Journal of Universal Computer Science, vol. 14, no. 12 (2008), 1929-1948
submitted: 1/9/07, accepted: 1/6/08, appeared: 28/6/08 © J.UCS

ASM Refinement Preserving Invariants

Gerhard Schellhorn
(University of Augsburg, Germany
schellhorn@informatik.uni-augsburg.de)

Abstract: This paper gives a definition of ASM refinement suitable for the verification that a protocol implements atomic transactions. We used this definition as the basis of the formal verification of the refinements of the Mondex case study with the interactive theorem prover KIV. The refinement definition we give differs from the one we gave in earlier work which preserves partial and total correctness assertions of ASM runs. The reason is that the main goal of the refinement of the Mondex protocol is to preserve a security invariant, while total correctness is not preserved. To preserve invariants, the definition of generalized forward simulation is limited to the use of "small" diagrams, which contain of a single protocol step. We show a technique that allows to use the natural "big" diagrams that consist of an atomic action being refined by a full protocol run.
Key Words: Refinement, Forward Simulation, Data Refinement, Commuting Diagrams, Abstract State Machines, Mondex, Electronic Purses, Security, Dynamic Logic, Weakest Preconditions, Interactive Theorem Proving
Categories: D.2.1, D.2.4, D.3.1, D.4.6, F.3.1, F.4.1

1 Introduction

This paper gives a formal definition of the ASM refinement theory underlying the mechanized verification of the Mondex protocol with the interactive theorem prover KIV [Reif et al., 1998] that is described in [Schellhorn et al., 2008] and also used in the second refinement to a security protocol in [Haneberg et al., 2008].

Although the theory developed is rather different, our work can be viewed as a parallel to [Cooper et al., 2002], which also defines a suitable data refinement theory (by adapting the contract approach of [Woodcock and Davies, 1996] to the needs of Mondex: both a final state and input/output are used) for the original paper-and-pencil verification of Mondex in [Stepney et al., 2000].

The Mondex case study deals with Mondex smart cards that implement an electronic purse [MCI 2008]. The main content of the case study is a refinement of the atomic action "transfer money" into a communication protocol between two purses. More details on this protocol are given in Section 2. At first glance the refinement looks very similar to problems in compiler verification which typically also refine one atomic step — execution of one source code instruction — to several steps of executing corresponding assembler code.

But there are several differences: the refinement must preserve an invariant, the presence of an attacker means that the concrete level may have infinite runs of failed attacks (termination is not preserved), and protocol steps are local steps of individual purses which can be interleaved. These differences are explained in detail in Section 4. To cope with them the formal definitions of ASM refinement in [Schellhorn, 2001] and [Schellhorn, 2005] (which were based on Börger's concepts [Börger, 1990, 2003], that he introduced into the framework of Gurevich's ASMs [Gurevich, 1995; Börger and Stärk, 2003]) had to be modified.

The original formalization is summarized in Section 3 for comparison and to provide the necessary notation. The new definition of *invariant preserving ASM refinement* is given in Section 4 based a simple notion of transition systems. Such transition systems (with algebras as states) can be used to define the semantics of ASMs. Section 5 transfers the result to the syntactic level of ASM rules: Dynamic Logic [Harel et al., 2000] and wp-calculus [Dijkstra, 1976] are used for this purpose. Both levels have been formally specified in KIV and can be browsed as part of the Web presentation of the Mondex proofs [KIV 2006].

To preserve invariants forces the use of "small" 0:1 and 1:1 commuting diagrams in the definition of forward simulations which verify a single step of the protocol against the corresponding abstract step or against no step (in data refinement terminology these protocol steps implement skip). Section 6 shows a technique to define simulation relations that either look into the future or the past of a protocol run. This technique allows to reduce the verification tasks to proofs for the natural "big" 1:n diagrams that prove that 1 abstract transaction is implemented by n protocol steps.

Finally, Section 7 gives related work and Section 8 concludes.

2 Mondex Purses

Mondex smart cards implement an electronic purse [MCI 2008]. They are intended to replace cash money when paying in shops. Typing in some amount into a terminal with two smart card slots, money is transfered from one card to another.

Mondex cards are famous for having been the target of the first ITSEC evaluation of the highest level E6 [Certification Body, 1999], which requires formal specification and verification.

The formal specification and paper-and-pencil proofs were done in [Stepney et al., 2000] using the Z specification language. Two models of the 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. The underlying data refinement theory is described in [Cooper et al., 2002].

The Mondex case study has been recently proposed as a challenge for theorem provers that we and several other groups [Jones and Woodcock, 2008] have solved. Initially we followed the original approach and verified the original backward simulation [Schellhorn et al., 2006] which used data refinement with KIV. As a second step we used ASM refinement to develop another proof [Schellhorn et al., 2008]. The resulting (generalized forward) simulation relation and invariants were more systematic and proofs could be better automated in KIV, than those of data refinement. The experiment using ASMs and ASM refinement also lead us to discover a weakness of the protocol, that required a small change. A detailed description of this weakness can be found in [Schellhorn et al., 2008].

Our work also extends the case study by developing verified Java code [Grandy et al., 2008], and by research on the Software Engineering aspects [Moebius et al., 2007].

In this paper we focus on the underlying ASM refinement theory needed for the refinements. Therefore only a short summary of the main ideas of the Mondex protocol refinement is given.

Both levels of the refinement are defined as abstract state machines (ASM). The abstract level consists of the following simple rule that describes money transfer between two smart cards.

```
TRANSFER#
choose from, to, value, fail?
with $\mathsf{from} \neq \mathsf{to} \wedge \mathsf{value} \leq \mathsf{balance(from)}$
in if ¬ fail? then $\mathsf{balance(from)} := \mathsf{balance(from)} - \mathsf{value}$
                   $\mathsf{balance(to)} := \mathsf{balance(to)} + \mathsf{value}$
              else $\mathsf{balance(from)} := \mathsf{balance(from)} - \mathsf{value}$
                   $\mathsf{lost(from)} := \mathsf{lost(from)} + \mathsf{value}$
```

The ASM rule chooses two different purses from and to and an amount value of money, that should be transferred from the from purse to the to purse. The amount is checked not to be higher than the available money balance(from) of the from card. If the randomly chosen boolean variable fail? is false, the transfer is successful and appropriate updates are done on balance(from) and balance(to). The transfer may also fail for various reasons, like the card being pulled out of the card reader, power failure, or lack of sufficient memory on the card. An attack on the transfer protocol using faked cards or a faked terminal might also cause the money transfer to fail. In the abstract rule all these cases are represented as fail? = true, and value is added to lost(from) in this case. One key goal of the refinement is to design a protocol, that will keep track of the money in lost (so that it is never truly lost!). The main advantage of the abstract specification is that the two key security properties of the final protocol are trivially provable as invariants. First, money transfer will never generate money, as the sum of all values balance(purse) for all authentic purses will never increase. Second, the sum of all balance and lost values will remain constant in all transfers, meaning all lost money has been accounted.

