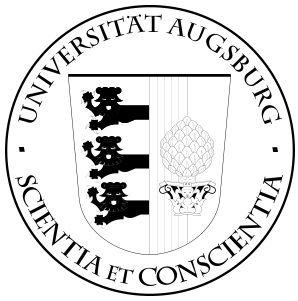
Universität Augsburg



The Mondex Challenge: Machine Checked Proofs for an Electronic Purse

G.Schellhorn, H.Grandy, D.Haneberg, W.Reif

Report 2006-02 February 2006

Institut für Informatik
D-86135 Augsburg

The Mondex Challenge: Machine Checked Proofs for an Electronic Purse

Gerhard Schellhorn, Holger Grandy, Dominik Haneberg, and Wolfgang Reif

Lehrstuhl für Softwaretechnik und Programmiersprachen,
Universität Augsburg, D-86135 Augsburg, Germany
{schellhorn,grandy,haneberg,reif}@informatik.uni-augsburg.de

Abstract. The Mondex case study about the specification and refinement of an electronic purse as defined in [SCJ00] has recently been proposed as a challenge for formal system-supported verification. This paper reports on the successful verification of the major part of the case study using the KIV specification and verification system. We demonstrate that even though the hand-made proofs were elaborated to an enormous level of detail we still could find small errors in the underlying data refinement theory as well as the formal proofs of the case study.
We also provide an alternative formalisation of the communication protocol using abstract state machines.
Finally the Mondex case study verifies functional correctness assuming a suitable security protocol. Therefore we propose to extend the case study to include the verification of a suitable security protocol.

1 Introduction

Mondex smart cards implement an electronic purse [MCI]. They have become famous for having been the target of the first ITSEC evaluation of the highest level E6 [CB99], which requires formal specification and verification.

The formal specification and proofs were done in [SCJ00] using the Z specification language [Spi92]. Two models of electronic purses were defined: an abstract one which models the transfer of money between purses as elementary transactions, and a concrete level that implements money transfer using a communication protocol that can cope with lost messages using a suitable logging of failed transfers. A suitable data refinement theory was developed in [CSW02].

Although the refinement proofs based on this theory were done manually (with an auxiliary type checker) they were elaborated to the details of almost calculus level. The Mondex case study has been recently proposed as a challenge for theorem provers [Woo06].

In this paper we show that verifying the refinement mechanically using the KIV theorem prover can be done within a few weeks of work. We verify the full Mondex case study except for the operations that archive failure logs from a smart card to a central archive. These are orthogonal to the protocol for money transfer.

The proofs of the Mondex case study are too big to be presented completely within a technical report ([SCJ00] has 240 pages, [CSW02] additional 54 pages, the proof details of the mechanized proofs which are even more detailed could easily fill hundreds of pages. Therefore, we will often refer to [SCJ00] for comparison. To view the details of the KIV proofs we have prepared and a Web presentation of the full KIV specifications and of all proofs, which can be found at [KIV]. The interested reader can find all details there.

Nevertheless we have tried to extract the core of the refinement problem and to give a concise definition of the case study in section 2. To this purpose we introduce the case study using abstract state machines (ASM, [Gur95], [BS03]). Since the relational approach of Z is quite different from the operational description of ASMs this paper can also be used to compare the two specification styles. To check the adequacy of the ASM formalization we have also verified the central proof obligations of [SCJ00]: backward simulation and an invariant for the concrete level. We discuss these proofs, their structure and the differences to the original proof in section 3. Doing them was sufficient to uncover small problems in the invariant of the concrete level.

While the proofs could be elaborated to a full ASM refinement proof which would be our traditional verification approach ([BR95], [Sch01], [Bör03]), we decided to mimic the data refinement proofs faithfully to succeed in verifying the challenge. Therefore we formalized the underlying data refinement theory. We report on a correction for this theory and an extension using invariants in section 4.

Finally we instantiated the data refinement theory with the operations of the Mondex case study. Our proofs improve the ones of [SCJ00] by using one refinement instead of two. Section 5 also reports on the additional complexity caused by using operations similar to Z instead of a simple ASM, and gives some statistics of the effort required.

When we discovered the Mondex case study, we were astonished to find that it has been given the highest security level ITSEC E6, when in fact the case study *assumes* a suitable security protocol rather than proving it. Since the real security protocol of Mondex smart cards has never been published, we discuss a probable security protocol in section 6 and propose a refinement of the concrete Mondex level to a specification, that includes such a security protocol as an extension of the case study.

Finally, we conclude in section 7.

2 Two Simple ASMs for the Mondex Case Study

The Mondex case study is based on smart cards that are being used as electronic purses. Each card has a balance and may be used to transfer money to other cards. Unfortunately it is very hard to get a clear picture of their use in real life. The original web site [MCI] says that the smart cards are used to transfer money over the internet using a card reader on each end. [RE03] says one card reader is used, the 'from' purse (where money is taken from) is first put in the card

reader, then the 'to' purse (which receives the money). This seems not really compatible with the protocol given later on. Finally, the Mondex paper [SCJ00] and the ITSEC evaluation [CB99] suggest an interface device, which seems to be a card reader with two slots, where both cards can be inserted simultaneously. It is also not clear how cryptography is used, [RSA04,CCW96] suggest, that this was never disclosed, and that the Mondex card therefore is a classical example of "security by obscurity". Maybe this is the reason why a security protocol is not considered in the Mondex case study.

The smart cards of the formal specification are specified on two levels: An abstract level which defines transfer of money between purses as an atomic transaction, and a concrete level which defines a protocol to transfer money.

In this section we now give an alternative version of the Mondex refinement problem using abstract state machines (ASMs, [Gur95], [BS03]) and algebraic specifications as used in KIV [RSSB98].

The abstract state machines can also be found on the Web [KIV] in the *Mondex* project as *simple-AASM* and *simple-BASM*. We have tried to stay as close as possible to the notation of the original Mondex case study, but we have removed all details that we thought were not essential to understand the problem described by the Mondex refinement.

2.1 The Abstract Level

The abstract specification of a purse consists of a function balance from purse names to their current balance. Since the transfer of money from one to another purse may fail (due to the card being pulled abruptly from the card reader, or for internal reasons like lack of memory) the state of an abstract purse also must log the amount of money that has been lost in such failed transfers.

In the formalism of ASMs this means that the abstract state consists of two dynamic functions

$$\mathsf{balance} : \mathsf{name} \to \mathbb{N}$$
$$\mathsf{lost} : \mathsf{name} \to \mathbb{N}$$

Purses may be faked, so we have a finite number of names which satisfy a predicate authentic[1]. How authenticity is checked (using secret keys, pins etc.) is left open on both levels of the specification, so the predicate is simply left unspecified. We will come back to this point in section 6.

Transfer of money between authentic purses is done with the following simple ASM rule[2]

[1] In the original Z specification, authentic is defined to be the domain of partial AbAuthPurse and ConAuthPurse functions. For simplicity, we use total functions instead, and use authentic to restrict their domain.

[2] By convention our rule names end with a # sign to distinguish them from predicates.

ABTRANSFER#
choose fail?, value, from, to
with authentic(from) $\wedge$ authentic(to) $\wedge$ from $\neq$ to $\wedge$ value $\leq$ balance(from)
in if $\neg$ fail?
  then balance(from) := balance(from) $-$ value
    balance(to) := balance(to) $+$ value
  else balance(from) := balance(from) $-$ value
    lost(from) := lost(from) $+$ value

The rule nondeterministically chooses two different, authentic purses with names from and to, and an amount value for which the from purse has enough money and transfers it. The transfer may fail for internal reasons as indicated by the randomly chosen boolean variable fail?. In this case the from purse logs the lost money in its lost component.

This already completes the specification of the abstract level. Compared to the Z specification in [SCJ00] we have left out the operation ABIGNORE# which skips (i.e. does nothing): In data refinement such a skip operation is needed, since every operation must be refined by a 1:1 diagram. ASM refinement directly allows to use 0:1 diagrams, therefore such a skip operation is not needed.

2.2 The Concrete Level

On the concrete level transferring money is done using a protocol with 5 steps. To execute the protocol, each purse needs a status that indicates how far it has progressed executing the protocol. The possible states a purse may be in are given by the enumeration status = idle | epr | epv | epa. Compared to [SCJ00] we have merged the two states eaFrom and eaTo into one idle state. The behavior of a purse in eaTo state is exactly the same as that of a purse in eaFrom state, so we saw no necessity to distinguish them.

Purses not participating in any transfer are in the idle state. To avoid replay attacks each purse stores a sequence number nextSeqNo that can be used in the next transaction. This number is incremented during any run of the protocol. During the run of the protocol each purse stores the current payment details in a variable pdAuth of type PayDetails. These are tuples consisting of the names of the from and to purse, the transaction numbers these use for this transaction and the amount of money that is transferred. In KIV we define a free data type
PayDetails =
  mkpd(. .from : name; . .fromno : nat; . .to : name; . .tono : nat; . .value : nat)
with postfix selectors (so pd.from is the name of the from purse stored in payment details pd). The state of a purse finally contains a log exLog of failed transfers represented by their payment details. The protocol is executed sending messages between the purses. The ether collects all messages that are currently available. A purse receives a message by selecting a message from the ether. Since the environment of the card is assumed to be hostile the message received may be any message that has already been sent, not just one that is directed to the card (this simple model of available messages is also used in many abstract specifications of security protocols, e.g. the traces of [Pau98]). The state of the concrete ASM therefore is:

balance : name → ℕ
state : name → status
pdAuth : name → PayDetails
exLog : name → set(PayDetails)
ether : set(message)

The protocol is started by two messages startFrom(msgna, value, msgno) and startTo(msgna, value, msgno) which are sent to the from and to purse respectively by the interface device. These two messages are assumed to be always available, so the initial ether already contains every such message. The arguments msgna and msgno of startFrom(msgna, value, msgno) are assumed to be the name and nextSeqNo of the to purse, value is the amount of value transfered. Similarly, for startTo(msgna, value, msgno) msgna and msgno are the corresponding data of the from purse.

