bool*, or abstract collection types such as sets, sequences or records. Data types may be further constrained through predicate expressions known as invariants. Operations update state variables and therefore describe the functionality of a class. Operations may be defined implicitly, as pre- and postcondition pairs, or explicitly, using imperative statements. Functions can be also be specified in a similar way to operations, but may not refer to state variables.

2.1.1 Proof in VDM

VDM is built on well-established proof theory [BFL⁺94]. This allows us to reason about models written in VDM. We use the VDM-SL dialect of VDM for generating and discharging the proof obligations (Section 6 discusses how VDM++ models are converted into VDM-SL models).

There are essentially two sets of proof obligations considered in this work: those that show the internal consistency of a single model; and those that demonstrate the refinement relation between the abstract and concrete models of the Mondex system.

An example of a proof obligation that demonstrates internal consistency of a model is that of *satisfiability*. This requires, of an operation *op*, that for any input that satisfies *pre-op* (the precondition of the operation) there must exist some state that satisfies *post-op* (the postcondition of the operation). In this work the internal consistency proof obligations were generated and proved automatically (see Section 6.4).

²<http://www.vdmtools.jp/en>

³<http://www.overturetool.org/>

2.1.2 Simulation using VDM models

The tools supporting VDM allow a user to exercise the executable parts of a model through an interpreter. This permits validation that the required behaviour has been accurately captured in the model [FLM⁺05]. A variety of testing and validation tools have been developed on this framework. In particular this study makes use of the combinatorial testing feature [LL09] provided in Overture and an animation tool that provides a GUI to the Mondex models [Cla09].

The combinatorial test feature in Overture automatically generates and conducts a large number of tests on the model through the use of regular expressions (known as traces). A trace is sequence of one or more operations with instantiated parameters. Each test can result in one of the following outcomes:

- *Pass* — The inputs are valid (the precondition of each operations holds at the time of execution) and executing each operation over its inputs provides valid results (the operation postcondition holds). None of the invariants are broken during the test.
- *Inconclusive* — The inputs of one of the operations are not valid (the precondition of that operation does not hold).
- *Fail* — A runtime error occurs or the outputs of an operation are invalid for valid inputs (the operation postcondition does not hold even though its precondition does).

As the combinatorial test feature of Overture executes the body of model operations to determine the test outcome, implicitly defined operations are not supported. This extends to explicitly defined operations which make operation calls to implicit operations.

VDMtools provides an interface to execute VDM from the Java programming language [FLM⁺05]. Using this interface it is possible to program a GUI to a VDM model, allowing the model to be exercised in interactive user-driven scenarios. Doing so enables easier validation by domain experts who may have little knowledge of the formal notation used in the model. Both the combinatorial testing and the animation carried out are discussed in Section 5.

2.2 Reification

An important aspect of many formal methods is *refinement*, the process of showing that a (more) concrete specification respects an abstract one. In [Jon89] this is referred to as *reification*, and we use that term here. This includes showing that more concrete data structures represent the same information as their abstract equivalents, known as *data reification*. In VDM a *retrieve function* (often abbreviated to *retr*) is used to map concrete data representations to their equivalent ones in the abstract model. For example the retrieve function, whose signature is shown below, converts a concrete world (*ConWorld*) into an abstract one (*AbWorld*).

retr: *ConWorld* → *AbWorld*

There are two properties that need to hold in order for the retrieve function to be correct. The function must be *total* – it must map every concrete state to an abstract state, and *adequate* – every abstract state must have a concrete representation.

More precisely, the totality proof obligation (called *retr-form*) requires that applying the retrieve function to a given state in the concrete specification gives a valid state in the abstract specification. The *retr-form* proof obligation is given below for concrete state *ConWorld* and abstract state *AbWorld*. *mk-ConWorld* is a constructor for the concrete world, so *mk-ConWorld*(*auth*,*cp*) creates a concrete world instantiated with the values *auth* and *cp*.

$$\boxed{\text{retr-form}} \frac{mk\text{-}ConWorld(auth, cp): ConWorld}{retr(mk\text{-}ConWorld(auth, cp)): AbWorld}$$

The adequacy proof obligation requires that for a given state in the abstract specification there exists a corresponding state in the concrete specification (i.e. that there exists some concrete state c such that $\text{retr}(c)$ returns the abstract state). For ConWorld and AbWorld this is formalised as follows:

$$\boxed{\text{retr-adequate}} \frac{a: \text{AbWorld}}{\exists c: \text{ConWorld} \cdot \text{retr}(c) = a}$$

Each operation in the concrete specification also needs to satisfy a further two proof obligations. Firstly, the *domain* proof obligation requires that the domain of the concrete operation is at least that of its abstract equivalent. Secondly, the *result* proof obligation requires that the concrete operation cannot achieve behaviour not permitted by its abstract equivalent. The retrieve function used in the Mondex case study, and the resulting proof obligations, are explained in more detail in Section 6.3.

For this case study the reification proof obligations were discharged by hand using the VDM proof theory rules found in [BFL⁺94].

3 Abstract Model

The Mondex system is a electronic purse system that hosts the purses on smart cards. Money may be transferred from one purse to another. It is a commercial development, but a commercially sanitised description is given in [SCW00], in which the state of the cards in the system and the money transfer protocol are modelled.

An abstract and a concrete model are developed, both modelled initially in VDM++. This section describes the abstract model, in which money is transferred between purses in single, atomic, operations. The model consists of an abstract purse and an abstract world, each of which are separate classes containing related operations. The full abstract model is given in Appendix A.

3.1 Abstract Purse

The *abstract purse* in the VDM++ model is represented by the class *AbPurse*, which consists of two instance variables, given in Figure 1. The variable *balance* stores the current balance of the purse, and *lost* stores the total of any money that has been lost in failed transactions.

instance variables

```
private balance: N;
private lost: N;
```

Figure 1: Instance variables of the class *AbPurse*

Objects of this class do not transfer money to another purse directly (this ability is in the class *AbWorld*). Instead they provide operations that allow these variables to be updated in a controlled way. These include operations to increase and decrease the balance of the purse, used when transferring money from one purse to another, and an operation to increase and decrease the value of the variable *lost*, used when transactions fail.

Some of the operations have a *precondition* governing their execution. For example, the *ReduceBalance* operation in Figure 2 has a precondition which ensures that it may only be executed if there are sufficient funds in the purse.

The class also contains accessor operations which simply return the values of the instance variables. These are used in the pre and postconditions of the money transfer operations in class *AbWorld*.

```

ReduceBalance:  $\mathbb{N} \rightarrow ()$ 
ReduceBalance(val)  $\triangleq$  balance := balance - val
pre balance  $\geq$  val;

```

Figure 2: The *ReduceBalance* operation from the class *AbPurse*

3.2 Abstract World

The abstract world is the world in which the abstract purses exist. It contains all authentic purses. In our abstract model the abstract world is represented by the class *AbWorld*, which contains a record (also called *AbWorld*) given in Figure 3. A class invariant ensures that all of the purses in the abstract world are authentic purses.

instance variables

```

private AbWorld :: authentic : PurseId-set
                abPurses : PurseId  $\xrightarrow{m}$  AbPurse
inv mk-AbWorld(auth, pm)  $\triangleq$   $\forall$  name  $\in$  dom pm  $\cdot$  name  $\in$  auth

```

Figure 3: Instance variables of the class *AbWorld*

AbWorld contains three operations: *AbTransferOk*, *AbTransferLost* and *AbIgnore*. Any purses involved in any of these operations must be in the set of authentic purses, and this is ensured by preconditions for each operation.

The *AbIgnore* operation does nothing, and has a postcondition that ensures that instance variables of the class remain unchanged during the operation. The *AbIgnore* operation will be refined by any operations in *ConWorld* which change only those variables introduced in *ConWorld* and consequently have no impact on the abstract world.

AbTransferOk represents a successful transfer of money between two purses, where an amount is taken from the balance of a *from* purse and added to the balance of a *to* purse. Certain security properties (described in more detail in Section 5.1) are ensured by preconditions. A transfer can only occur if both purses in the transaction are unique and authentic (security property 3), and if the *from* purse has sufficient funds to complete the transaction (security property 4). The postcondition of the operation requires that sum of the total (where total = balance + lost) of both purses is unchanged by the transaction as it was before (security property 2.1), that the lost component of the *to* and *from* purses remains unchanged (since this operation models a successful transfer), and that the balance and lost components of all other purses not involved in the transaction remain unchanged (security property 1). The postcondition also requires that no purses are added to the map of purses during the transaction.

The second of the transaction operations, *AbTransferLost*, models a failed transfer between two purses. The amount which was intended to have been transferred will be removed from the *balance* variable of the *from* purse and added to the *lost* variable of the *from* purse. This operation is again designed to respect the desired security properties through pre and postconditions: for *AbTransferOk* to occur, both purses in the transaction must be unique and authentic and sufficient funds must be present in the balance of the *from* purse. After the operation the total (where total = balance + lost) of the *from* purse remains unchanged and all other purses remain unchanged (including the *to* purse), and no purses are added to the set of purses during the transaction.

4 Concrete Model

The abstract world described in Section 3 above is just that — an ideal world where purses can instantaneously transfer sums of money between each other. While this allows examination of the problem, in the real, concrete system, purses must communicate with each other in a realizable way, while maintaining the security properties detailed in Section 5.1. The real Mondex system utilizes a message-passing protocol to allow cards to transfer money. This section details our concrete model, which models concrete purses that implement this message-passing protocol (as in [SCW00]). That this concrete model is a reification of the abstract model is considered in this paper in Section 6. The full concrete model is given in Appendix B.

As with the abstract model detailed above, the concrete model contains a concrete purse and concrete world. In addition, we have chosen to explicitly model the “ether” over which messages are passed, in order to explore the notion of messages going missing or being compromised by hostile parties.

Note that, as in [SCW00], the concrete model is still an abstraction of the real system. We do not, for example, model the encryption that protects messages that are transmitted over the ether. These details are not included in the “sanitized” Mondex specification. The model does not include timeouts on messages and assumes that the purses have unlimited memory. A second refinement to a cryptographically protected system is present in [HSGR08].

4.1 Concrete Purse

The instance variables (state) of the concrete purse are given in Figure 4. They are: *name* (the purse’s identity); *balance* (current balance); *nextSeqNo* (the sequence number to be used in the next transaction); *status* (the purse’s current status); *currTrans* (the details of the current transaction); and *exLog* (exception log recording failed transactions). As with the abstract purse, the concrete purse has various accessor methods to retrieve the values of state components; and a constructor to create a purse.

instance variables

```
public name: ConWorld·PurseId;  
public balance: nat;  
public nextSeqNo: nat;  
public status: Status;  
public currTrans: [TransDetails];  
public exLog: TransDetails-set;  
  
private ether: [Ether];
```

Figure 4: Instance variables (state) of a *ConPurse*

The specification of the concrete purse is more complex than the abstract purse, since it models the message-passing protocol upon which the card system operates. A number of new types are introduced in order to model this protocol. It is therefore useful to describe the class with reference to the protocol itself.

If there is a problem within the message protocol, a purse can abort and record the failed transaction in an exception log. This log performs the role of the abstract *lost* (which could be calculated as the sum of the value of the exceptions within the exception log). In this current model, the circumstances under which the purse aborts are not modelled, but these could include messages not being received (within a given time) or the receipt of a repeated message (which could indicate a playback attack).

A definition of the *Message* type, showing the kinds of messages used in the protocol, is given in Figure 5. A complete, successful transaction between two purses consists of five messages: *StartFrom*, *StartTo*, *Req*, *Val*, *Ack*. Recall that in a transaction there is the notion of a *from* purse (which is sending an amount of money) and a *to* purse (which is receiving this money). The *StartFrom* and *StartTo* messages indicate that

a transaction has been initiated. These messages carry details of the other purse involved in the transaction (represented by the type *CounterPartyDetails*).

```
public Message = StartFrom | StartTo | <READEXCEPTIONLOG> | Req | Val | Ack |
                ExceptionLogResult | ExceptionLogClear | <UNPROTECTED>;
```

Figure 5: Definition of the *Message* type, showing the various messages used for communication between *ConPurses*

The *to* purse then sends a *Req* message requesting the value be transferred. In response, the *from* purse sends a *Val* message (which is the point at which the money is transferred). The *to* purse must then acknowledge receipt of the money with an *Ack* message. These messages are represented by the union type *Message* in the concrete purse definition. Each of the *Req*, *Val*, *Ack* messages carry details of the current transaction using the type *TransDetails* (see below).

There are four other types of messages represented by the *Message* type (again, see Figure 5). These are: <READEXCEPTIONLOG> (representing a request to read the exception log); *ExceptionLogResult* (the result of reading the exception log); *ExceptionLogClear* (representing a request to clear the exception log); and <UNPROTECTED> (representing a message that is unencrypted and which therefore cannot be relied upon to be genuine).

The purse class handles those messages which invoke actions of the purse with the *RecMsg* operation, which uses a case statement to in turn invoke an operations corresponding to each message. For example, *StartFromOkay* responds to *StartFrom* messages, *OpReq* handles *Req* messages and so on.

In order to ensure that the messages sent and received by purses are both genuine and not repeated (as they would be in a replay attack), a sequence number from each purse is given to each transaction (hence two sequence numbers are recorded for each transaction).

The details of a transaction are recorded with the *TransDetails* (transaction details) type, which comprises: the identity of the two purses, the sequence number from both purses; and the value of the transaction. A definition for this type is given in Figure 6. This type forms the content of the *Req*, *Val* and *Ack* messages. The purse records the details of the current transaction in order to check that any messages received are genuine.

```
public TransDetails :: fromPurse : Conworld'PurseId
                    toPurse : Conworld'PurseId
                    fromSeqNo : nat
                    toSeqNo : nat
                    val : nat
```

Figure 6: Definition of the *TransDetails* type, which records information about a transaction

In addition, each purse tracks its position within the protocol with a status. This information is captured in the union type *Status*, presented in Figure 7. The four statuses are: <IDLE> (the purse may begin a transaction); <EPR> (the purse is expecting a *Req* message); <EPV> (the purse is expecting a *Val* message); <EPA> (the purse is expecting an *Ack* message).

```
private Status = <IDLE> | <EPR> | <EPV> | <EPA>;
```

Figure 7: Definition of the *Status* type, representing the status of a *ConPurse*

The purse should not respond to a message that is is not expecting (as this would break the protocol and may break the security properties). For example, the purse should not process a *Val* message if its state is not <EPV>, the purse should not begin a transaction if it is not <IDLE>, and so on. The various restrictions

detailed above are captured using preconditions on the message-handling operations. A partial definition for the *StartFromOkay* operation (showing the precondition) is given in Figure 8. This precondition ensures the following:

- that the purse is not entering a transaction with itself
- that the purse has a sufficient balance to make the transaction
- that the purse is in the <IDLE> state (ready for a transaction)
- that the purse knows about the ether (see Section 4.3).

```

StartFromOkay: CounterPartyDetails →  $\mathbb{B}$ 
StartFromOkay(cpd)  $\triangleq$ 
...
pre cpd.name ≠ name ∧ cpd.val ≤ balance ∧ status = <IDLE> ∧ ether ≠ nil;

```

Figure 8: Pre-condition for the *StartFromOkay* operation of *ConPurse*

The precondition for *StartToOkay* is the same, except it doesn't need to check the balance. The *OpReq*, *OpVal* and *OpAck* operations ensure that the purse is in the correct state to receive the message and that the message is part of the current transaction (i.e. that the transaction details of the message match those held by the purse). The exception log operations require that the purse is <IDLE>.

The concrete purse can also perform four “invisible” operations (those which don't result in messages being sent). These are: *Increase* (increase the sequence number); *Abort* (abort the current transaction); *LogIfNecessary* (adds the current transaction to the exception log if the purse aborted during a crucial step); *Image* (an underspecified operation responsible for ensuring that the exception log isn't cleared without first being read).

4.2 Concrete World

The *ConWorld* class is simpler than the *AbWorld* class, since much of the detail of the concrete model is handled within the *ConPurse* class. The instance variables of *ConWorld* are given in Figure 9. As with abstract world however, the concrete world contains a map from purse identities to purses and a set of authentic purse identities. In addition, the world contains an *ether*, which models how messages are sent between purses and is detailed below.

instance variables

```

private conauth: PurseId-set;
private conpurses: PurseId  $\xrightarrow{m}$  ConPurse := {};
private ether: Ether;
private previousmessages: ConPurse' TransDetails-set;
inv  $\forall name \in \mathbf{dom} \text{ conpurses} \cdot name \in \text{conauth}$ ;

```

Figure 9: Instance variables of the class *ConWorld*

The class has a constructor, accessor functions and a single transaction operation, *ConTransfer*, that initiates a transaction between two purses (this can be used for testing and animation, see Section 5).

4.3 Ether

In [SCW00], messages at the concrete level are considered to travel across the “ether”. All messages are considered to be sufficiently protected when traveling through the ether (e.g. through encryption). Our use of VDM however allows us to explicitly model this ether and explore how the model is affected when the ether is compromised.

We model this with an abstract Ether class. The main operation is *SendMsg*, which transmits a message to a purse. Each concrete purse contains an instance of an Ether class and uses this operation to send all messages through. The Ether class also contains four operations to allow the world to inject messages into the ether: *StartFrom*, *StartTo*, *ExceptionLogClear*, *ReadExceptionLog*. These messages are not generated by purses, rather they are generated in response to some external input. They are useful for testing and animation of the model.

Subclasses of Ether should then implement these five operations. We initially defined a PerfectEther class, which always delivers all messages successfully. We also created various subclasses of Ether that drop certain messages, in order to test the protocol. This is explored in Section 5.1. It would also be possible to create Ether classes that generated malicious messages, or replayed previous messages, to further test the protocol.

5 Testing and Animation

5.1 Testing

The Overture Integrated Development Environment (IDE) allows a user to create, syntax and type check models and use an interpreter to explore their behaviour. One aspect of Overture is a combinatorial testing feature which allows for automated high-volume testing of VDM++ models through the use of regular expressions, known as traces. Traces are an extension to the VDM++ language [LL09]. A trace consists of a let expression defining a variable list and a sequence of model operations. The combinatorial testing view within Overture reports the results of a test trace using the result categories in Section 2.1.2 (*Pass*, *Inconclusive* and *Fail*), together with any output generated by the sequence of model operations in the test trace.

Determining the test coverage obtained through combinatorial testing is not currently available in the Overture tool, and bespoke testing scripts must be used if test coverage data is required. Measuring test coverage is a planned feature for a future release of the tool [LFW⁺10]. Testing of VDM++ models in VDMTools⁴ is supported by coverage analysis tools, which have been successfully deployed in a number of industrial applications [FL06].

In the remainder of this section we discuss the combinatorial testing used to verify the security properties below. We address the test traces used to verify these properties in both the concrete and abstract models. This required the definition of new classes for the VDM++ versions of the abstract and concrete models: *AbTest* and *ConTest*. These classes include the instance variables, values and traces required for the tests performed and are given in full in Appendix C and Appendix D.

Security Properties

We consider the security properties defined in [SCW00]. Below, we provide a brief explanation of these properties and identify the models within which each one is checked.

Security Property 1: No value creation This property requires that no value may be created within the Mondex system. In other words, the sum of the balances of all purses must not increase. The property is checked within both models.

⁴Downloads are available via <http://www.vdmttools.jp/en/>

Security Property 2.1: All value accounted all value must be accounted for in the system. This property stipulates that the sum of all purses’ balances and lost components does not change. This property is checked within the abstract model.

Security Property 2.2: Exception logging This property is a slight variation on SP2.1 relating to the concrete model only. Within the concrete purse the lost component is replaced with an exception log. If a purse aborts a transfer where value may be lost, then the transaction details of that transfer must be logged.

Security Property 3: Authentic purses This property states that a transfer can occur only between purses deemed authentic. It applies to both abstract and concrete models.

Security Property 4: Sufficient funds The property requires that a transaction may only occur if there are sufficient funds in the balance of the *from* purse. This property applies to both models.

SP1: No Value Creation

The *no value creation* security property stipulates that no value may be created in the system. That is, the total balances of the abstract purses must not increase after any transfer operations. To aid in testing we modified the AbWorld class by adding an auxiliary operation, *TotalWorldBalance*, to determine the total amount in all purse balances. An example test trace from the AbTest class is shown in Figure 10. The trace performs three transfer operations, checking the world balance before and after these operations.

```

AbTestSP1:
  let x = abworldSP1 in
    let n ∈ {0, ..., 5} in
      (
        x.TotalWorldBalance();
        (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
        (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
        (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
        x.TotalWorldBalance()
      )

```

Figure 10: Extract from AbTest class, depicting trace for SP1

The test trace produces a total of 162 tests, all of which pass — demonstrating that no pre- or post-conditions were violated. We manually check the results of each test, comparing the returned value of the *TotalWorldBalance* before and after the 3 transfer operations. The postconditions of the transfer operations ensure the SP1 security property holds, and so the fact that all tests pass is a good indication that the abstract model conforms to SP1.

We take the same approach for the concrete model, though do not provide the details here. The full test class for the concrete model is included in Appendix D for reference.

SP2: All Value Accounted

The second security property, *all value accounted*, differs slightly between the abstract and concrete model. In the abstract world, SP2.1 ensures that the total value in the balance and lost fields of all purses remains the same after any transfer operation. In the concrete world, however, SP2.2 states that if a purse aborts transfer at a point in which value could be lost, then the transfer details are logged.

In the abstract model, we use the same testing tactic as SP1. A different auxiliary operation, *TotalWorldValue*, is used to determine the total value in the world in the manual check of test results. Due to the similarities

Ether	From		To	
	bal	lost	bal	lost
req	-	-	-	+n
val	-n	+n	-	+n
ack	-n	+n	+n	-

Table 1: Results of SP2 test showing how relation between the point at which messages are dropped and the change in purse balance and lost variables (for a transaction of value n)

with SP1, we do not include the details of this test trace here, however the tests developed for the concrete model do raise some interesting differences.

To perform the tests for the concrete model, we first created three new faulty ether classes to model messages being dropped at each stage of the message exchange protocol. The *FaultyEtherReq*, *FaultyEtherVal* and *FaultyEtherAck* ether classes drop request, value and acknowledgement messages respectively. Message dropping is handled by invoking the *abort* operation of both purses in a transaction upon the ether instance receiving the relevant message. We also modified the *ConWorld* constructor parameter list to include a character indicating the ether class to use in purse transactions.

The test trace *ConTestSP2*, given in Figure 11, uses one of three worlds; *conworldSP2ack*, *conworldSP2req* or *conworldSP2val*. These test worlds contain purses with the same contents, only the ether instance used in each case differs.

instance variables

```

public conworldSP2ack: ConWorld := new ConWorld({fid, tid}, {fid ↦ conpurseSP2af, ...}, 'a');
public conworldSP2req: ConWorld := new ConWorld({fid, tid}, {fid ↦ conpurseSP2bf, ...}, 'r');
public conworldSP2val: ConWorld := new ConWorld({fid, tid}, {fid ↦ conpurseSP2cf, ...}, 'v');
public conworldsSP2: ConWorld-set := {conworldSP2ack, conworldSP2req, conworldSP2val};

```

traces

```

ConTestSP2:
let x ∈ conworldsSP2 in
  let n ∈ {0, ..., 5} in
    (
      x.GetPurseBalLost(fid);
      x.GetPurseBalLost(tid);
      x.ConTransfer(fid, tid, n);
      x.GetPurseBalLost(fid);
      x.GetPurseBalLost(tid)
    )

```

Figure 11: Extract of *ConTest* class with instance variables and trace for SP2

The test trace results in 18 tests, all of which pass. The test trace includes a call to an accessor operation, *GetPurseBalLost*, in order to investigate the contents of the *from* and *to* purses. Table 1 below summarises the changes in the balance and exception logs of the purses involved in the transaction. The term *lost* denotes the total value of logged transactions in a purse's exception log. The resultant behaviour exhibited in the test trace conforms to the security property SP2.2. Purses update their log if the purse is in the <EPV> or <EPA> state — that is if the *to* purse is expecting a value message or the *from* purse is expecting an acknowledgement.

