Under consideration for publication in Formal Aspects of Computing

Atomic Actions, and their Refinements to Isolated Protocols

Richard Banach[1] and Gerhard Schellhorn[2]

[1]School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk,
[2]Lehrstuhl für Softwaretechnik und Programmiersprachen,
Universität Augsburg, D-86135 Augsburg, Germany
schellhorn@informatik.uni-augsburg.de

Abstract. Inspired by the properties of the refinement development of the Mondex Electronic Purse, we view an isolated atomic action as a family of transitions with a common before-state, and different after-states corresponding to different possible outcomes when the action is attempted. We view a protocol for an atomic action as a computation DAG, each path of which achieves in several steps one of the outcomes of the atomic action. We show that in this picture, the protocol can be viewed as a relational refinement of the atomic action in a number of ways. Firstly, it yields a 'big diagram' simulation à la ASM. Secondly, it yields a 'small diagram' simulation, in which the atomic action is synchronised with an individual step along each path through the protocol, and all the other steps of the path simulate skip. We show that provided each path through the protocol contains one step synchronised with the atomic action, the choice of synchronisation point can be made freely. We describe the relationship between such synchronisations and forward and backward simulations. We relate this theory to serialisations of system runs containing multiple interleaved transactions, showing how the clean picture of the refinement of an isolated atomic action to an isolated protocol becomes obscured by the details of the interleaving. In effect, the fact that protocols are typically executed by a number of co-operating agents, not all of which embark on executing the protocol at the same moment, results in 'ragged starts' and 'ragged ends' to protocol instantiations, leading to potential overlaps between unrelated protocol instances that the theory must handle. We show how existing Mondex refinements embody the ideas developed, and describe a mechanical verification of the results presented.

Keywords: Atomic Actions, Protocols, Synchronisation, Serialisation, 2-Phase Protocols, Forward and Backward Simulation, Refinement, Mondex.

Correspondence and offprint requests to: Richard Banach, School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K. email: banach@cs.man.ac.uk

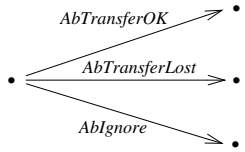

Fig. 1. The Mondex atomic actions.

1. Introduction

The Mondex Electronic Purse was developed formally in the mid-1990s using Z refinement. It was one of the first developments to achieve an ITSEC E6 security rating [DoTaI91].[1] Rather unusually for a commercial product, a sanitised version of the core of the formal development was made publicly available [SCW00]. Since then it has been a fertile ground for formal methods researchers — the original, human-built proofs of the security properties have been subjected to re-examination by contemporary techniques, and have stood up extremely well to the fiercest tool-based scrutiny achievable today, the first such mechanical verification being [SGHR06b].

The Mondex formal development featured a refinement proof from an atomic abstract model to a multi-step protocol at the concrete level. The principal component of this refinement proof was a backward simulation from abstract to concrete. At the time of the original development, the development team did try to construct a forward simulation, but were not successful — for a long time it was believed that a forward simulation refinement was impossible. It is nowadays known that a forward simulation is entirely possible, and more than one is now available in the literature [BPJS07, SGH$^+$07, HGS06]. The spur for the development of many of these was the Verification Grand Challenge, for which the mechanical verification of the Mondex protocol was the first major case study [JOW06, Woo06, WB07].

In this paper we explore the wider question regarding possible kinds of simulation for the refinement of an atomic action into a multi-step protocol, in order to settle the matter in the general case. We do this in the simplest possible relational framework in order to avoid complications that would distract from the main point.

In Mondex, the original refinement was done in a $(1, 1)$ manner, i.e. single concrete steps were made to refine single abstract ones. Consequently, since overall, there are more concrete steps than abstract ones, many concrete steps had to refine skip. Of course, one advantage of the $(1, 1)$ strategy is that, in the face of malevolent users or an unpredictable environment, the concrete protocol can be proved to refine the abstract atomic action, no matter how such a user might interrupt the intended playing out of the protocol — since every possible sequence of concrete steps that can be executed, corresponds to *some* abstract execution, even if it is one consisting entirely of skips.

In this, the original framework, the backward simulation correlated with an *early* synchronisation, i.e. the single non-trivial abstract step was $(1, 1)$ matched with a step that occurred early in protocol runs. By contrast, the more recently discovered forward simulations correlate with a *late* synchronisation, namely, the various possible non-trivial abstract steps are $(1, 1)$ matched with steps that occur late in protocol runs. Given the past uncertainty regarding forward and backward simulations in such contexts, our main aim in this paper is to give a general treatment.

The rest of this paper is thus as follows. In Section 2 we outline the operation of our motivating example, the Mondex Purse. In Section 3 we develop a theory of the refinement of a nondeterministic atomic action to a multi-step protocol in terms of computation DAGs. This explores the way that the single atomic action can be synchronised with an individual step of the protocol in a $(1, 1)$ refinement, and we see that there are a large number of possibilities for this which we call synchronisation assignments (SAs). We see that SAs are related to the possible choices of forward or backward simulations, according to the manner in which abstract outcomes are related to the details of the SA. In Section 4 we relate the rather abstract computation DAG view of protocols to a more conventional one, using event structures, and show that the histories generated by event structures yield computation trees in a natural way. In Section 5 we relate the preceding theory of an isolated protocol run to the more global picture needed to embed protocol runs into system runs, and we explore serialisability and the 2-phase property. Section 6 next, explores the relationship between serialisation and simulation. In particular it deals with the fact that protocol instantiations in a system run are normally executed by a collection of co-operating agents, not all of whom start and end the instantiation simultaneously, leading to overlaps of unrelated protocol instantiations which generates some technical complications. In the following Section 7 we apply the theory developed to the various refinements of Mondex available today,

[1] Nowadays, national standards like ITSEC have been superseded by the ISO Common Criteria standard [Int05]. The highest ITSEC level, E6, corresponds to the highest Common Criteria level, EAL7.

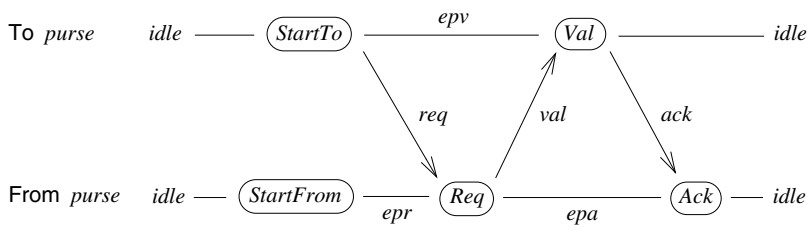

Fig. 2. The Mondex concrete protocol.

noting finally that there are in fact some non-2-phase corners of the original Mondex protocol (though none of them achieve anything observable at the abstract level, and are thus tolerated). The bulk of the preceding theory has been mechanically verified using KIV, and in Section 8 we review what has been achieved here. The final section concludes.

2. Mondex: A Motivating Example

Fundamentally, Mondex is a *smartcard purse*. Since it is a *purse*, it contains real money, and since it is a *smartcard*, it contains the money in digital form. This money is designed to be transferable from purse to purse. As for real money, the intention is that such transfers are normally performed in exchange for some desired purpose such as the purchase of goods or services, but equally —just as for real money— it is not the responsibility of the money itself to ensure that the transfer in which it engages is of a genuine nature. The only concern of money in general and of Mondex money in particular, is that it should be *unforgeable*.

The major objective of the original Mondex development was to develop a protocol for money transfer that ensured that:

1. Mondex money was unforgeable, even in the face of incomplete execution of the protocol or of malicious behaviour of the environment.
2. Any full or partial run of the protocol is equivalent to either a successful money transfer, or a traceably (and thus recoverably) lost-in-transit money transfer, or a null action.

These two properties are what make Mondex credible in the face of customer requirements: the first property, unforgeability, gives confidence in the value of Mondex money; while the second property, atomicity, gives comprehensibility when compared with the behaviour of conventional financial transactions. Fig. 1 shows the atomic abstraction that the Mondex protocol ensures, reflecting the three possibilities given in the above point 2. In Fig. 1 the nodes are states, and the arrows are the different atomic actions that the concrete protocol refines.

The essence of the Mondex concrete protocol is illustrated in Fig. 2 in activity diagram style. The source purse is the From purse while the destination purse is the To purse. The protocol begins with the two *Start* events (initiated from the environment as a result of the purses' owners typing in appropriate instructions at the interface device (the wallet) into which the two purses have been inserted). These are the *StartTo* event, performed by the To purse, and *StartFrom* event, performed by the From purse, both of which take their respective purse from the idle state to a 'busy' state: the *epr* state (expecting payment request) for the From purse, and the *epv* state (expecting payment value) for the To purse. The *StartTo* event sends a *req* message to the From purse. Upon arrival of the *req* message, the From purse performs a *Req* event and dispatches the money in a *val* message to the To purse, itself passing into the *epa* (expecting payment acknowledgement) state. Upon arrival of the *val* message, the To purse performs a *Val* event and sends an *ack* message to the From purse, itself passing back into the idle state. Receipt of the *ack* message in the *Ack* event by the From purse completes the protocol, and the From purse too passes back into the idle state. Note that in Fig. 2, the nodes are now events, edges are states, and arrows are messages.

The preceding described the workings of a successful run of the protocol. Beyond that, all events after the *Start* events can be replaced by *Abort* events, corresponding to runs of the protocol that are unsuccessful for whatever reason. The fact that despite *Abort* events, the protocol still enjoys the unforgeability and atomicity properties, is what makes Mondex non-trivial theoretically. However, the details of how this comes about do not concern us in this paper.

A further issue is that the Mondex protocol is *isolated*, i.e. once the protocol has commenced, the two purses take note only of the arrival of the next message expected in the playout of the protocol, and of calls to *Abort*, ignoring all other messages or calls from the environment and reserving the option of responding to such unexpected events by performing a self-initiated *Abort* whenever appropriate.