On receiving a startFrom message msg from ether in the idle state a purse named receiver[3] executes the following step:

```
STARTFROM#
let msgna = msg.name, value = msg.value, msgno = msg.nextSeqNo
in if     authentic(msgna) ∧ ¬ fail? ∧ receiver ≠ msgna
      ∧ value ≤ balance(receiver)
   then choose n with nextSeqNo(receiver) < n in
          pdAuth(receiver) := mkpd(receiver, nextSeqNo(receiver),
                                   msgna, msgno, value)
          status(receiver) := epr
          nextSeqNo(receiver) := n
          outmsg := ⊥
   else outmsg := ⊥
```

If the purse msgna which shall receive money is not authentic, the receiver purse has not enough money or the transition fails due to internal reasons (a flag fail? is used for this purpose just as on the abstract level), then the purse simply produces an empty output message ⊥ and does nothing else. Otherwise the purse stores the requested transfer in its pdAuth component, using its current nextSeqNo number as one component and proceeds to the epr state ("expecting request"). Thereby it becomes the from purse of the current transaction. nextSeqNo is incremented to make it unavailable in further transactions. An empty output message ⊥ is generated in the success case too that will be added to the ether (see the full ASM rule below).

The action for a purse receiving a startTo message in idle state is similar except that it goes into the epv state ("expecting value") and becomes the to purse of the transaction. Additionally it sends a request message to the from purse:

[3] receiver is always a purse receiving a message. This can be a from purse sending money as well as a to purse receiving money and should not be confused with the latter.

```
STARTTO#
let msgna = msg.name, value = msg.value, msgno = msg.nextSeqNo
in if authentic(msgna) ∧ ¬ fail ∧ receiver ≠ msgna
   then choose n with nextSeqNo(receiver) < n in
          pdAuth(receiver) := mkpd(msgna, msgno, receiver,
                                   nextSeqNo(receiver), value)
          state(receiver) := epv seq
          outmsg := req(pdAuth(receiver))
          nextSeqNo(receiver) := n
   else outmsg := ⊥
```

The request message req(pdAuth(receiver)) contains the payment details of the current transaction. Although this is not modeled, the message is assumed to be securely encrypted. Since an attacker can therefore never guess this message before it is sent, it is assumed that the initial ether does not contain any request message. When the from purse receives the request in state epr it executes

```
REQ#
if msg = req(pdAuth(receiver)) ∧ ¬ fail?
then balance(receiver) := balance(receiver) − pdAuth(receiver).value
       state(receiver) := epa
       outmsg := val(pdAuth(receiver))
else outmsg := ⊥
```

The message is checked to be consistent with the current transaction stored in pdAuth and if this is the case the money is sent with an encrypted value message val(pdAuth(receiver)). The state changes to epa ("expecting acknowledge"). On receiving the value the to purse does

```
VAL#
if msg = val(pdAuth(receiver)) ∧ ¬ fail?
then balance(receiver) := balance(receiver) + pdAuth(receiver).value
       state(receiver) := idle
       outmsg := ack(pdAuth(receiver))
else outmsg := ⊥
```

It adds the money to its balance, sends an encrypted acknowledge message back and finishes the transaction by going back to state idle. When this acknowledge message is received the from purse finishes similarly:

```
ACK#
if msg = ack(pdAuth(receiver)) ∧ ¬ fail?
then state(receiver) := idle
       outmsg := ⊥
else outmsg := ⊥
```

To put the steps together it finally remains to define the full ASM rule BOP#[4] which executes all the steps above:

[4] BOP# is called BSTEP# in the web presentation.

```
BOP#
choose msg, fail?, receiver with msg ∈ ether ∧ authentic(receiver) in
  if isStartTo(msg) ∧ state(receiver) = idle then STARTTO#
  else if isStartFrom(msg) ∧ state(receiver) = idle then STARTFROM#
  else if isreq(msg) ∧ state(receiver) = epr then REQ#
  else if isval(msg) ∧ state(receiver) = epv then VAL#
  else if isack(msg) ∧ state(receiver) = epa then ACK#
  else ABORT#
  seq ether := ether ++ outmsg
```

The ASM rule chooses an authentic receiver for some message msg from ether. Like in the abstract ASM the fail? flag indicates failure due to internal reasons. At the end of the rule the produced outmsg is added to the set ether of available messages. Therefore our ASM corresponds to the "between" level as defined in [SCJ00]. For the concrete level the ether is assumed to lose messages randomly (due to an attacker or technical reasons like power failure). Therefore an ASM that models the concrete level replaces the last assignment with

```
LOSEMSG#
choose ether′ with ether′ ⊆ ether ++ outmsg in ether := ether′
```

If a purse is sent an illegal message $\perp$ or a message for which it is not in the correct state the current transaction is aborted by

```
ABORT#
choose n with nextSeqNo(receiver) ≤ n in
  LOGIFNEEDED#
  state(receiver) := idle
  nextSeqNo(receiver) := n
  outmsg := ⊥

LOGIFNEEDED#
if state(receiver) = epa ∨ state(receiver) = epv
then exLog(receiver) := exLog(receiver) ++ pdAuth(receiver)
```

This action logs if money is lost due to aborting a transaction. The idea is that the lost money of the abstract level can be recovered by comparing the two logs of the from and to purse involved. Logging takes place if either the purse is a to purse in the critical state epv or a from purse in critical state epa.

This completes the description of the concrete level. It remains to summarize the free data type of messages:

```
message =
      startFrom(. .name : name; ..value : nat; . .nextSeqNo : nat; ) with isStartFrom
    | startTo(. .name : name; . .value : nat; . .nextSeqNo : nat; ) with isStartTo
    | req(. .pd : PayDetails)withisreq
    | val(. .pd : PayDetails)withisval
    | ack(. .pd : PayDetails)withisack
    | ⊥
```

Compared to the Z specification we have done several simplifications and modifications:

– We have removed the global input. Since input must be from ether anyway, it can be chosen from there directly. Of course we lose the correspondence of inputs, but this does not really seem relevant.
– STARTFROM# and STARTTO# are prefixed with ABORT# in the original specification: this is not really necessary since the ASM can execute an ABORT# at any time by choosing an input $\perp$ from ether. This simplification is also motivated by the fact that invariance and backward simulation proofs in [SCJ00] use the same lemma for ABORT# regardless whether it is used as prefix of STARTFROM# or separately.
– STARTFROM#, STARTTO# execute an ABORT# in the else case: same reason as for the previous simplification.
– All operations are in a disjunct with an IGNORE# operations in the original specification, which does nothing. This seems necessary only since the Z specification requires all operations to be total. While the ASM rule, which essentially is equivalent to the disjunction of all Z operations is total too, it is not necessary that each individual action is always applicable. If not, the ASM simply chooses another case. To be fully equivalent we could add a case that ignores the input message completely.
– The duplicate use of names as elements of the domain of ConAuthPurse as well as as components of purses has been avoided.
– The ASM specification is defined using operational style rules instead of relational style. From our point of view this is simpler: We have avoided to use promotion to lift specifications of one purse to the specification of several purses. We also avoided the numerous Z schemas which are necessary to say that some variables do *not* change in an operation (frame problem).

3 Verification of Backward Simulation and Invariance for the ASMs

The ASMs of the previous section were not intended to be a 1:1 representation of the original Z operations. Rather they were intended as a concise description of the essential refinement problem contained in the case study. To check this we tried to prove the main theorems of the Mondex case study for these ASMs, namely

– The concrete ASM preserves the invariant BINV, that is used to restrict the "concrete" state to the "between" state ([SCJ00], sections 28-29).
– The concrete ASM satisfies the correctness condition of backward refinement using a backward simulation ABINV ([SCJ00], sections 14-20).

This section reports on the results. The first thing we had to do is to extract the properties of the invariants from the Z specification. We found that they are distributed in 3 places:

– The property of payment details that requires pd.from $\neq$ pd.to for every relevant pd used (Section 4.3.2).

– The properties of purses P-1 to P-4 (section 4.6).
– The properties B-1 to B-16 of the intermediate state that define an invariant for the concrete state (section 5.3).

Collecting these properties and the required definitions of AuxWorld (section 5.2) results in the specification for BINV given in Appendix 8. These definitions can also be found in the specification *BINV* in project *Mondex* on the web [KIV].

Compared to the original definitions there are the following minor differences:

– The properties have been grouped into predicates lifting out the universal quantifiers. This allows to define rewrite rules for the predicates (to avoid unfolding altogether) and to instantiate the universal quantifier of several properties at once.
– The properties of state eaTo have been dropped altogether since this state has been merged with idle.
– The predicates are not defined in the context of a Z schema. We have to provide parameters explicitly.