SP3: Authentic Purses

The third security property, *authentic purses*, requires that transfers are only valid between those purses deemed authentic. Two test traces were added for the purposes of testing this property, and some additional purse identifiers added to the value list of the test class. A stronger property was also tested that denies transfers between authentic purse identifiers which are not linked to purse objects. In Figure 12, an extract of *AbTest* is given, detailing the relevant constructs for SP3.

```
AbTestSP3:
  let x = abworldSP3 in
  let n = 5 in
  let f = fid in
  let t ∈ {tid, undef1, undef2, undef3, undef4} in
  (
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n))
  )

AbTestSP3b:
  let x = abworldSP3 in
  let n = 5 in
  let t = tid in
  let f ∈ {fid, undef1, undef2, undef3, undef4} in
  (
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
    (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n))
  )
```

Figure 12: AbTest class extract containing test trace for SP3

AbTestSP3a attempts to transfer a set amount (5 units) from an authentic purse to a range of purse identifiers: *fid*, an authentic purse and one present in the *abPurses* domain; *undef1* and *undef2*, authentic purse identifiers that do not map to purse objects; and *undef3* and *undef4*, which are not authentic purse identifiers. *AbTestSP3b* attempts to transfer from this range of purses to an authentic purse.

The two test traces result in 80 test cases. Of these tests, the transfers between authentic purses respect the pre- and postconditions of the relevant operations and thus the test cases yield pass results. Those tests cases between authentic purses not mapped to purse objects, and non-authentic purse identifiers result in inconclusive results due to the precondition failure of the relevant transfer operation.

As with SP1, the tests are similar on the concrete world, and so are not given in this section.

SP4 : Sufficient Funds

The final security property, Sufficient Funds, dictates that the purse from which funds are transferred has sufficient funds in their balance. A simple boundary test trace for this property is required for both abstract and concrete worlds. The *AbTest* extract given in Figure 13 details the trace *AbTestSP4* transfers between 0 and 10 units from a purse containing only 5 units using both *TransferOK* and *TransferLost* operations.

The *AbTestSP4* trace results in 22 test cases. Of these test cases, 11 passed — those for which the amount being transferred was less than or equal to the starting balance of the *from* purse. Those transactions with the value being transferred greater than the purse balance, returned inconclusive due to precondition violation of either the *TransferOk* or *TransferLost* operations.

```

AbTestSP4
let  $x = \text{abworldSP4}$  in
  let  $n \in \{0, \dots, 10\}$ 
    ( $x.\text{AbTransferOk}(fid, tid, n) \mid x.\text{AbTransferLost}(fid, tid, n)$ )

```

Figure 13: AbTest class extract containing test trace for SP4

The test trace for the concrete world follows the same approach and is given in Appendix D.

5.2 Animation

VDMTools permits the control of the interpreter via an API that may be accessed through an application-specific GUI. This allows domain experts unfamiliar with the underlying formal notation an opportunity to exercise the model directly, rather than using code derived from the model. We refer to this as an *animation*-based approach to model validation. Such an approach was followed by Clarke in her animation of the concrete VDM++ model of Mondex developed in this paper [Cla09].

The animation uses an architecture based on the model-view-controller pattern used in the Dynamic Coalitions Workbench [FBG⁺08], which allows users to experiment with models of alternative information flow policies in a dynamically changing group of communicating agents.

Clarke’s Mondex GUI allows execution of user scenarios based on the operations modelling parts of a transaction, including abnormal behaviour, such as attempting to transfer funds to the same purse they came from, and modelling of a range of faults. Although validation of the model succeeded using this GUI, it is worth noting Clarke’s observation that reporting user-friendly information on the causes of a failed transaction necessitated embedding information about the meaning of preconditions into the interface, thus coupling the interface more tightly to the content of the model than might have been desirable.

6 Proof

A formal theory for reasoning about VDM models is provided in [BFL⁺94]. The axioms and inference rules of the theory refer to the VDM-SL dialect of VDM. We therefore begin, in Sections 6.1 and 6.2, by showing how we model the domain and operations of the object-oriented model in the VDM-SL dialect. The full VDM-SL model is given in Appendix E. Given VDM-SL models of the abstract and concrete worlds, and formal theory of VDM-SL, in Section 6.3 we present the retrieve function and describe the proof effort. In Section 6.4 we describe our results in the automation of the proof task and in Section 6.5 we record some observations on the modelling and proving exercise.

6.1 Domain Modelling

The object-oriented classes *AbWorld* and *AbPurse* are combined into a single VDM-SL module *abworld*. The *AbPurse* class becomes a record in *abworld* (Figure 14), with the private instance variables *balance* and *lost* as fields of the record.

```

publicAbPurse :: balance :  $\mathbb{N}$ 
                  lost :  $\mathbb{N}$ 

```

Figure 14: Definition of the AbPurse record.

The state of the *abworld* module is derived from the state of the *AbWorld* class presented in Section 3.2. Each instance variable in *AbWorld* (*authentic* and *abPurses*) has a corresponding state variable (*abauth* and *abpurses*). The type of *abauth* is the same as the type of *authentic*, and the type of *abpurses* is a map

from *PurseId* (which remains of type *token*) to records of type *AbPurse*. The state invariant is a logically equivalent translation from the class invariant, and the state initially contains no purses or purse identifiers.

```

state AbWorld of
  abauth: PurseId-set
  abpurses: PurseId  $\xrightarrow{m}$  AbPurse
inv mk_AbWorld(abauth, abpurses)  $\triangleq$  dom abpurses  $\subseteq$  abauth

```

Figure 15: The state of *AbWorld* in the VDM-SL model

The concrete model contains separate classes representing the world, purse and ether. As with in the abstract model, the state of the flattened *conworld* module is derived from the state of the *ConWorld* class. The instance variables of the *ConWorld* class becomes the state of the *conworld* module, and purses become records within that module. The concept of the ether is not translated into the VDM-SL model. The operations in the *conworld* module directly control the transfer of funds.

6.2 Operation Modelling

The *AbWorld* operations are translated into operations within the *abworld* module, and the operations in *ConWorld* are translated into the module *conworld*. Pre- and postconditions are given for all operations, as well as the operational definition.

In *conworld*, a successful run is modelled by the operation *ConTransferOk*. The other four operations represent failures of the protocol at different points, and are constructed as an initial sequence of the protocol operations (*OpReq*, *OpVal*, *OpAck*) followed by an *Abort* operation on each of the communicating purses. After this sequence, some resolution of the state of the purses is necessary to ensure that they are returned to a consistent state. This resolution represents the action of the bank. In most cases, the resolution required is to simply remove from the exception logs the transaction details that would be removed by the bank. In one case (*ConTransferLostValSucceed*) the presence of the transaction details in the purse exception log represents a genuine loss of money, which would be credited by the bank and must therefore be part of the resolution of the operation.

In the case of *ConTransferOK*, the three low-level protocol operations *OpReq*, *OpVal* and *OpAck* occur in sequence. Neither purse needs to abort, and no resolution is necessary. The full definition of the concrete operation *ConTransferOK* is given in Fig. 16. The precondition requires that the identifiers of the *from* and *to* purses be different, that they are both valid identifiers, and that the amount to be transferred is available in the *from* purse. We refer the reader to Appendix E for the definitions of the remaining operations.

ConTransferLostReq models the loss of the *req* message. The protocol therefore fails before completion of *OpReq*. The transaction details of the receiving purse are logged. The *from* purse begins in status $\langle \text{EPR} \rangle$, so the *LogIfNecessary* operation is not executed. This operation refines *AbIgnore*. Some resolution is necessary here because the receiving purse logs the transaction details, and for the constructed operation to model *AbIgnore*, these details must be removed from the transaction log of the receiving purse.

The case where the protocol fails because the *req* message is lost is non-deterministic, and is modelled by two operations at the concrete level. In each case *OpReq* completes, but *OpVal* does not. The non-determinism arises because we cannot tell whether the sum to be transferred has been placed in the receiving purse. The case where the sum has not yet been added to the receiving purse is modelled by *ConTransferLostValFail*, and the case where the sum has been added to the receiving purse is modelled by *ConTransferLostValSucceed*. In the first case, the value is removed from the sending purse, and the transaction details are logged. The receiving purse will log the transaction details, and so the resolution removes these, and *ConTransferLostValFail* refines *AbTransferLost*. Both purses log the transaction details in the operation *ConTransferLostValSucceed*, so in the resolution we must remove both of these. We also add the transferred sum to the receiving purse.

```

ConTransferOk: (PurseId, PurseId,  $\mathbb{N}$ )  $\rightarrow$  ()
ConTransferOk(fid, tid, val)  $\triangleq$ 
  let fseqno = conpurses(fid).seqno,
      tseqno = conpurses(tid).seqno,
      td = mk_TransDetails(fid, tid, fseqno, tseqno, val) in
  (
    StartFromOkay(fid, mk_CounterPartyDetails(tid, val, tseqno));
    StartToOkay(tid, mk_CounterPartyDetails(fid, val, fseqno));
    OpReq(fid, td);
    OpVal(tid, td);
    OpAck(fid, td)
  )
pre fid  $\neq$  tid  $\wedge$ 
  fid  $\in$  {dom conpurses}  $\wedge$ 
  tid  $\in$  {dom conpurses}  $\wedge$ 
  conpurses(fid).bal  $\geq$  val
post total(conpurses(fid)) + total(conpurses(tid)) =
  total( $\overline{\text{conpurses}}$ (fid)) + total( $\overline{\text{conpurses}}$ (tid))  $\wedge$ 
  conpurses(fid).bal + conpurses(tid).bal  $\leq$   $\overline{\text{conpurses}}$ (fid).bal +  $\overline{\text{conpurses}}$ (tid).bal  $\wedge$ 
  conpurses(fid).seqno  $\geq$   $\overline{\text{conpurses}}$ (fid).seqno  $\wedge$ 
  conpurses(tid).seqno  $\geq$   $\overline{\text{conpurses}}$ (tid).seqno  $\wedge$ 
  conpurses(fid).status = <IDLE>  $\wedge$ 
  conpurses(tid).status = <IDLE>  $\wedge$ 
  conpurses(fid).exlog =  $\overline{\text{conpurses}}$ (fid).exlog  $\wedge$ 
  conpurses(tid).exlog =  $\overline{\text{conpurses}}$ (tid).exlog  $\wedge$ 
  dom conpurses = dom  $\overline{\text{conpurses}}$   $\wedge$ 
   $\overline{\text{conauth}}$  = conauth  $\wedge$ 
   $\forall pid \in$  {dom conpurses  $\setminus$  {fid, tid}}  $\cdot$  conpurses(pid) =  $\overline{\text{conpurses}}$ (pid)

```

Figure 16: The *ConTransferOK* operation

The loss of the acknowledgement message is modelled by the top-level operation *ConTransferLostAck*. In this case, from the point of view of the receiving purse the protocol has terminated correctly, and so no resolution is necessary. The sending purse logs the transaction details when it fails to receive the acknowledgement message, and so these must be removed in the resolution.

6.3 The Retrieve Function and Formal Proof

To show that the concrete development in VDM-SL is a valid implementation of the abstract specification, we formally define the relationship between them. This relationship is given by the *retrieve* function (Figure 17). The retrieve function is a function from the concrete world to the abstract world which returns, for any concrete world, the abstract world of which it is an implementation.

$$\begin{aligned} &retr: ConWorld \rightarrow AbWorld \\ &retr(mk_ConWorld(auth, cp)) \triangleq \\ &mk_AbWorld(auth, pid \mapsto mk_AbPurse(cp(pid).bal, sumval(cp(pid).exlog)) \mid pid \in \{\mathbf{dom} \ cp\}) \end{aligned}$$

Figure 17: The retrieve function

This definition uses the function *sumval*, which maps a set of transaction details in the concrete implementation of a purse to the sum of their values in the abstract implementation. It is defined recursively as

$$\boxed{\text{sumval-def}} \frac{s: TD\text{-set}}{sumval(s) = \mathbf{if} \ s = \{\} \ \mathbf{then} \ 0 \ \mathbf{else} \ \mathbf{let} \ x \in s \ \mathbf{in} \ x.val + sumval(s \setminus \{x\})}$$

Figure 18: The *sumval* function

A retrieve function gives rise to a number of proof obligations, all of which need to be discharged if we are to assert that the concrete world is a valid implementation of the abstract one. The first proof obligation to discharge is that of *totality*, which states that every possible concrete world must be retrieved to an abstract one. In other words, all concrete worlds must have an abstract representation. The totality proof rule for the retrieve function above is also called the *formation* rule. It is given in Figure 19. The name of the rule is *retr-form*. The antecedent of the rule is given above the line, and the consequent below. The constructor *mk-ConWorld()* creates a new *ConWorld* using the set of authorised purse identities (*auth*), and a mapping from identities to members of *ConPurses* (*cp*).

$$\boxed{\text{retr-form}} \frac{mk_ConWorld(auth, cp): ConWorld}{retr(mk_ConWorld(auth, cp)): AbWorld}$$

Figure 19: The totality proof obligation

The proofs were carried out using the natural deduction style, and the full proof of the totality obligation is given in Figure 20. At each step, the deduction arrived at is recorded. The rule that was applied is recorded on the right hand side, and the parameters indicate the line or lines in the proof that justify applying this rule. The hypotheses of the rule are labelled implicitly, with *hx* standing for hypothesis *x*. The full set of rules that were used is recorded in Appendix F.

The second proof obligation to discharge is that of *adequacy*: the retrieve function must ensure that all abstract worlds have a concrete representation, or that for any abstract world *a*, there is a corresponding

```

from  $mk\text{-}ConWorld(auth, cp): ConWorld$ 
1    $inv\text{-}ConWorld(auth, cp)$   $inv\text{-}ConWorld\text{-}I(h1)$ 
2   dom  $cp \subseteq auth$   $unfolding(1)$ 
3    $mk\text{-}ConWorld(auth, cp).conpurses: PurseId \xrightarrow{m} ConPurse$   $conpurses\text{-}form(h1)$ 
4    $mk\text{-}ConWorld(auth, cp).conpurses = cp$   $conpurses\text{-}defn(h1)$ 
5    $cp: PurseId \xrightarrow{m} ConPurse$   $=\text{-type-inherit-right}(3,4)$ 
6   dom  $cp: PurseId\text{-set}$   $dom\text{-form}(5)$ 
7   from  $name: PurseId; name \in \mathbf{dom} cp$ 
7.1    $name \in \mathbf{dom} mk\text{-}ConWorld(auth, cp).conpurses$   $=\text{-subs-left}(b)(5,4,7.h2)$ 
7.2    $mk\text{-}AbPurse(mk\text{-}ConWorld(auth, cp).conpurses(name).bal,$ 
       $sumval(mk\text{-}ConWorld(auth, cp).conpurses(name).exlog): AbPurse$ 
       $ConWorld\text{-}AbWorld\text{-}form(7.h1,h1,7.2)$ 
infer  $mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog): AbPurse$   $=\text{-subs-right}(a)(3,4,7.2)$ 
8    $auth: PurseId\text{-set}$   $ConWorld\text{-}1\text{-form}(h1)$ 
9    $\{name \mapsto mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog))$ 
     $| name \in \mathbf{dom} cp\}: PurseId \xrightarrow{m} AbPurse$   $map\text{-}comp\text{-}form\text{-}left\text{-}set(6,7)$ 
10  dom  $\{name \mapsto mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog))$ 
     $| name \in \mathbf{dom} cp\} = \mathbf{dom} cp$   $dom\text{-}defn\text{-}map\text{-}comp\text{-}left\text{-}set(6,7)$ 
11  dom  $\{name \mapsto mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog))$ 
     $| name \in \mathbf{dom} cp\} \subseteq auth$   $=\text{-subs-left}(b)(6,10,2)$ 
12   $inv\text{-}AbWorld(auth,$ 
     $\{name \mapsto mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog))$ 
     $| name \in \mathbf{dom} cp\})$   $folding(11)$ 
13   $mk\text{-}AbWorld(auth,$ 
     $\{name \mapsto mk\text{-}AbPurse(cp(name).bal, sumval(cp(name).exlog))$ 
     $| name \in \mathbf{dom} cp\}): AbWorld$   $mk\text{-}AbWorld\text{-}form(8,9,12)$ 
infer  $retr(mk\text{-}ConWorld(auth, cp)): AbWorld$   $folding(13)$ 

```

Figure 20: The proof of the totality obligation

concrete world c such that $retr(c) = a$. The adequacy obligation is given in Figure 21 and is discharged in Appendix G.2.

$$\boxed{retr\text{-adequate}} \frac{a: AbWorld}{\exists c: ConWorld \cdot retr(c) = a}$$

Figure 21: The adequacy proof obligation

The adequacy and totality proof rules deal with the abstract and concrete states, and provide assurance that the retrieve function connects these states appropriately. The concrete world contains operations which correspond to the operations in the abstract world and the remaining obligations deal with linking the dynamic behaviour of the model in the abstract and the concrete states. They fall into two sets, and are known as *domain* obligations and *result* obligations, and together they show that a concrete operation is a valid implementation of an abstract operation.

The domain obligations show that the domains of aop and cop are properly linked by the retrieve function. This can be restated as follows: if a state in the concrete world is linked by the retrieve function to a state in the abstract world which is in the domain of aop (the precondition of aop holds), then the concrete state

must be in the domain of the corresponding concrete operation cop (the precondition of cop must hold in the concrete state). This is known as the *domain* obligation for operation aop . Formally, with $pre-op$ defined as the precondition of an operation op :

$$\forall c \cdot c \in ConWorld \cdot pre-aop(retr(c)) \Rightarrow pre-cop(c)$$

The *result* obligations concern the postconditions of operations. For any operation op and pair of states \overleftarrow{s} and s , let the postcondition of the operation be given by a function $post-op(\overleftarrow{s}, s)$, which evaluates to true when $op(\overleftarrow{s})$ results in the state s . The result proof obligation requires that if two concrete states c and \overleftarrow{c} are linked by a concrete operation cop , (i.e. $post-cop(\overleftarrow{c}, c)$ holds), and the precondition of the related abstract operation aop holds in the state $retr(\overleftarrow{c})$, then it must be the case that the postcondition of aop holds over the retrieved states $retr(\overleftarrow{c})$ and $retr(c)$. We write:

$$\forall \overleftarrow{c}, c \in ConWorld \cdot pre-aop(retr(\overleftarrow{c})) \wedge post-cop(\overleftarrow{c}, c) \Rightarrow post-aop(retr(\overleftarrow{c}), retr(c))$$

Due to limits of time, we considered only the concrete operation $ConTransferLostValFail$. This models one possible outcome of the abstract operation $AbTransferLost$ under the retrieve function. The domain obligation for these operations is given in Figure 22 and is discharged in Appendix H.1.1.

$$\boxed{\text{TransferLost-dom}} \frac{c: ConWorld; fid: PurseId; tid: PurseId; val: \mathbb{N};}{pre-AbTransferLost(retr(c), fid, tid, val) \Rightarrow pre-ConTransferLostValFail(c, fid, tid, val)}$$

Figure 22: The domain obligation for $TransferLost$

The result obligation for the $TransferLost$ operations is given in Figure 23. It is discharged in H.1.2.

$$\boxed{\text{TransferLost-res}} \frac{c: ConWorld; \overleftarrow{c}: ConWorld; fid: PurseId; tid: PurseId; val: \mathbb{N};}{pre-AbTransferLost(retr(c), fid, tid, val); post-ConTransferLostValFail(c, \overleftarrow{c}, fid, tid, val)} \\ post-AbTransferLost(retr(c), retr(\overleftarrow{c}), fid, tid, val)}$$

Figure 23: The result obligation for $TransferLost$

A number of useful results were formulated and proved as lemmas throughout the course of this work. These included lemmas about the retrieve function such as $dom-purses=$, which states that the purses used in a concrete world have the same identifiers when the corresponding abstract world was generated by the retrieve function. The rule and its proof are given in Appendix H.1.1.

Certain results about the subsidiary function $sumval$ (Figure 6.3) were also proved. These included well-formedness, and a proof that $sumval$ applied to a set containing details of exactly one transaction $\{a\}$ was the value $a.val$ of that transaction. The proofs of the well-formedness property can be found in Appendix G.2, and the proof the result of its application to a singleton set in Appendix G.3.

6.4 Automation

The work of Sander Vermolen [Ver07] gave us the option of using HOL4 [SN08] to automatically verify the consistency of the abstract VDM++ model, by showing that the model discharges all of its proof obligations. Vermolen developed a VDM++ to HOL a translation tool that translates functional VDM++ models into semantically equivalent HOL models, as well as translating the proof obligations generated from the VDM++ model. The proof obligations can then be automatically discharged using the HOL4 proof tool.

Using the translation tool, a functional version of the abstract VDM++ model was translated into a HOL model. The proof obligations for the model were generated by VDMTools and then translated to terms in the HOL logic. The model and the proof obligations were then passed to the HOL4 proof tool. To assist the proof process, we also used a set of custom tactics that were written specifically for models that had been translated from VDM++.

HOL was able to find proofs for all of the proof obligations of the abstract model without any problems. At the time the translator did not translate operations, and so only the abstract model was considered in this way.

6.5 Remarks

Some remarks about the team carrying out the work reported here are in order. The team comprised of six core members. One had significant previous experience in theorem proving. The other five had no or very little experience in theorem proving, although all were familiar with VDM specifications.

Each identified proof obligation was assigned to a member of the team. Completed proofs were checked by two other members of the team. We estimate that the work of developing and checking the proofs was about 3 person months in total.

Many proofs were straightforward, though often long and verbose, commonly due to typing judgements and equality rules. As a consequence, due to the fact line numbers are managed manually, considerable effort was taken up with such ‘clerical’ aspects. Whilst those proofs may be more easily discharged by automated proof tools and are often self-evident, we felt that it was important to be rigorous in our proofs. The individually numbered lines and arguments appealing to unique inference rules aids in the readability of natural deduction proofs. The checks only identified minor details such as missing justifications for the application of a proof rule. One of the most difficult proofs was the adequacy of the retrieve function. This was as much a matter of unwieldy definitions as inherent complexity. It was clear what the “target” was (a concrete world that retrieved to the hypothesised abstract one), but the proof itself required showing that several properties of the target concrete world held, and the linear format of the proof style meant that this rapidly became difficult to manage. In these cases the checking role became one of assisting with the proof.

The testing work required about 2 person-days in total, including time to achieve sufficient familiarity with the testing tool.

7 Related Work

In [FW08], Freitas and Woodcock explore the costs and benefits of using the Z/Eves theorem prover to mechanise the handwritten proofs found in the original Mondex monograph. The original Z specification of Mondex was changed as little as possible (and a prover that existed at the time was also used for the automation) to allow reflection on the effort that would have been required to include automated proof in the original project. However, some problems were discovered in the original specification, so changes were required to amend these. For example some finiteness assumptions were only informally made and a number of properties were omitted, thus allowing the possibility of inauthentic purses being created or manipulated. Freitas and Woodcock conclude with some insight into the cost of automating the proof: the whole project (covering approx. 90% of the proof effort) took eight man weeks to complete over six months; automated proof could have been included in the original research for just 10% extra effort.

In [BY08], Butler and Yadav present a development of the Mondex system in Event-B which uses the B4Free [Cle09] and Click’n’Prove [AC03] tools. Event-B promotes an incremental approach in which a series of intermediate models are developed and the refinement relations between these models are specified using gluing invariants. This incremental approach leads to a high degree of automatic proof. This is in contrast to our experience, where proofs were carried out by hand. The Event-B style of modelling is quite different from Z, and so Butler and Yadav treated [SCW00] as a requirements document, and (for example) redeveloped the invariants rather than use the ones given in [SCW00]. They capture the full behaviour of the protocol. They estimate that the development took two weeks of effort spread over several months. The bulk of this

work was in developing the Event-B models and constructing invariants between them, and over 97% of the proof obligations arising from the development were proved automatically. It is interesting to note that much of the manual proof required in [BY08] involved the *sum* function, that returns the total of a finite set of transactions. Much of our proof effort was also expended on proofs that involved our equivalent function, *sumval*.

In [HSGR08], Haneberg et al. describe a verification of the Mondex system using the KIV interactive theorem prover and ASMs (Abstract State Machines). The paper contains two main results. The first is a full verification of the original Mondex protocol (refinement between the abstract and concrete worlds). In the translation of the Z specification to KIV, the team discovered small errors and inconsistencies in the original development and proofs. They also formulated a single-step refinement between the abstract and concrete worlds (as opposed to the two-step refinement in [SCW00]). This is similar to our result. Estimated effort for the initial proofs of the protocol was one person month. In addition to verification of the original protocol, the paper describes a further refinement to a protocol that includes cryptography. As noted in Section 4, both the original verification and the other approaches presented here assume that messages are secure. In another paper, the group also describes an implementation of the Mondex system using JavaCard [GMB⁺06].