In this paper, rather than being concerned with *proving* that the atomicity and isolatedness properties are enjoyed by the protocol, we take properties such as these for granted, and instead, take an interest in simulation-theoretic properties —in a general sense, and for their own sake— of the refinement of an atomic action to a protocol with characteristics such as Mondex's. The isolated property makes these simulation-theoretic properties particularly convenient to study.

3. Isolated Atomic Actions and their Protocols

For both protocols and atomic actions, we will specify the transitions involved using a relational approach. The following statements summarise the assumptions we make about this setup.

Assumptions 3.1.

1. Relations are represented by predicates whose variables take values in suitable types.
2. Each relation used is deterministic, i.e. for each collection of values for the domain variables of the predicate representing the relation, there is a unique collection of values for the codomain variables that makes the relation true.
3. For each relation, for all values of domain and codomain variables that make the relation true, the domain values are reachable from an initial state.
4. Where nondeterminism (whether at the atomic or the protocol level) is needed, it is handled by having different relations for different outcomes.
5. Both atomic actions and protocols are represented by transition systems. At the atomic level, atomic actions are given by a collection of predicates whose interpretations are restricted to shallow computation forests (i.e. all maximal paths of length 1). At the protocol level, protocols are given by a collection of predicates whose interpretations are restricted to DAGs, all of whose paths are finite. A choice of initial state for a root of the interpreting forest of an atomic action picks out a unique tree, called the valid tree. A choice of an initial state for a root of the protocol DAG picks out a unique (maximal reachable) subDAG of the interpreting DAG, called the valid DAG.

Thus an atomic action will be specified by a (typically) finite collection of deterministic predicates $At_k(u,i,o,u')$ $k = 1\ldots$, in which u and u' are (variables denoting) the before- and after- states of the atomic action, i and o are the input and output of the action (these may in fact denote sequences, or more complex structures, of input and output values corresponding to the finer grained events in the protocol, if convenient), and the label k distinguishes the different deterministic outcomes for the same starting conditions. All together, the complete atomic specification of the protocol becomes:

$$Atomic(u,i,o,u') \equiv At_1(u,i,o,u') \vee At_2(u,i,o,u') \vee \ldots \tag{1}$$

where

$$(\forall\, u,i \bullet At_k(u,i,o_1,u'_1) \wedge At_k(u,i,o_2,u'_2) \Rightarrow o_1 = o_2 \wedge u'_1 = u'_2) \tag{2}$$

(and where it turns out that (2) is not actually needed in the ensuing mathematics, but helps for a convenient mental picture).

At the protocol level, the individual steps are described by a collection of deterministic predicates $St_\rho(v,j,p,v')$ where v and v' are the before- and after- states of the step, and j and p are the input and output of the step. The label ρ is an identifier which discriminates between different nondeterministic outcomes from the same before-state and input, and is required to be different for each step along a path through the protocol DAG,[2] but is otherwise available to conveniently label steps in an application-relevant way.

(Forward) paths through the protocol computation DAG are described by compound predicates:

$$\begin{aligned} &FPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(v_I,j_1,p_1,v_1,j_2,p_2,v_2,\ldots,v_{t-1},j_t,p_t,v_t) \equiv \\ &\quad St_\alpha(v_I,j_1,p_1,v_1) \wedge St_\beta(v_1,j_2,p_2,v_2) \wedge \ldots \wedge St_\gamma(v_{t-1},j_t,p_t,v_t) \end{aligned} \tag{3}$$

in which v_I is a possible initial state of the protocol, α labels a possible first step of the protocol, β labels a possible successor step of the α step of the protocol, and so on. As (3) indicates, if a step has a successor, the before-state of the successor must match the after-state of its predecessor. The length of the sequence of labels in the subscript of

[2] As for (2), determinism and path-uniqueness are not strictly necessary for ρ, but are conceptually convenient.

$FPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ must match both the number of inputs and outputs, and be one less than the number of states, in the argument list.

Maximal paths arise in the obvious way:

$$MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(\ldots) \equiv$$
$$FPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(\ldots) \wedge (\langle\alpha,\beta,\ldots,\gamma\rangle \text{ has no proper extension in the computation graph}) \tag{4}$$

From maximal and non-maximal paths, we can implicitly define a predicate *BPath* (backward paths) that describes extensions of non-maximal forward paths:

$$MPath_{\langle\alpha,\beta,\ldots,\gamma,\delta,\epsilon\ldots,\zeta\rangle}(v_I, j_1, p_1, v_1, \ldots, j_t, p_t, v_t, j_{t+1}, p_{t+1}, v_{t+1}\ldots, v_F) \equiv$$
$$FPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(v_I, j_1, p_1, v_1, \ldots, j_t, p_t, v_t) \wedge BPath_{\langle\delta,\epsilon\ldots,\zeta\rangle}(v_t, j_{t+1}, p_{t+1}, v_{t+1}\ldots, v_F) \tag{5}$$

In (5), v_F is a possible final state of the protocol.[3]

Finally, maximal paths give rise to the predicate $Protocol(v_I, js, ps, v_F)$, given by taking the disjunction over all maximal paths, existentially quantifying all intermediate states, and repackaging the inputs and outputs into sequences:

$$Protocol(v_I, js, ps, v_F) \equiv$$
$$\bigvee_{\substack{\text{maximal}\\ \langle\alpha,\beta,\ldots,\gamma\rangle}} \left(\begin{array}{l} (\exists\ j_1, p_1, v_1, j_2, p_2, v_2, \ldots, v_{t-1}, j_t, p_t \bullet \\ \quad MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(v_I, j_1, p_1, v_1, j_2, p_2, v_2, \ldots, v_{t-1}, j_t, p_t, v_F) \\ \quad \wedge\ js = \langle j_1, j_2, \ldots, j_t\rangle \wedge ps = \langle p_1, p_2, \ldots, p_t\rangle) \end{array} \right) \tag{6}$$

The fact that the protocol implements the atomic action is captured by relating the two via a retrieve relation *R*, input and output relations *Input* and *Output*, and demanding that an ASM-style [BS03] 'big-step' proof obligation holds. The retrieve relation is required to satisfy:

Assumptions 3.2.

1. $R(u, v)$ is a partial function from protocol states v to atomic states u. (7)
2. If v is a protocol state and v_{I1} and v_{I2} are initial protocol states, then
   $$FPath_{\langle\ldots\rangle}(v_{I1}\ldots v) \wedge FPath_{\langle\ldots\rangle}(v_{I2}\ldots v) \Rightarrow (\exists\ u_I \bullet R(u_I, v_{I1}) \wedge R(u_I, v_{I2})) \tag{8}$$
   (where u_I is obviously unique because of (7)).
3. $R(u, v)$ is 'not too big,' i.e. it concerns just the 'states of interest' for the overall protocol, i.e. the initial and final states:
   $$R(u, v) \Rightarrow (\exists\ js, ps, \tilde{v}\ \bullet Protocol(v, js, ps, \tilde{v}) \vee Protocol(\tilde{v}, js, ps, v)) \tag{9}$$

(As for (2), it turns out that (9) is not needed later, but helps for a convenient mental picture.) The big-step PO is now:

$$Protocol(v_I, js, ps, v_F) \Rightarrow$$
$$(\exists\ u_I, i, o, u_F\ \bullet R(u_I, v_I) \wedge Input(i, js) \wedge Atomic(u_I, i, o, u_F) \wedge Output(o, ps) \wedge R(u_F, v_F)) \tag{10}$$

Conditions (9) and (10) ensure that the hypotheses and conclusions of the big-step PO are valid exactly when the simulation predicate Σ:

$$\Sigma(u_I, i, o, u_F, v_I, js, ps, v_F) \equiv$$
$$Atomic(u_I, i, o, u_F) \wedge Protocol(v_I, js, ps, v_F) \wedge R(u_I, v_I) \wedge Input(i, js) \wedge Output(o, ps) \wedge R(u_F, v_F) \tag{11}$$

is true in the given types.

Now that we have connected together the atomic and finegrained descriptions of the protocol, our aim is to develop a general way of seeing how *some individual step* of a maximal path may be viewed as refining the atomic action, and the consequences of such a view. First we develop some technical machinery in the shape of past and future oriented retrieve relations — these show, in a generic way, how arbitrary points in the middle of the concrete protocol are related to the initial and final points of the abstract atomic action. Then we introduce synchronisation assignments, which delimit exactly how the choices of individual step within the protocol computation graph may be made. Finally we explore the consequences of these choices for proving the refinement via forward and backward simulation.

[3] Initial and final states of the protocol coincide exactly with the root and leaf states of the protocol computation graph.

First we get the 'past oriented' retrieve relation R^P:

$$R^P(u_I, v_t) \equiv (\exists\ v_I, j_1, p_1, v_1, \ldots, j_t, p_t, \langle\alpha, \beta, \ldots, \gamma\rangle \bullet R(u_I, v_I) \wedge FPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(v_I, j_1, p_1, \ldots, j_t, p_t, v_t)) \quad (12)$$

and the 'future oriented' retrieve relation R^F:

$$R^F(u_F, v_t) \equiv (\exists\ j_{t+1}, p_{t+1}, v_{t+1} \ldots, v_F, \langle\delta, \epsilon, \ldots, \zeta\rangle \bullet BPath_{\langle\delta,\epsilon\ldots,\zeta\rangle}(v_t, j_{t+1}, p_{t+1}, v_{t+1} \ldots, v_F) \wedge R(u_F, v_F)) \quad (13)$$

It is now easy to show the following:

Proposition 3.3.

$$R^P(u_I, v_t) \wedge R^F(u_F, v_t) \Rightarrow (\exists\ i, o\ \bullet\ Atomic(u_I, i, o, u_F)) \quad (14)$$

$$R^P(u_I, v_t) \Rightarrow (\exists\ i, o, u_F\ \bullet\ Atomic(u_I, i, o, u_F) \wedge R^F(u_F, v_t)) \quad (15)$$

$$R^F(u_F, v_t) \Rightarrow (\exists\ u_I, i, o\ \bullet\ R^P(u_I, v_t) \wedge Atomic(u_I, i, o, u_F)) \quad (16)$$