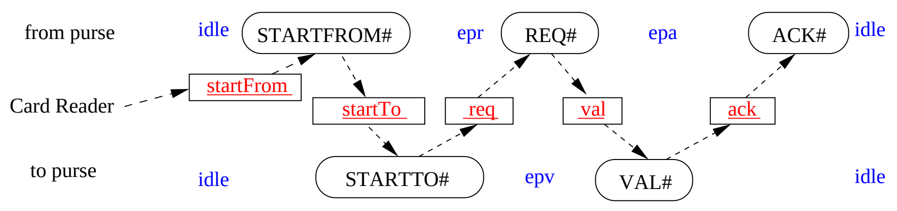

Figure 1: Activity chart for the Mondex protocol

The concrete level of Mondex defines a communication protocol between two purses. A successful protocol run is shown as an activity chart (drawn horizontally) in Fig. 1. The figure shows that the protocol uses five messages: the startFrom, startTo, req(uest), val(ue) and ack(nowledge) message. When a message is received the corresponding process step is taken, e.g. ASM rule REQ# is executed to process a request and to produce a value message. To keep track how far a purse has progressed in the protocol, the ASM uses a control state. Control state epa ("expecting acknowledge") indicates that the from purse has just sent the val message. A purse not running a protocol is in idle state.

The two first messages exchange relevant information to authenticate the two purses. The val message carries the money, so REQ# subtracts value from the from purse, and VAL# adds value to balance(to). Communication is done indirectly using a global set of messages, called the ether. Receiving a message is done by picking some message from ether, sending a message adds the message to ether. Receiving *any* message from ether, and not just picking the one that was previously sent, implicitly models an attacker that may record, replay or delete arbitrary messages. If a purse picks a message other than the one shown in the successful run in Fig. 1, it aborts the protocol.

When a to purse has sent a req message and entered state epv, but does not get a val message back it aborts the protocol run writing an exception log. In this case either req or val has been lost. Dually, the from purse creates an exception log, when it has received a req message and has sent the val, but does not get an ack in state epa. The key idea is that *both* exception logs are created if and only if the val message is lost. Losing the val message is the only case where money is lost, and is equivalent to the step on the abstract layer where money is added to lost. Therefore the abstract lost component is implemented as the sum of all values contained in pairs of matching exception logs. Purse owners who suffered from a failed transaction can go to the bank, show their two cards with matching exception logs, and the bank can deduce that it should give them back their money.

Details on the data structures used as local components of each purse state and the full ASM rules can be found in [Schellhorn et al., 2008]. For the ASM refinement theory the following facts should be remembered:

- Successful transfers are implemented by protocol runs as shown in Fig. 1
- Failed transfers are implemented by aborted protocol runs. There are aborted protocol runs which do not lose money. Those which do lose money create two matching exception logs.
- Protocols can be interleaved. A purse which has done its last protocol step can start a new protocol run even before the other purse has finished the run.
- The state of the ASM is composed of the states of the local purses, with the exception of the global set of messages ether
- The abstract level satisfies the invariant "no money generated or really lost", and the refinement should preserve this invariant.

3 ASM Refinement in Compiler Verification

ASM refinement theory was formalized in [Schellhorn, 2001] [Schellhorn, 2005] with compiler verification (in particular the Prolog-WAM compiler of [Börger and Rosenzweig, 1995]) as the intended target. In this section we will repeat key definitions and informally give the results of this work. This is done to establish the necessary notations for ASMs and commuting diagrams. They are needed to define the modifications necessary for the main result of the next section needed on protocol verification.

In compiler verification the semantics of the source code language is defined by giving an ASM $\mathsf{AM} = (\mathsf{AS}, \mathsf{AIN}, \mathsf{ARULE}, \mathsf{AOUT})$ that acts as an abstract interpreter of the source code language. AS are the states (algebras) of an ASM AM. The program source code and an initial program counter are stored in the possible initial states $\mathsf{AIN} \subseteq \mathsf{AS}$ of AM. One application of the ASM rule ARULE typically executes one source code instruction. Execution may terminate in a final state in AOUT. The compiler transforms the program to byte code or assembler code (again stored in the initial state CIN). An ASM $\mathsf{CM} = (\mathsf{CS}, \mathsf{CIN}, \mathsf{CRULE}, \mathsf{COUT})$ executes these instructions. A relation IR between the initial states gives compiler assumptions (this generalizes the case where a compiler function is given).

Typically one source code instruction is compiled to several assembler instructions (1:n diagrams), but occasionally it is useful to consider the more general case of m:n diagrams, where m source code instructions are compiled to n assembler instructions [Börger and Rosenzweig, 1995; Schellhorn and Ahrendt, 1998]. Corresponding states at the beginning and end of the diagrams are called *states of interest*. Refinement is defined relative to a correspondence relation IO

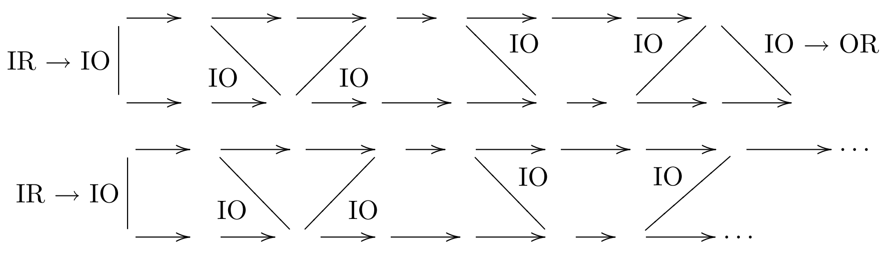

Figure 2: Refinement in Compiler Verification

that should hold for states of interest. IO typically states that inputs read and outputs produced are the same (relative to possible differences in representation) for both ASMs or that relevant parts of the states match. Given a finite concrete run, refinement requires that there must be a finite abstract run, such that IO holds initially and at the end of every diagram. For final states IO must imply some output relation OR, which typically says that executing both programs must have produced the same result. For infinite runs refinement requires that IO must hold infinitely often.