The main interesting modification is that we had to strengthen properties P-3 and P-4. We found that although the proofs of [SCJ00] are very detailed they *still* contain minor flaws. The problems were detected when the proof for invariance theorem *BINV* failed. This theorem is written using Dynamic Logic [HKT00] and proved in KIV using sequent calculus:

$$\mathsf{BINV}(\underline{\mathsf{cs}}) \vdash \langle\!|\mathsf{BOP}\#(;\ \underline{\mathsf{cs}})|\!\rangle\ \mathsf{BINV}(\underline{\mathsf{cs}})$$

$\vdash$ is the sequent arrow (semantics: the conjunction of antecedent formulas before the sequent arrow implies the disjunction of succedent formulas after the sequent arrow). $\underline{\mathsf{cs}}$ is the vector of variables that denote the concrete state, i.e. $\underline{\mathsf{cs}} = \mathsf{balance}, \mathsf{status}, \mathsf{nextSeqNo}, \mathsf{pdAuth}, \mathsf{ether}$. $\langle\!|\mathsf{BOP}\#(;\ \underline{\mathsf{cs}})|\!\rangle\ \mathsf{BINV}(\underline{\mathsf{cs}})$ states that all runs of BOP# terminate in a state where BINV holds again. Using wp-calculus this is $\mathsf{wp}(\mathsf{BOP}\#(;\ \mathsf{cs}), \mathsf{BINV}(\underline{\mathsf{cs}}))$. Using relational calculus and a relation BOP the property would be equivalent to

$$\forall\ \underline{\mathsf{cs}}.\ \mathsf{BINV}(\underline{\mathsf{cs}}) \rightarrow \underline{\mathsf{cs}} \in \mathsf{dom}(\mathsf{BOP}) \wedge \forall\ \underline{\mathsf{cs}'}.\ \mathsf{BOP}(\underline{\mathsf{cs}}, \underline{\mathsf{cs}'}) \rightarrow \mathsf{BINV}(\underline{\mathsf{cs}'})$$

The first proof for the invariance theorem used the original properties P-3 and P-4. Specification *BINV-orig* on [KIV] contains a failed proof attempt. Its first open premise is one of the subgoals for proving invariance of etherok in the proof of VAL#. The case can be traced back to the original Mondex paper. The problem is in section 29.4 in the proof of B-10 where it must be proved that $\mathsf{toInEpv} \vee \mathsf{toLogged} \Rightarrow \mathsf{req} \wedge \neg\ \mathsf{ack}$ for every payment details pd. Now the problem is as follows: the implication *can* be proved for pdAuth(receiver), where receiver is the (to) purse receiving the val message (to which it responds with an ack message). But this is not sufficient: if it would be possible that receiver is

different from na := pdAuth(receiver).to but has status(na) = epv and pdAuth(na) = pdAuth(receiver), then for *this* na the implication would be violated.

The open premise of the KIV proof shows exactly this situation[5].

$$\begin{array}{l}\mathsf{state}(\mathsf{receiver}) = \mathsf{epv}, \mathsf{state}(\mathsf{pdAuth}(\mathsf{receiver}).\mathsf{to}) = \mathsf{epv},\\ \mathsf{pdAuth}(\mathsf{receiver}) = \mathsf{pdAuth}(\mathsf{pdAuth}(\mathsf{receiver}).\mathsf{to}),\\ \mathsf{authentic}(\mathsf{pdAuth}(\mathsf{receiver}).\mathsf{from}), \mathsf{authentic}(\mathsf{receiver}),\\ \mathsf{val}(\mathsf{pdAuth}(\mathsf{receiver})) \in \mathsf{ether}, \mathsf{receiver} \neq \mathsf{pdAuth}(\mathsf{receiver}).\mathsf{to},\\ \mathsf{ack}(\mathsf{pdAuth}(\mathsf{receiver})) \notin \mathsf{ether}, \ldots \vdash\end{array}$$

The solution to this problem is obvious: add pdAuth(receiver).to = receiver when status(receiver) = epv to P-3.

A similar problem also exists for status(receiver) = epa (property P-4) where pdAuth(receiver).from = receiver has to be known (second open goal in the proof). Finally, we had to add the fact that every val(pd) message in the ether has authentic(pd.from). Like property authentic(pd.to) (B-1) is needed to make the application of partial function ConAuthPurse to pd.to defined in B-2, this property is needed in order to have a determined value for ConAuthPurse pd.from in B-3 (the proof of *BINV* in *BINV-orig* already has this property added).

We also added the requirement that pdAuth(receiver).to resp. .from must be authentic to P-3 and P-4. In early proof attempts this seemed necessary since these lacked the authentic clauses in the definition of the predicates toInEpr, toInEpv and toInEpa. After adding such clauses this addition to P-3 and P-4 may be redundant.

We did not copy the elaborated proof structure of [SCJ00], which uses a lot of lemmata (some of which require the state eaTo to be present). Rather the standard heuristics and automation features of KIV (simplifier rules and problem specific patterns to guide the proof search) as described in [RSSB98] were sufficient for the proof. Nevertheless, in some situations where it was not clear why our proof got stuck it was helpful to cross-check details in the original proofs. The resulting proof structure is shown in Fig. 3.

1. The proof starts with goal (1) and symbolic execution of BOP#. This builds up a proof tree of sequent calculus (with the root being the initial goal at the bottom, expanding leaves to give simpler and simpler goals). We get a number of subgoals of the form (2), one for each case as defined by the control structure of the operation. Γ_0 contains the information about the current case. $\underline{cs'}$ contains the appropriately modified state after execution of BOP#.
2. Now both definitions of BINV are unfolded, and after splitting cases we get 4 goals of the form (3), one for each predicate p_j = purseok, etherok, exLogok, logsfin. If the relevant arguments of these predicates are identical the goal is closed immediately. The goal for logsfin is always closed immediately using some rewrite rules.

[5] for better readability we only show some of the formulas and moved formulas in the succedent to the antecedent negating them.

3. For the remaining goals both definitions of p are unfolded. These definition all have the form $\mathsf{p}(\underline{\mathsf{x}}) \leftrightarrow \forall\, \mathsf{y}.\varphi(\underline{\mathsf{x}}, \mathsf{y})$ (where y is either some payment details or the name of a purse). In the succedent the quantifier becomes a new variable, the quantifier on the left is instantiated with the same variable. Since another instance of this quantifier is never needed it is discarded immediately, resulting in goal (4). Γ_1 contains the remaining predicates $\mathsf{p_i}$.
4. Up to this point the proof has been completely automatic and after some case splits we basically arrive at the level, where single properties have to be proved, comparable to the goals in sections 16–29 of [SCJ00].
5. Many of the goals are now proven automatically using simple rewrite rules[6] like

$$\begin{array}{l} \quad \mathsf{fromInEpa}(\mathsf{pd}, \mathsf{pdAuth}[\mathsf{receiver};\ \mathsf{pd_0}], \mathsf{state}[\mathsf{receiver};\ \mathsf{stat}]) \\ \leftrightarrow (\quad \mathsf{pd.from} = \mathsf{receiver} \wedge \mathsf{authentic}(\mathsf{receiver}) \\ \quad\ \supset \mathsf{stat} = \mathsf{epa} \wedge \mathsf{pd_0} = \mathsf{pd};\ \mathsf{fromInEpa}(\mathsf{pd}, \mathsf{pdAuth}, \mathsf{state})) \end{array}$$

   Some however need some additional interactions to unfold some of the other predicates from Γ_1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi_1(\underline{\mathsf{cs}}, \mathsf{y}), \varphi_2(\underline{\mathsf{cs}}, \mathsf{y}), \ldots, \varphi_k(\underline{\mathsf{cs}}, \mathsf{y}), \Gamma_0, \Gamma_1 \vdash \varphi_1(\underline{\mathsf{cs}'}, \mathsf{y}) \wedge \varphi_2(\underline{\mathsf{cs}'}, \mathsf{y}) \wedge \ \ldots\ \wedge \varphi_k(\underline{\mathsf{cs}'}, \mathsf{y})}}{\varphi(\underline{\mathsf{cs}}, \mathsf{y}), \Gamma_0, \Gamma_1 \vdash \varphi(\underline{\mathsf{cs}'}, \mathsf{y})}\ (4)}{\forall\, \mathsf{y}'.\varphi(\underline{\mathsf{cs}}, \mathsf{y}'), \Gamma_0, \Gamma_1 \vdash \varphi(\underline{\mathsf{cs}'}, \mathsf{y})}}{\mathsf{p_1}(\underline{\mathsf{cs}}), \ldots, \mathsf{p_i}(\underline{\mathsf{cs}}), \Gamma_0 \vdash \mathsf{p_j}(\underline{\mathsf{cs}'})}\ (3)}{\mathsf{p_1}(\underline{\mathsf{cs}}), \ldots, \mathsf{p_i}(\underline{\mathsf{cs}}), \Gamma_0 \vdash \mathsf{p_1}(\underline{\mathsf{cs}'}) \wedge \ \ldots\ \wedge \mathsf{p_i}(\underline{\mathsf{cs}'})}}{\mathsf{BINV}(\underline{\mathsf{cs}}), \Gamma_0 \vdash \mathsf{BINV}(\underline{\mathsf{cs}'})}\ (2)}{\mathsf{BINV}(\underline{\mathsf{cs}}) \vdash \langle\!|\mathsf{BOP}\#(;\ \underline{\mathsf{cs}})|\!\rangle\ \mathsf{BINV}(\underline{\mathsf{cs}})}\ (1)$$