In [KG08], Kuhlman and Gogolla describe a model of the Mondex system in UML (Universal Modelling Language). They describe both an imperative and declarative approach. OCL (Object Constraint Language) was used to describe constraints on the abstract world in terms of invariants and pre- and postconditions. The USE (UML Specification Environment) tool was used to check a series of test cases (both positive and negative). Their development only considers the abstract world (partially because there is no notion of formal refinement in UML). Therefore, they do not achieve similar verification results as other papers described above. They claim however that their development could be understood by a prospective client without a formal background.

In [GH07], the authors develop a stepwise refinement of the Mondex specification using the RAISE formal specification language RSL. The RAISE language permits a wide family of specification styles. The approach taken was in keeping with the RAISE method of refinement between specification levels, and closely followed the Z approach. Three levels were described, with the purpose in each being to contribute to the definition of the protocol operations. In the first level a definition of correctness is given for each operation, in the second the behaviour is specified axiomatically, and in the final level the operations are explicitly defined. Each specification level was translated to both the PVS theorem prover and the SAL model checker. With PVS, the consistency conditions at each level and the relations between levels generated a number of proof obligations. In some cases tactics were developed to deal with simple recurring patterns, and others were individually proved by experts. Interestingly, one of the most difficult cases involved the logs of transactions, here the difficulty arose in the proof that the logs were finite. The RSL to SAL translator was experimental, and only the middle level was translated. In this way some simple liveness properties were established, although the Mondex specification was limited to a small number of purses and possible transactions to reduce state space.

In [Ram07], the author checks a specification of Mondex developed the Alloy method. The Alloy method includes a modelling language based on first-order logic and the Alloy Analyzer tool, based on model-finding through SAT solving. The specification is translated into an SAT formula so that discovering an instance of the formula corresponds to finding a counter-example of the theorem being checked. The Alloy language is significantly different from the Z of the original specification, and the analyzer itself is limited in certain ways, for example, in not having a full representation of integers. It was possible to compensate for this however, because not all of the properties of integers were needed each time they were used, and so integers were replaced in each case by representations that retained the properties necessary and which were encoded more efficiently in Alloy.

8 Conclusions

In this paper we presented a development of the Mondex electronic purse system using VDM. The task was undertaken by a small group with a range of previous experience with formal modelling and VDM in particular. We successfully captured the Mondex problem in VDM and were able to explore and verify the resulting specifications using automated testing, animation and formal proof. The formal proofs make use of a single-step refinement, as achieved by other contemporary groups.

The model-based approach of VDM, and in particular the object-orientation of VDM++, intuitively capture the notion of a system of cards, where each individual card is represented by a single object. We would also argue that, although some features such as long preconditions can look overwhelming, overall the specifications in VDM are more readable than, for example, the original specification in Z [SCW00].

One of the clear strengths of choosing VDM is the robust set of tools available, which supported the construction of our specifications and made automated testing and animation possible. The rapid improvement of the open-source Overture tool in recent years, including the introduction of new features such as the combinatorial testing capabilities, has only helped in this regard. Overture has become an excellent addition to the tools available to the VDM user, complementing the industrial-strength, commercial tool set VDMTools.

An obvious downside of the use of VDM++ is the necessity to convert the specification into VDM-SL in order to perform formal proofs. The lack of a proof theory for the newer dialects of VDM has been identified by members of the VDM and Overture community as a key issue for the future direction of VDM. At a workshop in 2010, the community undertook to investigate this issue in the coming years [PPe10].

In addition to the restriction to VDM-SL, there is also a lack of tool support for proof in VDM. Although the mural tool developed in the early 1990s was perhaps ahead of its time [JJLM91], it is now all but lost. While some automated proof was achieved in our development [Ver07], the vast majority of the proof work was done by hand. This was a laborious process, however it was not a futile one. It proved to be an excellent learning experience for those involved, though perhaps not one they will wish to repeat soon. Having discussing our feelings after the proof work was completed, we felt that tool support could really help, however we didn't necessarily want to lose the option of producing hand-crafted proofs, since the process offers great insight into the studied problem and proof work in general.

Much of the tedium of hand-producing these proofs in the natural deduction style lies in the numbering of lines and justifications, and the requirement to manually update these as proofs evolve. Thus even a simple tool which aided in typesetting natural deduction proofs, which offered features such as automated line-numbering, would be of great help. At the next level of complexity, a proof checker would greatly reduce the human effort in checking proofs. In our process, each proof was painstakingly checked by hand at least twice, thus three separate people spent significant time on each proof.

Going further, a suite offering support for semi-automated, user-guided proof would begin to match the level of tool support for various other formal approaches. This was the key offering of the mural tool [JJLM91] and thus a resurrection or re-imagining of the mural tactic language / engine would be a great addition to a suite of tools. Finally, the ability to discharge large numbers of proofs automatically would bring tool-supported proof in VDM up to the level offered by other contemporary approaches, such as Rodin [Rod08]. The necessity to show adequacy however—which involves finding a witness value for an existential qualification—means that some user input will likely always be required. There is however ongoing research that may be able to help in this area, such as the AI4FM project [GJ10]. A tool that offered all of these features would be an excellent counterpart and counterpoint to the open-source Overture initiative.

It should be noted however that issues with formal proof (the current restriction to VDM-SL and lack of tool support for proofs) has not stopped the use of VDM in industry, suggesting that the well-supported techniques of specification, automated testing and animation are more important in the industrial domain. However there is a clear opportunity to improve what has been a somewhat neglected strength of VDM, with the current tools understandably focussing on the needs of industry for simulation, automated testing and animation.

References

- [AC03] J-R Abrial and D Cansell. Click'n'prove: Interactive proofs in set theory. In *Theorem Proving in Higher Order Logics*, volume 2758 of *LNCS*, pages 1–24, 2003.
- [BFL⁺94] Juan Bicarregui, John S. Fitzgerald, Peter A. Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practioner's Guide*. Springer-Verlag, 1994.
- [BY08] Michael Butler and Divakar Yadav. An incremental development of the Mondex system in Event-B. *Formal Asp. Comput.*, 20(1):61–77, 2008.
- [Cla09] Sarah Clarke. A Tool for Validating a Formal Model of an Electronic Purse. Master's thesis, School of Computing Science, Newcastle University, August 2009.
- [Cle09] Clearsy. B4free tool homepage. www.b4free.com, 2009.
- [FBG⁺08] J.S. Fitzgerald, J.W. Bryans, D. Greathead, C.B. Jones, and R. Payne. Animation-based Validation of a Formal Model of Dynamic Virtual Organisations. In P. Boca, J.P. Bowen, and P.G. Larsen, editors, *Proc. BCS-FACS Workshop on Formal Methods in Industry*, Electronic Workshops in Computing. British Computer Society, 2008.
- [FL06] John S. Fitzgerald and Peter Gorm Larsen. Triumphs and challenges for the industrial application of model-oriented formal methods. In *2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, 2006.
- [FLM⁺05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs For Object-Oriented Systems*. Springer-Verlag, 2005.
- [FLS08] John S. Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *SIGPLAN Notices*, 43(2):3–11, 2008.
- [FW08] Leo Freitas and Jim Woodcock. Mechanising Mondex with Z/Eves. *Formal Asp. Comput.*, 20(1):117–139, 2008.
- [GH07] Chris George and Anne E. Haxthausen. Specification, proof, and model checking of the Mondex electronic purse using RAISE. *Formal. Asp. Comput.*, 20:101–116, December 2007.
- [GJ10] Gudmund Grov and Cliff B. Jones. Ai4fm: A new project seeking challenges! In Rajeev Joshi, Tiziana Margaria, Peter Mueller, David Naumann, and Hongseok Yang, editors, *VSTTE 2010*, August 2010.
- [GMB⁺06] H Grandy, N. Moebius, M. Bischof, D. Haneberg, G. Schellhorn, K. Stenzel, and W. Reif. The Mondex Case Study: From Specifications to Code. Technical Report TR 2006-31, University of Augsburg, 2006.
- [HSGR08] Dominik Haneberg, Gerhard Schellhorn, Holger Grandy, and Wolfgang Reif. Verification of Mondex electronic purses with KIV: from transactions to a security protocol. *Formal Asp. Comput.*, 20(1):41–59, 2008.
- [JJLM91] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.
- [Jon89] Cliff B. Jones. Data reification. In John McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.
- [Jon90] Cliff B. Jones. *Systematic Software Development using VDM (2nd edition)*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990.

- [KG08] Mirco Kuhlmann and Martin Gogolla. Modeling and validating Mondex scenarios described in UML and OCL with USE. *Form. Asp. Comput.*, 20(1):79–100, 2008.
- [LFW⁺10] Peter Gorm Larsen, John Fitzgerald, Sune Wolff, Nick Battle, Kenneth Lausdahl, Augusto Ribeiro, and Kenneth Pierce. Tutorial for Overture/VDM++. Technical Report TR-004, The Overture Initiative, May 2010.
- [LL09] Peter Gorm Larsen and Kenneth Lausdahl. User Manual for the Overture Combinatorial Testing Plug-in. Technical Report TR-2009-01, The Overture Initiative, March 2009.
- [PPe10] Ken Pierce, Nico Plat, and Sune Wolff (eds.). Proceedings of the 8th Overture Workshop. Technical Report CS-TR 1224, Newcastle University, November 2010.
- [Ram07] Tahina Ramananandro. Mondex, an electronic purse: specification and refinement checks with the Alloy model-finding method. *Formal. Asp. Comput.*, 20:21–39, December 2007.
- [Rod08] Rodin, 2008. <http://sourceforge.net/projects/rodin-b-sharp/>.
- [SCW00] Susan Stepney, David Cooper, and Jim Woodcock. An Electronic Purse: Specification, Refinement and Proof. Technical Report PRG-126, Oxford University Computing Laboratory, July 2000.
- [SN08] Konrad Slind and Michael Norrish. A Brief Overview of HOL4. In Otmane Mohamed, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer Berlin / Heidelberg, 2008.
- [Ver07] S. D. Vermolen. Automatically Discharging VDM Proof Obligations using HOL. Master’s thesis, Radboud University, Nijmegen, 2007. Draft.
- [WB07] Jim Woodcock and Richard Banach. The Verification Grand Challenge. *Journal of Universal Computer Science*, 13(5):661–668, 2007.

A Abstract Model

A.1 AbPurse

class *AbPurse*

instance variables

private *balance*: \mathbb{N} ;

private *lost*: \mathbb{N} ;

operations

public *GetBalance*: $() \rightarrow \mathbb{N}$

GetBalance() \triangleq

(

*return**balance*

);

public *IncreaseBalance*: $\mathbb{N} \rightarrow ()$

IncreaseBalance(*val*) \triangleq

(

balance := *balance* + *val*

);

public *ReduceBalance*: $\mathbb{N} \rightarrow ()$

ReduceBalance(*val*) \triangleq

(

balance := *balance* - *val*

)

pre *balance* \geq *val*;

public *GetLost*: $() \rightarrow \mathbb{N}$

GetLost() \triangleq

(

*return**lost*

);

public *IncreaseLost*: $\mathbb{N} \rightarrow ()$

IncreaseLost(*val*) \triangleq

(

lost := *lost* + *val*

);

```

public GetTotal: () → ℕ
GetTotal()  $\triangleq$ 
(
  return balance + lost
);

public AbPurse: ℕ × ℕ → AbPurse
AbPurse(bal, lst)  $\triangleq$ 
(
  balance := bal;
  lost := lst
)

end AbPurse

```

A.2 AbWorld

```

class AbWorld

types

public PurseId = token;

instance variables

private abauth: PurseId-set;
private abpurses: PurseId  $\xrightarrow{m}$  AbPurse := { };
inv  $\forall name \in \mathbf{dom} \text{ } abpurses \cdot name \in abauth$ ;

operations

public AbIgnore: () → ()
AbIgnore()  $\triangleq$ 
  skip
post  $abauth = \overline{abauth} \wedge$ 
        $abpurses = \overline{abpurses}$ ;

```

```

public AbTransferOk: PurseId × PurseId × ℕ → ()
AbTransferOk(fid, tid, val)  $\triangleq$ 
(
  abpurses(fid).ReduceBalance(val);
  abpurses(tid).IncreaseBalance(val)
)
pre fid ≠ tid ∧
  fid ∈ dom abpurses ∧
  tid ∈ dom abpurses ∧
  abpurses(fid).GetBalance() ≥ val
post  $\overline{abpurses}(fid).GetTotal() + \overline{abpurses}(tid).GetTotal() =$ 
 $\overline{abpurses}(fid).GetTotal() + \overline{abpurses}(tid).GetTotal() \wedge$ 
 $\overline{abpurses}(fid).GetBalance() + \overline{abpurses}(tid).GetBalance() \geq$ 
 $\overline{abpurses}(fid).GetBalance() + \overline{abpurses}(tid).GetBalance() \wedge$ 
 $\forall name \in \mathbf{dom} \ abpurses \setminus fid, tid \cdot$ 
 $\overline{abpurses}(name).GetBalance() = \overline{abpurses}(name).GetBalance() \wedge$ 
 $\overline{abpurses}(name).GetLost() = \overline{abpurses}(name).GetLost() \wedge$ 
 $\overline{abpurses}(fid).GetLost() = \overline{abpurses}(fid).GetLost() \wedge$ 
 $\overline{abpurses}(tid).GetLost() = \overline{abpurses}(tid).GetLost() \wedge$ 
 $\mathbf{dom} \ \overline{abpurses} = \mathbf{dom} \ abpurses \wedge$ 
 $\overline{abauth} = abauth;$ 

```

```

public AbTransferLost: PurseId × PurseId × ℕ → ()
AbTransferLost(fid, tid, val)  $\triangleq$ 
(
  abpurses(fid).ReduceBalance(val);
  abpurses(fid).IncreaseLost(val)
)
pre fid ≠ tid ∧
  fid ∈ dom abpurses ∧
  tid ∈ dom abpurses ∧
  abpurses(fid).GetBalance() ≥ val
post  $\overline{abpurses}(fid).GetTotal() = \overline{abpurses}(fid).GetTotal() \wedge$ 
 $\overline{abpurses}(fid).GetBalance() \geq \overline{abpurses}(fid).GetBalance() \wedge$ 
 $\forall name \in (\mathbf{dom} \ abpurses) \setminus fid \cdot$ 
 $\overline{abpurses}(name).GetBalance() = \overline{abpurses}(name).GetBalance() \wedge$ 
 $\overline{abpurses}(name).GetLost() = \overline{abpurses}(name).GetLost() \wedge$ 
 $\mathbf{dom} \ \overline{abpurses} = \mathbf{dom} \ abpurses \wedge$ 
 $\overline{abauth} = abauth;$ 

```

```

public AbWorld: PurseId  $\xrightarrow{m}$  AbPurse  $\times$  PurseId-set  $\rightarrow$  AbWorld
AbWorld(purses, auth)  $\triangleq$ 
(
  abpurses := purses;
  abauth := auth
)
pre dom purses  $\subset$  auth

end AbWorld

```

B Concrete Model

B.1 ConPurse

```
class ConPurse

types

private Status = <IDLE> | <EPR> | <EPV> | <EPA>;

public TransDetails :: fromPurse : ConWorld'PurseId
                      toPurse : ConWorld'PurseId
                      fromSeqNo : ℕ
                      toSeqNo : ℕ
                      val : ℕ

public CounterPartyDetails :: name : ConWorld'PurseId
                              val : ℕ
                              seqNo : ℕ

public Message = StartFrom | StartTo | <READEXCEPTIONLOG> | Req | Val | Ack
                | ExceptionLogResult | ExceptionLogClear | <UNPROTECTED>;

public StartFrom = CounterPartyDetails;
public StartTo = CounterPartyDetails;

public Req = TransDetails;
public Val = TransDetails;
public Ack = TransDetails;

public ExceptionLogResult :: name : ConWorld'PurseId
                             td : TransDetails

public ExceptionLogClear :: name : ConWorld'PurseId
                            clear : token

instance variables

private name: ConWorld'PurseId;
private balance: ℕ;
private nextSeqNo: ℕ;
private status: Status;
private currTrans: [TransDetails];
private exLog: TransDetails-set;
private ether: [Ether];

operations

private Increase: () → ()
Increase()  $\triangleq$ 
  let delta  $\in$  {1, ..., 10} in
    nextSeqNo := nextSeqNo + delta;
```

```

private Abort: () → ()
Abort()  $\triangleq$ 
(
  self.LogIfNecessary();
  status := <IDLE>;
  self.Increase()
);

private LogIfNecessary: () → ()
LogIfNecessary()  $\triangleq$ 
  if status = <EPV>  $\vee$  status = <EPA>
  then exLog := exLog  $\cup$  {currTrans}
pre currTrans  $\neq$  nil;

private Image(tds: TransDetails-set)c: token
pre true
post true;

public RecMsg: Message → ()
RecMsg(m)  $\triangleq$ 
  cases m of
    mk-StartFrom(cpd) → StartFromOkay(cpd)
    mk-StartTo(cpd) → StartToOkay(cpd)
    <READEXCEPTIONLOG> → (Abort(); ReadExceptionLog())
    mk-Req(td) → OpReq(td)
    mk-Val(td) → OpVal(td)
    mk-Ack(td) → OpAck(td)
    mk-ExceptionLogClear(nm, clear) → (Abort(); ClearExceptionLog(nm, clear))
  others skip
end

private StartFromOkay: CounterPartyDetails → ()
StartFromOkay(cpd)  $\triangleq$ 
(
  currTrans := mk-TransDetails(name, cpd.name, nextSeqNo, cpd.seqNo, cpd.val);
  status := <EPR>;
  self.Increase();
  ether.SendMsg(<UNPROTECTED>)
)
pre cpd.name  $\neq$  name  $\wedge$  cpd.val  $\leq$  balance  $\wedge$  status = <IDLE>  $\wedge$  ether  $\neq$  nil;

private StartToOkay: CounterPartyDetails → ()
StartToOkay(cpd)  $\triangleq$ 
(
  currTrans := mk-TransDetails(cpd.name, name, cpd.seqNo, nextSeqNo, cpd.val);
  status := <EPV>;
  self.Increase();
  ether.SendMsg(mk-Req(currTrans))
)
pre cpd.name  $\neq$  name  $\wedge$  status = <IDLE>  $\wedge$  ether  $\neq$  nil;

```

```

private OpReq: TransDetails → ()
OpReq(td)  $\triangleq$ 
(
  balance := balance - td.val;
  status := <EPA>;
  ether.SendMsg(mk-Val(td))
)
pre status = <EPR>  $\wedge$  currTrans = td  $\wedge$  ether  $\neq$  nil;

private OpVal: TransDetails → ()
OpVal(td)  $\triangleq$ 
(
  balance := balance + td.val;
  status := <IDLE>;
  ether.SendMsg(mk-Ack(td))
)
pre status = <EPV>  $\wedge$  currTrans = td  $\wedge$  ether  $\neq$  nil;

private OpAck: TransDetails → ()
OpAck(td)  $\triangleq$ 
(
  status := <IDLE>;
  ether.SendMsg(<UNPROTECTED>)
)
pre status = <EPA>  $\wedge$  currTrans = td  $\wedge$  ether  $\neq$  nil;

private ReadExceptionLog: () → ()
ReadExceptionLog()  $\triangleq$ 
(
  if exLog =
  then ether.SendMsg(<UNPROTECTED>)
  else let logtd  $\in$  exLog in
    ether.SendMsg(mk-ExceptionLogResult(name, logtd))
)
pre status = <IDLE>  $\wedge$  ether  $\neq$  nil;

private ClearExceptionLog: ConWorld' PurseId  $\times$  token → ()
ClearExceptionLog(nm, c)  $\triangleq$ 
(
  exLog := {};
  ether.SendMsg(<UNPROTECTED>)
)
pre exLog  $\neq$  {}  $\wedge$  status = <IDLE>  $\wedge$  nm = name  $\wedge$  c = Image(exLog)  $\wedge$  ether  $\neq$  nil;

public GetBalance: () →  $\mathbb{N}$ 
GetBalance()  $\triangleq$ 
  return balance;

```

```

public GetLost: () → ℕ
GetLost()  $\triangleq$ 
(
  dcl lost: ℕ := 0;
  for all
    td ∈ exLog do lost := lost + td.val;
  return lost
);

public GetTotal: () → ℕ
GetTotal()  $\triangleq$ 
  return self.GetBalance() + self.GetLost();

public GetSeqNo: () → ℕ
GetSeqNo()  $\triangleq$ 
  return nextSeqNo;

public SetEther: Ether → ()
SetEther(ethr)  $\triangleq$ 
  ether := ethr;

public ConPurse: ConWorld' PurseId × ℕ → ConPurse
ConPurse(nm, bal)  $\triangleq$ 
(
  name := nm;
  balance := bal;
  status := <IDLE>;
  nextSeqNo := 0;
  currTrans := nil;
  exLog := { };
  ether := nil;
);

end ConPurse

```

B.2 ConWorld

```

class ConWorld

types

public PurseId = token

instance variables

private conauth: PurseId-set;
private conpurses: PurseId  $\xrightarrow{m}$  ConPurse := { };
private ether: Ether;
private previousmessages: ConPurse' TransDetails-set;
inv  $\forall$  name ∈ dom conpurses · name ∈ conauth;

```

operations

```
public ConTransfer: PurseId × PurseId ×  $\mathbb{N}$  → ()  
ConTransfer(fid, tid, val)  $\triangleq$   
(  
  let p1seqNo = conpurses(fid).GetSeqNo(), p2seqNo = conpurses(tid).GetSeqNo() in  
  (  
    ether.StartFrom(fid, mk-ConPurse‘CounterPartyDetails(tid, val, p2seqNo));  
    ether.StartTo(tid, mk-ConPurse‘CounterPartyDetails(fid, val, p1seqNo));  
  )  
)  
pre fid ∈ dom conpurses ∧ tid ∈ dom conpurses;  
  
public GetPurse: PurseId → ConPurse  
GetPurse(name)  $\triangleq$   
  return conpurses(name)  
pre name ∈ dom conpurses;  
  
public GetConPurses: () → PurseId  $\xrightarrow{m}$  ConPurse  
GetConPurses()  $\triangleq$   
  return conpurses;  
  
public ConWorld: PurseId-set → ConWorld  
ConWorld(names)  $\triangleq$   
(  
  ether := new PerfectEther(self);  
  conauth := names;  
  conpurses := name ↦ new ConPurse(name, 100) | name ∈ names;  
  for all purse ∈ rng conpurses do  
    purse.SetEther(ether);  
  previousmessages := { };  
)  
  
end ConWorld
```

B.3 Ether

```
class Ether
```

```
instance variables
```

```
protected world: ConWorld;
```

```
operations
```

```
public StartFrom: ConWorld‘PurseId × ConPurse‘CounterPartyDetails → ()  
StartFrom(pid, cpd)  $\triangleq$   
  is subclass responsibility;
```

```
public StartTo: ConWorld‘PurseId × ConPurse‘CounterPartyDetails → ()  
StartTo(pid, cpd)  $\triangleq$   
  is subclass responsibility;
```

```

public ExceptionLogClear: ConWorld' PurseId × token → ()
ExceptionLogClear(pid, clear)  $\triangleq$ 
  is subclass responsibility;

public ReadExceptionLog: ConWorld' PurseId → ()
ReadExceptionLog(pid)  $\triangleq$ 
  is subclass responsibility;

public SendMsg: ConPurse' Message → ()
SendMsg(m)  $\triangleq$ 
  is subclass responsibility;

public Ether: ConWorld → Ether
Ether(w)  $\triangleq$ 
  world := w;

end Ether

```

B.4 Perfect Ether

```

class PerfectEther is subclass of Ether

operations

public StartFrom: ConWorld' PurseId × ConPurse' CounterPartyDetails → ()
StartFrom(name, cpd)  $\triangleq$ 
  world.GetPurse(name).RecMsg(mk-ConPurse' StartFrom(cpd));

public StartTo: ConWorld' PurseId × ConPurse' CounterPartyDetails → ()
StartTo(name, cpd)  $\triangleq$ 
  world.GetPurse(name).RecMsg(mk-ConPurse' StartTo(cpd));

public ExceptionLogClear: ConWorld' PurseId × token → ()
ExceptionLogClear(name, clear)  $\triangleq$ 
  world.GetPurse(name).RecMsg(mk-ConPurse' ExceptionLogClear(name, clear));

public ReadExceptionLog: ConWorld' PurseId → ()
ReadExceptionLog(name)  $\triangleq$ 
  world.GetPurse(name).RecMsg(<READEXCEPTIONLOG>);

public SendMsg: ConPurse' Message → ()
SendMsg(m)  $\triangleq$ 
  cases m of
    mk-ConPurse' Req(td) → world.GetPurse(td.fromPurse).RecMsg(m)
    mk-ConPurse' Val(td) → world.GetPurse(td.toPurse).RecMsg(m)
    mk-ConPurse' Ack(td) → world.GetPurse(td.fromPurse).RecMsg(m)
    mk-ConPurse' ExceptionLogResult(-, -) → skip
    <UNPROTECTED> → skip

  others skip
end