Fig. 2 gives the two situations of finite and infinite runs. It can be shown that ASM refinement preserves partial and total correctness assertions modulo the relation IO.

To verify refinements a generalized forward simulation INV is defined. This relation always strengthens IO and sometimes splits the diagrams of Fig. 2 into smaller ones. The proof obligation for generalized forward simulation propagates INV forwards and reads informally as:

> Given two states as and cs of AM and CM for which $\mathsf{INV}(\mathsf{as}, \mathsf{cs})$ holds, every run of CM starting with cs must reach a state cs' such that a commuting diagram can be closed: a run of AM from as to some as' must exist, such that $\mathsf{INV}(\mathsf{as}', \mathsf{cs}')$ holds again.

Typically the number of steps m and n of CM and AM that are needed to get the next commuting m:n diagram can be determined from the states as,cs. Of course m:n must not both be zero. Triangular diagrams are allowed, but infinite chains of 0:n and m:0 diagrams must be forbidden using well-founded orders $<_{\mathsf{0n}}$ and $<_{\mathsf{m0}}$, otherwise the correspondence of *finite* runs to *finite* runs cannot be established. Also, if an application of CRULE fails (due to clashes, or infinite recursion in TurboASMs; see [Börger and Stärk, 2003]) it must be possible to construct a run of AM that leads to a failed application of ARULE. The four cases of the proof obligation are shown in Fig.3. ARULE^+ is a positive number of applications of ARULE and the $\perp$ in case (C) of Fig.3 indicates a failed rule.

$$\begin{tikzcd}
\text{as} \arrow[r, dotted, "\vdash \text{ARULE}^+"] \arrow[d, dash, "\text{INV}"'] & \vdash \text{as}' \\
\text{cs} \arrow[ur, dotted, dash, "\vdash \text{INV}"'] &
\end{tikzcd}$$

$\vdash (\text{as}', \text{cs}) <_{0n} (\text{as}, \text{cs})$

(A)

$$\begin{tikzcd}
\text{as} \arrow[r, dotted, "\vdash \text{ARULE}^+"] \arrow[d, dash, "\text{INV}"'] & \vdash \text{as}' \arrow[d, dotted, dash, "\vdash \text{INV}"] \\
\text{cs} \arrow[r, "\text{CRULE}^+"'] & \text{cs}'
\end{tikzcd}$$

(B)

$$\begin{tikzcd}
\text{as} \arrow[r, dotted, "\vdash \text{ARULE}^+"] \arrow[d, dash, "\text{INV}"'] & \vdash \bot \\
\text{cs} \arrow[r, "\text{CRULE}"'] & \bot
\end{tikzcd}$$

(C)

$$\begin{tikzcd}
\text{as} \arrow[d, dash, "\text{INV}"'] \arrow[dr, dotted, dash, "\vdash \text{INV}"] & \\
\text{cs} \arrow[r, "\vdash \text{CRULE}^+"'] & \text{cs}'
\end{tikzcd}$$

$\vdash (\text{as}, \text{cs}') <_{m0} (\text{as}, \text{cs})$

(D)

Figure 3: The commuting diagrams of ASM refinement

Dashed lines, as well as states and formulas after "⊢" have to be shown to exist or to be provable.

Formal proof obligations using either temporal operators or Dynamic Logic can be found in [Schellhorn, 2005].

4 ASM Refinement Preserving Invariants

At first glance, verification of the Mondex protocol looks quite similar to compiler verification: an atomic abstract step – money transfer – is refined by n smaller, concrete steps (here the 5 steps of the Mondex protocol), but there are three key differences.

First, compiler verification typically preserves termination. Executing the compiled code of a program should result in an infinite run only if the semantics of the original program allowed an infinite run too. But the Mondex refinement does not preserve termination because of failed protocol runs: those failed runs which do not create two matching exception logs correspond to doing nothing on the abstract layer (in the terminology of data refinement they implement "skip" steps). This happens e.g. if an attacker deletes all request messages. Therefore the protocol refinement may create an infinite chain of triangular 0:n diagrams. It only preserves partial correctness, but not termination and total correctness.

For protocol refinement this is clearly acceptable, and various other forms of refinement also ignore it, e.g. Abadi and Lamport [Abadi and Lamport., 1991] (by extending finite runs to infinite runs using stuttering), refinement of IO automata [Lynch and Vaandrager, 1995] and data refinement. Both the contract [Woodcock and Davies, 1996] as well as the behavioral approach [Bolton et al., 1999] preserve total correctness of *single operations* (when interpreting $\perp$ to be nontermination), but not termination of runs. An example that adds infinite runs is given in [Schellhorn, 2008] which gives a completeness proof for ASM refinement. In contrast, refinement in Event-B [Abrial and Hallerstede, 2007] uses a well-founded relation for steps implementing skip to preserves termination.

Second, the main requirement of the Mondex refinement is not (as for compiled code) that the abstract and concrete states correspond in states of interest, where some code sequence has been executed. The critical properties the refinement must preserve are the two security invariants, that say that money is never generated nor lost.

The original ASM refinement only weakly preserves invariants: an invariant AINV of the abstract level will only hold (modulo the IO correspondence) in every concrete state of interest.

Unfortunately this is not sufficient to establish security on the concrete level, where *all* intermediate states during protocols should satisfy the security invariant.

Finally, the third difference is that protocol runs can be interleaved. Intuitively, this does not matter since two interleaved protocol runs will modify disjoint pairs of purses, and the messages one protocol run generates will not be usable in other runs. But in particular the second fact is essential for security: replay attacks would be possible if it were false. Therefore it is not sufficient to prove that a non-interleaved protocol run implements an abstract step.