Fig. 1. Proof structure for BINV

After this proof we verified the backward simulation condition:

[6] $\varphi \supset \psi;\ \chi$ is if-then-else on formula level and $\mathsf{f}[\underline{\mathsf{x}};\ \mathsf{y}]$ modification of a dynamic function (i.e. a function variable). They can be viewed as abbreviations for $\varphi \wedge \psi \vee \neg\, \varphi \wedge \chi$ and $\lambda\, \underline{\mathsf{z}}.\ \underline{\mathsf{x}} = \underline{\mathsf{z}} \supset \mathsf{y};\ \mathsf{f}(\underline{\mathsf{x}})$

$$\mathsf{ABINV}(\underline{\mathsf{as}'}, \underline{\mathsf{cs}'}), \mathsf{BINV}(\underline{\mathsf{cs}}), \langle \mathsf{BOP}\#(\underline{\mathsf{cs}}) \rangle\ \underline{\mathsf{cs}} = \underline{\mathsf{cs}'}$$
$$\vdash \exists\ \underline{\mathsf{as}}.\mathsf{ABINV}(\underline{\mathsf{as}}, \underline{\mathsf{cs}}) \wedge (\langle \mathsf{AOP}\#(\mathsf{as}) \rangle\ \mathsf{as} = \mathsf{as}' \vee \mathsf{as} = \mathsf{as}')$$

ABINV(as, cs) is the backward simulation defined in Appendix 8. The definition is basically identical to the simulation relation defined in [SCJ00]. The only technical difference is, that we modified the maybeLost predicate (and similarly the definitelyLost predicate) to ensure that the payment details which satisfy it are always a *finite* set. This set is called maybelost with lower case letters in our maybeLost predicate.

The meaning of ⟨BOP#(cs)⟩ cs = cs' is that BOP# called with cs terminates and yields cs′. This is equivalent to COP(cs, cs′). The proof obligation for ASM refinement allows a 1:1 diagram, where the concrete rule BOP# operation refines an abstract operation AOP# as well as a 0:1 diagram, where the concrete operation refines skip (second disjunct).

The proof structure of this proof is very similar to the invariant proof above, with ABINV taking the role of BINV. BOP# is now symbolically executed in the antecedent instead of the succedent (we assume termination and a suitable execution of BOP# here, instead of proving termination for every execution). There are two differences: After symbolic execution of BOP# we first have to instantiate the existential quantifier in the succedent, i.e. we have to find suitable values for as = balance, lost. In most cases BOP# does not transfer money, so we simply choose as′. The only exceptions are REQ# and the case of ABORT#, where logging takes place. The second difference is that while unfolding BINV in the invariance proof simply gives a conjunction of properties, we now get a quantified formula over the sets maybelost, chosenlost and definitelylost. All three sets denote a set of payment details of transactions which are in progress: maybelost contains those transactions which may still succeed. chosenlost is the subset that will fail to transfer money in the rest of the run (since the simulation proceeds backwards this set is known). definitelylost are transactions where one party has already aborted during the critical phase of the protocol when the value message is in transit. How these sets change (backwards!) through each transition is the core of the proof and we have to give appropriate sets to instantiate the quantifier interactively. Unfortunately how the sets change is distributed in the proof structure of [SCJ00], so we summarize it here:

- None of the three sets changes in STARTFROM# and STARTTO# since aborting directly after these transitions will not lose money. Dually, the sets remain unchanged in ACK#, since the money has already been transfered successfully.
- When REQ#, VAL# or ACK# receive illegal input (the negative case of their conditional is executed) nothing changes.
- When REQ# executes successfully when receiving a request message, there are two cases to consider: either the to purse who sent the request is still in state epv: then the payment details of the request message enter maybelost and possibly chosenlost. Reasoning backwards the payment details must be deleted from both sets. Otherwise the to purse has already aborted the transaction after sending the request, so reasoning backwards the payment details

must be deleted from definitelylost. The two cases correspond to the two cases of successful and failed transaction in AOP# (all other operations refine skip).
- Successful execution of VAL# means that the payment details of the message leave maybelost, so reasoning backwards they must be added to maybelost.
- For ABORT# there are three cases to consider: First, if the purse does not log (in LOGIFNEEDED#), then no critical transaction is in progress and all sets remain unchanged. Second, if the purse logs and is in state epv, and if the corresponding from purse is either in state epa or has already logged the payment details, then aborting moves the payment details from maybelost and chosenlost to definitelylost. Reasoning backwards the payment details must be added to maybelost and chosenlost, and deleted from definitelylost. In the remaining third case the payment details are already in definitelylost and all sets remain unchanged.

The proof for the simulation condition has 655 proof steps and 197 interactions. Compared to the invariance proof, which has 447 proof steps with 71 interactions it is somewhat harder to automate, since the etherok property is needed many times, often indirectly: From the fact that e.g. the to purse is state epv conclude (with etherok) that some message is in ether and from this, conclude (using etherok again) that the from purse has some property. We have not tried, but it might be possible to get a better automation by improving the structuring of these properties.

The proofs can be found in project *Mondex* in the web presentation [KIV]. Specification *BINV* contains the proof for invariance (theorem *BINV*), specification *Mondex-ASM-refine* contains the proof for the simulation condition (theorem *correctness*).

4 Specifying the Data Refinement Theory

The data refinement theory underlying the Mondex case study is defined in [CSW02] in 3 steps: first, the general data refinement theory of [HHS86] is given (section 2). Second the contract embedding [WD96] of partial relations is defined and corresponding proof rules for forward and backward simulation are derived (section 3). Third the embedding of input and output into the state is discussed in Sect. 4.

We have formalised the first two parts of the theory already for [Sch05]. The corresponding algebraic specifications in KIV are available in the project named *DataRef* on the Web [KIV]. For the third part we have done the same in theory *Z-refinement*.

The central specification construct used in these projects (apart from standard constructs like enrichment, union, actualisation and renaming as present in all algebraic specification languages with lose semantics, e.g. CASL [CoF04]) is specification *instantiation*. Instantiation is an extended version of actualisation. Morphisms (called *mappings*) are used which which allow to map abstract sorts

like state to a tuple of dynamic functions (or variables) forming the concrete state. Mappings also allow to instantiate functions with arbitrary expressions. Another extension not used here allows to map types to subtypes or quotient types, similar to what is allowed in the modules defined in [Rei92]. Specification instantiation is very similar to theory interpretation in the IMPS system [Far94].

A specification SPEC = **instantiate** PSPEC < GSPEC **with** ASPEC **by** σ instantiates a subspecification PSPEC (the parameter) of a generic specification GSPEC with an actual specification ASPEC using a mapping σ. To prove correctness of the instantiation the instantiated axioms $\sigma(\mathsf{Ax}(\mathsf{PSPEC}))$ of PSPEC have to be proved over ASPEC. The resulting theory is $\sigma(\mathsf{GSPEC})$ with $\sigma(\mathsf{PSPEC})$ replaced by ASPEC. Therefore all theorems of ASPEC may be used. Instantiated specifications are used for 5 purposes:

- in project *DataRef* to prove that forward simulation implies refinement. Here PSPEC is the union of two theories ADT and CDT defining two data types used in the refinement. GSPEC extends this theory by the axiom stating that this is a correct refinement. ASPEC also extends PSPEC by axioms stating the conditions of forward simulation. SPEC uses the identity mapping, so it has to be proved that existence of a forward simulation implies refinement correctness.
- in project *DataRef* to prove that backward simulation is the dual of forward simulation: we instantiate all relations of forward simulation with their inverse relations and prove a) that the forward simulation conditions are exactly the backward simulation conditions for inverted relations and b) that the refinement theorem for inverted relations is equivalent to the original refinement theorem.
- in project *DataRef* to prove correctness of the proof obligations for backward simulation in the contract approach: this is done by instantiating the type of (abstract and concrete) states with states that include $\perp$, and instantiating partial operations with their totalised versions over this extended state space.
- in project *Z-Refinement* to instantiate the state space used in backward simulation and refinement (imported from *DataRef*) with states that include input and output as defined in Section 4.4 of [CSW02]. Additionally invariants are added for both data types (see below).
- in project *Mondex* to instantiate the theory defined in project *Z-Refinement* with the concrete data types used in the Mondex case study. As an example, the abstract state astate is mapped to the tuple balance, lost.

The specifications and proofs in project *DataRef* mimic the ones of chapter 2 and 3 of [CSW02], except that we have not restricted initialisation and finalisation operations of data types to be total. Instead we have added totality requirements in the proof obligations for forward and backward simulations only where necessary. For finalisation this means that we get two proof obligations, similar to correctness and applicability for operations. The motivation for this generalisation was the comparison with ASM refinement we did in [Sch05]: ASMs do not have a total finalisation relation (which would mean that the computation of an ASM might be finished at any time), but a specific set of final states.

The proof obligations in *Z-refinement* are different from the ones in [CSW02]. First we found, that the embedding used is not correct: input and output sequences are embedded into the initialisation and finalisation relation using an *empty[X, Y]* relation (e.g. $\mathsf{empty}[\mathsf{GO}, \mathsf{CO}]$ in section 4.4.1 to embed output in initialisation). This relation is defined in Appendix A.4 as the relation that comprises only a pair of empty sequences. But this is not a total relation, and would imply a partial initialisation relation. The correct definition should relate every sequence to the empty sequence (e.g. for $\mathsf{empty}[\mathsf{GO}, \mathsf{CO}]$ every global output GO present in the initial global state is discarded, so that the initial concrete state has empty output CO) just as it has been done in the closely related approach of [DB01], Definition 4.4.1 and p. 232.