The proofs are similar to the proofs of the more interesting following result, obtained by replacing the 'V' consisting of R^P and R^F in (14)-(16) by a 'U' consisting of R^P, St_ρ and R^F in (17)-(19):

Theorem 3.4.

$$R^P(u_I, v_{t-1}) \wedge St_\rho(v_{t-1}, j_t, p_t, v_t) \wedge R^F(u_F, v_t) \Rightarrow (\exists\ i, o, js^P, js^F, ps^P, ps^F \bullet$$
$$Input(i, js^P::\langle j_t\rangle::js^F) \wedge Atomic(u_I, i, o, u_F) \wedge Output(o, ps^P::\langle p_t\rangle::ps^F)) \quad (17)$$

$$R^P(u_I, v_{t-1}) \wedge St_\rho(v_{t-1}, j_t, p_t, v_t) \Rightarrow (\exists\ i, o, u_F, js^P, js^F, ps^P, ps^F \bullet$$
$$\wedge\ Input(i, js^P::\langle j_t\rangle::js^F) \wedge Atomic(u_I, i, o, u_F) \wedge Output(o, ps^P::\langle p_t\rangle::ps^F) \wedge R^F(u_F, v_t)) \quad (18)$$

$$St_\rho(v_{t-1}, j_t, p_t, v_t) \wedge R^F(u_F, v_t) \Rightarrow (\exists\ u_I, i, o, js^P, js^F, ps^P, ps^F \bullet$$
$$R^P(u_I, v_t) \wedge Input(i, js^P::\langle j_t\rangle::js^F) \wedge Atomic(u_I, i, o, u_F) \wedge Output(o, ps^P::\langle p_t\rangle::ps^F)) \quad (19)$$

Proof. For (17), from $R^P(u_I, v_{t-1})$ we know that there is a path through the computation tree from an initial v_I to v_{t-1}, satisfying (3), and such that $R(u_I, v_I)$ holds. Evidently $St_\rho(v_{t-1}, j_t, p_t, v_t)$ extends that path. From $R^F(u_F, v_t)$ we know that there is a completion of this path to a maximal path from v_I to some final v_F. This maximal path enables us to derive $R(u_F, v_F)$, and provides the witnessing js^P, js^F, ps^P, ps^F so that with j_t, p_t we can assemble $js = js^P::\langle j_t\rangle::js^F$ and $ps = ps^P::\langle p_t\rangle::ps^F$, and then assert $Protocol(v_I, js, ps, v_F)$.

Since we have $Protocol(v_I, js, ps, v_F)$, we can apply (10). The conclusions of (10) yield $R(\tilde{u}, v_I)$ for some $\tilde{u}$; and since R is functional (7), we must have $u_I = \tilde{u}$. The conclusions of (10) also yield $Atomic(u_I, i, o, \tilde{u}')$ and $R(\tilde{u}', v_F)$ for some $\tilde{u}'$. Again, since R is functional, we must have $u_F = \tilde{u}'$. From $Protocol(v_I, js, ps, v_F)$ we can also deduce $Input(i, js)$ and $Output(o, ps)$.

For (18), the argument is similar except that we do not have to use the functional nature of R to argue $u_F = \tilde{u}'$, since u_F is existentially quantified in the conclusion.

For (19), we note first that by Assumptions 3.1.3, v_t is reachable from some initial v_I. We use this to assert a u_I such that $R^P(u_I, v_t)$ holds, after which we argue as for case (17). We are done. □

Theorem 3.4 is a crucial observation, since it enables an arbitrary protocol step $St_\rho(v_{t-1}, j_t, p_t, v_t)$ to be singled out and made to correspond with a suitable abstract one $Atomic(u_I, i, o, u_F)$. For such a $St_\rho(v_{t-1}, j_t, p_t, v_t)$ step, let $\mathrm{Outcomes}(St_\rho, u_I)$ (where the dependence on v_{t-1}, j_t, p_t, v_t is understood) be given by:

$$\mathrm{Outcomes}(St_\rho, u_I) = \{u_F \mid R^P(u_I, v_{t-1}) \wedge St_\rho(v_{t-1}, j_t, p_t, v_t) \wedge R^F(u_F, v_t)\} \quad (20)$$

and $\mathrm{OD}(St_\rho, u_I)$ (outcome determinism of St_ρ, given u_I) be given by:

$$\mathrm{OD}(St_\rho, u_I) = \mid \mathrm{Outcomes}(St_\rho, u_I) \mid \quad (21)$$

If $\mathrm{OD}(St_\rho, u_I) = 1$ we say that St_ρ is outcome deterministic at u_I (St_ρ is OD at u_I), whereas if $\mathrm{OD}(St_\rho, u_I) > 1$ we say that St_ρ is outcome nondeterministic at u_I (St_ρ is ON at u_I).

Definition 3.5. Given an initial v_I, a synchronisation assignment ($\mathrm{SA}(v_I)$) for the relevant valid DAG of a protocol computation DAG is a subset of its steps, such that for each maximal path through the valid DAG from v_I, exactly one of its steps is in $\mathrm{SA}(v_I)$. Steps in $\mathrm{SA}(v_I)$ are called SA steps.

Fig. 3 shows a synchronisation assignment. The many-level computation graph at the bottom (which happens to be a tree) has thickened arrows which are the elements of the SA. The atomic action is at the top and plays no specific

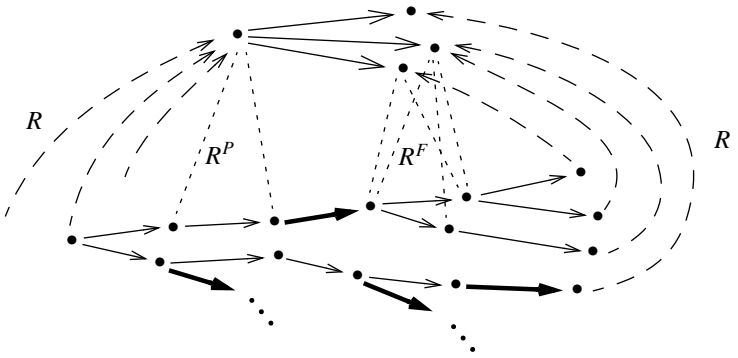

Fig. 3. A synchronisation assignment for a computation tree. The elements of the synchronisation assignment are shown bold.

part in the SA itself. Dashed arrows show the functional big-step retrieve relation R, while the dotted lines show some pieces from the R^P and R^F relations, for convenience below.

Definition 3.6. Given a protocol computation graph, an intial state v_I for the protocol, the atomic intial state u_I such that $R(u_I, v_I)$ holds, and a synchronisation assignment for the valid DAG determined by v_I, the steps of the valid DAG are classified as follows:

1. If a step is in the SA and is OD at u_I, it is called an outcome deterministic forward synchronisation (ODFS) step.
2. If a step is in the SA and is ON at u_I, it is called an outcome nondeterministic forward synchronisation (ONFS) step.
3. If a step is an immediate or later successor of an ONFS step, it is called a backward skip (BS) step.
4. Every step not covered by 1-3 is called a forward skip (FS) step.

This definition shows that every path through the protocol computation tree can be described by the following regular expression:

$$FS* \; ; \; (\; ODFS \; ; \; FS* \; + \; ONFS \;) \; ; \; BS* \tag{22}$$

Our aim is to show that when given a big-diagram refinement of an atomic action to a protocol of the kind we have described, if we wish to break the big-diagram refinement down into a collection of small-diagram refinements of zero or one atomic action steps to individual steps of the protocol, one can always use forward simulation reasoning, except for the BS steps. In fact one can use forward simulation reasoning for all steps except *branching BS steps* (a term explained below), though it comes at a price. Likewise, we have the option of using backward simulation reasoning for *all* steps if we so wish. We discuss these points later.

By an 'operation' we understand a transition relation such as those we have been using hitherto, but without assuming any specific additional properties.

Definition 3.7. Assume given an abstract operation $AOp(u, i, o, u')$, a concrete operation $COp(v, j, p, v')$, and retrieve, input and output relations, $R^1(u, v)$, $In^1(i, j)$ and $Out^1(o, p)$. Then *AOp* forward simulates *COp* iff:

$$R^1(u,v) \wedge COp(v,j,p,v') \Rightarrow (\exists \; i,o,u' \; \bullet \; In^1(i,j) \wedge AOp(u,i,o,u') \wedge Out^1(o,p) \wedge R^1(u',v')) \tag{23}$$

And *AOp* backward simulates *COp* iff:

$$COp(v,j,p,v') \wedge R^1(u',v') \Rightarrow (\exists \; u,i,o \; \bullet \; R^1(u,v) \wedge In^1(i,j) \wedge AOp(u,i,o,u') \wedge Out^1(o,p)) \tag{24}$$

In both cases, $In^1(i, j)$ and/or $Out^1(o, p)$ can be omitted where there is no input and/or output from *AOp* and/or *COp*, as applicable.

Theorem 3.8. Let there be a big-step refinement of an atomic action *Atomic* to a protocol *Protocol*, given by a retrieve relation R and input and output relations *Input* and *Output*, so that (10) holds. Let v_I be a fixed initial state such that $R(u_I, v_I)$ holds, and let $\mathrm{SA}(v_I)$ be a synchronisation assignment for the valid DAG rooted at v_I. Then the refinement of *Atomic* to *Protocol* can be decomposed into single step simulations such that:

1. If an FS step occurs before an SA step, it is forward simulated by the identity operation on u_I.
2. If an FS step occurs after an SA step, it is forward simulated by the identity operation on u_F, where u_F is some outcome of *Atomic*.
3. If St_ρ is an SA step, it is forward simulated by $Atomic(u_I, i, o, u_F)$ for every u_F in Outcomes(St_ρ, u_I).
4. Every BS step is backward simulated by the identity operation on some u_F.