Therefore, ASM refinement theory is specialized here to preserves invariants and to forward simulations that consist of n:1 diagrams only. Only these small diagrams propagate an abstract invariant (modulo IO) to *every* state of the protocol. For Mondex only 0:1 and 1:1 diagrams are used. We first define invariants:

Definition 1. [invariant]

A predicate $\mathsf{AINV}(\mathsf{as}_0, \mathsf{as})$ is an invariant of ASM AM, if

– $\mathsf{AINV}(\mathsf{as}_0, \mathsf{as}_0)$ holds for every initial state $\mathsf{as}_0 \in \mathsf{AIN}$
– $\mathsf{AINV}(\mathsf{as}_0, \mathsf{as})$ and $\mathsf{ARULE}(\mathsf{as}, \mathsf{as}')$ imply $\mathsf{AINV}(\mathsf{as}_0, \mathsf{as}')$.

This definition of invariants generalizes the usual one by allowing it to refer to the current state as *as well as* to the initial state as_0 from where the run started. This is necessary since the invariant of the abstract Mondex specification indeed compares the sum of balances of the initial state with those of the current state.

Definition 2. [Preservation of invariants]

Given two ASMs AM and CM, and correspondences IR, IO and OR between initial states, states of interest and final states, the refinement from AM to CM preserves invariants, if for every invariant AINV of AM

$$\mathsf{CINV}(\mathsf{cs_0}, \mathsf{cs}) := \exists\ \mathsf{as_0},\ \mathsf{as}.\ \mathsf{IR}(\mathsf{as_0}, \mathsf{cs_0}) \wedge \mathsf{IO}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{AINV}(\mathsf{as_0}, \mathsf{as})$$

is an invariant of CM.

Note that for IR and IO both being identity, the formula can be simplified to show that AINV itself is an invariant of CM.

For the Mondex case study IR and IO are identity for the $\mathsf{balance}$ of purses, but intersections of exception logs replace the lost component. Therefore the resulting invariant for the protocol is slightly more complex than the one of the abstract layer. Informally, the sum of money that is constant throughout all protocol runs is the sum of

– all balances of purses,
– all money that has been sent and can still be received (is “in transit”), and
– all money contained in matching pairs of exception logs.

If all purses are idle, the second summand is empty. Some details on the simulation relation IO for Mondex that is necessary to derive this invariant (by applying Theorem 3 below) will be given in the next section. A formal definition using predicates $\mathsf{intransit}$ and $\mathsf{lostafterabort}$ defined in [Schellhorn et al., 2008] can be found in the Web presentation of Mondex [KIV 2006] in specification *Refinement-preserves-SecProp*.

To verify that a refinement preserves invariants a generalized forward simulation INV is needed that uses n:1 diagrams only. This means that *all* concrete states will have a corresponding abstract state from which the invariant can be deduced using IO. As in [Schellhorn, 2001] ASM rules that never fail are considered first. For these the semantics of the ASM rule is a relation between states and the main proof obligation necessary to have a commuting now looks like

$$\begin{array}{lll} & \mathsf{INV}(\mathsf{as}, \mathsf{cs}) \wedge \mathsf{cs} \notin \mathsf{COUT} & \\ \rightarrow & \quad \mathsf{EF}^{+}(\mathsf{as}, \lambda\, \mathsf{as}'.\mathsf{INV}(\mathsf{as}', \mathsf{cs}) \wedge (\mathsf{as}', \mathsf{cs}) <_{\mathsf{m0}} (\mathsf{as}, \mathsf{cs})) & (1) \quad \text{(VC1)} \\ & \vee\ (\forall\ \mathsf{cs}'.\mathsf{CRULE}(\mathsf{cs}, \mathsf{cs}') \rightarrow \mathsf{EF}(\mathsf{as}, \lambda\, \mathsf{as}'.\mathsf{INV}(\mathsf{as}', \mathsf{cs}'))) & (2) \end{array}$$

For a pair of states with $\mathsf{INV}(\mathsf{as}, \mathsf{cs})$, where cs is not a final state in COUT the proof obligation requires, that one of the two possible commuting diagrams (1) and (2) may be added

– either there is a positive number of executions (temporal operator EF^{+}) of ARULE such that the state as' reached satisfies $\mathsf{INV}(\mathsf{as}', \mathsf{cs})$ and some well-founded order $<_{\mathsf{m0}}$ has been decremented (this prevents an infinite chain of m:0 diagrams) or

– for every state cs' that can be reached from cs by executing CRULE, there are some (possibly zero) abstract steps (operator EF) that lead from as to as' and $\mathsf{INV}(\mathsf{as}', \mathsf{cs}')$ holds at the end.

The first case allows finitely many m:0 diagrams just as case (A) of the original definition given in Fig. 3 does. The second case allows m:1 diagrams with $\mathsf{m} \geq 0$. This corresponds to cases (B) and (D) in Fig. 3. The requirement that a well-founded order decreases in 0:n diagrams has been dropped, since there is no need to preserve termination. In diagrams (B) and (D) there must now be one concrete step (CRULE) instead of any positive number (0:1 and m:1 diagrams instead of 0:n and m:n diagrams with $\mathsf{n} > 0$).

The proof obligations imply

Theorem 3. [Generalized forward simulation]

A refinement from AM to CM preserves invariants if

– $\forall\ \mathsf{cs} \in \mathsf{CIN}.\ \exists\ \mathsf{as} \in \mathsf{AIN}.\ \mathsf{IR}(\mathsf{as}, \mathsf{cs})$ (“initialize”)
– $\mathsf{IR}(\mathsf{as}, \mathsf{cs}) \rightarrow \mathsf{INV}(\mathsf{as}, \mathsf{cs})$ (“establish invariant”)
– verification condition (VC1) holds for nonfailing/potentially failing rules (“preserve simulation relation”)
– $\mathsf{INV}(\mathsf{as}, \mathsf{cs}) \rightarrow \mathsf{IO}(\mathsf{as}, \mathsf{cs})$ (“simulation relation implies refinement relation”)

Although the proof is rather tricky to do formally, since it involves infinite runs (the axiom of choice and a diagonalization argument are necessary to construct infinite abstract runs) the intuition behind the proof is simple: compose the commuting diagrams that (VC1) provides. The proof proceeds like the one in [Schellhorn, 2005], it is slightly simpler, since using m:n diagrams with n=1 allows to construct a corresponding abstract run to a finite concrete run by induction over the length of the concrete run.