```

```
public PerfectEther: ConWorld → PerfectEther  
PerfectEther(w)  $\triangleq$   
  world := w;  
end PerfectEther
```

C Abstract World Test Class

```
class AbTest
```

```
instance variables
```

```
public abworldSP1: AbWorld :=  
  new AbWorld({fid, tid}, {fid ↦ abpurseSP1f, tid ↦ abpurseSP1t});  
public abworldSP2: AbWorld :=  
  new AbWorld({fid, tid}, {fid ↦ abpurseSP2f, tid ↦ abpurseSP2t});  
public abworldSP3: AbWorld :=  
  new AbWorld({fid, tid, undef1, undef2}, {fid ↦ abpurseSP3f, tid ↦ abpurseSP3t});  
public abworldSP4: AbWorld :=  
  new AbWorld({fid, tid}, {fid ↦ abpurseSP4f, tid ↦ abpurseSP4t})
```

```
values
```

```
fid = mk-token(P1);  
tid = mk-token(P2);  
undef1 = mk-token(undef1);  
undef2 = mk-token(undef2);  
undef3 = mk-token(undef3);  
undef4 = mk-token(undef4);
```

```
abpurseSP1f: AbPurse = new AbPurse(50, 0);  
abpurseSP1t: AbPurse = new AbPurse(75, 0);
```

```
abpurseSP2f: AbPurse = new AbPurse(50, 0);  
abpurseSP2t: AbPurse = new AbPurse(75, 0);
```

```
abpurseSP3f: AbPurse = new AbPurse(50, 0);  
abpurseSP3t: AbPurse = new AbPurse(75, 0);
```

```
abpurseSP4f: AbPurse = new AbPurse(5, 0);  
abpurseSP4t: AbPurse = new AbPurse(10, 0)
```

```
traces
```

```
AbTestSP1:
```

```
let x = abworldSP1 in  
  let n ∈ {0, ..., 5} in  
  (  
    x.TotalWorldBalance();  
    (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());  
    (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());  
    (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());  
    x.TotalWorldBalance()  
  );
```

AbTestSP2:

```
let x = abworldSP2 in
  let n ∈ {0, ..., 5} in
    (
      x.TotalWorldValue();
      (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
      (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
      (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n) | x.AbIgnore());
      x.TotalWorldValue()
    );
```

AbTestSP3a:

```
let x = abworldSP3 in
  let n = 5 in
    let f = fid in
      let t ∈ {tid, undef1, undef2, undef3, undef4} in
        (
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n))
        );
```

AbTestSP3b:

```
let x = abworldSP3 in
  let n = 5 in
    let t = tid in
      let f ∈ {fid, undef1, undef2, undef3, undef4} in
        (
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n));
          (x.AbTransferOk(f, t, n) | x.AbTransferLost(f, t, n))
        );
```

AbTestSP4:

```
let x = abworldSP4 in
  let n ∈ {0, ..., 10} in
    (x.AbTransferOk(fid, tid, n) | x.AbTransferLost(fid, tid, n))
```

end *AbTest*

D Concrete World Test Class

```
class ConTest
```

```
instance variables
```

```
public conworldSP1: ConWorld :=  
  new ConWorld({fid, tid}, {fid ↦ conpurseSP1f, tid ↦ conpurseSP1t}, 'p');  
public conworldSP2ack: ConWorld :=  
  new ConWorld({fid, tid}, {fid ↦ conpurseSP2af, tid ↦ conpurseSP2at}, 'a');  
public conworldSP2req: ConWorld :=  
  new ConWorld({fid, tid}, {fid ↦ conpurseSP2bf, tid ↦ conpurseSP2bt}, 'r');  
public conworldSP2val: ConWorld :=  
  new ConWorld({fid, tid}, {fid ↦ conpurseSP2cf, tid ↦ conpurseSP2ct}, 'v');  
public conworldsSP2: ConWorld-set :=  
  {conworldSP2ack, conworldSP2req, conworldSP2val};  
public conworldSP3: ConWorld :=  
  new ConWorld({fid, tid, undef1, undef2}, {fid ↦ conpurseSP3f, tid ↦ conpurseSP3t}, 'p');  
public conworldSP4: ConWorld :=  
  new ConWorld({fid, tid}, {fid ↦ conpurseSP4f, tid ↦ conpurseSP4t}, 'p');
```

```
values
```

```
fid = mk_token(CP1);  
tid = mk_token(CP2);  
undef1 = mk_token(undef1);  
undef2 = mk_token(undef2);  
undef3 = mk_token(undef3);  
undef4 = mk_token(undef4);
```

```
conpurseSP1f: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP1t: ConPurse = new ConPurse(mk_token(CP2), 100);
```

```
conpurseSP2af: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP2at: ConPurse = new ConPurse(mk_token(CP2), 100);  
conpurseSP2bf: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP2bt: ConPurse = new ConPurse(mk_token(CP2), 100);  
conpurseSP2cf: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP2ct: ConPurse = new ConPurse(mk_token(CP2), 100);
```

```
conpurseSP3f: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP3t: ConPurse = new ConPurse(mk_token(CP2), 100);
```

```
conpurseSP4f: ConPurse = new ConPurse(mk_token(CP1), 50);  
conpurseSP4t: ConPurse = new ConPurse(mk_token(CP2), 100);
```

traces

ConTestSP1:

```
let x = conworldSP1 in
  let n ∈ {0, ..., 5} in
    (
      x.TotalWorldBalance();
      x.ConTransfer(tid, fid, n);
      x.TotalWorldBalance();
      x.GetPurseBalLost(fid);
      x.GetPurseBalLost(tid)
    )
```

ConTestSP2:

```
let x ∈ conworldsSP2 in
  let n ∈ {0, ..., 5} in
    (
      x.GetPurseBalLost(fid);
      x.GetPurseBalLost(tid);
      x.ConTransfer(fid, tid, n);
      x.GetPurseBalLost(fid);
      x.GetPurseBalLost(tid)
    )
```

ConTestSP3a:

```
let x = conworldSP3 in
  let n = 5 in
    let f = fid in
      let t ∈ {tid, undef1, undef2, undef3, undef4} in
        x.ConTransfer(t, f, n)
```

ConTestSP3b:

```
let x = conworldSP3 in
  let n = 5 in
    let t = tid in
      let f ∈ {fid, undef1, undef2, undef3, undef4} in
        x.ConTransfer(t, f, n)
```

ConTestSP4:

```
let x = conworldSP4 in
  let n ∈ {45, ..., 55} in
    x.ConTransfer(fid, tid, n)
```

end ConTest

E VDM-SL Model

E.1 abworld

module *abworld*

definitions

types

PurseId = *token*;

AbPurse :: *bal* : \mathbb{N}
lost : \mathbb{N} ;

state *AbWorld* **of**

abauth: *PurseId*-**set**

abpurses: *PurseId* \xrightarrow{m} *AbPurse*

inv *mk-AbWorld*(*abauth*, *abpurses*) \triangleq **dom** *abpurses* \subset *abauth*

init *a* \triangleq *a* = *mk-AbWorld*($\{\}$, $\{\mapsto\}$)

end

operations

AbIgnore: $() \rightarrow ()$

AbIgnore() \triangleq

skip

post *abauth* = $\overleftarrow{abauth} \wedge$
abpurses = $\overleftarrow{abpurses}$;

AbTransferOk: *PurseId* \times *PurseId* \times $\mathbb{N} \rightarrow ()$

AbTransferOk(*fid*, *tid*, *val*) \triangleq

(
abpurses(*fid*).*bal* := *abpurses*(*fid*).*bal* - *val*;
abpurses(*tid*).*bal* := *abpurses*(*tid*).*bal* + *val*
)

pre *fid* \neq *tid* \wedge

fid \in **dom** *abpurses* \wedge

tid \in **dom** *abpurses* \wedge

abpurses(*fid*).*bal* \geq *val*

post *total*(*abpurses*(*fid*)) + *total*(*abpurses*(*tid*)) = $\overleftarrow{total}(\overleftarrow{abpurses}(\overleftarrow{fid})) + \overleftarrow{total}(\overleftarrow{abpurses}(\overleftarrow{tid})) \wedge$

abpurses(*fid*).*bal* + *abpurses*(*tid*).*bal* \leq $\overleftarrow{abpurses}(\overleftarrow{fid}).\overleftarrow{bal} + \overleftarrow{abpurses}(\overleftarrow{tid}).\overleftarrow{bal} \wedge$

abpurses(*fid*).*lost* = $\overleftarrow{abpurses}(\overleftarrow{fid}).\overleftarrow{lost} \wedge$

abpurses(*tid*).*lost* = $\overleftarrow{abpurses}(\overleftarrow{tid}).\overleftarrow{lost} \wedge$

dom *abpurses* = **dom** $\overleftarrow{abpurses} \wedge$

abauth = $\overleftarrow{abauth} \wedge$

$\forall pid \in$ **dom** *abpurses* $\setminus \{fid, tid\} \cdot$ *abpurses*(*pid*) = $\overleftarrow{abpurses}(\overleftarrow{pid})$;

```

AbTransferLost: PurseId × PurseId × ℕ → ()
AbTransferLost(fid, tid, val) ≜
(
  abpurses(fid).bal := abpurses(fid).bal - val;
  abpurses(fid).lost := abpurses(fid).lost + val;
)
pre fid ≠ tid ∧
  fid ∈ dom abpurses ∧
  tid ∈ dom abpurses ∧
  abpurses(fid).bal ≥ val
post total(abpurses(fid)) = total(←abpurses(fid)) ∧
  abpurses(fid).bal ≤ ←abpurses(fid).bal ∧
  dom ←abpurses = dom abpurses ∧
  abauth = ←abauth ∧
  ∀pid ∈ dom abpurses \ {fid} · abpurses(pid) = ←abpurses(pid)

```

functions

```
total: AbPurse → ℕtotal(mk-AbPurse(bal, lost)) ≜ bal + lost;
```

end abworld

E.2 conworld

module conworld

definitions

types

```
PurseId = token;
```

```
ConPurse ::   bal : ℕ
              seqno : ℕ
              status : Status
              ctrans : [TransDetails]
              exlog : TransDetails-set;
```

```
Status = <IDLE> | <EPR> | <EPV> | <EPA>;
```

```
TransDetails ::   fid : PurseId
                  tid : PurseId
                  fseqno : ℕ
                  tseqno : ℕ
                  val : ℕ;
```

```
CounterPartyDetails ::   pid : PurseId
                          val : ℕ
                          seqno : ℕ;
```

```

state ConWorld of
  conauth: PurseId-set
  conpurses: PurseId  $\xrightarrow{m}$  ConPurse
inv mk-AbWorld(conauth, conpurses)  $\triangleq$  dom conpurses  $\subset$  conauth
init c  $\triangleq$  c = mk-ConWorld( $\{\}$ ,  $\{\mapsto\}$ )
end

```

operations

```

ConTransferOk: PurseId  $\times$  PurseId  $\times$   $\mathbb{N}$   $\rightarrow$  ()
ConTransferOk(fid, tid, val)  $\triangleq$ 
let fseqno = conpurses(fid).seqno,
     tseqno = conpurses(tid).seqno,
     td = mk-TransDetails(fid, tid, fseqno, tseqno, val) in
  (
    StartFromOkay(fid, mk-CounterPartyDetails(tid, val, tseqno));
    StartToOkay(tid, mk-CounterPartyDetails(fid, val, fseqno));
    OpReq(fid, td);
    OpVal(tid, td);
    OpAck(fid, td)
  )
pre fid  $\neq$  tid  $\wedge$ 
     fid  $\in$  dom conpurses  $\wedge$ 
     tid  $\in$  dom conpurses  $\wedge$ 
     conpurses(fid).bal  $\geq$  val
post total(conpurses(fid)) + total(conpurses(tid)) = total( $\overleftarrow{\text{conpurses}}$ (fid)) + total( $\overleftarrow{\text{conpurses}}$ (tid))  $\wedge$ 
     conpurses(fid).bal + conpurses(tid).bal  $\leq$   $\overleftarrow{\text{conpurses}}$ (fid).bal +  $\overleftarrow{\text{conpurses}}$ (tid).bal  $\wedge$ 
     conpurses(fid).seqno  $\geq$   $\overleftarrow{\text{conpurses}}$ (fid).seqno  $\wedge$ 
     conpurses(tid).seqno  $\geq$   $\overleftarrow{\text{conpurses}}$ (tid).seqno  $\wedge$ 
     conpurses(fid).status =  $\langle \text{IDLE} \rangle$   $\wedge$ 
     conpurses(tid).status =  $\langle \text{IDLE} \rangle$   $\wedge$ 
     conpurses(fid).exlog =  $\overleftarrow{\text{conpurses}}$ (fid).exlog  $\wedge$ 
     conpurses(tid).exlog =  $\overleftarrow{\text{conpurses}}$ (tid).exlog  $\wedge$ 
     dom  $\overleftarrow{\text{conpurses}}$  = dom  $\overleftarrow{\text{conpurses}}$   $\wedge$ 
      $\overleftarrow{\text{conauth}}$  = conauth  $\wedge$ 
      $\forall pid \in$  dom conpurses  $\setminus \{fid, tid\}$  . conpurses(pid) =  $\overleftarrow{\text{conpurses}}$ (pid);

```

$ConTransferLostReq: PurseId \times PurseId \times \mathbb{N} \rightarrow ()$
 $ConTransferLostReq(fid, tid, val) \triangleq$
let $fseqno = \text{compurses}(fid).seqno,$
 $tseqno = \text{compurses}(tid).seqno,$
 $td = mk\text{-}TransDetails(fid, tid, fseqno, tseqno, val)$ **in**
(

 $StartFromOkay(fid, mk\text{-}CounterPartyDetails(tid, val, tseqno));$
 $StartToOkay(tid, mk\text{-}CounterPartyDetails(fid, val, fseqno));$
 $Abort(fid);$
 $Abort(tid);$
 $\text{compurses}(td.tid).exlog := \text{compurses}(td.tid).exlog \setminus \{td\}$
)

pre $fid \neq tid \wedge$
 $fid \in \mathbf{dom} \text{compurses} \wedge$
 $tid \in \mathbf{dom} \text{compurses} \wedge$
 $\text{compurses}(fid).bal \geq val$
post $\text{compurses}(fid).bal = \overline{\text{compurses}(fid)}.bal \wedge$
 $\text{compurses}(tid).bal = \overline{\text{compurses}(tid)}.bal \wedge$
 $\text{compurses}(fid).seqno \geq \overline{\text{compurses}(fid)}.seqno \wedge$
 $\text{compurses}(tid).seqno \geq \overline{\text{compurses}(tid)}.seqno \wedge$
 $\text{compurses}(fid).status = \langle \text{IDLE} \rangle \wedge$
 $\text{compurses}(tid).status = \langle \text{IDLE} \rangle \wedge$
 $\text{compurses}(fid).exlog = \overline{\text{compurses}(fid)}.exlog \wedge$
 $\text{compurses}(tid).exlog = \overline{\text{compurses}(tid)}.exlog \wedge$
 $\mathbf{dom} \text{compurses} = \mathbf{dom} \overline{\text{compurses}} \wedge$
 $conauth = \overline{conauth} \wedge$
 $\forall pid \in \mathbf{dom} \text{compurses} \setminus \{fid, tid\} \cdot \text{compurses}(pid) = \overline{\text{compurses}(pid)};$

$ConTransferLostValFail: PurseId \times PurseId \times \mathbb{N} \rightarrow ()$
 $ConTransferLostValFail(fid, tid, val) \triangleq$
let $fseqno = conpurses(fid).seqno,$
 $tseqno = conpurses(tid).seqno,$
 $td = mk-TransDetails(fid, tid, fseqno, tseqno, val)$ **in**
(

 $StartFromOkay(fid, mk-CounterPartyDetails(tid, val, tseqno));$
 $StartToOkay(tid, mk-CounterPartyDetails(fid, val, fseqno));$
 $OpReq(fid, td);$
 $Abort(fid);$
 $Abort(tid);$
 $conpurses(td.tid).exlog := conpurses(td.tid).exlog \setminus td$
)

pre $fid \neq tid \wedge$
 $fid \in \mathbf{dom} conpurses \wedge$
 $tid \in \mathbf{dom} conpurses \wedge$
 $conpurses(fid).bal \geq val$
post $total(conpurses(fid)) = total(\overleftarrow{conpurses}(fid)) \wedge conpurses(fid).bal \leq \overleftarrow{conpurses}(fid).bal \wedge$
 $conpurses(tid).bal = \overleftarrow{conpurses}(tid).bal \wedge$
 $conpurses(fid).seqno \geq \overleftarrow{conpurses}(fid).seqno \wedge$
 $conpurses(tid).seqno \geq \overleftarrow{conpurses}(tid).seqno \wedge$
 $conpurses(fid).status = \langle IDLE \rangle \wedge$
 $conpurses(tid).status = \langle IDLE \rangle \wedge$
 $conpurses(fid).exlog \cap conpurses(fid).ctrans = \overleftarrow{conpurses}(fid).exlog \wedge$
 $conpurses(tid).exlog = \overleftarrow{conpurses}(tid).exlog \wedge$
 $\mathbf{dom} conpurses = \mathbf{dom} \overleftarrow{conpurses} \wedge$
 $conauth = \overleftarrow{conauth} \wedge$
 $\forall pid \in \mathbf{dom} conpurses \setminus \{fid, tid\} \cdot conpurses(pid) = \overleftarrow{conpurses}(pid);$

$ConTransferLostValSucceed: PurseId \times PurseId \times \mathbb{N} \rightarrow ()$
 $ConTransferLostValSucceed(fid, tid, val) \triangleq$
let $fseqno = conpurses(fid).seqno,$
 $tseqno = conpurses(tid).seqno,$
 $td = mk-TransDetails(fid, tid, fseqno, tseqno, val)$ **in**
(

 $StartFromOkay(fid, mk-CounterPartyDetails(tid, val, tseqno));$
 $StartToOkay(tid, mk-CounterPartyDetails(fid, val, fseqno));$
 $OpReq(fid, td);$
 $Abort(fid);$
 $Abort(tid);$
 $conpurses(td.fid).exlog := conpurses(td.fid).exlog \setminus \{td\};$
 $conpurses(td.tid).exlog := conpurses(td.tid).exlog \setminus \{td\};$
 $conpurses(td.tid).bal := conpurses(td.tid).bal + td.val$
)

pre $fid \neq tid \wedge$
 $fid \in \mathbf{dom} conpurses \wedge$
 $tid \in \mathbf{dom} conpurses \wedge$
 $conpurses(fid).bal \geq val$
post $total(conpurses(fid)) + total(conpurses(tid)) = total(\overline{conpurses}(fid)) + total(\overline{conpurses}(tid)) \wedge$
 $conpurses(fid).bal + conpurses(tid).bal \leq \overline{conpurses}(fid).bal + \overline{conpurses}(tid).bal \wedge$
 $conpurses(fid).seqno \geq \overline{conpurses}(fid).seqno \wedge$
 $conpurses(tid).seqno \geq \overline{conpurses}(tid).seqno \wedge$
 $conpurses(fid).status = \langle \text{IDLE} \rangle \wedge$
 $conpurses(tid).status = \langle \text{IDLE} \rangle \wedge$
 $conpurses(fid).exlog = \overline{conpurses}(fid).exlog \wedge$
 $conpurses(tid).exlog = \overline{conpurses}(tid).exlog \wedge$
 $\mathbf{dom} conpurses = \mathbf{dom} \overline{conpurses} \wedge$
 $\overline{conauth} = conauth \wedge$
 $\forall pid \in \mathbf{dom} conpurses \setminus \{fid, tid\} \cdot conpurses(pid) = \overline{conpurses}(pid);$

$ConTransferLostAck: PurseId \times PurseId \times \mathbb{N} \rightarrow ()$
 $ConTransferLostAck(fid, tid, val) \triangleq$
let $fseqno = \text{compurses}(fid).seqno,$
 $tseqno = \text{compurses}(tid).seqno,$
 $td = mk\text{-}TransDetails(fid, tid, fseqno, tseqno, val)$ **in**
(

 $StartFromOkay(fid, mk\text{-}CounterPartyDetails(tid, val, tseqno));$
 $StartToOkay(tid, mk\text{-}CounterPartyDetails(fid, val, fseqno));$
 $OpReq(fid, td);$
 $OpVal(tid, td);$
 $Abort(fid);$
 $Abort(tid);$
 $\text{compurses}(td.fid).exlog := \text{compurses}(td.fid).exlog \setminus \{td\}$
)

pre $fid \neq tid \wedge$
 $fid \in \mathbf{dom} \text{compurses} \wedge$
 $tid \in \mathbf{dom} \text{compurses} \wedge$
 $\text{compurses}(fid).bal \geq val$
post $total(\text{compurses}(fid)) + total(\text{compurses}(tid)) = total(\overline{\text{compurses}}(fid)) + total(\overline{\text{compurses}}(tid)) \wedge$
 $\text{compurses}(fid).bal + \text{compurses}(tid).bal \leq \overline{\text{compurses}}(fid).bal + \overline{\text{compurses}}(tid).bal \wedge$
 $\text{compurses}(fid).seqno \geq \overline{\text{compurses}}(fid).seqno \wedge$
 $\text{compurses}(tid).seqno \geq \overline{\text{compurses}}(tid).seqno \wedge$
 $\text{compurses}(fid).status = \langle \text{IDLE} \rangle \wedge$
 $\text{compurses}(tid).status = \langle \text{IDLE} \rangle \wedge$
 $\text{compurses}(fid).exlog = \overline{\text{compurses}}(fid).exlog \wedge$
 $\text{compurses}(tid).exlog = \overline{\text{compurses}}(tid).exlog \wedge$
 $\mathbf{dom} \text{compurses} = \mathbf{dom} \overline{\text{compurses}} \wedge$
 $\text{conauth} = \overline{\text{conauth}} \wedge$
 $\forall pid \in \mathbf{dom} \text{compurses} \setminus \{fid, tid\} \cdot \text{compurses}(pid) = \overline{\text{compurses}}(pid);$

$Increase: PurseId \rightarrow ()$
 $Increase(pid) \triangleq$
let $\delta \in \{1, \dots, 10\}$ **in**
 $\text{compurses}(pid).seqno := \text{compurses}(pid).seqno + \delta$
post $\text{compurses}(pid).bal = \overline{\text{compurses}}(pid).bal \wedge$
 $\text{compurses}(pid).seqno \geq \overline{\text{compurses}}(pid).seqno \wedge$
 $\text{compurses}(pid).status = \overline{\text{compurses}}(pid).status \wedge$
 $\text{compurses}(pid).ctrans = \overline{\text{compurses}}(pid).ctrans \wedge$
 $\text{compurses}(pid).exlog = \overline{\text{compurses}}(pid).exlog \wedge$
 $\mathbf{dom} \text{compurses} = \mathbf{dom} \overline{\text{compurses}} \wedge$
 $\text{conauth} = \overline{\text{conauth}} \wedge$
 $\forall pid \in \mathbf{dom} \text{compurses} \setminus \{pid\} \cdot \text{compurses}(pid) = \overline{\text{compurses}}(pid);$

LogIfNecessary: $PurseId \rightarrow ()$
LogIfNecessary(pid) \triangleq
if $compurses(pid).status = \langle EPV \rangle \vee compurses(pid).status = \langle EPA \rangle$
then $compurses(pid).exlog := compurses(pid).exlog \cup \{compurses(pid).ctrans\}$
pre $compurses(pid).ctrans \neq nil$
post $compurses(pid).bal = \overline{compurses}(pid).bal \wedge$
 $compurses(pid).seqno \geq \overline{compurses}(pid).seqno \wedge$
 $compurses(pid).status = \overline{compurses}(pid).status \wedge$
 $compurses(pid).ctrans = \overline{compurses}(pid).ctrans \wedge$
 $compurses(pid).exlog \setminus \{compurses(pid).ctrans\} = \overline{compurses}(pid).exlog \wedge$
dom $compurses = \mathbf{dom} \overline{compurses} \wedge$
 $conauth = \overline{conauth} \wedge$
 $\forall pid \in \mathbf{dom} compurses \setminus \{pid\} \cdot compurses(pid) = \overline{compurses}(pid);$