Proof. We start by defining R^1, which is:

$$R^1(u,v) \equiv \left(\begin{array}{l} \exists \text{ a maximal path from some initial } \tilde{v}_I, \text{ and} \\ \left(\begin{array}{l} (v \text{ precedes an SA step along this path, and } R^P(u,v) \text{ holds}) \vee \\ (v \text{ follows an SA step along this path, and } R^F(u,v) \text{ holds}) \end{array} \right) \end{array} \right) \tag{25}$$

Also, we must define the single step input and output relations In^1 and Out^1; these however are only needed for the SA steps themselves.

$$In^1(i,j) \equiv (\exists \text{ an SA step } St_\rho(v_{t-1}, j, p_t, v_t), js^B, js^F \bullet Input(i, js^P :: \langle j \rangle :: js^F)) \tag{26}$$

$$Out^1(o,p) \equiv (\exists \text{ an SA step } St_\rho(v_{t-1}, j_t, p, v_t), ps^B, ps^F \bullet Output(o, ps^P :: \langle p \rangle :: ps^F)) \tag{27}$$

In fact we prove slightly more than we strictly need.

For 1, let $St_\rho(v_{t-1}, j_t, p_t, v_t)$ be the FS step in question. Since the SA is defined with respect to paths reachable from v_I, and FS steps are defined with respect to the SA, v_{t-1} must be reachable from v_I. To prove forward simulation, assume $R^1(u, v_{t-1})$ holds. Then there is a maximal path from some initial $\tilde{v}_I$ that reaches v_{t-1} such that $R^P(u, v_{t-1})$ holds. From (12) there is a path from some initial $\tilde{\tilde{v}}_I$ that reaches v_{t-1} such that $R(\tilde{u}_I, \tilde{\tilde{v}}_I)$ holds for some initial $\tilde{u}_I$. By (7) and (8), $\tilde{u}_I = u = u_I$. So in fact $R^1(u_I, v_{t-1})$ and $R^P(u_I, v_{t-1})$ both hold. Since $St_\rho(v_{t-1}, j_t, p_t, v_t)$ obviously extends the paths that witness $R^P(u_I, v_{t-1})$, the extensions witness $R^P(u_I, v_t)$ and $R^1(u_I, v_t)$ too, which is what is required for forward simulation of the identity on u_I.

For 2, let $St_\rho(v_{t-1}, j_t, p_t, v_t)$ be the FS step in question. Since it occurs after an SA step, it must again be reachable from v_I. To prove forward simulation, assume $R^1(u, v_{t-1})$ holds. Then there is a maximal path from some initial $\tilde{v}_I$ that reaches v_{t-1} such that $R^F(u, v_{t-1})$ holds. From (13) there is a path from v_{t-1} to some final v_F such that $R(u_F, v_F)$ holds, where u_F is the unique abstract outcome, that witnesses that the SA step that $St_\rho(v_{t-1}, j_t, p_t, v_t)$ follows, is outcome deterministic. By (7), $u = u_F$, so that $R^F(u_F, v_{t-1})$ holds, whereby $R^1(u_F, v_{t-1})$ holds too. Truncating the first step of the path from v_{t-1} to v_F that witnesses $R(u_F, v_F)$, gives a path that witnesses $R^F(u_F, v_t)$ and hence $R^1(u_F, v_t)$, which is what is required for forward simulation of the identity on u_F.

For 3, let $St_\rho(v_{t-1}, j_t, p_t, v_t)$ be the SA step in question. Obviously it is reachable from v_I. To prove forward simulation, assume $R^1(u, v_{t-1})$. Then we can deduce $R^1(u_I, v_{t-1})$ and $R^P(u_I, v_{t-1})$ exactly as in case 1. For any u_F in Outcomes(St_ρ, u_I), we know that $Atomic(u_I, i, o, u_F)$ holds. Also, we can deduce $R^F(u_F, v_t)$ and hence $R^1(u_F, v_t)$ exactly as in case 2. Since $St_\rho(v_{t-1}, j_t, p_t, v_t)$ occurs on a maximal path from v_I to some final v_F, the totality of inputs along the path, both js^P before j_t, and js^F after j_t, will witness that $Input(i, js^P :: \langle j_t \rangle :: js^F)$ holds, giving $In^1(i, j_t)$ as required. The reasoning for outputs is similar. So we have all the conclusions of (23), which is what is required for forward simulation of $Atomic(u_I, i, o, u_F)$.

For 4, let $St_\rho(v_{t-1}, j_t, p_t, v_t)$ be the BS step in question. Since it occurs after an SA step, it must be reachable from v_I. To prove backward simulation, assume $R^1(u, v_t)$ holds. Then there is a maximal path from some initial $\tilde{v}_I$ that reaches v_t such that $R^F(u, v_t)$ holds. From (13) there is a path from v_t to some final v_F such that $R(u_F, v_F)$ holds, where u_F is some abstract outcome, that witnesses that the SA step that $St_\rho(v_{t-1}, j_t, p_t, v_t)$ follows, is outcome nondeterministic. By (7), $u = u_F$ for some such u_F, so let us assume that $R^F(u_F, v_t)$ holds, whereby $R^1(u_F, v_t)$ holds too. Prepending $St_\rho(v_{t-1}, j_t, p_t, v_t)$ to the path from v_t to v_F that witnesses $R(u_F, v_F)$, gives a path that witnesses $R^F(u_F, v_{t-1})$, and hence $R^1(u_F, v_{t-1})$, which is what is required for backward simulation of the identity on u_F. □

Since at the abstract level, the transpose of the step relation is a partial function, backward simulation is always aligned with a decrease of nondeterminism in both abstract and protocol transition functions. Therefore we get the following (cf. [LV93]).

Corollary 3.9. Under the assumptions of Theorem 3.8, one can always use single step backward simulations throughout.

Corollary 3.9 might seem strange in the light of the well known fact that backward simulation alone is not complete for data refinement. The explanation comes from the fact that we have an asymmetry between forward and backward

directions in our setup. While we can never lose 'abstract backward nondeterminism' by simulating the protocol backward (due to (8)), we *can* lose 'abstract forward nondeterminism' by simulating the protocol forward. We also have the following.

Corollary 3.10. Under the assumptions of Theorem 3.8, suppose there are no BS steps (i.e. all SA steps are OD). Then single step forward simulations can be used throughout.

Obviously, choosing the SA as the last step of each maximal path through the protocol satisfies the hypotheses of Corollary 3.10.

Corollary 3.11. Let $MPath(v_I, \ldots, v_F)$ be a maximal path from an initial v_I to a final v_F, such that (10) holds (for suitably chosen other quantities). Let $St_\rho(v_{t-1}, j_t, p_t, v_t)$ be the SA(v_I) step along $MPath(v_I, \ldots, v_F)$. Then the simulation of $MPath(v_I, \ldots, v_F)$ by $Atomic(u_I, i, o, u_F)$ can be decomposed as follows:

1. If $St_\rho(v_{t-1}, j_t, p_t, v_t)$ is an ODFS step, the simulation of $MPath(v_I, \ldots, v_F)$ may be established by inductively forward simulating the steps of $MPath(v_I, \ldots, v_F)$ from v_I up to a state $v_{\bar{t}}$ (which does not precede v_t), and backward simulating the steps of $MPath(v_I, \ldots, v_F)$ from v_F up to $v_{\bar{t}}$ (if $v_{\bar{t}} \neq v_F$), such that:
   (a) predecessors of $St_\rho(v_{t-1}, j_t, p_t, v_t)$ are forward simulated by the identity operation on u_I,
   (b) $St_\rho(v_{t-1}, j_t, p_t, v_t)$ is forward simulated by $Atomic(u_I, i, o, u_F)$ where u_F is the unique element of Outcomes(St_ρ, u_I), establishing $R^F(u_F, v_t)$,
   (c) FS successors of $St_\rho(v_{t-1}, j_t, p_t, v_t)$ are forward simulated from v_t by the identity operation on u_F, establishing $R^F(u_F, v_{\bar{t}})$,
   (d) BS successors of $St_\rho(v_{t-1}, j_t, p_t, v_t)$ are backward simulated from v_F by the identity operation on u_F, establishing $R^F(u_F, v_{\bar{t}})$.
2. If $St_\rho(v_{t-1}, j_t, p_t, v_t)$ is an ONFS step, the simulation of $MPath(v_I, \ldots, v_F)$ may be established by inductively forward simulating the steps of $MPath(v_I, \ldots, v_F)$ from v_I up to and including $St_\rho(v_{t-1}, j_t, p_t, v_t)$, and inductively backward simulating the steps of $MPath(v_I, \ldots, v_F)$ from v_F up to v_t, such that:
   (a) predecessors of $St_\rho(v_{t-1}, j_t, p_t, v_t)$ are forward simulated by the identity operation on u_I,
   (b) $St_\rho(v_{t-1}, j_t, p_t, v_t)$ is forward simulated by $Atomic(u_I, i, o, u_F)$, for each u_F in Outcomes(St_ρ, u_I), establishing $R^F(u_F, v_t)$,
   (c) successors of $St_\rho(v_{t-1}, j_t, p_t, v_t)$ are backward simulated from v_F by the identity operation on u_F, establishing $R^F(u_F, v_t)$.

Why are the above results useful? We can give a couple of reasons.

Firstly, they are illuminative. One can be convinced of the correctness of a protocol with respect to an atomic action, without having the details of a refinement already worked out. In such a situation, it may not be clear how to synchronise the atomic action with the lower level description. Theorem 3.8 shows that one can choose this synchronisation relatively freely, within the parameters of allowable synchronisation assignments.

Secondly, once having chosen a synchronisation, it is much easier to write down the 'big-step' retrieve relation and associated input and output relations, than to discover the more finegrained single step ones. Theorem 3.8 shows that with the big-step retrieve relation fixed, the single step ones, R^P and R^F may simply be *calculated*. Their generic form needs to be instantiated with the details of the protocol and big-step retrieve relation, and then one must eliminate as many existential quantifiers as possible in order to arrive at a closed form. Making clear that there *is* such a strategy to follow is a considerable improvement over the hit-and-miss approach one would otherwise need, especially when combined with uncertainty regarding synchronisation.