5 Proving ASM Refinements Using Dynamic Logic

The theorems of the previous section were based on the semantics of ASMs as transition systems. Indeed, the KIV specifications [KIV 2006] have a first layer that defines refinement for transition systems with an arbitrary set S of states, where $\mathsf{RULE} \subseteq \mathsf{S} \times \mathsf{S}$ is a partial relation on this state set. Final states are those outside the domain of RULE. For ASMs the set of states consists of (first-order) algebras. An additional $\perp$ state is included to indicate a failed rule application.

To effectively reason with ASMs, KIV uses a higher-order variant of wp-calculus [Dijkstra, 1976], using notation from Dynamic Logic [Harel et al., 2000]. This Logic allows to reason with syntactic rules directly, avoiding the need to encode them in relational calculus.

Three operators are defined for programs α, which may be either one ASM rule or the iterated execution of rules in a loop:

– $[\alpha]\ \varphi$ means "all terminating runs of α end in a state where φ holds" and corresponds to $\mathsf{wlp}(\alpha, \varphi)$ in wp-calculus.
– $\langle\!|\alpha|\!\rangle\ \varphi$ means "all runs of α terminate and end in a state where φ holds" and corresponds to $\mathsf{wp}(\alpha, \varphi)$.
– $\langle\alpha\rangle\ \varphi$ means "there is a terminating run of α which ends in a state where φ holds" and is equivalent to $\neg\ \mathsf{wlp}(\alpha, \neg\ \varphi)$.

With these operator the relational definitions can be recast using a syntactic ASM rule $\mathsf{RULE(s)\#}$ [see 1] which modifies dynamic functions in s instead of its semantic relation RULE. As an example

$$\mathsf{CRULE}(\mathsf{cs}, \mathsf{cs}') \rightarrow \varphi(\mathsf{cs}') \text{ is replaced by } [\mathsf{CRULE\#}(\mathsf{cs})]\varphi(\mathsf{cs})$$

which means "all terminating executions of CRULE# result in a state where φ holds". Operators EF and AF can also be translated to this logic. For an always terminating rule $\mathsf{ARULE\#}$

$$\mathsf{EF}^{+}(\mathsf{as}, \lambda\,\mathsf{as}', \psi(\mathsf{as}')) \text{ becomes } \exists\ \mathsf{i}.\langle \mathsf{ARULE\#}(\mathsf{as})^{\mathsf{i+1}}\rangle\ \psi(\mathsf{as})$$

which says "some positive iterated execution of ARULE# will terminate and lead to a state with ψ".

For more general ASM rules that may fail a set of states is used which in addition to the algebras over some signature includes a special $\perp$ element to indicate failure. The semantic relation of an ASM rule is then a strict relation over such states. The fact, whether an ASM rule may fail or must fail can be encoded using divergence to represent all failures as:

$$\mathsf{mayfail(cs)} := \neg\ \langle\!|\mathsf{CRULE\#(cs)}|\!\rangle\ \mathsf{true},$$
$$\mathsf{mustfail(cs)} := \neg\ \langle\mathsf{CRULE\#(cs)}\rangle\ \mathsf{true}$$

Informally these predicates say: "not every run of CRULE# terminates (in a state where true holds)" and "no run of CRULE# terminates".

Using these definitions, the proof obligation using states with $\perp$ can be simplified such that all occurrences of $\perp$ are removed. The process of removing $\perp$ from proof obligations is in principle the same as in data refinement. It is complicated since m:1 instead of 1:1 diagrams are used, and because ASM rules already have an erratic semantics [deBakker, 1980] over states with $\perp$: data refinement operations are usually have a simpler semantics derived by either using the behavioral [Bolton et al., 1999] or the contract [Woodcock and Davies, 1996] embedding (see [Derrick and Boiten, 2001] for an overview), which in essence yields two specific types of ASM rules as described in [Schellhorn, 2005]. The resulting proof obligation is

[1] Adding a # sign is the KIV convention to distinguish ASM rules from predicates

$$\begin{array}{l} \mathsf{INV}(\mathsf{as},\mathsf{cs}) \land \mathsf{as} = \mathrm{as}_0 \land \neg\, \mathsf{cs} \in \mathsf{COUT} \\ \rightarrow \quad \exists\, \mathsf{i}.\ \langle \mathsf{ARULE\#}(\mathsf{as})^{\mathsf{i}+1}\rangle\ (\mathsf{INV}(\mathsf{as},\mathsf{cs}) \land (\mathsf{as},\mathsf{cs}) <_{\mathsf{m0}} (\mathrm{as}_0,\mathsf{cs})) \\ \quad \lor \quad [\mathsf{CRULE\#}(\mathsf{cs})]\ \exists\, \mathsf{i}.\langle \mathsf{ARULE\#}(\mathsf{as})^{\mathsf{i}}\rangle\ \mathsf{INV}(\mathsf{as},\mathsf{cs}) \\ \qquad \land (\mathsf{mayfail}(\mathsf{cs}) \rightarrow \exists\, \mathsf{i}.\langle \mathsf{ARULE\#}(\mathsf{as})^{\mathsf{i}}\rangle\ (\neg\, \mathsf{final}(\mathsf{as}) \land \mathsf{mayfail}(\mathsf{as}))) \end{array} \qquad \text{(VC2)}$$

It has an additional case (last line) which corresponds to case (C) of Fig. 3. Like before it can be proved

Theorem 4. [Generalized forward simulation for ASMs]
A refinement from AM to CM preserves invariants if

– $\forall$ cs $\in$ CIN. $\exists$ as $\in$ AIN. IR(as, cs) ("initialize")
– IR(as, cs) $\rightarrow$ INV(as, cs) ("establish invariant")
– verification condition (VC2) holds ("preserve simulation relation")
– INV(as, cs) $\rightarrow$ IO(as, cs) ("simulation relation implies refinement relation")

6 Simulation Relations that Look into the Future/Past

The refinement theory of the previous section restricts possible commuting diagrams to "small" m:1 diagrams (where in fact m will be 0 or 1) and to define a simulation relation that relates every intermediate protocol state to some abstract state. This is unfortunate, since the basic idea of protocol verification is to show commutativity of a "big" 1:5 diagram that consists of abstract money transfer and the 5 protocol steps.

The small diagrams are needed only to preserve the security invariant for every intermediate state, while the states of interest in protocol verification are only those states where every purse has control state idle, i.e. no protocol is running. In this section we show a technique that allows to essentially verify such a big diagram again. Although the formulas are explained for Mondex, the reader should have no trouble to see that the idea is applicable for any protocol refinement which consists of the local states of the individual participants and a global communication state. Section 7 will give several other examples where the same idea has been successfully applied. In Mondex the local states are the local purse states, and the global communication state is the set ether of messages that were already sent.