Abort: $PurseId \rightarrow ()$
Abort(pid) \triangleq
(

 LogIfNecessary(pid);
 $compurses(pid).status := \langle IDLE \rangle;$
 Increase(pid)
)

post $compurses(pid).bal = \overline{compurses}(pid).bal \wedge$
 $compurses(pid).seqno \geq \overline{compurses}(pid).seqno \wedge$
 $compurses(pid).status = \langle IDLE \rangle \wedge$
 $compurses(pid).exlog \setminus compurses(pid).ctrans = \overline{compurses}(pid).exlog \wedge$
dom $compurses = \mathbf{dom} \overline{compurses} \wedge$
 $conauth = \overline{conauth} \wedge$
 $\forall pid \in \mathbf{dom} compurses \setminus \{pid\} \cdot compurses(pid) = \overline{compurses}(pid);$

StartFromOkay: $PurseId \times CounterPartyDetails \rightarrow ()$
StartFromOkay(pid, cpd) \triangleq
(

 $compurses(pid).ctrans := mk-TransDetails(pid, cpd.pid, compurses(pid).seqno, cpd.seqno, cpd.val);$
 $compurses(pid).status := \langle EPR \rangle;$
 Increase(pid)
)

pre $cpd.pid \neq pid \wedge$
 $cpd.val \leq compurses(pid).bal \wedge$
 $compurses(pid).status = \langle IDLE \rangle$
post $compurses(pid).bal = \overline{compurses}(pid).bal \wedge$
 $compurses(pid).seqno \geq \overline{compurses}(pid).seqno \wedge$
 $compurses(pid).status = \langle EPR \rangle \wedge$
 $compurses(pid).ctrans = mk-TransDetails(pid, cpd.pid, compurses(pid).seqno, cpd.seqno, cpd.val) \wedge$
 $compurses(pid).exlog = \overline{compurses}(pid).exlog \wedge$
dom $compurses = \mathbf{dom} \overline{compurses} \wedge$
 $conauth = \overline{conauth} \wedge$
 $\forall pid \in \mathbf{dom} compurses \setminus \{pid\} \cdot compurses(pid) = \overline{compurses}(pid);$

StartToOkay: $PurseId \times CounterPartyDetails \rightarrow ()$
StartToOkay(*pid*, *cpd*) \triangleq
(

 compurses(*pid*).*ctrans* := *mk-TransDetails*(*cpd.pid*, *pid*, *cpd.seqno*, *compurses*(*pid*).*seqno*, *cpd.val*);
 compurses(*pid*).*status* := <EPV>;
 Increase(*pid*)
)

pre *cpd.pid* \neq *pid* \wedge
 compurses(*pid*).*status* = <IDLE>
post *compurses*(*pid*).*bal* = $\overline{\text{compurses}}(\text{pid}).\text{bal} \wedge$
 compurses(*pid*).*seqno* \geq $\overline{\text{compurses}}(\text{pid}).\text{seqno} \wedge$
 compurses(*pid*).*status* = <EPV> \wedge
 compurses(*pid*).*ctrans* = *mk-TransDetails*(*pid*, *cpd.pid*, *compurses*(*pid*).*seqno*, *cpd.seqno*, *cpd.val*) \wedge
 compurses(*pid*).*exlog* = $\overline{\text{compurses}}(\text{pid}).\text{exlog} \wedge$
 dom *compurses* = **dom** $\overline{\text{compurses}} \wedge$
 conauth = $\overline{\text{conauth}} \wedge$
 $\forall \text{pid} \in \text{dom } \text{compurses} \setminus \{\text{pid}\} \cdot \text{compurses}(\text{pid}) = \overline{\text{compurses}}(\text{pid});$

OpReq: $PurseId \times TransDetails \rightarrow ()$
OpReq(*pid*, *td*) \triangleq
(

 compurses(*pid*).*bal* := *compurses*(*pid*).*bal* - *td.val*;
 compurses(*pid*).*status* := <EPA>
)

pre *compurses*(*pid*).*status* = <EPR> \wedge
 compurses(*pid*).*ctrans* = *td*
post *compurses*(*pid*).*bal* = $\overline{\text{compurses}}(\text{pid}).\text{bal} - \text{td.val} \wedge$
 compurses(*pid*).*seqno* \geq $\overline{\text{compurses}}(\text{pid}).\text{seqno} \wedge$
 compurses(*pid*).*status* = <EPA> \wedge
 compurses(*pid*).*ctrans* = *td* \wedge
 compurses(*pid*).*exlog* = $\overline{\text{compurses}}(\text{pid}).\text{exlog} \wedge$
 dom *compurses* = **dom** $\overline{\text{compurses}} \wedge$
 conauth = $\overline{\text{conauth}} \wedge$
 $\forall \text{pid} \in \text{dom } \text{compurses} \setminus \{\text{pid}\} \cdot \text{compurses}(\text{pid}) = \overline{\text{compurses}}(\text{pid});$

OpVal: $PurseId \times TransDetails \rightarrow ()$
OpVal(*pid*, *td*) \triangleq
(

 compurses(*pid*).*bal* := *compurses*(*pid*).*bal* + *td.val*;
 compurses(*pid*).*status* := <IDLE>
)

pre *compurses*(*pid*).*status* = <EPV> \wedge *compurses*(*pid*).*ctrans* = *td*
post *compurses*(*pid*).*bal* = $\overline{\text{compurses}}(\text{pid}).\text{bal} + \text{td.val} \wedge$
 compurses(*pid*).*seqno* \geq $\overline{\text{compurses}}(\text{pid}).\text{seqno} \wedge$
 compurses(*pid*).*status* = <IDLE> \wedge
 compurses(*pid*).*ctrans* = *td* \wedge
 compurses(*pid*).*exlog* = $\overline{\text{compurses}}(\text{pid}).\text{exlog} \wedge$
 dom *compurses* = **dom** $\overline{\text{compurses}} \wedge$
 conauth = $\overline{\text{conauth}} \wedge$
 $\forall \text{pid} \in \text{dom } \text{compurses} \setminus \{\text{pid}\} \cdot \text{compurses}(\text{pid}) = \overline{\text{compurses}}(\text{pid});$

$OpAck: PurseId \times TransDetails \rightarrow ()$
 $OpAck(pid, td) \triangleq$
 $($
 $\quad conpurses(pid).status := \langle IDLE \rangle;$
 $)$
pre $conpurses(pid).status = \langle EPA \rangle \wedge$
 $\quad conpurses(pid).ctrans = td$
post $conpurses(pid).bal = \overline{conpurses}(pid).bal \wedge$
 $\quad conpurses(pid).seqno \geq \overline{conpurses}(pid).seqno \wedge$
 $\quad conpurses(pid).status = \langle IDLE \rangle \wedge$
 $\quad conpurses(pid).exlog = \overline{conpurses}(pid).exlog \wedge$
dom $conpurses = \mathbf{dom} \overline{conpurses} \wedge$
 $\quad conauth = \overline{conauth} \wedge$
 $\quad \forall pid \in \mathbf{dom} conpurses \setminus \{pid\} \cdot conpurses(pid) = \overline{conpurses}(pid);$

functions

$total: ConPurse \rightarrow \mathbb{N}$
 $total(mk-ConPurse(bal, -, -, -, exlog)) \triangleq$
 $\quad bal + sumval(exlog);$

$sumval: TransDetails\text{-set} \rightarrow \mathbb{N}$
 $sumval(s) \triangleq$
 $\quad \mathbf{if} \ s = \{\} \ \mathbf{then} \ 0 \ \mathbf{else} \ \mathbf{let} \ x \in s \in x.val + sumval(s \setminus \{x\});$

end *conworld*

F Rules Derived from Models

AbWorld rules

$$\begin{array}{c}
 \boxed{\text{mk-AbWorld-form}} \frac{a: \text{PurseId-set} \\ p: \text{PurseId} \xrightarrow{m} \text{AbPurse} \\ \text{inv-AbWorld}(a, p)}{mk\text{-AbWorld}(a, p): \text{AbWorld}} \\
 \\
 \boxed{\text{inv-AbWorld-form}} \frac{mk\text{-AbWorld}(a, p): \text{AbWorld}}{\text{inv-AbWorld}(a, p)} \\
 \\
 \boxed{\text{mk-AbWorld-defn}} \frac{a: \text{AbWorld}}{mk\text{-AbWorld}(a.abauth, a.abpurses) = a} \\
 \\
 \boxed{\text{mk-AbPurse-form}} \frac{b: \mathbb{N}; l: \mathbb{N}}{mk\text{-AbPurse}(b, l): \text{AbPurse}} \\
 \\
 \boxed{\text{mk-AbPurse-defn}} \frac{a: \text{AbPurse}}{mk\text{-AbPurse}(a.bal, a.lost) = a} \\
 \\
 \boxed{\text{AbPurse-bal-form}} \frac{a: \text{AbPurse}}{a.bal: \mathbb{N}} \\
 \\
 \boxed{\text{AbPurse-lost-form}} \frac{a: \text{AbPurse}}{a.lost: \mathbb{N}} \\
 \\
 \boxed{\text{AbPurse-bal-defn}} \frac{mk\text{-AbPurse}(b, l): \text{AbPurse}}{mk\text{-AbPurse}(b, l).bal = b} \\
 \\
 \boxed{\text{AbPurse-lost-defn}} \frac{mk\text{-AbPurse}(b, l): \text{AbPurse}}{mk\text{-AbPurse}(b, l).lost = l} \\
 \\
 \boxed{\text{abpurses-form}} \frac{a: \text{AbWorld}}{a.abpurses: \text{PurseId} \xrightarrow{m} \text{AbPurse}} \\
 \\
 \boxed{\text{abpurses-defn}} \frac{mk\text{-AbWorld}(a, p): \text{AbWorld}}{mk\text{-AbWorld}(a, p).abpurses = p} \\
 \\
 \boxed{\text{TD-form}} \frac{n1: \text{PurseId}; n2: \text{PurseId} \\ fsq: \mathbb{N}; tsq: \mathbb{N}; val: \mathbb{N}}{mk\text{-TD}(n1, n2, fsq, tsq, val): \text{TD}} \\
 \\
 \boxed{\text{abauth-form}} \frac{a: \text{AbWorld}}{a.abauth: \text{PurseId-set}} \\
 \\
 \boxed{\text{abauth-defn}} \frac{mk\text{-AbWorld}(a, p): \text{AbWorld}}{mk\text{-AbWorld}(a, p).abauth = a} \\
 \\
 \boxed{=mk\text{-AbPurse}} \frac{x: \mathbb{N}; y: \mathbb{N}; x = a; y = b}{mk\text{-AbPurse}(x, y) = mk\text{-AbPurse}(a, b)}
 \end{array}$$

$$\begin{array}{l}
a: AbWorld \\
b: AbWorld \\
a.abauth = b.abauth \\
a.abpurses = b.abpurses \\
\boxed{=AbWorld} \frac{}{a = b}
\end{array}$$

ConWorld rules

$$\boxed{mk-ConWorld-form} \frac{
\begin{array}{l}
a: abworld' PurseId\text{-set} \\
p: abworld' PurseId \xrightarrow{m} ConPurse \\
inv-ConWorld(a, p)
\end{array}
}{mk-ConPurse(a, p): ConWorld}$$

$$\boxed{inv-ConWorld-I} \frac{mk-ConWorld(a, p): ConWorld}{inv-ConWorld(a, p)}$$

$$\boxed{conauth-defn} \frac{mk-ConWorld(a, p): ConWorld}{mk-ConWorld(a, p).conauth = a}$$

$$\boxed{conauth-form} \frac{mk-ConWorld(a, p): ConWorld}{mk-ConWorld(a, p).conauth: abworld' PurseId\text{-set}}$$

$$\boxed{compurses-defn} \frac{mk-ConWorld(a, p): ConWorld}{mk-ConWorld(a, p).compurses = p}$$

$$\boxed{compurses-form} \frac{mk-ConWorld(a, p): ConWorld}{mk-ConWorld(a, p).compurses: abworld' PurseId \xrightarrow{m} ConPurse}$$

$$\boxed{mk-ConPurse-form} \frac{
\begin{array}{l}
x: PurseId \\
n1: \mathbb{N} \\
n2: \mathbb{N} \\
s: Status \\
ex: TD\text{-set}
\end{array}
}{mk-ConPurse(x, n1, n2, s, nil, ex, nil): ConPurse}$$

$$\boxed{ConPurse-exlog-form} \frac{cp: ConPurse}{cp.exlog: TD\text{-set}}$$

$$\boxed{ConPurse-exlog-defn} \frac{mk-ConPurse(-, -, -, -, -, ex, -): ConPurse}{mk-ConPurse(-, -, -, -, -, ex, -).exlog = ex}$$

$$\boxed{ConPurse-bal-form} \frac{cp: ConPurse}{cp.bal: \mathbb{N}}$$

$$\boxed{ConPurse-bal-defn} \frac{mk-ConPurse(-, v, -, -, -, -): ConPurse}{mk-ConPurse(-, v, -, -, -, -).bal = v}$$

$$\boxed{\langle Idle \rangle\text{-form}} \frac{}{\langle IDLE \rangle: \langle IDLE \rangle}$$

$$\boxed{Status-form} \frac{}{Status = \langle IDLE \rangle \mid \langle EPR \rangle \mid \langle EPV \rangle \mid \langle EPA \rangle}$$

$$\boxed{bal-form} \frac{mk-ConPurse(b, -, -, -, -)}{b: \mathbb{N}}$$

Other rules used

$$\boxed{\text{dom-form}} \frac{m: A \xrightarrow{m} B}{\mathbf{dom} \ m: A\text{-set}}$$

$$\boxed{\text{Quote-type-extend}} \frac{T = \langle A \rangle \mid B; e: \langle A \rangle}{e: T}$$

$$\boxed{\text{retr-defn}} \frac{c: \text{ConWorld}}{\text{retr}(c) = \text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid \text{name} \in \mathbf{dom} \ c.\text{conpurses} \})}$$

$$\boxed{\text{sumval-defn}} \frac{s: TD\text{-set}}{\text{sumval}(s) = \mathbf{if} \ s = \{ \} \ \mathbf{then} \ 0 \ \mathbf{else} \ \mathbf{let} \ x \in s \ \mathbf{in} \ x.\text{val} + \text{sumval}(s \setminus \{x\})}$$

$$\boxed{\text{total-C-form}} \frac{p: \text{ConPurse}}{\text{totalC}(p): \mathbb{N}}$$

$$\boxed{\text{totalC-defn}} \frac{\text{purse}: \text{ConPurse}}{\text{totalC}(\text{purse}) = \text{purse}.\text{bal} + \text{sumval}(\text{purse}.\text{exlog})}$$

$$\boxed{\text{totalA-defn}} \frac{\text{purse}: \text{AbPurse}}{\text{totalA}(\text{purse}) = \text{purse}.\text{bal} + \text{purse}.\text{lost}}$$

$$\boxed{\{a,b\}\text{-form}} \frac{a: A; b: A}{\{a, b\}: A\text{-set}}$$

$$\boxed{\{a,b\}\text{-defn}} \frac{a: A; b: A}{\{a, b\} = \text{add}(a, \text{add}(b, \{ \}))}$$

$$\boxed{\forall\text{-elem-I}} \frac{a: A; P(a); s: A\text{-Set}; \forall b \in s \setminus \{a\} \cdot P(b)}{\forall b \in s \cdot P(b)}$$

$$\boxed{\text{TD-val-form}} \frac{t: TD}{t.\text{val}: \mathbb{N}}$$

$$\boxed{\text{TD-val-defn}} \frac{\text{mk-TD}(_, _, _, v): TD}{\text{mk-TD}(_, _, _, v).\text{val} = v}$$

$$\boxed{\text{let-defn1}} \frac{a: A}{\mathbf{let} \ x \in \{a\} \ \mathbf{in} \ P(x) = P(a)}$$

$$\boxed{\text{let-}\in\text{-form}} \frac{a: A; P(a): B}{(\mathbf{let} \ x \in \{a\} \ \mathbf{in} \ P(x)): B}$$

G Data Reification Proofs

G.1 Formation of the retrieve function

Conjecture

$$\boxed{\text{retr-form}} \frac{mk\text{-}ConWorld(auth, cp): ConWorld}{retr(mk\text{-}ConWorld(auth, cp)): AbWorld}$$

Proof

```

from mk-ConWorld(auth, cp): ConWorld
1   inv-ConWorld(auth, cp)                                inv-ConWorld-I(h1)
2   dom cp ⊆ auth                                        unfolding(1)
3   mk-ConWorld(auth, cp).conpurses: PurseId  $\xrightarrow{m}$  ConPurse    conpurses-form(h1)
4   mk-ConWorld(auth, cp).conpurses = cp                conpurses-defn(h1)
5   cp: PurseId  $\xrightarrow{m}$  ConPurse                                =-type-inherit-right(3,4)
6   dom cp: PurseId-set                                  dom-form(5)
7   from name: PurseId; name ∈ dom cp
7.1   name ∈ dom mk-ConWorld(auth, cp).conpurses        =-subs-left(b)(5,4,7.h2)
7.2   mk-AbPurse(mk-ConWorld(auth, cp).conpurses(name).bal,
      sumval(mk-ConWorld(auth, cp).conpurses(name).exlog): AbPurse
      ConWorld-AbWorld-form(7.h1,h1,7.2)
infer mk-AbPurse(cp(name).bal, sumval(cp(name).exlog): AbPurse =-subs-right(a)(3,4,7.2)
8   auth: PurseId-set                                    ConWorld-1-form(h1)
9   {name ↦ mk-AbPurse(cp(name).bal, sumval(cp(name).exlog))
    | name ∈ dom cp}: PurseId  $\xrightarrow{m}$  AbPurse                map-comp-form-left-set(6,7)
10  dom {name ↦ mk-AbPurse(cp(name).bal, sumval(cp(name).exlog))
      | name ∈ dom cp} = dom cp                            dom-defn-map-comp-left-set(6,7)
11  dom {name ↦ mk-AbPurse(cp(name).bal, sumval(cp(name).exlog))
      | name ∈ dom cp} ⊆ auth                                =-subs-left(b)(6,10,2)
12  inv-AbWorld(auth,
    {name ↦ mk-AbPurse(cp(name).bal, sumval(cp(name).exlog))
    | name ∈ dom cp})
    folding(11)
13  mk-AbWorld(auth,
    {name ↦ mk-AbPurse(cp(name).bal, sumval(cp(name).exlog))
    | name ∈ dom cp}): AbWorld                            mk-AbWorld-form(8,9,12)
infer retr(mk-ConWorld(auth, cp)): AbWorld                folding(13)

```

G.2 Adequacy of the retrieve function

Conjecture

$$\boxed{\text{retr-adequate}} \frac{a: AbWorld}{\exists c: ConWorld \cdot retr(c) = a}$$

Proof

$x = mk\text{-}ConWorld(a.abauth, \{name \mapsto mk\text{-}ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$

$\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil \mid name \in \mathbf{dom} a.abpurses\}$

```

from  $a: AbWorld$ 
1    $x: ConWorld$  con-assign(h1)
2    $x.conauth: PurseId\text{-set}$  conauth-form(1)
3    $x.conpurses: PurseId \xrightarrow{m} ConPurse$  conpurses-form(1)
4   dom  $x.conpurses: PurseId\text{-set}$  dom-form(3)
5   from  $name: PurseId; name \in \mathbf{dom} x.conpurses$ 
5.1    $x.conpurses(name): ConPurse$  at-form(5.h1, 3, 5.h2)
5.2    $x.conpurses(name).bal: \mathbb{N}$  ConPurse-bal-form(5.1)
5.3    $x.conpurses(name).exlog: TD\text{-set}$  ConPurse-exlog-form(5.1)
5.4    $sumval(x.conpurses(name).exlog): \mathbb{N}$  sumval-form(5.3)
infer  $mk\text{-}AbPurse(x.conpurses(name).bal,$ 
       $sumval(x.conpurses(name).exlog)): AbPurse$  mk-AbPurse-form(5.2, 5.4)
6    $\{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}: PurseId \xrightarrow{m} AbPurse$  map-comp-form-left-set(4, 5)
7    $inv\text{-}ConWorld(x.conauth, x.conpurses)$  inv-ConWorld-I(1)
8   dom  $x.conpurses \subseteq x.conauth$  unfolding(7)
9   dom  $\{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\} = \mathbf{dom} x.conpurses$  dom-defn-map-comp-left-set(4, 5)
10  dom  $\{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}: PurseId$  dom-form(6)
11  dom  $\{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\} \subseteq x.conauth$  =sub-left(a)(10, 9, 8)
12   $inv\text{-}AbWorld(x.conauth, \{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\})$  folding(11)
13   $mk\text{-}AbWorld(x.conauth, \{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}): AbWorld$  mk-AbWorld-form(2, 6, 12)
14   $x.conauth = a.abauth$  conauth-defn(1)
15   $a.abauth: PurseId\text{-set}$  abauth-form(h1)
16   $mk\text{-}AbWorld(x.conauth, \{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}).abauth = x.conauth$  abauth-defn(13)
17   $mk\text{-}AbWorld(x.conauth, \{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}).abauth = a.abauth$  =-trans(b)(2, 16, 14)
18   $a.abauth = mk\text{-}AbWorld(x.conauth, \{name \mapsto$ 
       $mk\text{-}AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$ 
       $name \in \mathbf{dom} x.conpurses\}).abauth$  =-symm(b)(15, 17)
19   $a.abpurses: PurseId \xrightarrow{m} AbPurse$  abpurses-form(h1)
20  dom  $a.abpurses: PurseId\text{-set}$  dom-form(19)

```

from $a: AbWorld$
...
21 **from** $name: PurseId; name \in \mathbf{dom} a.abpurses$
21.1 $a.abpurses(name): AbPurse$ at-form(21.h1, 19, 21.h2)
21.2 $a.abpurses(name).bal: \mathbb{N}$ AbPurse-bal-form(21.1)
21.3 $a.abpurses(name).lost: \mathbb{N}$ AbPurse-lost-form(21.1)
21.4 $0: \mathbb{N}$ 0-form
21.5 $\langle IDLE \rangle: \langle IDLE \rangle$ $\langle IDLE \rangle$ -form
21.6 $Status = \langle IDLE \rangle \mid \langle EPR \rangle \mid \langle EPV \rangle \mid \langle EPA \rangle$ Status-form
21.7 $\langle IDLE \rangle: Status$ Quote-type-extend(21.6, 21.5)
21.8 $mk-TD(name, name, 0, 0,$
 $a.abpurses(name).lost): TD$ TD-form(21.h1, 21.h1, 21.4, 21.4, 21.3)
21.9 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}: TD\text{-set}$ $\{a\}$ -form(21.8)
infer $mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil): ConPurse$
 $mk-ConPurse$ -form(21.h1, 21.2, 21.4, 21.7, 21.9)
22 **dom** $\{name \mapsto$
 $mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid$
 $name \in \mathbf{dom} a.abpurses\} = \mathbf{dom} a.abpurses$ dom-defn-map-comp-left-set(20,21)
23 $x.conpurses = \{name \mapsto mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid$
 $name \in \mathbf{dom} a.abpurses\}$ concurses-defn(1)
24 **dom** $x.conpurses = \mathbf{dom} a.abpurses$ =-subs-left(a)(3, 23, 22)
25 **dom** $a.abpurses = \mathbf{dom} x.conpurses$ =-symm(a)(3, 24)
26 **dom** $a.abpurses = \mathbf{dom} \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}$ =-trans-right(c)(4, 25, 9)
27 **from** $y: PurseId; y \in \mathbf{dom} a.abpurses$
27.1 $y \in \mathbf{dom} x.conpurses$ =-subs-right(a)(20, 25, 27.h2)
27.2 $a.abpurses(y): AbPurse$ at-form(27.h1, 19, 27.h2)
27.3 $x.conpurses(y): ConPurse$ at-form(27.h1, 3, 27.1)
27.4 $x.conpurses(y).bal: \mathbb{N}$ ConPurse-bal-form(27.3)
27.5 $x.conpurses(y) = \{name \mapsto mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid$
 $name \in \mathbf{dom} a.abpurses\}(y)$ =-extend(a)(3, 23, 27.3)
27.6 $\{name \mapsto mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid$
 $name \in \mathbf{dom} a.abpurses\}(y) = mk-ConPurse(y, a.abpurses(y).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil)$
 $at-defn-map-comp-left-set(27.h1, 20, 27.h2, 21)$
27.7 $a.abpurses(y).bal: \mathbb{N}$ AbPurse-bal-form(27.2)
27.8 $a.abpurses(y).lost: \mathbb{N}$ AbPurse-lost-form(27.2)
27.9 $0: \mathbb{N}$ 0-form
27.10 $\langle IDLE \rangle: \langle IDLE \rangle$ $\langle IDLE \rangle$ -form
27.11 $Status = \langle IDLE \rangle \mid \langle EPR \rangle \mid \langle EPV \rangle \mid \langle EPA \rangle$ Status-form
27.12 $\langle IDLE \rangle: Status$ Quote-type-extend(27.10, 27.11)

from $a: AbWorld$

...