With this corrected definition some of the proofs in Section 4.4 must be modified. This results in an additional proof obligation (from the finalisation proof) that every concrete input must be related to some abstract input (via relation ι_{b} in section 4.4, RIn in Appendix C.2; our web presentation calls this relation IT using the notation of [DB01], which we formalised earlier). At first it seems that this condition might be avoided using a partial finalisation where states are final only when there is no more input available. But this is not true: when proving the applicability condition of backward simulation for some operations COP and AOP we have to construct a state as such that a diagram commutes with some cs and $\mathsf{as}' = \mathsf{cs}' = \bot$ (i.e. we have $\mathsf{cs} \notin \mathsf{dom}(\mathsf{COP})$ and have to find $\mathsf{as} \notin \mathsf{dom}(\mathsf{COP})$). In the concrete scenario cs and as contain input that must be related with IT which imposes the requirement of IT to be total. Totality of IT is also required in theorem 10.5.2 of [DB01]. Adding the proof obligation it can be proved that backward simulation implies refinement correctness.

In the Mondex case study the proof obligations are applied restricting the state space of the concrete level to those states for which an invariant holds: this implies that all refinement proof obligations can assume the invariant for every concrete state. While this is adequate for the total operations of Mondex it seems there is a problem when using invariants to restrict the state space for the general case of partial operations. . Consider e.g. abstract states consisting of two variables x, y and an invariant $\mathsf{x} < \mathsf{y}$. Assume we have proven that $\mathsf{x} < \mathsf{y} \wedge \mathsf{AOP}(\mathsf{x}, \mathsf{y}, \mathsf{x}', \mathsf{y}') \rightarrow \mathsf{x}' < \mathsf{y}'$ i.e. that operations preserve the invariant whenever they are defined. Now for a partial operation AOP with $(1, 2) \notin \mathsf{dom}(\mathsf{AOP})$ the contract embedding says that any implementation of the operation should be allowed for this input (either abort to $\bot$ or choose an arbitrary next state). If the state space consists of states only that satisfy the invariant this will restrict the freedom of implementing the operation: a defined implementation will have to return a state that satisfies the invariant. An implementation of AOP with COP that relates the state $(1, 2)$ to $(1, 1)$ is no longer possible. But it is not necessary to restrict the state space and thereby the potential implementations: proof obligations that simply add the invariants as assumptions can still be shown to be sufficient for refinement correctness even if we allow the full state space.

For the proof obligations of forward simulations the proof that this can be done is simple: add invariants for the abstract and concrete data type as con-

juncts to the simulation relation. The simulation proof will then carry the invariants with it (see Theorem 2.4.2 in [DB01]). For backwards simulation such a proof cannot be done by instantiating the contract approach, since invariants do not propagate backwards through operations.

Therefore we derived a backward simulation theorem that allows invariants directly by instantiating the original approach of [HHS86].

We could prove the following two theorems for the contract approach without and with IO. The first is given here with the slight simplification of having total initialisation and finalisation using relational notation. The second is given in purely algebraic form, not using relational notation. While we used relational operators like $\lhd$, $\lhd\!\!\!\cdot$ a lot in the specifications of project *DataRef* we found that they only caused overhead since nearly every proof has to unfold their definition. Therefore we avoided them in later specifications. The proof obligations can also be found as axioms in the theories *conbackward-INV* in project *DataRef* (standard contract approach) and *IOconbackward-INV* in project *Z-refinement*:

Theorem 1. (Backward Simulation using Invariants)
Given an abstract data type $\mathsf{ADT} = (\mathsf{AINIT}, \mathsf{AIN}, \mathsf{AOP}, \mathsf{AFIN}, \mathsf{AOUT})$ *with total* $\mathsf{AINIT} \subseteq \mathsf{GS} \times \mathsf{AS}$*,* $\mathsf{AOP_i} \subseteq \mathsf{AS} \times \mathsf{AS}$*, total* $\mathsf{AFIN} \subseteq \mathsf{AS} \times \mathsf{GS}$*, a similar data type* $\mathsf{CDT} = (\mathsf{CINIT}, \mathsf{CIN}, \mathsf{COP}, \mathsf{CFIN}, \mathsf{COUT})$ *which uses states from* CS *instead of* AS*, a backward simulation* $\mathsf{T} \subseteq \mathsf{CS} \times \mathsf{AS}$ *and two invariants* $\mathsf{AINV} \subseteq \mathsf{AS}$ *and* $\mathsf{CINV} \subseteq \mathsf{CS}$*, then the refinement is correct using the contract approach provided the following proof obligations hold:*

- $\mathsf{CINIT} \subseteq \mathsf{CINV}, \mathsf{AINIT} \subseteq \mathsf{AINV}$ *(initially invariants)*
- $\mathrm{ran}(\mathsf{AINV} \lhd \mathsf{AOP}) \subseteq \mathsf{AINV}, \mathrm{ran}(\mathsf{CINV} \lhd \mathsf{COP}) \subseteq \mathsf{CINV}$ *(invariance)*
- $(\mathsf{CINIT} \rhd \mathsf{CINV}) \fatsemi \mathsf{T} \subseteq \mathsf{AINIT}$ *(initialisation)*
- $(\mathsf{CINV} \lhd \mathsf{CFIN}) \subseteq \mathsf{T} \fatsemi (\mathsf{AINV} \lhd \mathsf{AFIN})$ *(finalisation)*
- $\mathrm{dom}(\mathsf{COP_i}) \lhd\!\!\!\cdot\, \mathsf{CINV} \subseteq \mathrm{dom}((\mathsf{T} \lhd \mathsf{AINV}) \lhd\!\!\!\cdot\, \mathrm{dom}(\mathsf{AOP_i}))$ *(applicability)*
- $\mathrm{dom}(\mathsf{T} \lhd\!\!\!\cdot\, \mathrm{dom}(\mathsf{AOP_i})) \lhd\!\!\!\cdot\, (\mathsf{COP_i} \fatsemi \mathsf{T}) \subseteq \mathsf{T} \fatsemi (\mathsf{AINV} \lhd \mathsf{AOP_i})$ *(correctness)*

Instead of the usual embedding of the contract approach $\overset{\circ}{\mathsf{T}} = \mathsf{T} \cup \{\bot\} \times \mathsf{AS}_\bot$ the proof uses $\overset{\circ}{\mathsf{T}} = (\mathsf{T} \rhd \mathsf{AINV}) \cup \{\mathsf{CS}_\bot \setminus \mathsf{CINV}\} \times \mathsf{AS}_\bot$. The idea is that the concrete states that do not satisfy the invariant behave like the undefined $\bot$ state and therefore get mapped to every abstract state. The proof proceeds as usual by eliminating $\bot$ from the resulting proof obligations.

Theorem 2. (Backward Simulation with IO using Invariants)
Assume an abstract data type $\mathsf{ADT} = (\mathsf{AINIT}, \mathsf{AIN}, \mathsf{AOP}, \mathsf{AFIN}, \mathsf{AOUT})$ *where*

- $\mathsf{AINIT} \subseteq \mathsf{AS}$ *(the set of initial states)*
- $\mathsf{AIN} \subseteq \mathsf{GI} \times \mathsf{AI}$*, (inputs initialised from global inputs)*
- $\mathsf{AOP_i} \subseteq \mathsf{AI} \times \mathsf{AS} \times \mathsf{AS} \times \mathsf{AO}$ *(abstract operations read an input, modify the state and produce output)*
- $\mathsf{AFIN} \subseteq \mathsf{AS} \times \mathsf{GS}$ *(finalising a local state gives a global state)*
- $\mathsf{AOUT} \subseteq \mathsf{AO} \times \mathsf{GO}$ *(finalising output to global output)*

together with an invariant $\mathsf{AINV} \subseteq \mathsf{AS}$ *and a similar concrete data type* $\mathsf{CDT} = (\mathsf{CINIT}, \mathsf{CIN}, \mathsf{COP}, \mathsf{CFIN}, \mathsf{COUT})$ *with invariant* $\mathsf{CINV} \subseteq \mathsf{CS}$ *which uses the same global data* $\mathsf{GI}, \mathsf{GS}, \mathsf{GO}$ *but different local data* $\mathsf{CI}, \mathsf{CS}, \mathsf{CO}$ *are given. Then a backward simulation consisting of* $\mathsf{IT} \subseteq \mathsf{CI} \times \mathsf{AI}$, $\mathsf{T} \subseteq \mathsf{CS} \times \mathsf{AS}$ *and* $\mathsf{OT} \subseteq \mathsf{CO} \times \mathsf{AO}$ *proves correctness of the refinement (in the same sense as in [CSW02]), if the following proof obligations can be verified:*