A first technical observation is, that even to verify the "big" 1:n diagram an invariant CINV of the protocol ASM is needed. Formally, an invariant is easily integrated into the forward simulation INV by defining

$$\mathsf{INV}(\mathsf{as},\mathsf{cs}) := \mathsf{ACINV}(\mathsf{as},\mathsf{cs}) \land \mathsf{CINV}(\mathsf{cs})$$

where ACINV is the "real" simulation relation (which is identical to the refinement relation IO in this case). Splitting INV allows to develop ACINV and CINV separately, both times using an instance of the technique described in the following. We will therefore talk about a "(coupling) invariant" when both ACINV and CINV are meant.

The idea is to first define a *simple* (coupling) invariant SCINV/SACINV that only needs to hold, when purses are idle. For Mondex, SACINV states that abstract and concrete balances are equal and that lost is the intersection of exception logs. SCINV needs to say nothing about local purse states, as they are irrelevant when no protocol is running. But it needs to state the security assumption for the global ether, which says that messages are suitably encrypted such that the ether never contains messages that will be used in future protocol runs. Indeed, if such messages would be available to an attacker, he could load these messages on a faked card, and could use this card to gain money. This property was not completely true for the original Mondex protocol, which lead us to discover the attack described in [Schellhorn et al., 2008]: the protocol described in Section 2 has therefore been slightly changed (the startTo message is now an encrypted answer to startFrom) from the original protocol in [Stepney et al., 2000]).

The main question to solve now is: how should a (coupling) invariant for intermediate states of the protocol be derived? The idea is to do this *schematically* using a (coupling) invariant that either *looks into the future or the past* of a protocol run. Two possibilities exist:

– Future: from the current state a future state is reachable, where the simple (coupling) invariant holds or
– Past: Every state is reachable from some state of interest, where the simple (coupling) invariant held

Looking into the future or past is easy to express in Dynamic Logic, since it just corresponds to iterated execution of CRULE#. For ACINV the resulting formulas are

– $\langle\!| \textbf{while} \neg \mathsf{SACINV}(\mathsf{cs}, \mathsf{as}) \textbf{ do } \mathsf{CRULE\#}(\mathsf{cs}) |\!\rangle \, \mathsf{SACINV}(\mathsf{cs}, \mathsf{as})$
– $\exists \, \mathsf{i}, \mathsf{cs}_0 . \mathsf{SACINV}(\mathsf{cs}_0, \mathsf{as}) \wedge \langle \mathsf{CRULE\#}(\mathsf{cs}_0)^{\mathsf{i}} \rangle \, \mathsf{cs}_0 = \mathsf{cs}$

Informally, these formulas say "all runs of protocol steps will eventually reach a state with SACINV" and "the current state has been reached by steps that started in a state of interest cs_0 where SACINV was true". For CINV the abstract state as has to be dropped. For a (coupling) invariant looking into the future verifying a commuting diagram for all but the first step of a protocol run creates a trivial 0:1 diagram, since after one step of the protocol the states that will be reached at the end of the protocol are still the same states that were reachable before the step. The only interesting steps to verify are initial states of a protocol, since the future states reachable after one step into the protocol are the ones at the end of the diagram. This means the proof obligation for this case will be exactly the 1:n diagrams that we intended to verify. Looking into the past dually creates trivial 0:1 diagrams for all but the last step of a protocol run, and the 1:n diagram for the last step.

For the simulation relation of Mondex, looking into the future is the simpler approach, since all purses can simply abort the protocol in one step. On the abstract level, corresponding actions are failed transfers for those purses from and to, where a val message containing an amount of money value has been sent by from, but not yet received by to. This set is called maybelost in Mondex. Therefore, ACINV(as, cs) is defined as

$$\langle\!|\textbf{forall}\ \mathsf{purses}\ \textbf{do}\ \mathsf{ABORT\#(purse)}|\!\rangle\ \langle\textbf{forall}\ (\mathsf{from}, \mathsf{to}, \mathsf{value}) \in \mathsf{maybelost}\ \textbf{do}\ \mathsf{TFAIL\#}\rangle\ \mathsf{SACINV}(\mathsf{as}, \mathsf{cs})$$

where TFAIL# is the case of TRANSFER# from Section 2 where $\mathsf{fail?} = \mathsf{true}$. The steps can be done in parallel (execution order does not matter) since they just set the local control state to idle and create (again local!) exception logs (in states epv and epa). The expression can be simplified to get the simulation relation already described informally in Section 2. The resulting formal definition is given in [Schellhorn et al., 2008].

The approach of looking into the future does not work for the invariant needed, since by resetting the control state to idle the abort step forgets all information about the current state, so the invariant will be trivial and not provide the information needed to verify the big 1:n diagram.

To get this information, it is necessary to look into its past. This can be done locally for each purse, except for the global ether.

$$\mathsf{CINV}(\mathsf{cs}) \leftrightarrow \forall\ \mathsf{purse}.\exists\ \mathsf{ps}_0.\ \mathsf{LCINV}(\mathsf{ps}_0, \mathsf{cs}(\mathsf{purse}), \mathsf{ether})$$

where ps_0 is the old local state of the purse at the beginning of the current protocol run and cs(purse) is the current local state. This state includes the control state of the purse, which is enough to give complete information as to how the current state was reached:

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{LCINV}(\mathsf{ps}_0, \mathsf{ps}, \mathsf{ether})} \\
:= & \textbf{case}\ \mathsf{controlstate}\ \textbf{of} \\
 & \quad \mathsf{idle} : \mathsf{ps}_0 = \mathsf{ps} \\
 & \quad \mathsf{epr} : \langle \mathsf{STARTFROM\#}(\mathsf{ps}_0)\rangle\ (\mathsf{ps}_0 = \mathsf{ps} \wedge \varphi_{\mathsf{epr}}) \\
 & \quad \mathsf{epv} : \langle \mathsf{STARTTO\#}(\mathsf{ps}_0)\rangle\ (\mathsf{ps}_0 = \mathsf{ps} \wedge \varphi_{\mathsf{epv}}) \\
 & \quad \mathsf{epa} : \langle \mathsf{STARTFROM\#}(\mathsf{ps}_0);\ \mathsf{REQ\#}(\mathsf{ps}_0)\rangle\ (\mathsf{ps}_0 = \mathsf{ps} \wedge \varphi_{\mathsf{epa}})
\end{array}$$

The formula just says (compare to Fig. 1): if the control state is idle, the purse has executed no step since the last state of interest ps_0, so the current state ps is equal to ps_0. For state epr a STARTFROM# has been executed to reach the current state, and similarly for the two other cases.