27.13 $mk-TD(y, y, 0, 0, a.abpurses(y).lost): TD$ TD-form(27.h1, 27.h1, 27.9, 27.9, 27.8)

27.14 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}: TD\text{-set}$ $\{a\}$ -form(27.13)

27.15 $mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\},$
 $nil): ConPurse$ mk-conPurse-form(27.h1, 27.7, 27.9, 27.12, 27.14)

27.16 $mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil).bal = a.abpurses(y).bal$
ConPurse-bal-defn(27.15)

27.17 $mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil).exlog: TD\text{-set}$ ConPurse-exlog-form(27.15)

27.18 $mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil).exlog$
 $= \{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}$ ConPurse-exlog-defn(27.15)

27.19 $x.conpurses(y) = mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil)$ $=\text{-trans}(a)$ (27.3, 27.5, 27.6)

27.20 $x.conpurses(y).exlog = \{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}$
 $=\text{-subs-left}(a)$ (27.3, 27.19, 27.18)

27.21 $sumval(mk-ConPurse(y, a.abpurses(y).bal, 0, <IDLE>, nil,$
 $\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\}, nil).exlog): \mathbb{N}$ sumval-form(27.17)

27.22 $x.conpurses(y).exlog: TD\text{-set}$ ConPurse-exlog-form(27.3)

27.23 $sumval(x.conpurses(y).exlog): \mathbb{N}$ sumval-form(27.22)

27.24 $sumval(x.conpurses(y).exlog)$
 $= sumval(\{mk-TD(y, y, 0, 0, a.abpurses(y).lost)\})$ $=\text{-extend}(a)$ (27.22, 27.20, 27.23)

27.25 $sumval(mk-TD(y, y, 0, 0, a.abpurses(y).lost)) =$
 $mk-TD(y, y, 0, 0, a.abpurses(y).lost).val$ sumval-defn1(27.13)

27.26 $mk-TD(y, y, 0, 0, a.abpurses(y).lost).val = a.abpurses(y).lost$ TD-val-defn(27.13)

27.27 $sumval(mk-TD(y, y, 0, 0, a.abpurses(y).lost)) = a.abpurses(y).lost$
 $=\text{-trans}(c)$ (27.8, 27.25, 27.26)

27.28 $sumval(x.conpurses(y).exlog) = a.abpurses(y).lost$ $=\text{-trans}(c)$ (27.8, 27.24, 27.27)

27.29 $x.conpurses(y).bal = a.abpurses(y).bal$ $=\text{-subs-left}(a)$ (27.3, 27.19, 27.16)

27.30 $mk-AbPurse(x.conpurses(y).bal, sumval(x.conpurses(y).exlog)) =$
 $mk-AbPurse(a.abpurses(y).bal, a.abpurses(y).lost)$ $=\text{-mk-AbPurse}$ (27.4, 27.23, 27.29, 27.28)

27.31 $mk-AbPurse(a.abpurses(y).bal, a.abpurses(y).lost) = a.abpurses(y)$ mk-AbPurse-defn(27.2)

27.32 $mk-AbPurse(x.conpurses(y).bal, sumval(x.conpurses(y).exlog)) = a.abpurses(y)$
 $=\text{-trans}(c)$ (27.2, 27.30, 27.31)

27.33 $\{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}(y) =$
 $mk-AbPurse(x.conpurses(y).bal, sumval(x.conpurses(y).exlog))$
at-defn-map-comp-left-set(27.h1, 4, 27.1, 5)

27.34 $\{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}(y) = a.abpurses(y)$ $=\text{-trans}(c)$ (27.2, 27.33, 27.32)

infer $a.abpurses(y) = \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}(y)$ $=\text{-symm}(b)$ (27.2, 27.34)

from $a: AbWorld$

...

28 $\forall p \in \mathbf{dom} a.abpurses \cdot a.abpurses(p) = \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}(p)$ \forall -I-set(20,27)

29 $a.abpurses = \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}$ $=$ -map-defn(19,6,26,28)

30 $mk-AbWorld(x.conauth, \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}).abpurses = \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}$ abpurses-defn(13)

31 $mk-AbWorld(x.conauth, \{name \mapsto mk-AbPurse(x.conpurses(name).bal,$
 $sumval(x.conpurses(name).exlog)) \mid name \in$
 $\mathbf{dom} x.conpurses\}).abpurses = a.abpurses$ $=$ -subs-left(a)(19, 29, 30)

32 $a.abpurses = mk-AbWorld(x.conauth, \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}).abpurses$ $=$ -symm(b)(19, 31)

33 $a = mk-AbWorld(x.conauth, \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}$ $=$ -AbWorld(h1, 13, 18, 32)

34 $retr(x) = mk-AbWorld(x.conauth, \{name \mapsto$
 $mk-AbPurse(x.conpurses(name).bal, sumval(x.conpurses(name).exlog)) \mid$
 $name \in \mathbf{dom} x.conpurses\}$ retr-defn(1)

35 $retr(x) = a$ $=$ -trans-right(c)(13, 34, 33)

infer $\exists c: ConWorld \cdot retr(c) = a$ \exists -I(1, 35)

Lemma con-assign

$$\frac{a: AbWorld}{\boxed{\text{con-assign}} \quad mk-ConWorld(a.abauth, \{name \mapsto$$

 $mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$
 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid name \in \mathbf{dom} a.abpurses\}): ConWorld$

Abbreviations used:

$$cp = \{name \mapsto mk-ConPurse(name, a.abpurses(name).bal, 0, \langle IDLE \rangle, nil,$$

 $\{mk-TD(name, name, 0, 0, a.abpurses(name).lost)\}, nil) \mid name \in \mathbf{dom} a.abpurses\}$

	from $a: TD$	
1	$\{a\}: TD\text{-set}$	$\{a\}$ -form(h1)
2	$\{\}: TD\text{-set}$	$\{\}$ -form
3	$a \notin \{\}$	$\{\}$ -is-empty(h1)
4	$add(a, \{\}): TD\text{-set}$	unfolding(1)
5	$card(\{\}): \mathbb{N}$	card-form(2)
6	$card(add(a, \{\})): \mathbb{N}$	card-form(4)
7	$succ(card(\{\})) \neq 0$	succ $\neq 0$ (5)
8	$card(add(a, \{\})) = succ(card(\{\}))$	card-defn-add(h1,2,3)
9	$card(add(a, \{\})) \neq 0$	=-subs-left(a)(6,8,7)
10	$add(a, \{\}) \neq \{\}$	card $\neq 0$ -E(4,9)
11	$\{a\} \neq \{\}$	unfolding(9)
12	$sumval(\{a\}): \mathbb{N}$	sumval-form(1)
13	$sumval(\{\}): \mathbb{N}$	sumval-form(2)
14	$\{a\} \setminus \{a\} = \{\}$	diff-self(1)
15	$0: \mathbb{N}$	0-form
16	$\{\} = \{\}$	=-self-I(2)
17	$0 = 0$	=-self-I(15)
18	$\{a\} \setminus \{a\}: TD\text{-set}$	=-type-inherit-left(2,14)
19	if $\{\} = \{\}$ then 0 else let $x \in \{\}$ in $x.val + sumval(\{\} \setminus \{a\}) = 0$	condition-true(15,16)
20	$sumval(\{a\} \setminus \{a\}): \mathbb{N}$	=sub-left(a)(18,14,13)
21	$sumval(\{\}) = 0$	folding(19)
22	$a.val: \mathbb{N}$	TD-val-form(h1)
23	$a.val + 0 = a.val$	+ -defn-0-right(22)
24	$\{a\} \setminus \{a\}: \mathbb{N}$	diff-form(1,1)
25	$sumval(\{a\} \setminus \{a\}) = 0$	=-subs-left(a)(24,14,21)
26	$a.val + sumval(\{a\} \setminus \{a\}): \mathbb{N}$	+form(22,20)
27	let $x \in \{a\}$ in $x.val + sumval(\{x\} \setminus \{a\}) = a.val + sumval(\{a\} \setminus \{a\})$	let-defn1(1)
28	$a.val + sumval(\{a\} \setminus \{a\}) = a.val$	=-subs-left(a)(20,25,23)
29	if $\{a\} = \{\}$ then 0 else let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}): \mathbb{N}$	unfolding(12)
30	(let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}): \mathbb{N}$	let- \in -form(1,26)
31	if $\{a\} = \{\}$ then 0 else let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}) =$ $\text{let } x \in \{a\} \text{ in } x.val + sumval(\{a\} \setminus \{x\})$	condition-false(30,11)
32	let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}) =$ if $\{a\} = \{\}$ then 0 else let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\})$	=symm(b)(29,31)
33	let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}) = a.val$	=-trans(b)(26,27,28)
34	let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}): \mathbb{N}$	=-type-inherit-left(29,32)
35	if $\{a\} = \{\}$ then 0 else let $x \in \{a\}$ in $x.val + sumval(\{a\} \setminus \{x\}) =$ $a.val$	=-subs-right(a)(34,32,33)
	infer $sumval(\{a\}) = a.val$	folding(35)

G.3 Formation of sumval

Conjecture

$$\boxed{\text{sumval-form}} \frac{s: TD\text{-set}}{sumval(s): \mathbb{N}}$$

Proof

First, to enhance readability of the proof, make the following syntactic definition:

$$\Sigma s = (\text{if } s = \{\} \text{ then } 0 \text{ else let } x \in s \text{ in } x.\text{val} + \text{sumval}(s \setminus \{x\}))$$

from $s: TD\text{-set}$	
1 card $s: \mathbb{N}$	card -form(h1)
2 $0: \mathbb{N}$	0-form
3 from $t: TD\text{-set}$	
3.1 from card $t = 0$	
3.1.1 $t = \{\}$	card -0-E(3.h1, 3.1.h1)
3.1.2 $\Sigma t = 0$	condition-true(2, 3.1.1)
infer $\Sigma t: \mathbb{N}$	=-type-inherit-left(2, 3.1.2)
3.2 card $t: \mathbb{N}$	card -form(3.h1)
3.3 $\delta(\text{card } t = 0)$	δ -=-I(3.2, 2)
infer card $t = 0 \Rightarrow \Sigma t: \mathbb{N}$	\Rightarrow -I(3.3, 3.1)
4 $\forall t: TD\text{-set} \cdot \text{card } t = 0 \Rightarrow \Sigma t: \mathbb{N}$	\forall -I(3)
5 from $n: \mathbb{N}; \forall t: TD\text{-set} \cdot \text{card } t = n \Rightarrow \Sigma t: \mathbb{N}$	
5.1 from $u: TD\text{-set}$	
5.1.1 from card $u = \text{succ}(n)$	
5.1.1.1 from $x: TD; x \in u$	
5.1.1.1.1 card $u \setminus \{x\} = n$	card -s\{x\}=n(5.1.h1, 5.1.1.1.h2, 5.h1, 5.1.1.h1)
5.1.1.1.2 $\{x\}: TD\text{-set}$	$\{a\}$ -form(5.1.1.1.h1)
5.1.1.1.3 $u \setminus \{x\}: TD\text{-set}$	diff-form(5.1.h1, 5.1.1.1.2)
5.1.1.1.4 card $u \setminus \{x\} = n \Rightarrow \Sigma u \setminus \{x\}: \mathbb{N}$	\forall -E(5.1.1.1.3, 5.h2)
5.1.1.1.5 $\Sigma u \setminus \{x\}: \mathbb{N}$	\Rightarrow -E-left(5.1.1.1.4, 5.1.1.1.1)
5.1.1.1.6 $x.\text{val}: \mathbb{N}$	TD-val-form(5.1.1.1.h1)
infer $(x.\text{val} + \Sigma u \setminus \{x\}): \mathbb{N}$	+-form(5.1.1.1.6, 5.1.1.1.5)
5.1.1.3 $(\text{let } x \in u \text{ in } x.\text{val} + \Sigma u \setminus \{x\}): \mathbb{N}$	let- \in -form(5.1.h1, 5.1.1.1)
5.1.1.4 $\{ \}: TD\text{-set}$	$\{ \}$ -form
5.1.1.5 $\delta(u = \{ \})$	δ -=-I(5.1.h1, 5.1.1.4)
infer $(\text{if } u = \{ \} \text{ then } 0 \text{ else let } x \in u \text{ in } x.\text{val} + \Sigma(u \setminus \{x\})): \mathbb{N}$	ITE-form(5.1.1.5, 2, 5.1.1.3)
5.1.2 card $u: \mathbb{N}$	card -form(5.1.h1)
5.1.3 succ $(n): \mathbb{N}$	succ -form(5.h1)
5.1.4 $\delta(\text{card } u = \text{succ}(n))$	δ -=-I(5.1.2, 5.1.3)
infer card $u = \text{succ}(n) \Rightarrow \Sigma u: \mathbb{N}$	\Rightarrow -I(5.1.4, 5.1.1)
infer $\forall t: TD\text{-set} \cdot \text{card } t = \text{succ}(n) \Rightarrow \Sigma t: \mathbb{N}$	\forall -I(5.1)
6 $\forall u: TD\text{-set} \cdot \text{card } u = \text{card } s \Rightarrow \Sigma u: \mathbb{N}$	\mathbb{N} -indn(1, 4, 5)
7 card $s = \text{card } s \Rightarrow \Sigma s: \mathbb{N}$	\forall -E(h1, 6)
8 card $s = \text{card } s$	=-self-I(1)
infer $\Sigma s: \mathbb{N}$	\Rightarrow -E-left(7, 8)

Lemma diff- $\{a\}$ - \subset

$$\boxed{\text{diff-}\{a\}\text{-}\subset} \frac{s: A\text{-set}; x \in s; x: A}{s \setminus \{x\} \subset s}$$

	from $s: A\text{-set}; x \in s; x: A$	
1	$\{x\}: A\text{-set}$	$\{x\}$ -form(h3)
2	$s \setminus \{x\}: A\text{-set}$	diff-form(h1,1)
3	$s \setminus \{x\} \subset s \Leftrightarrow s \setminus \{x\} \subseteq s \wedge s \setminus \{x\} \neq s$	\subset -defn(2,h1)
4	$s \setminus \{x\} \subseteq s \Leftrightarrow \forall a \in s \setminus \{x\} \cdot a \in s$	\subseteq -defn(2,h1)
5	from $a: A$	
5.1	from $a \in s \setminus \{x\}$	
	infer $a \in s$	\in -diff-E-right(5.h1,h1,1,5.1.h1)
5.2	$\delta(a \in s \setminus \{x\})$	δ - \in (5.h1,1)
	infer $a \in s \setminus \{x\} \Rightarrow a \in s$	\Rightarrow -I(5.2,5.1)
6	$\forall a \in s \setminus \{x\} \cdot a \in s$	\forall -I(5)
7	$\delta(s \setminus \{x\} = s)$	δ -= -I (2,h1)
8	from $s \setminus \{x\} = s$	
8.1	$s \setminus \{x\} \subseteq s \wedge s \subseteq s \setminus \{x\}$	=-set-E(1,8.h1)
8.2	$s \subseteq s \setminus \{x\}$	\wedge -E-L(8.1)
8.3	$x \in s \setminus \{x\}$	\subseteq -E(h3,h1,2,h2,8.2)
8.4	$x \notin \{x\}$	\in -diff-E-left(h3,h1,1,8.3)
8.5	$x \in \{x\}$	\in - $\{a\}$ -I(h3)
	infer false	contradiction(8.4,8.5)
9	$s \setminus \{x\} \subseteq s$	\Leftrightarrow -E-R(4,6)
10	$s \setminus \{x\} \neq s$	false -contr(7,8)
11	$s \setminus \{x\} \subseteq s \wedge s \setminus \{x\} \neq s$	\wedge -I(9,10)
	infer $s \setminus \{x\} \subset s$	\Leftrightarrow -E-R(3,11)

H Operation Modelling Proofs

H.1 Operation Modelling Proof for ConTransferLostValFail

We show that the concrete operation `ConTransferLostValFail` models the abstract operation `AbTransferLost` under the retrieve function. We examine the domain and result obligations in turn.

H.1.1 Domain Obligation

Conjecture

$$\boxed{\text{TransferLost-dom}} \quad c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N}; \\ \hline \text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val}) \Rightarrow \text{pre-ConTransferLostValFail}(c, \text{fid}, \text{tid}, \text{val})$$

Proof

$$\begin{array}{ll} \text{from } c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N}; & \\ 1. \quad \text{from } \text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val}) & \\ 1.1 \quad \text{fid} \neq \text{tid} \wedge \text{fid} \in \mathbf{dom} \text{ retr}(c).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \text{ retr}(c).\text{abpurses} \wedge & \\ \quad \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val} & \text{unfolding(1.h1)} \\ 1.2 \quad \text{fid} \neq \text{tid} & \wedge\text{-E-right(1.1)} \\ 1.3 \quad \text{fid} \in \mathbf{dom} \text{ retr}(c).\text{abpurses} & \wedge\text{-E-right}(\wedge\text{-E-left(1.1)}) \\ 1.4 \quad \text{tid} \in \mathbf{dom} \text{ retr}(c).\text{abpurses} & \wedge\text{-E-left}(\wedge\text{-E-right(1.1)}) \\ 1.5 \quad \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val} & \wedge\text{-E-left(1.1)} \\ 1.6 \quad \mathbf{dom} \text{ retr}(c).\text{abpurses} = \mathbf{dom} \text{ c.conpurses} & \text{dom-purses-=(h1)} \\ 1.7 \quad \text{retr}(c): \text{AbWorld} & \text{retr-form(h1)} \\ 1.8 \quad \text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse} & \text{abpurses-form(1.7)} \\ 1.9 \quad \mathbf{dom} \text{ retr}(c).\text{abpurses}: \text{PurseId-set} & \text{dom-form(1.8)} \\ 1.10 \quad \text{fid} \in \mathbf{dom} \text{ c.conpurses} & \text{=-subs-right(a)(1.9, 1.6, 1.3)} \\ 1.11 \quad \text{tid} \in \mathbf{dom} \text{ c.conpurses} & \text{=-subs-right(a)(1.9, 1.6, 1.4)} \\ 1.12 \quad \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} = \text{c.conpurses}(\text{fid}).\text{bal} & \text{bal-purses-=(h1, h2, 1.10, 1.3)} \\ 1.13 \quad \text{retr}(c).\text{abpurses}(\text{fid}): \text{AbPurse} & \text{at-form(h2, 1.8, 1.3)} \\ 1.14 \quad \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal}: \mathbb{N} & \text{AbPurse-bal-form(1.13)} \\ 1.15 \quad \text{c.conpurses}(\text{fid}).\text{bal} \geq \text{val} & \text{=-subs-right(a)(1.14, 1.12, 1.5)} \\ 1.16 \quad \text{fid} \neq \text{tid} \wedge \text{fid} \in \mathbf{dom} \text{ c.conpurses} \wedge \text{tid} \in \mathbf{dom} \text{ c.conpurses} \wedge & \\ \quad \text{c.conpurses}(\text{fid}).\text{bal} \geq \text{val} & \wedge\text{-I(1.2, } \wedge\text{-I(1.10, } \wedge\text{-I(1.11, 1.15)))} \\ \text{infer } \text{pre-ConTransferLostValFail}(c, \text{fid}, \text{tid}, \text{val}) & \text{folding(1.16)} \\ 2. \quad \delta(\text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val})) & \delta\text{-pre-AbTransferLost(h1, h2, h3, h4)} \\ \text{infer } \text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val}) \Rightarrow & \\ \quad \text{pre-ConTransferLostValFail}(c, \text{fid}, \text{tid}, \text{val}) & \Rightarrow \text{-I(1,2)} \end{array}$$

Lemma dom-purses-=

$$\boxed{\text{dom-purses-=}} \quad c: \text{ConWorld} \\ \hline \mathbf{dom} \text{ retr}(c).\text{abpurses} = \mathbf{dom} \text{ c.conpurses}$$

```

from  $c: \text{ConWorld}$ 
1    $c.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$  conpurses-form(h1)
2   dom  $c.\text{conpurses}: \text{PurseId}\text{-set}$  dom-form(1)
3    $\text{retr}(c): \text{AbWorld}$  retr-form(h1)
4    $\text{retr}(c) = \text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \})$  retr-defn(h1)
5    $\text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$  abpurses-form(3)
6   dom  $\text{retr}(c).\text{abpurses}: \text{PurseId}\text{-set}$  dom-form(5)
7    $\text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \})$  =-type-inherit-right(3,4)
8    $\text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \}).\text{abpurses}$ 
      =
       $\{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$  abpurses-defn(7)
9   from  $\text{name}: \text{PurseId}; \text{name} \in \mathbf{dom} \ c.\text{conpurses}$ 
   infer  $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal},$ 
       $\text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}): \text{AbPurse}$  ConWorld-AbWorld-form(9.h1, h1, 9.h2)
10   $\text{retr}(c).\text{abpurses} = \text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \}).\text{abpurses}$  =-extend(a)(3,4,5)
11   $\text{retr}(c).\text{abpurses} = \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$  =-trans(a)(5,10,8)
12  dom  $\text{retr}(c).\text{abpurses} = \mathbf{dom} \ \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$  =-extend(a)(5,11,6)
13  dom  $\{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} \ c.\text{conpurses} \} = \mathbf{dom} \ c.\text{conpurses}$  dom-defn-map-comp-left-set(2,9)
infer dom  $\text{retr}(c).\text{abpurses} = \mathbf{dom} \ c.\text{conpurses}$  =-trans(c)(2,12,13)

```

Lemma $\text{bal-purses} =$

bal-purses-equal	$c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{fid} \in \mathbf{dom} \ c.\text{conpurses}; \text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses}$
	$\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} = c.\text{conpurses}(\text{fid}).\text{bal}$

```

from  $c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{fid} \in \mathbf{dom} \ c.\text{conpurses}; \text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses}$ 
1   $\text{retr}(c) = \text{mk-AbWorld}(c.\text{conauth},$ 
    $\{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}))$ 
    $| \text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$ 
   retr-defn(h1)
2   $\text{retr}(c): \text{AbWorld}$ 
   retr-form(h1)
3   $\text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$ 
   abpurses-form(2)
4   $\text{retr}(c).\text{abpurses} = \text{retr}(c).\text{abpurses}$ 
   =-self-I(3)
5   $\text{retr}(c).\text{abpurses} = \text{mk-AbWorld}(\dots).\text{abpurses}$ 
   =-subs-right(a)(2, 1, 4)
6   $\text{mk-AbWorld}(\dots).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$ 
   =-type-inherit-right(3, 5)
7   $\text{mk-AbWorld}(\dots): \text{AbWorld}$ 
   =-type-inherit-right(2, 1)
8   $\text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid \text{name} \in \mathbf{dom} \ c.\text{conpurses} \}).\text{abpurses} =$ 
    $\{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid \text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$ 
   abpurses-defn(7)
9   $\text{retr}(c).\text{abpurses} = \{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid \text{name} \in \mathbf{dom} \ c.\text{conpurses} \}$ 
   =-subs-left(b)(6, 5, 8)
10  $\text{retr}(c).\text{abpurses}(\text{fid}): \text{AbPurse}$ 
   at-form(h2, 3, h4)
11  $\text{retr}(c).\text{abpurses}(\text{fid}) = \text{retr}(c).\text{abpurses}(\text{fid})$ 
   =-self-I(10)
12  $\text{retr}(c).\text{abpurses}(\text{fid}) = \{ \text{name} \mapsto \dots \}(\text{fid})$ 
   =-subs-right(a)(3, 9, 11)
13  $\{ \text{name} \mapsto \dots \}(\text{fid}): \text{AbPurse}$ 
   =-type-inherit-right(10, 12)
14  $c.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$ 
   conpurses-form(h1)
15 dom  $c.\text{conpurses}: \text{PurseId-set}$ 
   dom-form(14)
16 from  $\text{name}: \text{PurseId}; \text{name} \in \mathbf{dom} \ c.\text{conpurses}$ 
infer  $\text{mk-AbPurse}(\dots): \text{AbPurse}$ 
   ConWorld-AbWorld-form(16.h1, h1, 16.h2)
17  $\{ \text{name} \mapsto \text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid \text{name} \in \mathbf{dom} \ c.\text{conpurses} \}(\text{fid}) =$ 
    $\text{mk-AbPurse}(c.\text{conpurses}(\text{fid}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{fid}).\text{exlog}))$ 
   at-defn-map-comp-left-set(h2, 15, h3, 16)
18  $\text{retr}(c).\text{abpurses}(\text{fid}) = \text{mk-AbPurse}(c.\text{conpurses}(\text{fid}).\text{bal},$ 
    $\text{sumval}(c.\text{conpurses}(\text{fid}).\text{exlog}))$ 
   =-subs-left(b)(13, 12, 17)
19  $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal}: \mathbb{N}$ 
   AbPurse-bal-form(10)
20  $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} = \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal}$ 
   =-self-I(19)
21  $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} = \text{mk-AbPurse}(\dots).\text{bal}$ 
   =-subs-right(a)(10, 18, 20)
22  $\text{mk-AbPurse}(\dots).\text{bal}: \mathbb{N}$ 
   =-type-inherit-right(19, 21)
23  $\text{mk-AbPurse}(\dots): \text{AbPurse}$ 
   =-type-inherit-right(10, 18)
24  $\text{mk-AbPurse}(c.\text{conpurses}(\text{fid}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{fid}).\text{exlog})).\text{bal} =$ 
    $c.\text{conpurses}(\text{fid}).\text{bal}$ 
   AbPurse-bal-defn(23)
infer  $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} = c.\text{conpurses}(\text{fid}).\text{bal}$ 
   =-subs-left(b)(22, 21, 24)