- *Initialisation (state)*
  $\mathsf{CINIT}(\mathsf{cs}), \mathsf{T}(\mathsf{cs}, \mathsf{as}) \vdash \mathsf{AINIT}(\mathsf{as})$
- *Initialisation (input)*
  $\mathsf{CIN}(\mathsf{gin}, \mathsf{cin}), \mathsf{IT}(\mathsf{cin}, \mathsf{ain}') \vdash \mathsf{AIN}(\mathsf{gin}, \mathsf{ain}')$
- *Applicability*
  $\mathsf{CINV}(\mathsf{cs}), (\mathsf{cin}, \mathsf{cs}) \notin \mathrm{dom}(\mathsf{COP_i})$
  $\vdash \exists\ \mathsf{as}, \mathsf{ain}.\ \mathsf{T}(\mathsf{cs}, \mathsf{as}) \wedge \mathsf{AINV}(\mathsf{as}) \wedge \mathsf{IT}(\mathsf{cin}, \mathsf{ain}) \wedge (\mathsf{ain}, \mathsf{as}) \notin \mathrm{dom}(\mathsf{AOP_i})$
- *Correctness*
  $\mathsf{CINV}(\mathsf{cs}), \mathsf{COP_i}(\mathsf{cin}, \mathsf{cs}, \mathsf{cs}', \mathsf{cou}'), \mathsf{T}(\mathsf{cs}', \mathsf{as}'), \mathsf{AINV}(\mathsf{as}'), \mathsf{OT}(\mathsf{cou}', \mathsf{aou}'),$
  $\forall\ \mathsf{as}, \mathsf{ain}.\ \mathsf{T}(\mathsf{cs}, \mathsf{as}) \wedge \mathsf{AINV}(\mathsf{as}) \wedge \mathsf{IT}(\mathsf{cin}, \mathsf{ain}) \rightarrow (\mathsf{ain}, \mathsf{as}) \in \mathrm{dom}(\mathsf{AOP_i})$
  $\vdash \exists\ \mathsf{as}, \mathsf{ain}.\ \mathsf{IT}(\mathsf{cin}, \mathsf{ain}) \wedge \mathsf{T}(\mathsf{cs}, \mathsf{as}) \wedge \mathsf{AINV}(\mathsf{as}) \wedge \mathsf{AOP_i}(\mathsf{ain}, \mathsf{as}, \mathsf{as}', \mathsf{aou}')$
- *Finalisation (state)*
  $\mathsf{CINV}(\mathsf{cs}), \mathsf{CFIN}(\mathsf{cs}, \mathsf{gs}) \vdash \exists\ \mathsf{as}.\ \mathsf{AINV}(\mathsf{as}) \wedge \mathsf{T}(\mathsf{cs}, \mathsf{as}) \wedge \mathsf{AFIN}(\mathsf{as}, \mathsf{gs})$
- *Finalisation (output)*
  $\mathsf{COUT}(\mathsf{cou}, \mathsf{gou}') \vdash \exists\ \mathsf{aou}.\ \mathsf{OT}(\mathsf{cou}, \mathsf{aou}) \wedge \mathsf{AOUT}(\mathsf{aou}, \mathsf{gou}')$
- *Totality of finalisation (state)*
  $\mathsf{CINV}(\mathsf{cs}) \vdash \exists\ \mathsf{gs}'.\ \mathsf{CFIN}(\mathsf{cs}, \mathsf{gs}')$
- *Totality of finalisation (output)*
  $\vdash \exists\ \mathsf{gou}'.\ \mathsf{COUT}(\mathsf{cou}, \mathsf{gou}')$
- *Totality of input*
  $\vdash \exists\ \mathsf{ain}.\ \mathsf{IT}(\mathsf{cin}, \mathsf{ain})$
- *Totality of initialisation (state)*
  $\exists\ \mathsf{cs}.\ \mathsf{CINIT}(\mathsf{cs})$
- *Totality of initialisation (input)*
  $\exists\ \mathsf{cin}'.\mathsf{CIN}(\mathsf{gin}, \mathsf{cin}')$
- *Initially abstract invariant*
  $\mathsf{AINIT}(\mathsf{as}) \vdash \mathsf{AINV}(\mathsf{as})$
- *Initially concrete invariant*
  $\mathsf{CINIT}(\mathsf{cs}) \vdash \mathsf{CINV}(\mathsf{cs})$
- *Abstract invariant preserved*
  $\mathsf{CINV}(\mathsf{cs}), \mathsf{COP_i}(\mathsf{cin}, \mathsf{cs}, \mathsf{cs}', \mathsf{cou}') \vdash \mathsf{CINV}(\mathsf{cs}')$
- *Concrete invariant preserved*
  $\mathsf{AINV}(\mathsf{as}), \mathsf{AOP_i}(\mathsf{ain}, \mathsf{as}, \mathsf{as}', \mathsf{aou}') \vdash \mathsf{AINV}(\mathsf{as}')$

The proof of this theorem uses the corrected *empty*-relation and instantiates the previous theorem, but otherwise proceeds routinely like the one in [CSW02].

5 Verification of the Data Refinement

Our specification of the operations of the two data types used for the Mondex refinement is based on two ASMs in specifications *AASM* and *CASM*. These

two ASMs try to mimic the semantics of the data types as faithfully as possible (see [Sch05] for the general translation). We have not used these ASMs directly (so the top level ASM rules may still have flaws), but we have used parts of the ASM rules to define the individual operations of the data type by axioms in specifications *Mondex-AOP* and *Mondex-COP* like

$$\mathsf{COP}(\mathsf{cs}, \mathsf{cs}') \leftrightarrow \langle \mathsf{RULE\#}(\mathsf{cs}) \rangle\ \mathsf{cs} = \mathsf{cs}'$$

This equivalence defines the $\mathsf{COP}(\mathsf{cs}, \mathsf{cs}')$ to hold if and only if ASM rule RULE# started with cs has a terminating run that computes cs' as its result. We could have defined the operations without referring to an ASM altogether, doing so has the advantage that the principle of symbolic execution can still be used to automate proofs.

Apart from the auxiliary use of operational definitions instead of pure relations the specification mimics the structure of the Z specifications faithfully: STARTFROM# is now prefixed with ABORT#, input is read from a list of inputs, disjunctions with IGNORE# operations that skip have been added etc.

The use of auxiliary operational definitions has the effect that the main proof obligations for data refinement, "Correctness" and "Concrete invariant preserved", have proofs which are nearly identical to the ones we did for ASM refinement (see the proofs of theorems *correctness* and *cinv-ok* in specification *Mondex-refine* in project *Mondex* on the web [KIV]). The only important differences are that instead of one proof for the full ASM rule we now have several proof obligations for the individual operations corresponding to cases in the ASM proof (lemmas *ABORT-ACINV*, *REQ-ACINV* etc. for correctness, *ABORT-CINV*, *REQ-CINV* etc. for invariance) and that the lemmas for ABORT# and IGNORE# are used several times, since several operations now refer to it.

We have decided to merge the two refinements of the Mondex case study into one, so each operation calls LOSEMSG# at the end, just as described for the ASM at the end of section 2.

This means that our concrete invariant cannot be BINV since the properties of the ether which have been specified with a predicate $\mathsf{etherok}(\mathsf{ether}, \ldots)$ that is part of the definition of BINV (see Appendix 8 for the exact definitions) do not hold for an ether where messages have been dropped. Instead we replace the old definition of etherok with

$$\mathsf{newetherok}(\mathsf{ether}, \ldots) \leftrightarrow \exists\ \mathsf{fullether}.\ \mathsf{ether} \subseteq \mathsf{fullether} \wedge \mathsf{etherok}(\mathsf{fullether}, \ldots)$$

The new predicate[7] claims the existence of fullether, where no messages have been dropped, such that fullether has the properties specified in the old etherok predicate. fullether does not change during LOSEMSG#, otherwise it is modified just like ether. The new definition of etherok is used in the definition of the new

[7] The web presentation [KIV] uses the modified etherok definition given in specification *Mondex-CINV*, not a new predicate.

invariant CINV for the concrete level. The backward simulation ABINV is left unchanged. It is just renamed to ACINV.

Summarizing, there is a little extra work required to cope with the redundancy of operations and the lossy ether, but essentially proofs are done by "copy-paste" from the ASM proofs. Of course a little additional effort is also needed to prove the remaining proof obligations (like totality of operations) and the security properties *NoValueCreated* and *AllValueAccounted* of the abstract specification. These are proved in specification *Mondex-Secprop*).

Summarizing, the effort to do the full case study was as follows

- 1 week was needed to get familiar with the case study and to set up the initial ASMs (Section 2).
- 1 week was needed to prove the essential proof obligations "correctness" and "invariance" for the ASM refinement as shown in (Section 3).
- 1 week was needed to specify the Mondex refinement theory of [CSW02] and to generalize the proof obligations to cope with invariants (Section 4).
- Finally, 1 week was necessary to prove the data refinement and to polish the theories for the web presentation (this section).

Of course the time needed to do the verification is influenced by the level of expertise with formal verification in general and with the KIV system in particular. Getting the work done in 4 weeks was helped by having a (nearly) correct simulation relation. Usually most of the time is not needed to verify the correct solution, but to find invariants and simulation relations incrementally. On the other hand, sticking to ASM refinement would have shortened the verification time. The main data refinement proofs for the Mondex refinement consist of 1839 proof steps with 372 interactions.

The effort required can be compared to the effort required for refinement proofs from another application domain which we did at around the same time as the original Mondex case study: verification of a compiler that compiles Prolog to code of the Warren abstract machine ([SA97], [SA98], [Sch99], [Sch01]). This case study required 9 refinements, and the statistical data ([Sch99], Chapter 19) show that each refinement in this case study needed on average about the same number of proof steps in KIV as the Mondex case study.