The theorem and its corollaries also provoke the following considerations.

One can replace some backward simulation by forward simulation. Given a synchronisation assignment, a branching BS step is a BS step $St_\theta(v_s, \ldots, v'_{s,1})$ for which there is another BS step $St_\phi(v_s, \ldots, v'_{s,2})$ (with $v'_{s,1} \neq v'_{s,2}$) such that the abstract outcomes $u_{F,1}, u_{F,2}$ corresponding to the completions of the paths from $v'_{s,1}$ and $v'_{s,2}$ are different, $u_{F,1} \neq u_{F,2}$.[4] In such a case, one *cannot* make a forward simulation inference succeed.

[4] Since we speak of a BS step, there must exist such $u_{F,1} \neq u_{F,2}$, as the nondeterminism in $Atomic(u_I, i, o, u_F)$ has been resolved earlier than at this BS step.

To see this, recall that forward simulation demands that (23) can be proved *no matter what* instantiations one chooses for the hypotheses. Now, suppose the first hypothesis of (23) is made true by $R^1(u_{F,1}, v_s)$, and the second hypothesis is made true by $St_\phi(v_s, \ldots, v'_{s,2})$. Then the first hypothesis demands that u_F be chosen to be $u_{F,1}$, while the second hypothesis demands that u_F be chosen to be $u_{F,2}$, a contradiction. This is the standard backward simulation counterexample.

In Fig. 3, the SA element along the upper thread of the computation tree is an ONFS step, since it can reach two concrete final states that retrieve to two different abstract outcomes. Accordingly, the two BS steps immediately following it (and the two following the topmost of them along the upper thread) are branching BS steps, since they too can individually reach different concrete final states that retrieve to the two different abstract outcomes. With the dotted lines depicting R^F, it is easy to see that these steps illustrate what we have just discussed.

However, if a BS step is *not* branching, i.e. there is only one protocol successor state v'_s to v_s, then the preceding problem cannot arise since the unique successor cannot force a distinction between the choices for u_F. So for non-branching BS steps, a forward simulation inference will succeed. However, it comes at a price. If a forward simulating BS step immediately follows a backward simulating BS step, the $R^1(u_F, v)$ value at the v state that they share, occurs as a hypothesis in both the backward PO (24) and the forward PO (23). It thus remains as an unproved assumption in the overall single-step verification of the big-step refinement. As such it allows the verification to succeed vacuously (i.e. using the 'don't care' interpretation of the POs' implications that comes into play when their hypotheses are false). For this reason we phrased Corollary 3.11.2 as two inductive processes that meet in the middle, since it is much better to verify some $R^1(u_F, v)$ twice independently, than to leave some other $R^1(u_F, v)$ unproved, thus undermining the whole verification.

Lastly, Theorem 3.8 offers a different strategy for addressing global correctness. Normally, to prove a protocol (such as the one we have been considering) globally correct, one chooses either forward or backward simulation, establishes that each protocol step refines some atomic option or skip, and this then extends to an inductive proof for global executions as a whole. With Theorem 3.8, we can envisage a different approach, structured as follows.

1. We first study the 'big-step' refinement of atomic action to protocol, determining the protocol computation DAG and the big-step retrieve relation.
2. Next we choose a suitable synchronisation assignment.
3. Next we determine which combination of forward and backward simulations are appropriate for the synchronisation assignment.
4. Next we calculate the necessary single step retrieve relation, breaking down the big-step refinement into single step refinements.
5. Finally, we determine how runs of the protocol can interleave to make global executions.

While the first four of these points have been discussed above, the fifth point is elaborated in the remainder of the paper (see especially Sections 5 and 6). The alternative approach advocated, separates concerns, and in cases where a complex protocol is concerned, may offer some advantages. In any event, the mere awareness of the possibility of such an approach may make the more monolithic standard approach more tractable, since it can show that certain subgoals of the standard approach are achievable in advance.

In choosing between the new approach and the traditional approach, it is important to appreciate that the choice represents a value judgement about the very notion of what it means for a protocol to be correct. The standard inductive approach, and the more separated-concerns approach developed in this paper, are (mathematically) different statements, though not unrelated as discussed at the end of Section 6. Both however offer the same coverage of concrete steps, and both establish simulations between protocol and atomic action, albeit in different ways. We make further remarks to this effect below.

4. Event Structures and Protocol Computation Trees

Step 1 of the alternative verification strategy just suggested relies on determining the protocol computation DAG. Usually, consideration of this computation structure is not itself the means by which a protocol is invented, so the computation DAG might well be derived from alternative starting points.

A common way of inventing a protocol is to say 'this happens after that' for a sufficiently large number of cases. Such a train of thought can be formalised quite effectively using event structures of various kinds [WN95, Bou90, NPW81, Win86, Win88, BC88, PP95]. Accordingly, we use event structures with symmetric conflict relations to encode

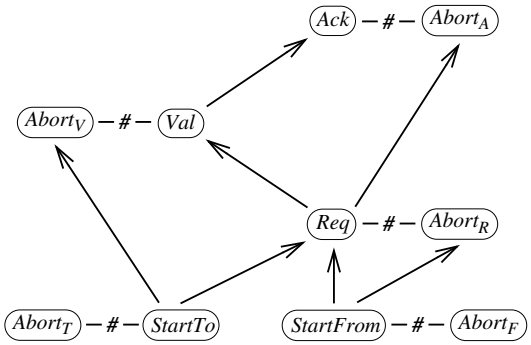

Fig. 4. An event structure for the Mondex protocol.

possible playouts of a protocol, and show how to derive a computation tree from an underlying event structure of this kind. Once there, one can map the tree to a more convenient DAG if one wishes.

Definition 4.1. A (symmetric flow) event structure $\mathcal{E}$ is a triple $(E, \prec, \#)$ such that:

1. E is a set (of events).
2. $\prec$ is an asymmetric causal flow relation on E (whose transitive (resp. reflexive transitive) closure is written $<$ (resp. $\leq$)).
3. $\#$ is an irreflexive symmetric conflict relation on E compatible with $\leq$, i.e. such that $x \# y \leq z \Rightarrow x \# z$.

The preceding is a very simple definition which will do for our immediate purposes. Generalisations arise by eg. allowing the conflict relation to be asymmetric; see some of the cited literature.

An event structure defines which events may occur once other events have already occurred. Collections of events are called configurations, and the legal configurations and legal ways of passing from one configuration to a successor configuration are packaged up in the following definition.

Definition 4.2. Let $\mathcal{E} = (E, \prec, \#)$ be an event structure. The set $\mathcal{X}_\mathcal{E} \subseteq \mathbb{P}\, E$ of (legal) configurations of $\mathcal{E}$, and the legal ways of moving from a legal configuration X of $\mathcal{E}$ to a successor legal configuration Y are given by the following rules.

1. $\varnothing \in \mathcal{X}_\mathcal{E}$.
2. $X \in \mathcal{X}_\mathcal{E}, x \in E - X, (\forall\, x' \in E \bullet x' \prec x \Rightarrow x' \in X), (\forall\, x' \in E \bullet x' \# x \Rightarrow x' \notin X) \quad \vdash \quad X \cup \{x\} \in \mathcal{X}_\mathcal{E}$.

In Fig. 4 we show an event structure for the Mondex protocol, adapted from the activity diagram of Fig. 2 to include all the 'abnormal' ways that the protocol can play out, and flowing up the page. The constituent events are in the labelled nodes, while the arrows show the elements of the flow relation $\prec$, and the #-labelled edges show a generating set for the conflict relation. In the Mondex documentation [SCW00] the various $Abort_x$ events are all part of a single *Abort* operation, which has been split into five pieces in Fig. 4 according to which 'normal' event the *Abort* is in conflict with.

In Fig. 4 there are two root events, *StartFrom* and *StartTo*, either of which can start an 'execution' of the event structure. (For the time being, we ignore the possibility of starting with one or both of the $Abort_T$ or $Abort_F$ events, which lead to 'stillborn' executions; they are included in Fig. 4 for later convenience.) Once the first event has taken place, we have a (different) choice of two next events (depending on which *Start* event went first). If the next event is the other *Start* event, then we have a choice of three subsequent events ... and so on. Working out all the possible orderings of events yields a quite complex structure, and it is clear that the event structure formalism captures all these possibilities in a compact and convenient way.

In general, an event structure is executed by starting with the empty configuration, and then one executes one event at a time, adding a new event x to the existing configuration X, as sanctioned by the rules in Definition 4.2. So Definition 4.2 provides a proof system that enables us to derive sequences of event occurrences. The set of sequences obtained thereby can be turned into DAG-shaped and forest-shaped transition relations by accumulating the information encountered in the course of assembling these sequences in suitable ways: if one adds elements to a set of events one generates a DAG; if one appends elements to a sequence of events one generates a forest.

Definition 4.3. Let $\mathcal{E} = (E, \prec, \#)$ be an event structure. The transition system $\mathcal{E}^{\mathsf{DAG}}$ associated with $\mathcal{E}$ is defined by:

1. the states are the configurations $X \in \mathcal{X}_\mathcal{E}$, with $\varnothing$ as intial state,
2. the transitions are the steps $X \xrightarrow{x} X'$ iff $X \in \mathcal{X}_\mathcal{E} \ldots \vdash X' = X \cup \{x\} \in \mathcal{X}_\mathcal{E}$ is a valid inference according to Definition 4.2.

Evidently $\mathcal{E}^{\mathsf{DAG}}$ is a DAG.

Definition 4.4. Let $\mathcal{E} = (E, \prec, \#)$ be an event structure and $\mathcal{E}^{\mathsf{DAG}}$ its associated transition system. The transition forest $\mathcal{E}^{\mathsf{FOR}}$ associated with $\mathcal{E}$ is defined by:

1. the states are the paths $\langle \varnothing, \ldots, X \rangle$ in $\mathcal{E}^{\mathsf{DAG}}$ which start at the initial $\mathcal{E}^{\mathsf{DAG}}$ state, with the empty path as initial $\mathcal{E}^{\mathsf{FOR}}$ state,
2. the transitions are the steps $\langle \varnothing, \ldots, X \rangle \xrightarrow{x} \langle \varnothing, \ldots, X, X' \rangle$ iff $X \xrightarrow{x} X'$ is a step of $\mathcal{E}^{\mathsf{DAG}}$.