```

Lemma conworld-abworld-form

ConWorld-AbWorld-form	$n: \text{PurseId}; c: \text{ConWorld}; n \in \mathbf{dom} \ c.\text{conpurses}$ $\text{mk-AbPurse}(c.\text{conpurses}(n).\text{bal}, \text{sumval}(c.\text{conpurses}(n).\text{exlog})): \text{AbPurse}$
-----------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	from $n: \text{PurseId}; c: \text{ConWorld}; n \in \mathbf{dom} \ c.\text{conpurses}$	
1	$c.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$	conpurses-form(h2)
2	$c.\text{conpurses}(n): \text{ConPurse}$	at-form(h1,1,h3)
3	$c.\text{conpurses}(n).\text{exlog}: \text{TD-set}$	ConPurse-exlog-form(2)
4	$\text{sumval}(c.\text{conpurses}(n).\text{exlog}): \mathbb{N}$	sumval-form(3)
5	$c.\text{conpurses}(n).\text{bal}: \mathbb{N}$	ConPurse-bal-form(2)
	infer $\text{mk-AbPurse}(c.\text{conpurses}(n).\text{bal},$ $\text{sumval}(c.\text{conpurses}(n).\text{exlog})): \text{AbPurse}$	mk-AbPurse-form(5,4)

Lemma δ -pre-abtransferlost

$$\boxed{\delta\text{-pre-AbTransferLost}} \frac{c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N};}{\delta(\text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val}))}$$

	from $c: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N};$	
1	$\delta(\text{fid} \neq \text{tid})$	$\delta\text{-}\neq\text{-I}(\text{h2}, \text{h3})$
2	$\text{retr}(c): \text{AbWorld}$	retr-form(h1)
3	$\text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$	abpurses-form(2)
4	dom $\text{retr}(c).\text{abpurses}: \text{PurseId-set}$	dom-form(3)
5	$\delta(\text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses})$	$\delta\text{-}\in(\text{h2}, 4)$
6	$\delta(\text{tid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses})$	$\delta\text{-}\in(\text{h3}, 4)$
7	$\delta(\text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses})$	$\delta\text{-}\wedge\text{-inherit}(5, 6)$
8	from $\text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses}$	
8.1	$\text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses}$	$\wedge\text{-E-right}(8.\text{h1})$
8.2	$\text{retr}(c).\text{abpurses}(\text{fid}): \text{AbPurse}$	at-form(h2, 3, 8.1)
8.3	$\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal}: \mathbb{N}$	AbPurse-bal-form(8.2)
	infer $\delta(\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val})$	$\delta\text{-}\geq(8.3, \text{h4})$
9	$\delta(\text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge$ $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val})$	$\delta\text{-}\wedge\text{-inherit-sqrt}(7, 8)$
10	$\delta(\text{fid} \neq \text{tid} \wedge \text{fid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \wedge$ $\text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val})$	$\delta\text{-}\wedge\text{-inherit}(1, 9)$
	infer $\delta(\text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val}))$	folding(10)

Lemma $\delta\text{-}\geq$

$$\boxed{\delta\text{-}\geq} \frac{a: \mathbb{N}; b: \mathbb{N}}{\delta(a \geq b)}$$

from $a: \mathbb{N}; b: \mathbb{N}$	
1 from $k: \mathbb{N}$	
1.1 $(b + k): \mathbb{N}$	+-form(h2, 1.h1)
infer $\delta(b + k = a)$	$\delta=-\text{I}(1.1, \text{h1})$
2 $\delta(\exists k: \mathbb{N} \cdot b + k = a)$	$\delta-\exists\text{-inherit}(1)$
infer $\delta(a \geq b)$	folding(2)

H.1.2 Result Obligation

Conjecture

$c: \text{ConWorld}; \overleftarrow{c}: \text{ConWorld}; fid: \text{PurseId}; tid: \text{PurseId}; val: \mathbb{N}$	
TransferLost-res	$\frac{pre\text{-}AbTransferLost(retr(c), fid, tid, val); post\text{-}ConTransferLostValFail(c, \overleftarrow{c}, fid, tid, val)}{post\text{-}AbTransferLost(retr(c), retr(\overleftarrow{c}), fid, tid, val)}$

Proof

from $c: \text{ConWorld}; \overleftarrow{c}: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N};$
 $\text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val});$
 $\text{post-ConTransferLostValFail}(c, \overleftarrow{c}, \text{fid}, \text{tid}, \text{val})$

1 $\text{fid} \neq \text{tid} \wedge \text{fid} \in \mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses} \wedge \text{tid} \in \mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses} \wedge$
 $\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{fid}).\text{bal} \geq \text{val}$ unfolding(h6)

2 $\text{totalC}(\overleftarrow{c}.\text{compurses}(\text{fid})) = \text{totalC}(c.\text{compurses}(\text{fid})) \wedge$
 $\overleftarrow{c}.\text{compurses}(\text{fid}).\text{bal} \geq c.\text{compurses}(\text{fid}).\text{bal} \wedge$
 $c.\text{compurses}(\text{tid}).\text{bal} = \overleftarrow{c}.\text{compurses}(\text{tid}).\text{bal} \wedge$
 $c.\text{compurses}(\text{tid}).\text{exlog} = \overleftarrow{c}.\text{compurses}(\text{tid}).\text{exlog} \wedge$
 $c.\text{compurses}(\text{fid}).\text{seqno} \geq \overleftarrow{c}.\text{compurses}(\text{fid}).\text{seqno} \wedge$
 $c.\text{compurses}(\text{tid}).\text{seqno} \geq \overleftarrow{c}.\text{compurses}(\text{tid}).\text{seqno} \wedge$
 $c.\text{compurses}(\text{fid}).\text{status} = \langle \text{IDLE} \rangle \wedge c.\text{compurses}(\text{tid}).\text{status} = \langle \text{IDLE} \rangle \wedge$
 $\mathbf{dom} \overleftarrow{c}.\text{compurses} = \mathbf{dom} c.\text{compurses} \wedge \overleftarrow{c}.\text{conauth} = c.\text{conauth} \wedge$
 $\forall \text{pid} \in (\mathbf{dom} c.\text{compurses} \setminus \{\text{fid}, \text{tid}\}).$
 $\overleftarrow{c}.\text{compurses}(\text{pid}) = c.\text{compurses}(\text{pid})$ unfolding(h7)

3 $\text{totalC}(\overleftarrow{c}.\text{compurses}(\text{fid})) = \text{totalC}(c.\text{compurses}(\text{fid}))$ \wedge -E-right(\wedge -E-left(2))

4 $\overleftarrow{c}.\text{compurses}(\text{fid}).\text{bal} \geq c.\text{compurses}(\text{fid}).\text{bal}$ \wedge -E-right(\wedge -E-left(2))

5 $\mathbf{dom} \overleftarrow{c}.\text{compurses} = \mathbf{dom} c.\text{compurses}$ \wedge -E-left(\wedge -E-right(2))

6 $\overleftarrow{c}.\text{conauth} = c.\text{conauth}$ \wedge -E-left(\wedge -E-right(2))

7 $\forall \text{pid} \in (\mathbf{dom} c.\text{compurses} \setminus \{\text{fid}, \text{tid}\}). \overleftarrow{c}.\text{compurses}(\text{pid}) = c.\text{compurses}(\text{pid})$
 \wedge -E-left(2)

8 $c.\text{compurses}(\text{tid}).\text{bal} = \overleftarrow{c}.\text{compurses}(\text{tid}).\text{bal}$ \wedge -E-right(\wedge -E-left(2))

9 $c.\text{compurses}(\text{tid}).\text{exlog} = \overleftarrow{c}.\text{compurses}(\text{tid}).\text{exlog}$ \wedge -E-right(\wedge -E-left(2))

10 $\text{fid} \in \mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses}$ \wedge -E-right(\wedge -E-left(1))

11 $\text{tid} \in \mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses}$ \wedge -E-right(\wedge -E-left(1))

12 $\mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses} = \mathbf{dom} \text{retr}(c).\text{abpurses}$ post-dom-=(h1, h2, 5)

13 $\text{retr}(c): \text{AbWorld}$ retr-form(h1)

14 $\text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$ abpurses-form(13)

15 $\mathbf{dom} \text{retr}(c).\text{abpurses}: \text{PurseId-set}$ dom-form(14)

16 $\text{retr}(\overleftarrow{c}): \text{AbWorld}$ retr-form(h2)

17 $\text{retr}(\overleftarrow{c}).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$ abpurses-form(16)

18 $\mathbf{dom} \text{retr}(\overleftarrow{c}).\text{abpurses}: \text{PurseId-set}$ dom-form(17)

19 $\text{fid} \in \mathbf{dom} \text{retr}(c).\text{abpurses}$ \in -set(h3, 18, 15, 10, 12)

20 $\text{tid} \in \mathbf{dom} \text{retr}(c).\text{abpurses}$ \in -set(h4, 18, 15, 11, 12)

from $c: \text{ConWorld}; \overleftarrow{c}: \text{ConWorld}; \text{fid}: \text{PurseId}; \text{tid}: \text{PurseId}; \text{val}: \mathbb{N};$
 $\text{pre-AbTransferLost}(\text{retr}(c), \text{fid}, \text{tid}, \text{val});$
 $\text{post-ConTransferLostValFail}(c, \overleftarrow{c}, \text{fid}, \text{tid}, \text{val})$
...
21 **dom** $\text{retr}(c).\text{abpurses} = \text{dom } c.\text{conpurses}$ dom-purses=(h1)
22 **dom** $\text{retr}(\overleftarrow{c}).\text{abpurses} = \text{dom } \overleftarrow{c}.\text{conpurses}$ dom-purses=(h2)
23 $c.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$ conpurses-form(h1)
24 **dom** $c.\text{conpurses}: \text{PurseId}\text{-set}$ dom-form(23)
25 $\overleftarrow{c}.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$ conpurses-form(h2)
26 **dom** $\overleftarrow{c}.\text{conpurses}: \text{PurseId}\text{-set}$ dom-form(25)
27 $\text{fid} \in \text{dom } c.\text{conpurses}$ $\in\text{-set}(\text{h3}, 15, 24, 19, 21)$
28 $\text{tid} \in \text{dom } c.\text{conpurses}$ $\in\text{-set}(\text{h4}, 15, 24, 20, 21)$
29 $\text{fid} \in \text{dom } \overleftarrow{c}.\text{conpurses}$ $\in\text{-set}(\text{h3}, 18, 26, 10, 22)$
30 $\text{tid} \in \text{dom } \overleftarrow{c}.\text{conpurses}$ $\in\text{-set}(\text{h4}, 18, 26, 11, 22)$
31 $\text{totalA}(\text{retr}(c).\text{abpurses}(\text{fid})) = \text{totalA}(\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{fid}))$
post-total=(h1, h2, h3, 3, 19, 10, 27, 29)
32 $\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{fid}).\text{bal} \geq \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal}$
post-bal- \geq (h1, h2, h3, 19, 10, 27, 29, 4)
33 $\text{retr}(\overleftarrow{c}).\text{abauth} = \text{retr}(c).\text{abauth}$ post-abauth=(h2, h1, 6)
34 $\forall \text{pid} \in (\text{dom } \text{retr}(c).\text{abpurses} \setminus \{\text{fid}\}).$
 $\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{pid}) = \text{retr}(c).\text{abpurses}(\text{pid})$
post- \forall =(h1, h2, h3, h4, 20, 11, 28, 30, 12, 7, 8, 9)
35 $\text{totalA}(\text{retr}(c).\text{abpurses}(\text{fid})) = \text{totalA}(\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{fid})) \wedge$
 $\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{fid}).\text{bal} \geq \text{retr}(c).\text{abpurses}(\text{fid}).\text{bal} \wedge$
 $\text{dom } \text{retr}(\overleftarrow{c}).\text{abpurses} = \text{dom } \text{retr}(c).\text{abpurses} \wedge$
 $\text{retr}(\overleftarrow{c}).\text{abauth} = \text{retr}(c).\text{abauth} \wedge$
 $\forall \text{pid} \in (\text{dom } \text{retr}(c).\text{abpurses} \setminus \{\text{fid}\}).$
 $\text{retr}(\overleftarrow{c}).\text{abpurses}(\text{pid}) = \text{retr}(c).\text{abpurses}(\text{pid})$
 $\wedge\text{-I}(31, \wedge\text{-I}(32, \wedge\text{-I}(12, \wedge\text{-I}(33, 34))))$
infer $\text{post-AbTransferLost}(\text{retr}(c), \text{retr}(\overleftarrow{c}), \text{fid}, \text{tid}, \text{val})$ folding(35)

Lemma rule- $\in\text{-set}$

$$\boxed{\in\text{-set}} \frac{a: A; s1: A\text{-Set}; s2: A\text{-set}; a \in s1; s1 = s2}{a \in s2}$$

from $a: A; s1: A\text{-Set}; s2: A\text{-set}; a \in s1; s1 = s2$
1 $(s1 \subseteq s2) \wedge (s2 \subseteq s1)$ $\text{=set-E}(\text{h2}, \text{h5})$
2 $s1 \subseteq s2$ $\wedge\text{-E-right}(1)$
infer $a \in s2$ $\subseteq\text{-E}(\text{h1}, \text{h2}, \text{h3}, \text{h4}, 2)$

Lemma post-dom-=-

$$\boxed{\text{post-dom-=-}} \frac{a: \text{ConWorld}; b: \text{ConWorld}; \mathbf{dom} \ b.\text{compurses} = \mathbf{dom} \ a.\text{compurses}}{\mathbf{dom} \ \text{retr}(b).\text{abpurses} = \mathbf{dom} \ \text{retr}(a).\text{abpurses}}$$

from $a: \text{ConWorld}; b: \text{ConWorld}; \mathbf{dom} \ b.\text{compurses} = \mathbf{dom} \ a.\text{compurses}$

1	$b.\text{compurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$	compurse-form(h2)
2	$\mathbf{dom} \ b.\text{compurses}: \text{PurseId-set}$	dom-form(1)
3	$a.\text{compurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$	compurse-form(h1)
4	$\mathbf{dom} \ a.\text{compurses}: \text{PurseId-set}$	dom-form(3)
5	$\mathbf{dom} \ \text{retr}(b).\text{abpurses} = \mathbf{dom} \ b.\text{compurses}$	dom-purses-=(h2)
6	$\mathbf{dom} \ \text{retr}(a).\text{abpurses} = \mathbf{dom} \ a.\text{compurses}$	dom-purses-=(h1)
7	$\mathbf{dom} \ \text{retr}(b).\text{abpurses} = \mathbf{dom} \ a.\text{compurses}$	=-trans(b)(2, 5, h3)
	infer $\mathbf{dom} \ \text{retr}(b).\text{abpurses} = \mathbf{dom} \ \text{retr}(a).\text{abpurses}$	=-trans-right(a)(4, 6, 7)

Lemma post-total-=-

$$\boxed{\text{post-total-=-}} \frac{a: \text{ConWorld}; b: \text{ConWorld}; \text{pid}: \text{PurseId}; \\ \text{totalC}(b.\text{compurses}(\text{pid})) = \text{totalC}(a.\text{compurses}(\text{pid})); \text{pid} \in \mathbf{dom} \ \text{retr}(a).\text{abpurses}; \\ \text{pid} \in \mathbf{dom} \ \text{retr}(b).\text{abpurses}; \text{pid} \in \mathbf{dom} \ a.\text{compurses}; \text{pid} \in \mathbf{dom} \ b.\text{compurses};}{\text{totalA}(\text{retr}(a).\text{abpurses}(\text{pid})) = \text{totalA}(\text{retr}(b).\text{abpurses}(\text{pid}))}$$

from $a: \text{ConWorld}; b: \text{ConWorld}; \text{pid}: \text{PurseId}; \\ \text{totalC}(b.\text{compurses}(\text{pid})) = \text{totalC}(a.\text{compurses}(\text{pid})); \text{pid} \in \mathbf{dom} \ \text{retr}(a).\text{abpurses}; \\ \text{pid} \in \mathbf{dom} \ \text{retr}(b).\text{abpurses}; \text{pid} \in \mathbf{dom} \ a.\text{compurses}; \text{pid} \in \mathbf{dom} \ b.\text{compurses}$

1	$\text{totalC}(a.\text{compurses}(\text{pid})) = \text{totalA}(\text{retr}(a).\text{abpurses}(\text{pid}))$	total-purses-=(h1, h3, h7, h5)
2	$\text{totalC}(b.\text{compurses}(\text{pid})) = \text{totalA}(\text{retr}(b).\text{abpurses}(\text{pid}))$	total-purses-=(h2, h3, h8, h6)
3	$b.\text{compurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$	compurses-form(h2)
4	$b.\text{compurses}(\text{pid}): \text{ConPurse}$	at-form(h3, 3, h8)
5	$\text{totalC}(b.\text{compurses}(\text{pid})): \mathbb{N}$	total-C-form(4)
6	$\text{totalC}(b.\text{compurses}(\text{pid})) = \text{totalA}(\text{retr}(a).\text{abpurses}(\text{pid}))$	=-trans(a)(5, h4, 1)
	infer $\text{totalA}(\text{retr}(a).\text{abpurses}(\text{pid})) = \text{totalA}(\text{retr}(b).\text{abpurses}(\text{pid}))$	=-trans-left(a)(5, 6, 2)

Lemma total-C-form

$$\boxed{\text{total-C-form}} \frac{p: \text{ConPurse}}{\text{totalC}(p): \mathbb{N}}$$

from $p: \text{ConPurse}$

1	$p.\text{bal}: \mathbb{N}$	ConPurse-bal-form(h1)
2	$p.\text{exlog}: \text{TD-set}$	ConPurse-exlog-form(h1)
3	$\text{sumval}(p.\text{exlog}): \mathbb{N}$	sumval-form(2)
4	$(p.\text{bal} + \text{sumval}(p.\text{exlog})): \mathbb{N}$	+-form(1,3)
	infer $\text{totalC}(p): \mathbb{N}$	folding(4)

Lemma total-purses-=-

$$\boxed{\text{total-purses-=-}} \quad c: \text{ConWorld}; pid: \text{PurseId}; pid \in \mathbf{dom} \ c.\text{conpurses}; pid \in \mathbf{dom} \ \text{retr}(c).\text{abpurses} \\ \text{totalC}(c.\text{conpurses}(pid)) = \text{totalA}(\text{retr}(c).\text{abpurses}(pid))$$

- from** $c: \text{ConWorld}; pid: \text{PurseId}; pid \in \mathbf{dom} \ c.\text{conpurses}; pid \in \mathbf{dom} \ \text{retr}(c).\text{abpurses}$
1. $c.\text{conpurses}: \text{PurseId} \xrightarrow{m} \text{ConPurse}$ conpurses-form(h1)
 2. $c.\text{conpurses}(pid): \text{ConPurse}$ at-form(h2, 1, h3)
 3. $\text{totalC}(c.\text{conpurses}(pid)) =$
 $\quad c.\text{conpurses}(pid).\text{bal} + \text{sumval}(c.\text{conpurses}(pid).\text{exlog})$ totalC-defn(2)
 4. $\text{totalC}(c.\text{conpurses}(pid)): \mathbb{N}$ totalC-form(2)
 5. $\text{retr}(c): \text{AbWorld}$ retr-form(h1)
 6. $\text{retr}(c).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$ abpurses-form(5)
 7. $\text{retr}(c).\text{abpurses}(pid): \text{AbPurse}$ at-form(h2, 6, h4)
 8. $\text{totalA}(\text{retr}(c).\text{abpurses}(pid)) =$
 $\quad \text{retr}(c).\text{abpurses}(pid).\text{bal} + \text{retr}(c).\text{abpurses}(pid).\text{lost}$ totalA-defn(7)
 9. $\text{retr}(c).\text{abpurses}(pid) = \text{mk-AbPurse}(c.\text{conpurses}(pid).\text{bal},$
 $\quad \text{sumval}(c.\text{conpurses}(pid).\text{exlog}))$ at-mk-AbPurse-defn(h1, h2, h3, h4)
 10. $\text{mk-AbPurse}(c.\text{conpurses}(pid).\text{bal}, \text{sumval}(c.\text{conpurses}(pid).\text{exlog})):$
 $\quad \text{AbPurse}$ =-type-inherit-right(7,9)
 11. $\text{mk-AbPurse}(c.\text{conpurses}(pid).\text{bal}, \text{sumval}(c.\text{conpurses}(pid).\text{exlog})).\text{bal} =$
 $\quad c.\text{conpurses}(pid).\text{bal}$ abpurse-bal-defn(10)
 12. $\text{mk-AbPurse}(c.\text{conpurses}(pid).\text{bal}, \text{sumval}(c.\text{conpurses}(pid).\text{exlog})).\text{lost} =$
 $\quad \text{sumval}(c.\text{conpurses}(pid).\text{exlog})$ abpurse-lost-defn(10)
 13. $\text{retr}(c).\text{abpurses}(pid).\text{bal} = c.\text{conpurses}(pid).\text{bal}$ =-subs-left(a)(7,9,11)
 14. $\text{retr}(c).\text{abpurses}(pid).\text{lost} = \text{sumval}(c.\text{conpurses}(pid).\text{exlog})$ =-subs-left(a)(7,9,12)
 15. $\text{retr}(c).\text{abpurses}(pid).\text{bal}: \mathbb{N}$ AbPurse-bal-form(7)
 16. $c.\text{conpurses}(pid).\text{bal}: \mathbb{N}$ ConPurse-bal-form(2)
 17. $\text{retr}(c).\text{abpurses}(pid).\text{lost}: \mathbb{N}$ AbPurse-lost-form(7)
 18. $\text{retr}(c).\text{abpurses}(pid).\text{bal} + \text{retr}(c).\text{abpurses}.\text{lost} =$
 $\quad c.\text{conpurses}(pid).\text{bal} + \text{sumval}(c.\text{conpurses}(pid).\text{exlog})$ +-(13, 14, 15, 16, 17)
 19. $\text{totalC}(c.\text{conpurses}(pid)) =$
 $\quad \text{retr}(c).\text{abpurses}(pid).\text{bal} + \text{retr}(c).\text{abpurses}.\text{lost}$ =-trans-right(a)(4,3,18)
- infer** $\text{totalC}(c.\text{conpurses}(pid)) = \text{totalA}(\text{retr}(c).\text{abpurses}(pid))$ =-trans-right(a)(4,19,8)

Lemma rule-+-=

$$\boxed{+-=} \quad \frac{a = b; c = d; a: \mathbb{N}; b: \mathbb{N}; c: \mathbb{N}}{a + c = b + d}$$

	from $a = b; c = d; a: \mathbb{N}; b: \mathbb{N}; c: \mathbb{N}$	
1	$a + 0 = a$	+ -defn-0-right(h3)
2	$b + 0 = b$	+ -defn-0-right(h4)
3	$a + 0 = b$	=-trans(b)(h3, 1, h1)
4	$a + 0 = b + 0$	=-trans-right(c)(h4, 3, 2)
5	from $k: \mathbb{N}; a + k = b + k$	
5.1	$a + k: \mathbb{N}$	+ -form(h3, 5.h1)
5.2	$b + k: \mathbb{N}$	+ -form(h4, 5.h1)
5.3	$a + \text{succ}(k) = \text{succ}(a + k)$	+ -defn-succ-right(h3, 5.h1)
5.4	$b + \text{succ}(k) = \text{succ}(b + k)$	+ -defn-succ-right(h4, 5.h1)
5.5	$\text{succ}(b + k): \mathbb{N}$	succ-form(5.2)
5.6	$a + \text{succ}(k) = \text{succ}(b + k)$	=-subs-right(a)(5.1, 5.h2, 5.3)
	infer $a + \text{succ}(k) = b + \text{succ}(k)$	=-trans-right(c)(5.5, 5.6, 5.4)
6	$a + c = b + c$	\mathbb{N} -indn(h5, 4, 5)
	infer $a + c = b + d$	=-subs-right(a)(h5, h2, 6)

