The Mondex challenge: machine checked proofs for an electronic purse

Gerhard Schellhorn, Holger Grandy, Dominik Haneberg, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Schellhorn, Gerhard, Holger Grandy, Dominik Haneberg, and Wolfgang Reif. 2006. "The Mondex challenge: machine checked proofs for an electronic purse." In *FM 2006: Formal Methods: 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, proceedings*, edited by Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, 16–31. Berlin: Springer. https://doi.org/10.1007/11813040_2.

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 detail 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 independent of the protocol for money transfer.

The Mondex case study is too big to be presented completely within a paper of 16 pages ([SCJ00] has 240 pages, [CSW02] additional 54 pages). Therefore we unfortunately will have to refer to these papers quite often. To view the details of the KIV proofs we have prepared a technical report [SGHR06] 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 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 formalised 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.

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, [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]

$$\begin{array}{l}
\mathsf{ABTRANSFER\#} \\
\textbf{choose } \mathsf{from}, \mathsf{to}, \mathsf{value}, \mathsf{fail?} \\
\textbf{with } \mathsf{authentic}(\mathsf{from}) \wedge \mathsf{authentic}(\mathsf{to}) \wedge \mathsf{from} \neq \mathsf{to} \wedge \mathsf{value} \leq \mathsf{balance}(\mathsf{from}) \\
\textbf{in if } \neg\, \mathsf{fail?} \textbf{ then } \mathsf{balance}(\mathsf{from}) := \mathsf{balance}(\mathsf{from}) - \mathsf{value} \\
\qquad\qquad\qquad\qquad \mathsf{balance}(\mathsf{to}) := \mathsf{balance}(\mathsf{to}) + \mathsf{value} \\
\qquad\qquad\quad \textbf{else } \mathsf{balance}(\mathsf{from}) := \mathsf{balance}(\mathsf{from}) - \mathsf{value} \\
\qquad\qquad\qquad\quad \mathsf{lost}(\mathsf{from}) := \mathsf{lost}(\mathsf{from}) + \mathsf{value}
\end{array}$$

[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.

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 → $\mathbb{N}$
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 (selecting a message from ether is defined in the full ASM rule BOP# at the end of this section) 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) ∧ receiver ≠ msgna
      ∧ value ≤ balance(receiver) ∧ ¬ fail?
   then choose n with nextSeqNo(receiver) < n in
           pdAuth(receiver) := mkpd(receiver, nextSeqNo(receiver),
                                    msgna, msgno, value)
           state(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:

```
STARTTO#
let msgna = msg.name, value = msg.value, msgno = msg.nextSeqNo
in if authentic(msgna) ∧ receiver ≠ msgna ∧ ¬ fail?
   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 := ⊥
```

[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.

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:

BOP#
choose msg, receiver, fail? **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

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

the end of the rule the produced message 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 the ASM rule COP# that models the concrete level replaces ether := ether ++ outmsg in BOP# with LOSEMSG# where:

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

If a purse is sent an illegal message ⊥ 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 purses 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. Although the ASM is much simpler than the full Z specification (no promotion was used, none of the Z schemas in [SCJ00] that describe which variables are *not* allowed to change in operations are necessary, STARTFROM# is not prefixed with ABORT#, since the ASM can do this step separately by choosing ⊥ from ether, etc.) it still captures the essence of the refinement as we will see in Section 5.

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 in [SCJ00]: 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) and 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) gives a suitable definition of BINV: full details can be found in the technical report [SGHR06] and in specification *BINV* in project *Mondex* [KIV].

There is one interesting modification: 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. $\underline{\mathsf{cs}} = \mathsf{balance}, \mathsf{state}, \mathsf{nextSeqNo}, \mathsf{pdAuth}, \mathsf{ether}$. $\langle\!|\mathsf{BOP}\#(;\ \underline{\mathsf{cs}})|\!\rangle\ \mathsf{BINV}(\underline{\mathsf{cs}})$ states that all runs of BOP# terminate in a state where BINV holds again.

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 for the VAL# rule. 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 some $\mathsf{na} := \mathsf{pdAuth}(\mathsf{receiver}).\mathsf{to}$ but has $\mathsf{state}(\mathsf{na}) = \mathsf{epv}$ and $\mathsf{pdAuth}(\mathsf{na}) = \mathsf{pdAuth}(\mathsf{receiver})$, then for *this* na the implication would be violated. The solution to this problem is obvious: add $\mathsf{pdAuth}(\mathsf{receiver}).\mathsf{to} = \mathsf{receiver}$ when $\mathsf{state}(\mathsf{receiver}) = \mathsf{epv}$ to P-3.

A similar problem also exists for $\mathsf{state}(\mathsf{receiver}) = \mathsf{epa}$ (property P-4) where $\mathsf{pdAuth}(\mathsf{receiver}).\mathsf{from} = \mathsf{receiver}$ 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.

With these additions the invariant proof was successful. 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. None of the complex lemma structure of [SCJ00] was necessary, although in some situations where it was not clear why our proof got stuck, it was helpful to cross-check details with the original proofs.

After this proof we verified the backward simulation condition:

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

$\mathsf{ABINV}(\underline{\mathsf{as}}, \underline{\mathsf{cs}})$ is the backward simulation. The definition is basically identical to the simulation relation defined in [SCJ00].

The meaning of $\langle \mathsf{BOP}\#(\underline{\mathsf{cs}})\rangle\ \underline{\mathsf{cs}} = \underline{\mathsf{cs}'}$ is that $\mathsf{BOP}\#$ called with $\underline{\mathsf{cs}}$ terminates and yields $\underline{\mathsf{cs}'}$. This is equivalent to $\mathsf{BOP}(\underline{\mathsf{cs}}, \underline{\mathsf{cs}'})$. The proof obligation for ASM refinement allows a 1:1 diagram, where the concrete rule $\mathsf{BOP}\#$ refines an abstract operation $\mathsf{AOP}\#$ as well as a 0:1 diagram, where the concrete operation refines skip (second disjunct).