The ratio of interactions to proof steps is somewhat better in the WAM case study, since automation of refinement proofs increases over time: investing time to improve automation by adding rewrite rules becomes more important when similar steps are necessary in several refinements and when developing simulation relations iteratively. Summarizing, the Mondex case study is a medium-sized case study and a good benchmark for interactive theorem provers.

6 A Security Model for Mondex Purses

Although the Mondex case study was the basis of an ITSEC E6 certification ([CB99]), the formal model abstracts away an important part of the security of the application. As the cryptographic protocols used to realize the value transfer

were and are still, to our knowledge, undisclosed ([RSA04],[CCW96]) the formal model assumes the existence of unforgeable messages for requesting, transferring and acknowledging the transfer of a value. To complete the analysis of the application a model based on a theory of messages with abstract representations of the used cryptographic mechanisms should be specified and used to proof that the 'dangerous' messages actually cannot be forged.

The Mondex application is prepared to use different cryptographic algorithms in the value transfer protocol. It is generally assumed that DES and RSA were used to authenticate the value transfer ([DFG98],[CB99]). It is not too difficult to come up with a cryptographic protocol that ensures that its messages have the properties that are required for the abstract messages req, val and ack. Using DES as cryptographic algorithm a shared secret key is used for authentication of messages ([BGJY98]). A possible protocol written in a commonly used standard notation for cryptographic protocols ([Car94])is:
1. $\mathsf{to} \rightarrow \mathsf{from} : \{\mathrm{REQ},\mathsf{pdAuth(to)}\}_{\mathsf{K_S}}$
2. $\mathsf{from} \rightarrow \mathsf{to} : \{\mathrm{VAL},\mathsf{pdAuth(from)}\}_{\mathsf{K_S}}$
3. $\mathsf{to} \rightarrow \mathsf{from} : \{\mathrm{ACK},\mathsf{pdAuth(to)}\}_{\mathsf{K_S}}$

In this protocol $\mathsf{K_S} : \mathsf{key}$ denotes a secret key shared between all valid Mondex cards. REQ, VAL and ACK are pairwise distinct constants used to distinguish the three message types.

Using RSA makes things somewhat more complicated since individual key pairs and digital certificates should then be used. To ensure security for the next years keys with at least 1024 Bit length must be used. Given this key size the public key and the associated certificate of a Mondex card and the payload of the protocol messages cannot be transferred to the smart card in one step, due to restrictions of the communication interface of smart cards. Therefore some of the steps that are atomic on the concrete level of Mondex would have to be split up into several steps on the implementation level. This further complicates the refinement.

Assuming the DES-based protocol, the challenge to be solved is to verify the security of the Mondex application with this real cryptographic protocol instead of the special messages postulated as unforgeable in the Mondex case study in Z. Possible approaches generally used in the verification of cryptographic protocols are model-checking ([Low96], [BMV03]) or interactive verification ([Pau98],[Eva03]). Paulson's inductive approach has proven to be especially powerful by tackling complex industrial protocols ([Pau01]). We plan to use our ASM-based model for cryptographic protocols ([HGRS05]) for verification. Particularly interesting is the question whether the protocol with cryptographic operations can be proven to be a refinement of the concrete protocol of the original Mondex case study. We think such a refinement is possible, and the Mondex case study shows an elegant way to separate functional correctness and security into two refinements.

7 Conclusion and Further Work

We have specified and formally verified the full communication protocol of the Mondex case study with the KIV system. We have slightly improved the protocol to use one idle instead of two eaFrom and eaTo states. We have improved the theory of backward simulation in the contract approach to include invariants for the data types. Using the improved theory, the correctness proof for Mondex could be done as one refinement instead of two. We feel that the effort to do this was rather small compared to the effort we assume it has taken to write down proofs in [SCJ00] at nearly calculus level. Despite this great detail we were still able to find two small flaws: one in the underlying data refinement theory, where a proof obligation was missing and one in the invariant, where we had to add a totality property. Therefore we feel justified to recommend doing machine proofs as a means to increase confidence in the results.

As a second contribution we gave an alternative, concise specification of the refinement problem using ASMs. The fact that the main proofs are nearly identical to those for the original refinement indicates, that the ASMs are a good starting point to further improve the invariant and the verification technique.

One idea for further work is therefore to take the ideas of [HGRS05] to do a proper ASM refinement proof (that probably would use generalized forward simulation [Sch01] instead of backward simulation).

Another idea contained in the Mondex case study that we will try to address is that functional correctness and a security protocol as proposed Section 6 may be verified independently as two separate refinements.

Acknowledgement We like to thank Prof. Börger for pointing out the Mondex challenge to us.

References

[BGJY98] M. Bellare, J. Garay, C. Jutla, and M. Yung. VarietyCash: a Multipurpose Electronic Payment System. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*. USENIX, September 1998. URL: http://citeseer.ist.psu.edu/bellare98varietycash.html.

[BMV03] David Basin, Sebastian Mödersheim, and Luca Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proceedings of Esorics'03*, LNCS 2808, pages 253–270. Springer-Verlag, Heidelberg, 2003.

[Bör03] E. Börger. The ASM Refinement Method. *Formal Aspects of Computing*, 15 (1–2):237–257, November 2003.

[BR95] E. Börger and D. Rosenzweig. The WAM—definition and compiler correctness. In Christoph Beierle and Lutz Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artificial Intelligence*, pages 20–90. North-Holland, Amsterdam, 1995.

[BS03] E. Börger and R. F. Stärk. *Abstract State Machines—A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[Car94] Ulf Carlsen. Generating formal cryptographic protocol specifications. In *IEEE Symposium on Research in Security and Privacy*, pages 137–146. IEEE Computer Society, 1994.

[CB99] UK ITSEC Certification Body. UK ITSEC SCHEME CERTIFICATION REPORT No. P129 MONDEX Purse. Technical report, UK IT Security Evaluation and Certification Scheme, 1999. URL: http://www.cesg.gov.uk/site/iacs/itsec/media/certreps/CRP129.pdf.

[CCW96] E. K. Clemons, D. C. Croson, and B. W. Weber. Reengineering Money: The Mondex Stored Value Card and Beyond. In *Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences 1996*. IEEE, 1996. URL: http://doi.ieeecomputersociety.org/10.1109/HICSS.1996.495345.

[CoF04] CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS 2960 (IFIP Series). Springer, 2004.

[CSW02] D. Cooper, S. Stepney, and J. Woodcock. Derivation of Z Refinement Proof Rules: forwards and backwards rules incorporating input/output refinement. Technical Report YCS-2002-347, University of York, 2002. URL: http://www-users.cs.york.ac.uk/∼susan/bib/ss/z/zrules.htm.

[DB01] J. Derrick and E. Boiten. *Refinement in Z and in Object-Z : Foundations and Advanced Applications*. FACIT. Springer, 2001.

[DFG98] R. D'Amico, F. Fonseca, and D. Guerin. EPOS: Europe-wide IS-card based Payment for Online services deliverable 1 volume 2 of 4. Technical report, EURESCOM Project 705, February 1998. URL: http://www.eurescom.de/public/projectresults/P700-series/705d1.htm.

[Eva03] Neil Evans. *Investigating Security Through Proof*. PhD thesis, Royal Holloway University of London, 2003.

[Far94] W. M. Farmer. Theory interpretation in simple type theory. In J. Heering et al., editor, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer-Verlag, 1994.

[Gur95] M. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9 – 36. Oxford University Press, 1995.

[HGRS05] D. Haneberg, H. Grandy, W. Reif, and G. Schellhorn. Verifying Security Protocols: An ASM Approach. In D. Beauquier, E. Börger, and A. Slissenko, editors, *12th Int. Workshop on Abstract State Machines, ASM 05*. University Paris 12 – Val de Marne, Créteil, France, March 2005.

[HHS86] He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, volume 213 of *Lecture Notes in Computer Science*, pages 187–196. Springer-Verlag, 1986.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[KIV] Web presentation of the mondex case study in KIV. URL: http://www.informatik.uni-augsburg.de/swt/projects/mondex.html.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, 1996.

[MCI] MasterCard International Inc. *Mondex*. URL: http://www.mondex.com.

[Pau98] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *J. Computer Security*, 6:85–128, 1998.

[Pau01] L. C. Paulson. Verifying the set protocol. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001: International Joint Conference on Automated Reasoning*, Siena, Italy, 2001. Springer LNCS 2083.

[RE03] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley & Sons, 3rd edition, 2003.

[Rei92] W. Reif. Correctness of Generic Modules. In Nerode and Taitslin, editors, *Symposium on Logical Foundations of Computer Science*, LNCS 620, Berlin, 1992. Logic at Tver, Tver, Russia, Springer.

[RSA04] RSA. What is Mondex?, 2004. URL: http://www.rsasecurity.com/rsalabs/node.asp?id=2288.

[RSSB98] W. Reif, G. Schellhorn, K. Stenzel, and M. Balser. Structured specifications and interactive proofs with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II: Systems and Implementation Techniques, chapter 1: Interactive Theorem Proving, pages 13 – 39. Kluwer Academic Publishers, Dordrecht, 1998.

[SA97] G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM Case Study. *Journal of Universal Computer Science (J.UCS)*, 3(4):377–413, 1997. URL: http://hyperg.iicm.tu-graz.ac.at/jucs/.