Lemma post-bal- \geq

	$a: \text{ConWorld}; b: \text{ConWorld}; pid: \text{PurseId}; pid \in \mathbf{dom} \text{retr}(a).\text{abpurses};$
	$pid \in \mathbf{dom} \text{retr}(b).\text{abpurses}; pid \in \mathbf{dom} a.\text{compurses}; pid \in \mathbf{dom} b.\text{compurses};$
post-bal- \geq	$b.\text{compurses}(pid).\text{bal} \geq a.\text{compurses}(pid).\text{bal}$
	$\text{retr}(b).\text{abpurses}(pid).\text{bal} \geq \text{retr}(a).\text{abpurses}(pid).\text{bal}$

	from $a: \text{ConWorld}; b: \text{ConWorld}; pid: \text{PurseId}; pid \in \mathbf{dom} \text{retr}(a).\text{abpurses};$	
	$pid \in \mathbf{dom} \text{retr}(b).\text{abpurses}; pid \in \mathbf{dom} a.\text{compurses}; pid \in \mathbf{dom} b.\text{compurses};$	
	$b.\text{compurses}(pid).\text{bal} \geq a.\text{compurses}(pid).\text{bal}$	
1	$\text{retr}(b).\text{abpurses}(pid).\text{bal} = b.\text{compurses}(pid).\text{bal}$	bal-purses-equal(h2, h3, h7, h5)
2	$\text{retr}(a).\text{abpurses}(pid).\text{bal} = a.\text{compurses}(pid).\text{bal}$	bal-purses-equal(h1, h3, h6, h4)
3	$\text{retr}(a).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$	abpurse-form(h1)
4	$\text{retr}(a).\text{abpurses}(pid): \text{AbPurse}$	at-form(h3, 3, h4)
5	$\text{retr}(a).\text{abpurses}(pid).\text{bal}: \mathbb{N}$	bal-form(4)
6	$a.\text{compurses}(pid).\text{bal} = \text{retr}(a).\text{abpurses}(pid).\text{bal}$	=-symm(a)(5, 2)
7	$\text{retr}(b).\text{abpurses}: \text{PurseId} \xrightarrow{m} \text{AbPurse}$	abpurse-form(h2)
8	$\text{retr}(b).\text{abpurses}(pid): \text{AbPurse}$	at-form(h3, 7, h5)
9	$\text{retr}(b).\text{abpurses}(pid).\text{bal}: \mathbb{N}$	bal-form(8)
10	$b.\text{compurses}(pid).\text{bal} \geq \text{retr}(a).\text{abpurses}(pid).\text{bal}$	=-subs-right(b)(5, 6, h8)
11	$b.\text{compurses}(pid).\text{bal} = \text{retr}(b).\text{abpurses}(pid).\text{bal}$	=-symm(a)(9, 1)
	infer $\text{retr}(b).\text{abpurses}(pid).\text{bal} \geq \text{retr}(a).\text{abpurses}(pid).\text{bal}$	=-subs-right(b)(9, 11, 10)

from $a: ConWorld; b: ConWorld; fid: PurseId; tid: PurseId;$
 $tid \in \mathbf{dom} \text{retr}(a).abpurses; tid \in \mathbf{dom} \text{retr}(b).abpurses;$
 $tid \in \mathbf{dom} a.conpurses; tid \in \mathbf{dom} b.conpurses;$
 $\mathbf{dom} \text{retr}(b).abpurses = \mathbf{dom} \text{retr}(a).abpurses;$
 $\forall pid \in (\mathbf{dom} a.conpurses \setminus \{fid, tid\}) \cdot b.conpurses(pid) = a.conpurses(pid);$
 $a.conpurses(tid).bal = b.conpurses(tid).bal;$
 $a.conpurses(tid).exlog = b.conpurses(tid).exlog$
...
22 **from** $y: PurseId; y \in \mathbf{dom} \text{retr}(a).abpurses \setminus \{fid, tid\}$
22.1 $y \in \mathbf{dom} \text{retr}(b).abpurses \setminus \{fid, tid\}$ =-subs-left(b)(17, h9, 22.h2)
22.2 $\mathbf{dom} \text{retr}(b).abpurses: PurseId\text{-set}$ dom-form(10)
22.3 $y \in \mathbf{dom} \text{retr}(b).abpurses$ ∈-diff-E-right(22.h1, 22.2, 18, 22.1)
22.4 $\mathbf{dom} \text{retr}(b).abpurses = \mathbf{dom} b.conpurses$ dom-purses-=(h2)
22.5 $y \in \mathbf{dom} b.conpurses$ =-subs-right(a)(22.2, 22.4, 22.3)
22.6 $\text{retr}(b).abpurses(y) = mk\text{-}AbPurse(b.conpurses(y).bal,$
 $\text{sumval}(b.conpurses(y).exlog))$ at-mk-AbPurse-defn(h2, 22.h1, 22.6, 22.3)
22.7 $y \in \mathbf{dom} \text{retr}(a).abpurses$ ∈-diff-E-right(22.h1, 17, 18, 22.h2)
22.8 $y \in \mathbf{dom} a.conpurses$ =-subs-right(a)(17, 20, 22.7)
22.9 $\text{retr}(a).abpurses(y) = mk\text{-}AbPurse(a.conpurses(y).bal,$
 $\text{sumval}(a.conpurses(y).exlog))$ at-mk-AbPurse-defn(h1, 22.h1, 22.8, 22.7)
22.10 $a.conpurses(y): ConPurse$ at-form(22.h1, 1, 22.8)
22.11 $b.conpurses(y) = a.conpurses(y)$ ∀-E-set(22.h1, 19, 22.h2, 21)
22.12 $\text{retr}(a).abpurses(y) = mk\text{-}AbPurse(b.conpurses(y).bal,$
 $\text{sumval}(b.conpurses(y).exlog))$ =-subs-left(b)(22.10, 22.11, 22.9)
22.13 $\text{retr}(b).abpurses(y): AbPurse$ at-form(22.h1, 10, 22.3)
infer $\text{retr}(b).abpurses(y) = \text{retr}(a).abpurses(y)$ =-trans-right(a)(22.13, 22.6, 22.12)
23 $\forall pid \in (\mathbf{dom} \text{retr}(a).abpurses \setminus \{fid, tid\}).$
 $\text{retr}(b).abpurses(pid) = \text{retr}(a).abpurses(pid)$ ∀-I-set(19, 22)
24 $\{fid\}: \{PurseId\}$ {a}-form(h3)
25 $\mathbf{dom} \text{retr}(a).abpurses \setminus \{fid\}: \{PurseId\}$ diff-form(17,24)
infer $\forall pid \in (\mathbf{dom} \text{retr}(a).abpurses \setminus \{fid\}).$
 $\text{retr}(b).abpurses(pid) = \text{retr}(a).abpurses(pid)$ ∀-elem-I(h4, 16, 25, 23)

Lemma at-mk-abpurse-defn

at-mk-AbPurse-defn $\frac{c: ConWorld; pid: PurseId; pid \in \mathbf{dom} c.conpurses; pid \in \mathbf{dom} \text{retr}(c).abpurses}{\text{retr}(c).abpurses(pid) = mk\text{-}AbPurse(c.conpurses(pid).bal,$
 $\text{sumval}(c.conpurses(pid).exlog))}$

```

from  $c: ConWorld; pid: PurseId; pid \in \mathbf{dom} c.conpurses; pid \in \mathbf{dom} retr(c).abpurses$ 
1   $retr(c) = mk\text{-}AbWorld(c.conauth,$ 
    $\{name \mapsto mk\text{-}AbPurse(c.conpurses(name).bal, sumval(c.conpurses(name).exlog))$ 
    $| name \in \mathbf{dom} c.conpurses\})$  retr-defn(h1)
2   $retr(c): AbWorld$  retr-form(h1)
3   $retr(c).abpurses: PurseId \xrightarrow{m} AbPurse$  abpurses-form(2)
4   $retr(c).abpurses = retr(c).abpurses$  =-self-I(3)
5   $retr(c).abpurses = mk\text{-}AbWorld(\dots).abpurses$  =-subs-right(a)(2, 1, 4)
6   $mk\text{-}AbWorld(\dots).abpurses: PurseId \xrightarrow{m} AbPurse$  =-type-inherit-right(3, 5)
7   $mk\text{-}AbWorld(\dots): AbWorld$  =-type-inherit-right(2, 1)
8   $mk\text{-}AbWorld(c.conauth, \{name \mapsto mk\text{-}AbPurse(c.conpurses(name).bal,$ 
    $sumval(c.conpurses(name).exlog)) | name \in \mathbf{dom} c.conpurses\}).abpurses =$ 
    $\{name \mapsto mk\text{-}AbPurse(c.conpurses(name).bal,$ 
    $sumval(c.conpurses(name).exlog)) | name \in \mathbf{dom} c.conpurses\}$  abpurses-defn(7)
9   $retr(c).abpurses = \{name \mapsto mk\text{-}AbPurse(c.conpurses(name).bal,$ 
    $sumval(c.conpurses(name).exlog)) | name \in \mathbf{dom} c.conpurses\}$  =-subs-left(b)(6, 5, 8)
10  $retr(c).abpurses(pid): AbPurse$  at-form(h2, 3, h4)
11  $retr(c).abpurses(pid) = retr(c).abpurses(pid)$  =-self-I(10)
12  $retr(c).abpurses(pid) = \{name \mapsto \dots\}(pid)$  =-subs-right(a)(3, 9, 11)
13  $\{name \mapsto \dots\}(pid): AbPurse$  =-type-inherit-right(10, 12)
14  $c.conpurses: PurseId \xrightarrow{m} ConPurse$  conpurses-form(h1)
15 dom  $c.conpurses: PurseId\text{-set}$  dom-form(14)
16 from  $name: PurseId; name \in \mathbf{dom} c.conpurses$ 
   infer  $mk\text{-}AbPurse(\dots): AbPurse$  ConWorld-AbWorld-form(16.h1, h1, 16.h2)
17  $\{name \mapsto mk\text{-}AbPurse(c.conpurses(name).bal,$ 
    $sumval(c.conpurses(name).exlog)) | name \in \mathbf{dom} c.conpurses\}(pid) =$ 
    $mk\text{-}AbPurse(c.conpurses(pid).bal,$ 
    $sumval(c.conpurses(pid).exlog))$  at-defn-map-comp-left-set(h2, 15, h3, 16)
infer  $retr(c).abpurses(pid) = mk\text{-}AbPurse(c.conpurses(pid).bal,$ 
    $sumval(c.conpurses(pid).exlog))$  =-subs-left(b)(13, 12, 17)

```

Lemma $\{a, b\}$ -form

$$\boxed{\{a, b\}\text{-form}} \frac{a: A; b: A}{\{a, b\}: A\text{-set}}$$

```

from  $a: A; b: A$ 
1.  $\{a, b\} = add(a, add(b, \{\}))$  \{a,b\}-defn(h1, h2)
2.  $\{\}: A\text{-set}$  \{\}-form
3.  $add(a, add(b, \{\})): A\text{-set}$  add-add-form(h1, h2, 3)
infer  $\{a, b\}: A\text{-set}$  =-type-inherit-left(3, 1)

```

Lemma \forall -elem-I

$$\boxed{\forall\text{-elem-I}} \frac{a: A; P(a); s: A\text{-Set}; \forall b \in s \setminus \{a\} \cdot P(b)}{\forall b \in s \cdot P(b)}$$

$$\begin{array}{l}
\text{from } a: A; P(a); s: A\text{-set}; \forall b \in s \setminus \{a\} \cdot P(b) \\
1. \quad \{a\}: A\text{-set} \quad \{a\}\text{-form}(h1) \\
2. \quad \text{from } y: A; y \in s \\
2.1 \quad y \in \{a\} \vee y \notin \{a\} \quad \in\text{-}\notin(2.h1, 1) \\
2.2 \quad \text{from } y \in \{a\} \\
2.2.1 \quad y = a \quad \in\text{-}\{a\}\text{-E}(h1, 2.h1, 2.2.h1) \\
\quad \text{infer } P(y) \quad =\text{-subs-left}(b)(h1, 2.2.1, h2) \\
2.3 \quad \text{from } y \notin \{a\} \\
2.3.1 \quad s \setminus \{a\}: A\text{-set} \quad \text{diff-form}(h3, 1) \\
2.3.2 \quad y \in s \setminus \{a\} \quad \in\text{-diff-I}(2.h1, h3, 1, 2.h2, 2.3.h1) \\
\quad \text{infer } P(y) \quad \forall\text{-E-set}(2.h1, 2.3.1, 2.3.2, h4) \\
\quad \text{infer } P(y) \quad \forall\text{-E}(2.1, 2.2, 2.3) \\
\text{infer } \forall b \in s \cdot P(b) \quad \forall\text{-I-set}(h3, 2)
\end{array}$$

Lemma post-abauth-=-

$$\boxed{\text{post-abauth-=-}} \frac{a: \text{ConWorld}; b: \text{ConWorld}; a.\text{conauth} = b.\text{conauth}}{\text{retr}(a).\text{abauth} = \text{retr}(b).\text{abauth}}$$

Proof

$$\begin{array}{l}
\text{from } a: \text{ConWorld}; b: \text{ConWorld}; a.\text{conauth} = b.\text{conauth} \\
1 \quad b.\text{conauth}: \text{PurseId-set} \quad \text{conauth-form}(h2) \\
2 \quad a.\text{conauth}: \text{PurseId-set} \quad \text{conauth-form}(h1) \\
3 \quad b.\text{conauth} = \text{retr}(b).\text{abauth} \quad \text{auth-}=(h2) \\
4 \quad a.\text{conauth} = \text{retr}(a).\text{abauth} \quad \text{auth-}=(h1) \\
5 \quad a.\text{conauth} = \text{retr}(b).\text{abauth} \quad =\text{-trans}(a)(2, h3, 3) \\
\text{infer } \text{retr}(a).\text{abauth} = \text{retr}(b).\text{abauth} \quad =\text{-trans-left}(a)(2, 4, 5)
\end{array}$$

Lemma auth-=-

$$\boxed{\text{auth-=-}} \frac{c: \text{ConWorld}}{c.\text{conauth} = \text{retr}(c).\text{abauth}}$$

```

from  $c: \text{ConWorld}$ 
1    $c.\text{conauth}: \text{PurseId-set}$                                 conauth-form(h1)
2    $\text{retr}(c): \text{AbWorld}$                                        retr-form(h1)
3    $\text{retr}(c) = \text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} c.\text{conpurses} \})$                                 retr-defn(h1)
4    $\text{retr}(c).\text{abauth}: \text{PurseId-set}$                                 abauth-form(2)
5    $\text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} c.\text{conpurses} \}): \text{AbWorld}$                     =-type-inherit-right(2, 3)
6    $\text{mk-AbWorld}(c.\text{conauth}, \{ \text{name} \mapsto$ 
       $\text{mk-AbPurse}(c.\text{conpurses}(\text{name}).\text{bal}, \text{sumval}(c.\text{conpurses}(\text{name}).\text{exlog}) \mid$ 
       $\text{name} \in \mathbf{dom} c.\text{conpurses} \}).\text{abauth} = c.\text{conauth}$         abauth-defn(5)
7    $\text{retr}(c).\text{abauth} = c.\text{conauth}$                                 =-subs-left(a)(2,3,6)
infer  $c.\text{conauth} = \text{retr}(c).\text{abauth}$                             =-symm(b)(1,7)

```

Lemma =-mk-abpurse

$$\boxed{\text{=-mk-AbPurse}} \frac{x: \mathbb{N}; y: \mathbb{N}; x = a; y = b}{\text{mk-AbPurse}(x, y) = \text{mk-AbPurse}(a, b)}$$

```

from  $x: \mathbb{N}; y: \mathbb{N}; x = a; y = b$ 
1    $\text{mk-AbPurse}(x, y): \text{AbPurse}$                                 mk-AbPurse-form(h1, h2)
2    $\text{mk-AbPurse}(x, y) = \text{mk-AbPurse}(x, y)$                         =-self-I(1)
3    $\text{mk-AbPurse}(x, y) = \text{mk-AbPurse}(a, y)$                     =-subs-right(a)(h1, h3, 2)
infer  $\text{mk-AbPurse}(x, y) = \text{mk-AbPurse}(a, b)$                     =-subs-right(a)(h2, h4, 3)

```

I Miscellaneous Proofs

I.1 Miscellaneous proofs in underlying theories

$$\boxed{\text{diff-not-eq}} \frac{a \in t; a: X; t: X\text{-set}}{\neg(t \setminus \{a\} = t)}$$

	from $a \in t; a: X; t: X\text{-set}$	
1	$a \in \{a\}$	$\in\text{-}\{a\}\text{-I}=(\text{h2})$
2	$\{a\}: X\text{-set}$	$\{a\}\text{-form}(\text{h2})$
3	$t \setminus \{a\}: X\text{-set}$	$\text{diff-form}(\text{h3,2})$
4	$\neg\neg(a \in \{a\})$	$\neg\neg\text{-I}(1)$
5	$\neg(a \notin \{a\})$	$\text{folding}(4)$
6	$\neg(a \in t \wedge a \notin \{a\})$	$\neg\wedge\text{-I-left}(5)$
7	$a \in t \setminus \{a\} \Leftrightarrow a \in t \wedge a \notin \{a\}$	$\in\text{-diff-defn}(\text{h2,h3,2})$
8	$\neg(a \in t \setminus \{a\})$	$\Leftrightarrow\text{-E-right-}\neg(7,6)$
9	$\neg(\forall x \in t \cdot x \in t \setminus \{a\})$	$\neg\forall\text{-I-}\neg(\text{h2,8})$
10	$t \subseteq t \setminus \{a\} \Leftrightarrow \forall x \in t \cdot x \in t \setminus \{a\}$	$\subseteq\text{-defn}(\text{h3,3})$
11	$\neg(t \subseteq t \setminus \{a\})$	$\Leftrightarrow\text{-E-right-}\neg(10,9)$
12	$\neg(t \subseteq t \setminus \{a\} \wedge t \setminus \{a\} \subseteq t)$	$\neg\wedge\text{-I-right}(11)$
13	$t = t \setminus \{a\} \Leftrightarrow t \subseteq t \setminus \{a\} \wedge t \setminus \{a\} \subseteq t$	$=\text{-set-defn}(\text{h3,3})$
	infer $\neg(t \setminus \{a\} = t)$	$\Leftrightarrow\text{-E-right-}\neg(13,12)$

$$\boxed{\text{complete-set-indn}} \frac{\begin{array}{l} s: A\text{-set}; \\ \exists a: A \cdot a \notin s; \\ \forall t: A\text{-set} \cdot \forall t': A\text{-set} \cdot t' \subset t \Rightarrow P(t') \end{array}}{P(s)}$$

	from $s: A\text{-set}; \exists a: A \cdot a \notin s;$	
	$\forall t: A\text{-set} \cdot \forall t': A\text{-set} \cdot t' \subset t \Rightarrow P(t')$	
1	from $a: A; a \notin s$	
1.1	$\text{add}(a, s): A\text{-set}$	$\text{add-form}(1.\text{h1,h1})$
1.2	$s \subset \text{add}(a, s)$	$\subset\text{-add}(1.\text{h1,h1,1.h2})$
1.3	$\forall t': A\text{-set} \cdot t' \subset \text{add}(a, s) \Rightarrow P(t')$	$\forall\text{-E}(1.1,\text{h3})$
1.4	$s \subset \text{add}(a, s) \Rightarrow P(s)$	$\forall\text{-E}(\text{h1,1.3})$
	infer $P(s)$	$\Rightarrow\text{-E-left}(1.4,1.2)$
	infer $P(s)$	$\exists\text{-E}(\text{h2,1})$

$$\boxed{\neq\text{-add}} \frac{a: A; s: A\text{-set}; a \notin s}{s \neq \text{add}(a, s)}$$

	from $a: A; s: A\text{-set}; a \notin s$	
1	$add(a, s): A\text{-set}$	add-form(h1,h2)
2	$a \in add(a, s)$	\in -add-I-elem(h1,h2)
3	$\exists x \in add(a, s) \cdot x \notin s$	\exists -I-set(h1,1,2,h3)
4	$\neg\neg(\exists x \in add(a, s) \cdot x \notin s)$	$\neg\neg$ -I(3)
5	$\neg(\forall x \in add(a, s) \cdot x \in s)$	folding(4)
6	$add(a, s) \subseteq s \Leftrightarrow \forall x \in add(a, s) \cdot x \in s$	\subseteq -defn(1,h2)
7	$\neg(add(a, s) \subseteq s)$	\Leftrightarrow -E-right- \neg (6,5)
8	$\neg(s \subseteq add(a, s) \wedge add(a, s) \subseteq s)$	\neg - \wedge -I-left(7)
9	$s = add(a, s) \Leftrightarrow (s \subseteq add(a, s) \wedge add(a, s) \subseteq s)$	$=$ -set-defn(h2,1)
	infer $s \neq add(a, s)$	\Leftrightarrow -E-right- \neg (9,8)

$$\boxed{\subseteq\text{-add}} \frac{a: A; s: A\text{-set}; a \notin s}{s \subset add(a, s)}$$

	from $a: A; s: A\text{-set}; a \notin s$	
1	$add(a, s): A\text{-set}$	add-form(h1, h2)
2	$s \subset add(a, s) \Leftrightarrow s \subseteq add(a, s) \wedge s \neq add(a, s)$	\subset -defn(h2, 1)
3	$s \subseteq add(a, s) \Leftrightarrow \forall x \in s \cdot x \in add(a, s)$	\subseteq -defn(h2, 1)
4	from $x: A; x \in s$	
4.1	$x = a \vee x \in s$	\vee -I-left(4.h2)
4.2	$x \in add(a, s) \Leftrightarrow x = a \vee x \in s$	\in -add-defn(4.h1, h1, h2)
	infer $x \in add(a, s)$	\Leftrightarrow -E-right(4.2, 4.1)
5	$\forall x \in s \cdot x \in add(a, s)$	\forall -I-set(h2, 4)
6	$s \subseteq add(a, s)$	\Leftrightarrow -E-right(3, 5)
7	$s \neq add(a, s)$	\neq -add(h1, h2, h3)
8	$s \subseteq add(a, s) \wedge s \neq add(a, s)$	\wedge -I(6, 7)
	infer $s \subset add(a, s)$	\Leftrightarrow -E-right(2, 8)

$$\boxed{\text{card}=0\text{-E}} \frac{s: A\text{-set}; \text{card } s = 0}{s = \{\}}$$

	from $s: A\text{-set}; \text{card } s = 0$	
1	$\{\}: A\text{-set}$	$\{\}$ -form
2	$\{\} = \{\}$	=-self-I(1)
3	card $\{\} = 0 \Rightarrow \{\} = \{\}$	\Rightarrow -I-left-vac(2)
4	from $a: A; s': A\text{-set}; \text{card } s' = 0 \Rightarrow s' = \{\}; a \notin s'$	
4.1	card $s': \mathbb{N}$	card -form(4.h2)
4.2	succ (card $s') \neq 0$	succ $\neq 0$ (4.1)
4.3	card $\text{add}(a, s') = \text{succ}(\text{card } s')$	card-defn-add(4.h1, 4.h2, 4.h4)
4.4	succ (card $s'): \mathbb{N}$	succ -form(4.1)
4.5	card $\text{add}(a, s') \neq 0$	=-subs-left(b)(4.4, 4.3, 4.2)
4.6	$\neg(\text{card } \text{add}(a, s') = 0)$	unfolding(4.5)
	infer card $\text{add}(a, s') = 0 \Rightarrow \text{add}(a, s') = \{\}$	\Rightarrow -I-right-vac(4.6)
5	card $s = 0 \Rightarrow s = \{\}$	set-indn(h1, 3, 4)
	infer $s = \{\}$	\Rightarrow -E-left(5, h2)