The preceding gets us some way towards the provisions of Section 3. However we are not there yet. Section 3 is couched in relational terms. So events have to correspond to relations, and the enabledness or otherwise of these relations in any state must correspond to what the flow and conflict relations of the event structure permit in given configurations. In general, the process will be application-specific, since it will depend on many factors, such as how many protagonists participate in the protocol, what their local state is envisaged to be, what knowledge of the global state they have, the role of I/O, etc. However, in the context of designing a protocol to accomplish some identified atomic goal, the process of reconciling these two approaches can provide a useful consistency/correctness check on the design activity.

Beyond that, our event structure account of Mondex left out certain state components, such as the details of purse balances and amounts transfered etc., that a full account must include — i.e. the event structure was deliberately intended to be generic. Reinstating the omitted components generates a replication of the forest, indexed by the reinstated values, corresponding to the full computation forest.[5]

Once the event description is in place, and one is confident that it properly corresponds to the relational picture, we can extract a computation forest via the constructions of Definitions 4.3 and 4.4.

By construction, the nodes of the forest shaped computational DAG $\mathcal{E}^{\mathsf{FOR}}$ incorporate the full history of the protocol up to the given point. Such history information is often needed in reasoning about protocols, since protocol properties frequently depend not only on knowing that the protocol has arrived at a certain point, but that certain other things must have necessarily happened prior to that point. Such facts can be trivially extracted from the full history, so our formulation may be regarded as a multipurpose canonical description, useful for things other than just the concerns of this paper. However, since different paths can often arrive at 'essentially the same' state eg. via interchanges of causally independent steps somewhere in the interior of the protocol, it is just as useful to be able to forget aspects of history, and identify common suffixes of certain paths. The duality between $\mathcal{E}^{\mathsf{DAG}}$ and $\mathcal{E}^{\mathsf{FOR}}$ (given in one direction by the construction of $\mathcal{E}^{\mathsf{FOR}}$ from $\mathcal{E}^{\mathsf{DAG}}$, and in the other by forgetting all but the last component of each state in $\mathcal{E}^{\mathsf{FOR}}$) bears out the compatibility of these different points of view.

Another aspect that should be discussed is I/O. At the atomic level, the I/O for the single step that takes place must inevitably concern the environment, since there is no internal structure to engage in internal communication. At the protocol level however, I/O can either be between the environment and the protocol, or be purely internal to the protocol. In the latter case, the only restriction that we need is that messages must be produced before they can be consumed, a fact we insist on also when we come to consider multiple protocol instatiations and the possibility that one protocol instatiation outputs a message that is consumed by another protocol instatiation. There is of course the option of representing messages in flight within a suitable state component —such a state component can model properties of the communication medium, eg. unreliablity— however we do not need to insist on that for the serialisation discussed in the next section.

5. Interleaving and Serialising Individual Protocol Runs

Thus far, although using language such as 'protocol,' in reality we have only discussed some properties of acyclic transition systems. In genuine protocols, various agents interact by performing events and sending/receiving messages etc. We must connect our theory to this world.

[5] N. B. This picture incidentally yields one useful convention for the ρ labels of the step relations St_ρ of Section 3: namely to tag each edge of the 'generic' forest by a distinct label (corresponding to the relevant event name in the event structure picture), and then to retain these labels in each replicated forest, making the labels akin to names of 'operations' at what would be the code level.

The basic idea is that the previous section should be understood as describing (the various possibilities for) a single isolated protocol run, performed by however many agents would be appropriate in practice, with the protocol state recording in principle the full history of the protocol so far (regardless of whether such knowledge can indeed be posessed by the individual agents), and ignoring the rest of the universe. The latter not only regarding other agents/activities in the rest of the universe, but also regarding what the agents of the single protocol run might do both before and after the run itself. So the previous section described an idealised *pattern* or *template* for what collections of agents might do over some period of time towards the achievement of some goal described by the atomic action that the protocol implements.

Patterns or templates are normally made to correspond with what happens in the real world by some process of matching, and that is the basis of our approach too. Since we have remarked that our protocol states can in principle include unrealistically detailed history information, our matching process must include a projection mechanism to allow the unrealistic parts to be forgotten. In such a scenario, protocol states that were previously distinct can be matched to the same system state, just as we described in the previous section.

Definition 5.1. A *system* consists of a number of *agents*, $A_a, A_b, \ldots$ each with its agent state subspace $W_a, W_b, \ldots$. Thus the system state space is $W = W_a \times W_b \times \ldots$, agent A_a's instantaneous state is some $w_a \in W_a$, and the system's instantaneous state is $\overline{w} \equiv (w_a, w_b, \ldots)$.

Each agent is a transition system, i.e. the agent can move between different elements of its state space in discrete steps, leaving the state of every other agent unaffected. The enabledness of any agent's transitions is independent of the state of any other agent. Each step can also consume input and produce output, and the I/O policy described in the previous section applies again: i.e. I/O may either be with the environment, or it may be internal to the system, and any internal message that is consumed must earlier have been produced.

The system's transitions are described by a predicate Sy_A similar to *St* in Section 3, where the subscript '$_A$' refers to the agent performing the step, and each Sy_A step modifies only its own agent's state subspace. The transitions of the system as a whole are the interleaved agent transitions of the system's agents, each extended with skip on the irrelevant part of the total system state. The skip-extended transitions are written $\overline{Sy}_A$.

Definition 5.2. Let S be a system with agents $A_a, A_b, \ldots$. The sequence $\mathcal{T} \equiv \langle \overline{w}_I, (k_1, A_1, q_1), \overline{w}_1, (k_2, A_2, q_2), \overline{w}_2, \ldots \rangle$ is a run of the system iff:

1. $\overline{w}_I$ is an initial state of the system,
2. A_1 is the agent that performs the first step,
3. k_1 is the input consumed by A_1 during the first step,
4. q_1 is the output produced by A_1 during the first step,
5. $\overline{w}_1$ is the result state of the first step,
6. the change of state $\overline{w}_I \rightarrow \overline{w}_1$ involves a change $w_I \rightarrow w_1$ to the state space W_1 of A_1 only; the step $Sy_{A_1}(w_I, k_1, q_1, w_1)$ is the *agent step* of the first transition of the run; the state spaces of agents other than A_1 remain unchanged,
7. ... and analogously for subsequent system transitions.

Definition 5.3. Let *Protocol* be a protocol in the sense of the previous section. An agent decomposition for the protocol is a decomposition of the protocol state space V into a cartesian product of agent subspaces $V = V_1 \times V_2 \times \ldots$, such that each step of the protocol modifies[6] at most one component in the product, leaving the other components unaltered.

The decomposition into agent subspaces just described, represents the fact that an instantiation of a protocol is normally executed by a number of agents inside a real system.[7] However a real agent in a real system can play many roles during the running of the system, including acting out different roles in different instances of the same protocol at different times. So we need to distinguish the various agent roles in a protocol definition from the different instantiations of these during system runs. The next definition introduces the technical machinery for this.

Definition 5.4. Let *Atomic*, *Protocol*, ... (with all the attendant machinery) be a protocol implementing an atomic action in the sense of the previous section. We say that system run $\mathcal{T}$ instantiates *Protocol* iff there is a maximal path

[6] Here, and in the remainder of the paper, 'modifies' should be understood to mean 'is deemed to modify' or, 'is permitted to modify in the syntactic description of the step,' since it is intended to cover not only non-trivial update, but also cases of read-only access, and cases in which the agent in fact chooses not to access the state at all (even though the syntactic description, of which the step is a specific instantiation, permits it).

[7] In Mondex there will be two agents, the To agent and the From agent, so that $V = V_{\mathsf{From}} \times V_{\mathsf{To}}$. Similarly for any protocol that works by exchanging messages between two agents.

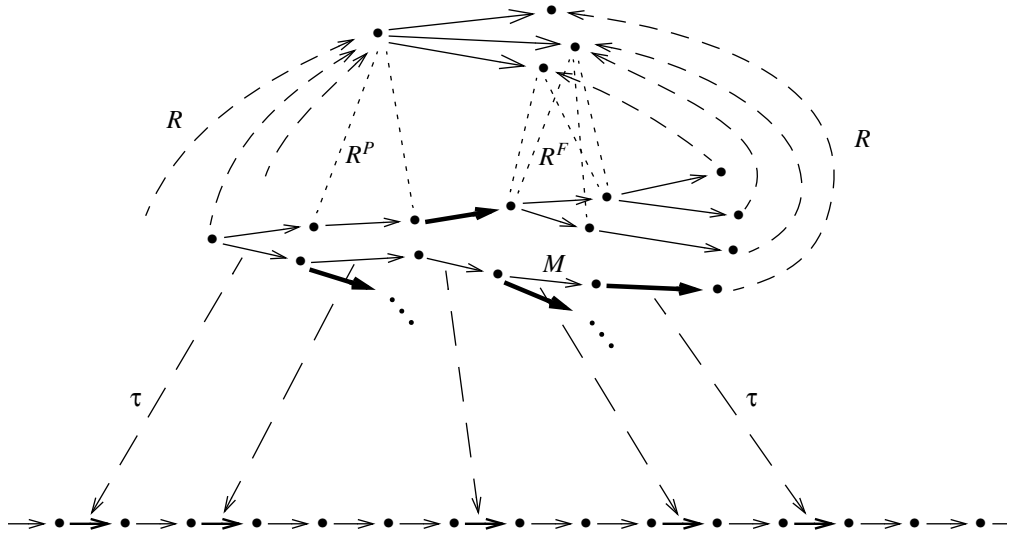

Fig. 5. An atomic action, a protocol which implements it, and a system run containing an instance of a maximal path through the protocol. The steps of the instance are shown bold.