This approach would work without any problems if Mondex had local state only, the tricky bit is to handle the global state ether in the formulas φ_{epr}, φ_{epv} and φ_{epa}. Fortunately, the property that has to be stated about the ether in intermediate states can be derived systematically for every protocol that sends messages forth and back between two participants: it simply consists of three parts saying which messages *must* have been sent to reach the current control state, which response *may* have been sent, and which messages *have not* been sent in the current protocol.

Consider e.g. a purse p in control state epv (compare Fig. 1 again) that communicates with a purse p′. φ_{epv} says:

– startFrom, startTo and req have been sent
– val has been sent, iff the opposite purse p′ has sent it entering state epa and is either still there or has aborted the protocol run
– ack has not yet been sent

The other two formulas φ_{epr} and φ_{epa} are similar, formal definitions are in [Schellhorn et al., 2008].

The proof that the invariant CINV is preserved in steps of purse q then reduces to a lemma that the local LCINV is preserved for every p. When $p = q$, LCINV is trivially invariant for all steps into the protocol. The only interesting steps are aborting the protocol and finishing it with VAL# or ACK#. These reduce to a proof of the invariance of SCINV for full protocol runs.

If p and q are different, and q is not the opposite purse p′ to p in the current protocol, the invariance can be proved, since the states of the purses are disjoint and since the protocol run of q does not create usable messages for the protocol run of p and p′. For $q = p'$ the critical properties to be shown are the properties φ_{epr}, φ_{epv} and φ_{epa} of the ether. E.g. when p′ sends the val message and p expects it in epv, it enters epa so φ_{epv} remains true.

7 Related Work

Our work on Mondex is based on [Stepney et al., 2000] and a lot of related work will be available in [Jones and Woodcock, 2008] (see also [Schellhorn et al., 2008] for more work on this case study).

The definition of ASM refinement preserving invariants and its proof obligations for generalized forward simulation are closely related to refinement of IO automata [Lynch and Vaandrager, 1995]: an IO automaton can be directly encoded as an ASM, and using the identity relation on actions as IO shows that preserving partial correctness for the ASM is equivalent to refinement of the IO automata. The n:1 diagrams given here for forward simulation are the same as those for IO automata.

The idea of using local invariants is a common idea in ASM refinement, it is used e.g. as the core idea in [Börger and Mazzanti, 1996], which verifies refinements from sequential to pipelined execution of instructions of the DLX processor using local invariants for each pipeline stage. The idea is also not specific to ASM refinement, it can be found in other refinement notions, e.g. in work that relates promotion in Z specifications and data refinement (see [Derrick and Boiten, 2001] for an overview).

Our use of states of interest and the use of m:n diagrams seems rather particular to ASM refinement. It was used informally in [Börger and Rosenzweig, 1995] for the compilation of Prolog to WAM, in our formal proofs to verify them [Schellhorn and Ahrendt, 1998]. The term *states of interest* itself was coined in [Börger, 2003]. Past and future simulation relations are also an important idea of ASM refinement. One instance of a future simulation relation was used already in the proof of ASM refinement correctness (Corollary 2 in [Schellhorn, 2001]). In data refinement, coupled refinement [Derrick and Wehrheim, 2003] uses past states of interest [Schellhorn, 2005].

A formal definition of future and past simulation relations in a data refinement setting is given in [Banach and Schellhorn, 2007]. This paper also shows that a variety of combinations between both approaches is possible, and that the technique of *purse local* invariants used in this paper can be improved to use *protocol local* invariants. This improvement allows to develop a slightly simplified invariant compared to the one of Section 6, which does not need the formulas φ_{epr}, φ_{epv} and φ_{epa}. The improved technique was used in [Schellhorn and Banach, 2008], where the original Mondex refinement is split into three refinements, one for each of the important concepts.

For invariants the idea is closely related to using invariants that "sometimes" instead of "always" hold [Burstall, 1974]. Such invariants have been used in KIV for a long time [Heisel et al., 1989]. Recently our group used an invariant that looks into the future to analyze security protocols [Haneberg et al., 2007]. The idea there is to focus on future states *after* all possible attacks have been tried.

The general topic of refining atomic actions into protocols has similarities to many other refinement problems, e.g. the refinement of distributed transactions. This idea was taken up in the Event-B work on Mondex (see again [Jones and Woodcock, 2008]).

8 Conclusion and Further Work

In this paper we have defined invariant preserving ASM refinement, the theory underlying the Mondex protocol refinement in KIV. To preserve an invariant for all states, the theory is limited to m:1 diagrams. We have shown a systematic way, how to get from these small diagrams back to the intuitive 1:n diagrams, that refine one abstract step by the steps of a protocol run. The method is based on using simulation relations and invariants that look into the future or the past. The method results in extra conditions which show in essence, that interleaved executions of protocols do not interfere. Although the details of the invariants and simulation relations that result applying the method are specific to the Mondex protocol, all indications we have from other case studies suggest that the idea is applicable in many contexts.

References

[Abadi and Lamport., 1991] Abadi, M., Lamport., L.: "The existence of refinement mappings"; Theoretical Computer Science; 2 (1991), 253–284; also appeared as SRC Research Report 29.

[Abrial and Hallerstede, 2007] Abrial, J.-R., Hallerstede, S.: "Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B"; Fundamenta Informaticae; 77 (2007).

[Banach and Schellhorn, 2007] Banach, R., Schellhorn, G.: "On the refinement of atomic actions"; Proceedings of REFINE 2007; volume 201; 3 – 30; ENTCS, 2007.

[Bolton et al., 1999] Bolton, C., Davies, J., Woodcock, J.: "On the refinement and simulation of data types and processes"; K. Araki, A. Galloway, K. Taguchi, eds., Proceedings of the International conference of Integrated Formal Methods (IFM); 273–292; Springer, 1999.