[SA98] G. Schellhorn and W. Ahrendt. The WAM Case Study: Verifying Compiler Correctness for Prolog with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications*, pages 165 – 194. Kluwer Academic Publishers, Dordrecht, 1998.

[Sch99] Gerhard Schellhorn. *Verification of Abstract State Machines*. PhD thesis, Universität Ulm, Fakultät für Informatik, 1999. URL: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/publications/.

[Sch01] G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *Journal of Universal Computer Science (J.UCS)*, 7(11):952–979, 2001. URL: http://hyperg.iicm.tu-graz.ac.at/jucs/.

[Sch05] G. Schellhorn. ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison. *Journal of Theoretical Computer Science*, vol. 336, no. 2-3:403–435, May 2005.

[SCJ00] S. Stepney, D. Cooper, and Woodcock J. AN ELECTRONIC PURSE Specification, Refinement, and Proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000. URL: http://www-users.cs.york.ac.uk/∼susan/bib/ss/z/monog.htm.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[WD96] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[Woo06] J. Woodcock. Mondex case study, 2006. URL: http://qpq.csl.sri.com/vsr/shared/MondexCaseStudy/.

8 Appendix: Definition of the Invariant and the Simulation Relation

The backward simulation used in the ASM proofs as well as in the data refinement proof is ACINV = ABINV, the invariant used in the ASM refinement proof is BINV. The invariant CINV for the data refinement proof is derived from BINV as described in Section 5. The definitions are copied from the specifications *BINV* and *Mondex-ASM-refine* in project *Mondex* on the Web [KIV]. Remarks at the end of the line give the correspondence to the properties of [SCJ00]: PD is the payment details property of section 4.3.2, P-1 to P-4 are the purse properties of section 4.6, and B-1 to B-16 are the invariants of section 5.3. Property B-14

and B-15 are the only properties that refers to logs that have been moved to a central archive, so it was left off. Property B-16 is implied by B-10 and B-12 and therefore removed.

BINV(balance, exLog, state, nextSeqNo, pdAuth, ether)
↔ purseok(balance, state, nextSeqNo, pdAuth)
∧ etherok(exLog, state, nextSeqNo, pdAuth, ether)
∧ exLogok(exLog, state, nextSeqNo, pdAuth)
∧ logsfin(exLog) B-13

purseok(balance, state, nextSeqNo, pdAuth)
↔ ∀ na. authentic(na)
→ (state(na) = epr
→ pdAuth(na).from = na
∧ pdAuth(na).from ≠ pdAuth(na).to PD
∧ pdAuth(na).value ≤ balance(na) P-2
∧ pdAuth(na).fromno < nextSeqNo(na)) P-2
∧ ((state)(na) = epv
→ pdAuth(na).to = na added to P-3
∧ pdAuth(na).from ≠ pdAuth(na).to PD
∧ pdAuth(na).tono < nextSeqNo(na) P-3
∧ authentic((pdAuth)(na).from)) added to P-3
∧ ((state)(na) = epa
→ pdAuth(na).from = na added to P-4
∧ pdAuth(na).from ≠ pdAuth(na).to PD
∧ pdAuth(na).fromno < nextSeqNo(na) P-4
∧ authentic(pdAuth(na).to)) added to P-4

etherok(exLog, state, nextSeqNo, pdAuth, ether)
↔ ∀ pd. (req(pd) ∈ ether
→ pd.tono < (nextSeqNo)(pd.to) B-2
∧ authentic(pd.to)) B-1
∧ (val(pd) ∈ ether ∨ ack(pd) ∈ ether
→ pd.fromno < (nextSeqNo)(pd.from) B-3, B-4
∧ pd.tono < (nextSeqNo)(pd.to) B-3, B-4
∧ authentic(pd.to) added to B-3, B-4
∧ authentic(pd.from)) added to B-3, B-4
∧ (fromInEpr(pd, pdAuth, state)
→ ¬ val(pd) ∈ ether ∧ ¬ ack(pd) ∈ ether) B-9
∧ (req(pd) ∈ ether ∧ ¬ ack(pd) ∈ ether
↔ toInEpv(pd, pdAuth, state)
↔ ∨ toLogged(pd, exLog)) B-10
∧ (val(pd) ∈ ether ∧ toInEpv(pd, pdAuth, state)
→ fromInEpa(pd, pdAuth, state)
∨ fromLogged(pd, exLog)) B-11
∧ (fromInEpa(pd, pdAuth, state)
∨ fromLogged(pd, exLog)
→ req(pd) ∈ ether) B-12

exLogok(exLog, state, nextSeqNo, pdAuth)
$\leftrightarrow \forall$ pd, na. (fromLogged(pd, exLog)
$\rightarrow$ pd.fromno $<$ (nextSeqNo)(pd.from) B-5
$\wedge$ (state(pd.from) = epr
$\vee$ state(pd.from) = epa
$\rightarrow$ pd.fromno $<$ pdAuth(pd.from).fromno)) B-7
$\wedge$ (toLogged(pd, exLog)
$\rightarrow$ pd.tono $<$ nextSeqNo(pd.to) B-6
$\wedge$ (state(pd.to) = epv
$\rightarrow$ pd.tono $<$ pdAuth(pd.to).tono)) B-8
$\wedge$ (authentic(na) $\wedge$ pd $\in$ exLog(na)
$\rightarrow$ (pd.to = na $\vee$ pd.from = na) P-1
$\wedge$ pd.to $\neq$ pd.from) PD

fromLogged(pd, exLog)
$\leftrightarrow$ authentic(pd.from) $\wedge$ pd $\in$ exLog(pd.from)

toLogged(pd, exLog)
$\leftrightarrow$ authentic(pd.to) $\wedge$ pd $\in$ exLog(pd.to)

fromInEpa(pd, pdAuth, state)
$\leftrightarrow$ authentic(pd.from) $\wedge$ pd = pdAuth(pd.from)
$\wedge$ state(pd.from) = epa

fromInEpr(pd, pdAuth, state)
$\leftrightarrow$ authentic(pd.from) $\wedge$ pd = pdAuth(pd.from)
$\wedge$ state(pd.from) = epr

toInEpv(pd, pdAuth, state)
$\leftrightarrow$ authentic(pd.to) $\wedge$ pd = pdAuth(pd.to) $\wedge$ state(pd.to) = epv

logsfin(exLog)
$\leftrightarrow (\exists$ pds. $\forall$ pd. toLogged(pd, exLog) $\leftrightarrow$ pd $\in$ pds)

ABINV(balance, lost, balance$_0$, exLog, state,
nextSeqNo, pdAuth, ether, outmsg)
$\leftrightarrow \exists$ maybelost, chosenlost, definitelylost.
maybeLost(exLog, pdAuth, state, maybelost)
$\wedge$ definitelyLost(exLog, pdAuth, state, definitelylost)
$\wedge$ chosenlost $\subseteq$ maybelost
$\wedge$ balandlostok(balance, lost, balance$_0$, chosenlost,
definitelylost, maybelost)

$$
\begin{array}{l}
\quad \text{maybeLost(exLog, pdAuth, state, maybelost)} \\
\leftrightarrow \forall\ \text{pd.} \quad \text{pd} \in \text{maybelost} \\
\qquad \leftrightarrow \quad \text{toInEpv(pd, pdAuth, state)} \\
\qquad\quad \wedge\ (\quad \text{fromLogged(pd, exLog)} \\
\qquad\qquad \vee\ \text{fromInEpa(pd, pdAuth, state))}
\end{array}
$$

$$
\begin{array}{l}
\quad \text{definitelyLost(exLog, pdAuth, state, definitelylost)} \\
\leftrightarrow \forall\ \text{pd.} \quad \text{pd} \in \text{definitelylost} \\
\qquad \leftrightarrow \quad \text{toLogged(pd, exLog)} \\
\qquad\quad \wedge\ (\quad \text{fromLogged(pd, exLog)} \\
\qquad\qquad \vee\ \text{fromInEpa(pd, pdAuth, state))}
\end{array}
$$

$$
\begin{array}{l}
\quad \text{balandlostok(balance, lost, balance}_0\text{,} \\
\qquad\qquad \text{chosenlost, definitelylost, maybelost)} \\
\leftrightarrow \forall\ \text{na.} \quad \text{authentic(na)} \\
\qquad \rightarrow \quad \text{(lost)(na)} = \Sigma\ \text{filter}(\lambda\ \text{pd. pd.from} = \text{na}, \\
\qquad\qquad\qquad\qquad\qquad \text{definitelylost} \cup \text{chosenlost}) \\
\qquad\quad \wedge \quad \text{(balance)(na)} \\
\qquad\qquad = (\text{balance}_0)(\text{na}) + \\
\qquad\qquad\quad \Sigma\ \text{filter}(\lambda\ \text{pd. pd.to} = \text{na}, \text{maybelost} \setminus \text{chosenlost})
\end{array}
$$

$$\Sigma\ \emptyset = 0$$

$$\Sigma(\text{pds} +\!+ \text{pd}) = \text{pd} \in \text{pds} \supset \Sigma\ \text{pds}; \Sigma\ \text{pds} + \text{pd.value}$$

$$\text{filter(Ppd, } \emptyset) = \emptyset$$

$$
\begin{array}{l}
\text{filter(Ppd, pds} +\!+ \text{pd)} = \text{(Ppd)(pd)} \supset \text{filter(Ppd, pds)} +\!+ \text{pd;} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{filter(Ppd, pds)}
\end{array}
$$