through the protocol $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}(v_I, j_1, p_1, v_1, j_2, p_2, v_2, \ldots, v_{F-1}, j_F, p_F, v_F)$ and there are two maps: τ_A and τ_S such that:

1. there is a cartesian product of disjoint functions $\tau_{A,l} : V_l \to W_{a_l}$ from all of the agent components of V to a (possibly proper) subset of distinct agent subspaces of W, and $\tau_A = \Pi_l\, \tau_{A,l}$,
2. τ_S is an injective function from the steps of the maximal path $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ to agent steps of $\mathcal{T}$,
3. τ_S is order preserving, i.e. if St_β precedes St_γ in $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$, then $\tau_S(St_\beta)$ precedes $\tau_S(St_\gamma)$ in $\mathcal{T}$,
4. for each protocol step $St_\beta(v_{t-1}, j_t, p_t, v_t)$ in the domain of τ_S, if V_l is the agent component of V modified during the step, then $\tau_{A,l}(V_l)$ is the agent subspace modified during the step $\tau_S(St_\beta(v_{t-1}, j_t, p_t, v_t))$,
5. for each protocol step $St_\beta(v_{t-1}, j_t, p_t, v_t)$ in the domain of τ_S, if $\tau_S(St_\beta(v_{t-1}, j_t, p_t, v_t)) = Sy_{A_l}(w_{s-1}, k_s, q_s, w_s)$, then $\tau_{A,l}(v_{t-1}) = w_{s-1}$, $j_t = k_s$, $p_t = q_s$, $\tau_{A,l}(v_t) = w_s$,
6. if protocol step St_β modifies V_l and protocol step St_γ is the next protocol step along $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ that modifies V_l, then no step of $\mathcal{T}$ between $\tau_S(St_\beta)$ and $\tau_S(St_\gamma)$ modifies $\tau_{A,l}(V_l)$.

When we want to emphasise the details, we say that system run $\mathcal{T}$ instantiates *Protocol* via $\tau \equiv (\tau_A, \tau_S)$ at step $\tau_S(St_\alpha(v_I, j_1, p_1, v_1))$ of $\mathcal{T}$, where $St_\alpha(v_I, j_1, p_1, v_1)$ is the initial step in $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$.

In Fig. 5 we show how a particular maximal path, M say, through the protocol illustrated in Fig. 3, might be mapped, via an instantiation function τ, to a selection of steps in a system run. The system state in the run is now 'real world' state, eschewing the maximal knowledge that the idealised protocol formulation allows. In between the steps of $\tau(M)$, other protocols are being instantiated by other agents, though without interfering with the state of $\tau(M)$, by Definition 5.4.6.

Definition 5.5. Let $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ be a maximal path in *Protocol*. Step $St_\beta(v_{t-1}, j_t, p_t, v_t)$ of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ is a first use of agent subspace V_l iff: it modifies V_l, and no earlier step of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ modifies V_l. Similarly $St_\beta(v_{t-1}, j_t, p_t, v_t)$ is a last use of V_l iff: it modifies V_l, and no later step of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ modifies V_l. We say that *Protocol* is 2-phase (2P) along $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ iff all first uses of all agent subspaces of *Protocol* precede any last use of any agent subspace of *Protocol* along $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$.

Definition 5.6. Let $\overline{Sy}_A(\overline{w}_{s-1}, k_s, q_s, \overline{w}_s)$ and $\overline{Sy}_B(\overline{w}_s, k_{s+1}, q_{s+1}, \overline{w}_{s+1})$ be two successive steps of a run $\mathcal{T}$ of the system. We say that $\overline{Sy}_A(\ldots)$ and $\overline{Sy}_B(\ldots)$ can be commuted iff there is a state $\tilde{w}_s$ such that $\overline{Sy}_A(\tilde{w}_s, k_s, q_s, \overline{w}_{s+1})$ and $\overline{Sy}_B(\overline{w}_{s-1}, k_{s+1}, q_{s+1}, \tilde{w}_s)$ are valid steps of the system, and the pair $\overline{Sy}_A(\overline{w}_{s-1}, k_s, q_s, \overline{w}_s)\,;\, \overline{Sy}_B(\overline{w}_s, k_{s+1}, q_{s+1}, \overline{w}_{s+1})$ can be replaced in $\mathcal{T}$ by $\overline{Sy}_B(\overline{w}_{s-1}, k_{s+1}, q_{s+1}, \tilde{w}_s)\,;\, \overline{Sy}_A(\tilde{w}_s, k_s, q_s, \overline{w}_{s+1})$, yielding $\mathcal{T}'$, where $\mathcal{T}'$ is a valid run.

Lemma 5.7. If $\overline{Sy}_A(\ldots)$ and $\overline{Sy}_B(\ldots)$ as in Definition 5.6, are two successive steps performed by two different agents, then, provided both inputs are available in state $\overline{w}_{s-1}$, $\overline{Sy}_A(\ldots)$ and $\overline{Sy}_B(\ldots)$ can be commuted.

Proof. Since $\overline{Sy}_A(\ldots)$ and $\overline{Sy}_B(\ldots)$ are performed by different agents, the two agent subspaces modified by these steps are disjoint, so the changes of state can be swapped, easily yielding the state $\tilde{w}_s$ required by Definition 5.6. If both inputs are available in state $\overline{w}_{s-1}$, then the $\overline{Sy}_B(\ldots)$ is enabled in state $\overline{w}_{s-1}$ and can be performed first. Since the input to $\overline{Sy}_A(\ldots)$ is not removed by doing $\overline{Sy}_B(\ldots)$, $\overline{Sy}_A(\ldots)$ can follow $\overline{Sy}_B(\ldots)$. That this generates a valid run is now straightforward. □

Since our formulation of a protocol does not consider the protocol's context, the only way that a protocol, as formulated in Section 3, can interact with the rest of the universe, is via I/O with the environment. In the system context, this leads to a distinction within the internal system messages, between messages that are produced and consumed by the same protocol instance (which should thus correspond to internal communications of the protocol itself), and those which are produced and consumed by different protocol instances (which should thus correspond to communications with the environment in the protocol model). (System level communications with the environment must of course also correspond with protocol communications with the environment.) Since inter-protocol communications must comply with normal causality considerations, these communications must fit well with the 2-phase property for protocol state components. The next definition introduces the needed technicalities.

Definition 5.8. Suppose given a maximal path $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ of a protocol, which is 2P. An external dependency definition (XDD) for it is a pair of (not necessarily disjoint) sets (IDS, ODS) of steps of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$. *IDS* is the input dependency set: the set of steps of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ during which an external input (i.e. one originating from outside $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$) is received; and *ODS* is the output dependency set: the set of steps of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ during which an external output (i.e. one destined to outside $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$) is delivered. A protocol is 2PXDD-normal iff:

1. all *IDS* steps occur no later than any *ODS* step along $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$,
2. the producer of every input of every protocol step other than an *IDS* step is some other step of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$,
3. the consumer of every output of every protocol step other than an *ODS* step is some other step of $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$,
4. each *IDS* step occurs no later than any last use of the state,
5. each *ODS* step occurs no earlier than any first use of the state.

For a maximal path through the Mondex protocol, the IDS steps are the *StartFrom* and *StartTo* events. There are no ODS steps since Mondex is designed to change state, rather than produce output. As we mention in Section 7.8, there are some paths through the Mondex protocol which are not 2-phase, though none of them do anything 'useful'.

Definition 5.9. An instantiation of a 2PXDD-normal protocol is called a (2PXDD-normal) transaction.

For the rest of this paper all transactions will be 2PXDD-normal.

Theorem 5.10. Let $\mathcal{T}_0$ be a run of a system which consists entirely of the steps of transactions of a family of protocols.[8] Then there is a serialisation $\mathcal{T}_\infty$ of $\mathcal{T}_0$, generated by commuting adjacent steps, in which each instantiation occurs as a contiguous series of steps.

Proof. Consider the directed graph Dep_0 whose nodes are the transactions of $\mathcal{T}_0$, and whose edges are given by: $\tau_1 \to \tau_2$ iff:

1. an output of an *ODS* step of τ_1 is an input of an *IDS* step of τ_2, or,
2. an agent subspace V_l is used by both τ_1 and τ_2, but τ_1's modifications of V_l occur earlier in $\mathcal{T}_0$ than τ_2's.

Claim 5.10.1 Dep_0 is acyclic.

Proof of Claim. Let V be the state space of a transaction τ. Since the last first use of V precedes the first last use of V in τ, and all all *IDS* steps precede all *ODS* steps in τ, by Definition 5.8.4-5, we can deduce that there is a step in τ (which we will call the pivot), that precedes neither the last first use of V nor any *IDS* step, and simultaneously follows neither the first last use of V nor any *ODS* step (there are four cases). We identify each transaction in $\mathcal{T}_0$ with (some choice for) its pivot. Since steps are interleaved, there is a total order on the transactions, inherited from that on their pivots.

[8] So there is a set of maximal paths through a set of 2PXDD-normal protocols, and a set of instantiations of them in $\mathcal{T}_0$, and the set of steps of $\mathcal{T}_0$ is the disjoint union of these instantiations.

We show that Dep_0 can be interpreted in the set of pivots, and that each edge in the interpretation is oriented towards the future, yielding the acyclicity of Dep_0 immediately. For a Dep_0 edge of type 1, note that it is oriented towards the future by straightforwards causality. So pretending that the requisite message was sent during the producing transaction's pivot step, and pretending that it arrived during the consuming transaction's pivot step can increase its time of flight, but not change its orientation towards the future. For a Dep_0 edge of type 2, since the pivot steps are contained within the uses of transactions' state which do not overlap by Definition 5.4.6, and these are oriented towards the future by 2, the orientation is preserved in the interpretation. We have our claim. □□

We serialise $\mathcal{T}_0$ stage by stage. At each stage there are serialised and unserialised transactions. We call the boundary between the serialised and unserialised transactions the horizon. So at the beginning there are no serialised transactions, and the horizon lies just before the first step of $\mathcal{T}_0$. At the n'th stage, which starts with $\mathcal{T}_n$, whose unserialised transactions comprise Dep_n (a subgraph of Dep_0), we choose an unserialised transaction which is a root of Dep_n, and we serialise it, whereupon its steps —in contiguous sequence— are both appended to the serialised part, and removed from the unserialised part of the partly serialised run, moving the horizon to just beyond the newly serialised steps, and yielding $\mathcal{T}_{n+1}$ and Dep_{n+1}. If $\mathcal{T}_0$ is infinite, then the serialisation process continues forever, and every finite prefix of $\mathcal{T}_0$ has all its steps eventually included in the serialised part. If $\mathcal{T}_0$ is finite, the process stops when the last transaction of $\mathcal{T}_0$ has been serialised.