[Börger, 1990] Börger, E.: "A Logical Operational Semantics for Full Prolog. Part I: Selection Core and Control"; E. Börger, H. Kleine Büning, M. M. Richter, W. Schönfeld, eds., CSL'89. 3rd Workshop on Computer Science Logic; volume 440 of LNCS; 36–64; Springer, 1990.

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

[Börger and Mazzanti, 1996] Börger, E., Mazzanti, S.: "A Practical Method for Rigorously Controllable Hardware Design"; J. Bowen, M. Hinchey, D. Till, eds., ZUM'97: The Z Formal Specification Notation; volume 1212 of LNCS; 151–187; Springer, 1996.

[Börger and Rosenzweig, 1995] Börger, E., Rosenzweig, D.: "The WAM—definition and compiler correctness"; C. Beierle, L. Plümer, eds., Logic Programming: Formal Methods and Practical Applications; Studies in Computer Science and Artificial Intelligence 11; 20–90; North-Holland, Amsterdam, 1995.

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

[Burstall, 1974] Burstall, R. M.: "Program proving as hand simulation with a little induction"; Information processing 74; (1974), 309–312.

[Certification Body, 1999] Certification Body, U.: "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.

[Cooper et al., 2002] Cooper, D., Stepney, S., Woodcock, J.: "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.

[deBakker, 1980] deBakker, J.: Mathematical Theory of Program Correctness; International Series in Computer Science; Prentice-Hall, 1980.

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

[Derrick and Wehrheim, 2003] Derrick, J., Wehrheim, H.: "Using Coupled Simulations in Non-atomic Refinement"; D. Bert, J. Bowen, S. King, M. Walden, eds., ZB 2003: Formal Specification and Development in Z and B; volume 2651; 127–147; Springer LNCS, 2003.

[Dijkstra, 1976] Dijkstra, E. W.: A Discipline of Programming; chapter 14; Prentice-Hall, Englewood Cliffs, N. J., 1976.

[Grandy et al., 2008] Grandy, H., Bischof, M., Schellhorn, G., Reif, W., Stenzel, K.: "Verification of Mondex Electronic Purses with KIV: From a Security Protocol to Verified Code"; Proceedings of the 15th International Symposium on Formal Methods (FM 2008); volume 5014 of LNCS; Springer, 2008.

[Gurevich, 1995] Gurevich, Y.: "Evolving algebras 1993: Lipari guide"; E. Börger, ed., Specification and Validation Methods; 9 – 36; Oxford Univ. Press, 1995.

[Haneberg et al., 2007] Haneberg, D., Grandy, H., Reif, W., Schellhorn, G.: "Verifying Smart Card Applications: An ASM Approach"; International Conference on integrated Formal Methods (iFM) 2007; volume 4591 of LNCS; Springer, 2007.

[Haneberg et al., 2008] Haneberg, D., Schellhorn, G., Grandy, H., Reif, W.: "Verification of Mondex Electronic Purses with KIV: From Transactions to a Security Protocol"; Formal Aspects of Computing; 20 (2008), 1.

[Harel et al., 2000] Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic; MIT Press, 2000.

[Heisel et al., 1989] Heisel, M., Reif, W., Stephan, W.: "A Dynamic Logic for Program Verification"; A. Meyer, M. Taitslin, eds., Logical Foundations of Computer Science; LNCS 363; 134–145; Logic at Botik, Pereslavl-Zalessky, Russia; Springer, Berlin, 1989.

[Jones and Woodcock, 2008] Jones, C., Woodcock, J., eds.: Formal Aspects of Computing; volume 20 (1); Springer, 2008.

[KIV 2006] KIV 2006: "Web presentation of the Mondex case study in KIV"; (2006); URL: http://www.informatik.uni-augsburg.de/swt/projects/mondex.html.

[Lynch and Vaandrager, 1995] Lynch, N., Vaandrager, F.: "Forward and Backward Simulations – Part I: Untimed systems"; Information and Computation; 121(2) (1995), 214–233; also: Technical Memo MIT/LCS/TM-486.b, Laboratory for Computer Science, MIT.

[MCI 2008] MCI 2008: Mondex; MasterCard International Inc. (2008); URL: http://www.mondex.com.

[Moebius et al., 2007] Moebius, N., Haneberg, D., Schellhorn, G., Reif, W.: "A Modeling Framework for the Development of Provably Secure E-Commerce Applications"; International Conference on Software Engineering Advances 2007; IEEE Press, 2007.

[Reif et al., 1998] Reif, W., Schellhorn, G., Stenzel, K., Balser, M.: "Structured specifications and interactive proofs with KIV"; W. Bibel, P. Schmitt, eds., Automated Deduction—A Basis for Applications; volume II; 13 – 39; Kluwer Academic Publishers, 1998.

[Schellhorn, 2001] Schellhorn, G.: "Verification of ASM Refinements Using Generalized Forward Simulation"; Journal of Universal Computer Science (J.UCS); 7 (2001), 11, 952–979.

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

[Schellhorn, 2008] Schellhorn, G.: "Completeness of asm refinement"; Proceedings of REFINE 2008; ENTCS, 2008; to appear.

[Schellhorn and Ahrendt, 1998] Schellhorn, G., Ahrendt, W.: "The WAM Case Study: Verifying Compiler Correctness for Prolog with KIV"; W. Bibel, P. Schmitt, eds., Automated Deduction—A Basis for Applications; volume III; 165 – 194; Kluwer, 1998.

[Schellhorn and Banach, 2008] Schellhorn, G., Banach, R.: "A concept-driven construction of the mondex protocol using three refinements"; Proceedings of ABZ 2008; LNCS; Springer, 2008; (to appear).

[Schellhorn et al., 2008] Schellhorn, G., Grandy, H., Haneberg, D., Moebius, N., Reif, W.: "A Systematic Verification Approach for Mondex Electronic Purses using ASMs"; U. G. J.-R. Abrial, ed., Dagstuhl Seminar on Rigorous Methods for Software Construction and Analysis; volume 5115 of LNCS; Springer, 2008.

[Schellhorn et al., 2006] Schellhorn, G., Grandy, H., Haneberg, D., Reif, W.: "The Mondex Challenge: Machine Checked Proofs for an Electronic Purse"; J. Misra, T. Nipkow, E. Sekerinski, eds., Formal Methods 2006, Proceedings; volume 4085 of LNCS; 16–31; Springer, 2006.

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

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