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 achieve a high degree of automation due to the more complex quantifier structure of ABINV compared to BINV.

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. Second the contract embedding [WD96] of partial relations is defined and corresponding proof rules for forward and backward simulation are derived. Third the embedding of input and output into the state is discussed.

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]. The third part is formalised 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*. Instantiating a subspecification PSPEC (the parameter) of a generic specification GSPEC with an actual specification ASPEC using a mapping σ (a morphism, that allows to instantiate sorts resp. operations with tuples of types resp. arbitrary expressions) requires to prove the axioms of $\sigma(\mathsf{PSPEC})$ over ASPEC. The resulting specification is $\sigma(\mathsf{GSPEC})$, with all theorems of ASPEC available as lemmas. Instantiating specifications is used to prove that forward and backward simulation imply refinement correctness and to prove

that the contract approach instantiates the original approach [HHS86] for total relations (sections 3 and 4 in [CSW02]).

While the specifications and proofs in project *DataRef* mimic the ones of chapter 2 and 3 of [CSW02], those in *Z-refinement* differ from the ones in chapter 4. 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. This is not a total relation, and leads to a partial initialisation relation. The correct definition should relate every sequence to the empty sequence (e.g. for $\mathsf{empty}[\mathsf{GO}, \mathsf{CO}]$ the global output GO of 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].

The correction results in an additional proof obligation (from the finalisation proof) for refinement correctness: every concrete input must be related to some abstract input via relation ι_{b}: Our web presentation calls this relation IT using the notation of theorem 10.5.2 in [DB01], which also requires totality. 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. More details on this can be found in the technical report [SGHR06].

Nevertheless it is possible to use invariants without restricting the state space, but a backward simulation theorem using invariants cannot be derived as an instance of the contract approach. Therefore we proved the following theorem directly by instantiating the original approach of [HHS86]. The theorem is given here for the approach without IO in a slightly simplified form with total initialisation and finalisation relations. The theorem with IO can be derived from this theorem with a similar proof as in [CSW02] (but with the corrected *empty* relation). It is given in the technical report [SGHR06]. The proof obligations can also be found as axioms without and with IO in the theories *conbackward-INV* in project *DataRef* 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)*
- $\mathrm{dom}(\mathsf{COP_i}) \lhd \mathsf{CINV} \subseteq \mathrm{dom}((\mathsf{T} \lhd \mathsf{AINV}) \lhd \mathrm{dom}(\mathsf{AOP_i}))$ *(applicability)*
- $\mathrm{dom}(\mathsf{T} \lhd \mathrm{dom}(\mathsf{AOP_i})) \lhd (\mathsf{COP_i} \fatsemi \mathsf{T}) \subseteq \mathsf{T} \fatsemi (\mathsf{AINV} \lhd \mathsf{AOP_i})$ *(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 those 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.

5 Verification of the Data Refinement

Our specification faithfully replicates the data types and operations of the original Mondex refinement. The operations are defined in the specifications *Mondex-AOP* and *Mondex-COP*. The only difference to the original Mondex refinement is that we used ASM rules as an auxiliary means to specify operations:

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

This equivalence defines the relation $\mathsf{OP}(\mathsf{cs}, \mathsf{cs}')$ to hold if and only if ASM rule OP# started with cs can compute cs′ as one possible result. Because of the relational semantics of programs, ASM rules adequately represent operation schemas: schema composition becomes sequential composition of programs. For example the composition $\mathsf{ABORT} \fatsemi \mathsf{STARTFROM} \lor \mathsf{IGNORE}$ is represented as $\langle \mathsf{ABORT\#};\ \mathsf{STARTFROM\#}\ \mathbf{or}\ \mathsf{IGNORE\#} \rangle$ where **or** is the nondeterministic choice between two programs. Compared to using operations on relations directly, using auxiliary ASM rules allows to execute programs symbolically (see [RSSB98]), which improves proof automation.

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 etherok(ether, ...), that is part of the definition of BINV, 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[5] 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 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.

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 generalise 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 four weeks needed for verification are not comparable to effort for the original case study, which had to develop the formal specifications, refinement notions and proofs from scratch: in private mail, Jim Woodcock sent us an estimation of 1.5 pages of specification/proof per day, which results in at least 10 person months of effort.

The task we solved here is the mechanisation of an already existing proof. This time was of course significantly reduced by having a (nearly) correct simulation, since usually most of the time is needed 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 proving each refinement needed on average about the same number of proof steps in KIV as the Mondex case study.

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

The ratio of interactions to proof steps is somewhat better in the Prolog 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 our proof effort shows that 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 ([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 ([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 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]). 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 think that the additional effort to do this was rather small compared to the effort needed to write down the 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 generalised 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.

[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/$\sim$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.

[Gur95] Yuri Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9 – 36. Oxford Univ. 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] Gavin 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] Lawrence 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.

[RSSB98] Wolfgang Reif, Gerhard Schellhorn, Kurt Stenzel, and Michael 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] Gerhard Schellhorn and Wolfgang 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] Gerhard Schellhorn and Wolfgang 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`.

[SGHR06] Gerhard Schellhorn, Holger Grandy, Dominik Haneberg, and Wolfgang Reif. The Mondex Challenge: Machine Checked Proofs for an Electronic Purse. Technical Report 2006-2, Universität Augsburg, 2006.

[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/`.