Stage n: A root transaction τ_n of Dep_n is chosen. By assumption, all transactions on which τ_n is dependent, whether through the state space, or via τ_n's *IDS* messages, have been serialised, i.e. their steps lie beyond the horizon. So any step of $\mathcal{T}_n$ that lies between the horizon and τ_n's first step neither uses any state used by τ_n's first step, nor produces a message consumed by τ_n's first step. So there is no obstacle to commuting the first step of τ_n towards the past until it it arrives immediately after the horizon. Similarly the dependencies for the second step lie either beyond the horizon, or arise from the first step, so the second step of τ_n can be commuted towards the past until it arrives immediately after the first. The process continues until the last step of τ_n has been commuted until it arrives immediately after its predecessor. This yields $\mathcal{T}_{n+1}$. Transaction τ_n is removed from Dep_n, yielding Dep_{n+1}, and the horizon is moved to just after τ_n's last step. *End Stage n*. □

The preceding amounts to a sketch of a relatively standard 2-phase serialisation proof process [BHG87, GR93, BN97, WV02]. And once the run has been serialised, it is unsurprising to observe that each transaction of the serialised run is a refinement of its corresponding atomic action via a retrieve function that forgets the part of the system state not relevant to the transaction. We examine the details of this, and associated issues, in the next section.

6. Interleaved Protocol Runs as Simulations

Consider a serial run $\mathcal{T}_{\text{ser}}$, as manufactured in the discourse above. If we pick on a particular state in the run, say $\overline{w}$, it will be the representative in the system run of a protocol state, say v, of the protocol being executed at that point of the run. Say it is the g'th protocol instantiation being executed, *Protocol-g*. Then *Protocol-g* itself is a refinement of an atomic action *Atomic-g*, via a retrieve relation $R^1_{Protocol\text{-}g}$, constructed as in Section 3.

We know that W consists of agent subspaces. Knowing that $\overline{w}$ relates to a step of *Protocol-g* allows us to split the agent subspaces of W into two sets. The members of the first set, $W_{A,Protocol\text{-}g}$ are the state subspaces belonging to those agents actually executing *Protocol-g*. The second set $W_{\overline{A,Protocol\text{-}g}}$ contains any agent subspaces of W that remain.

Continuing to use the notations of the previous section, we know that *Protocol-g* is instantiated via $\tau_{Protocol\text{-}g} = (\tau_{A,Protocol\text{-}g}, \tau_{S,Protocol\text{-}g})$. Therefore the codomain of $\tau_{A,Protocol\text{-}g}$ is $W_{A,Protocol\text{-}g}$, and $\overline{w}$ is the tuple consisting of w (the projection of $\overline{w}$ onto $W_{A,Protocol\text{-}g}$), and some other state elements belonging to $W_{\overline{A,Protocol\text{-}g}}$ (if the latter is nonempty). We thereby derive the following.

Proposition 6.1. Let $\mathcal{T}_{\text{ser}}$ be a serial run of a collection of protocol instantiations by a set of agents. Then for each step $\langle \overline{w}_{d-1}, k_d, A_d, q_d, \overline{w}_d \rangle$ of $\mathcal{T}_{\text{ser}}$, there is an atomic action, *Atomic-g* say, such that $\overline{w}_{d-1}$ and $\overline{w}_d$ are related to: (a) the identity on an initial state of *Atomic-g*, or, (b) a step of *Atomic-g*, or, (c) the identity on a final state of *Atomic-g*, via the retrieve relation:

$$S_g(u, \overline{w}) \equiv R^1_{Protocol\text{-}g}(u, v) \wedge \tau^{-1}_{A,Protocol\text{-}g}(v, w) \tag{28}$$

where $R^1_{Protocol\text{-}g}$ is the R^1 retrieve relation for *Protocol-g*, constructed as described in Section 3. Furthermore, in the case of option (b), the I/O of the entire transaction of which $\langle \overline{w}_{d-1}, k_d, A_d, q_d, \overline{w}_d \rangle$ forms a part, is mapped to the I/O of the *Atomic-g* step as in (26) and (27).

Proof. By construction, we know that there is a step of *Protocol-g*, $St_\gamma(v_{t-1}, j_t, p_t, v_t)$ say, which is mapped to

$\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$ of $\mathcal{T}_{\text{ser}}$ by $\tau_{S,Protocol\text{-}g}$, where w_{d-1} and w_d are the projections of $\overline{w}_{d-1}$ and $\overline{w}_d$ onto $W_{A,Protocol\text{-}g}$. Thus the presence of $\overline{w}$ in the left hand side of (28) and the presence of w in the right hand side of (28) serves to project out the unneeded system state. Next, we claim that $\tau^{-1}_{A,Protocol\text{-}g}$ connects w_{d-1} and w_d to v_{t-1} and v_t respectively. Finally, $R^1_{Protocol\text{-}g}$ connects v_{t-1} and v_t respectively to: (a) the identity on an initial state of *Atomic-g*, or, (b) a step of *Atomic-g*, or, (c) the identity on a final state of *Atomic-g*, depending on whether $St_\gamma(v_{t-1}, j_t, p_t, v_t)$: (a) occurs before the SA step, or, (b) *is* the SA step, or, (c) occurs after the SA step, for the maximal path of *Protocol-g* instantiated in this portion of $\mathcal{T}_{\text{ser}}$.

We now briefly substantiate our claim about $\tau^{-1}_{A,Protocol\text{-}g}$. $W_{A,Protocol\text{-}g}$ splits into two parts. The first part just contains $W_{A,Protocol\text{-}g,d}$ the agent subspace of the system agent A_d actually executing $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$. The second part contains the agent subspaces of all the system agents involved in executing this protocol instantiation, except for A_d. A similar split occurs at the *Protocol-g* level. Therefore, by Definition 5.4.1, $\tau^{-1}_{A,Protocol\text{-}g}$ splits into pieces, one for each corresponding pair of (protocol/system) agent subspaces involved ('$\wedge$' in predicates, product relationally). For A_d, we know that the relevant piece, $\tau^{-1}_{A,Protocol\text{-}g,d}$, holds, by Definition 5.4.5. For every other agent, since the agent is involved in the protocol instantiation, there is some other step of the instantiation, occurring either earlier or later than $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$, for which the appropriate $\tau^{-1}_{A,Protocol\text{-}g,-}$ piece holds; and so, since both protocols and instantiations skip on non-involved state during execution steps, the truth of such $\tau^{-1}_{A,Protocol\text{-}g,-}$ pieces propagates throughout the whole of the instantiation. So we have the claim. □

Proposition 6.1 establishes a 1-step simulation property $\Sigma^1_{\mathcal{T}_{\text{ser}}}$ which is analogous to (11). It shows that a single protocol instantiation step $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$ and a suitable single (trivial or non-trivial) atomic step, together with the relations S_g that connect them at before- and after- states (and with I/O relations where appropriate) all hold.

By combining the 1-step simulation relations thus calculated for all the protocol instantiations in $\mathcal{T}_{\text{ser}}$, we see that the whole of $\mathcal{T}_{\text{ser}}$ is simulated by a sequence of such instantiations, as one would expect. And system states in $\mathcal{T}_{\text{ser}}$ that lie at the end of one protocol instantiation and at the beginning of the next, are evidently in the domain of the S_g relations for two consecutive g's.

Unlike most formulations of simulation between a system run and its abstraction, in the above, each atomic abstraction exists in isolation, thus helping to isolate any tricky technical aspects involved in its refinement from complications arising from the involvement of system state. Of course, on the abstract side, one could set up instantiations in an abstract system world in the same way as was done for the concrete protocol instantiations, and thereby recover a more conventional view.

One could then ask what the benefit of the more complicated formulation was. The answer is that it separates concerns rather neatly. Any technical difficulties in the refinement of the atomic action to the protocol, can be investigated within a 'bubble' consisting of a template atomic action and a template protocol. After this, the connection between the templates and the whole-system view, can be made relatively routinely, as we have just shown. The reader should note in particular that the synchronisation needed for refinement (via the SA steps) has been completely decoupled from the synchronisation needed for serialisation (which depends on first/last uses of state and analogous I/O properties).

Furthermore, although one could certainly invent contrived counterexamples, the connection between the state in a protocol transition system and the state in a protocol instantiation will, in practical examples, be very simple. The former typically contains more history information than a real system state would; the latter will typically contain information about all sorts of system state that is irrelevant to the protocol instantiation itself (specifically, state belonging to agents not involved in that particular protocol instantiation). For this reason we state the following.

Remark 6.2. We anticipate that in all cases of practical interest, the connection between protocol state and system state will be achievable via simple forward induction techniques.

So far then, the picture looks rosy enough. But there is one remaining snag: not all system runs are serial. To see the issues this raises, we note that once a step $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$ is chosen, the discussion in Proposition 6.1 splits the system state space into three: (a) agent A_d's agent subspace, (b) the agent subspaces of the other agents participating in the protocol instantiation, (c) the agent subspaces of agents not covered by (a) or (b). For (a) we calculated the contribution to S_g directly via the approproiate τ_A function, for (b) we enlarged this contribution by propagating the effects of other steps in the maximal path being instantiated, for (c) we projected that state away.

The latter two are problematic in a non-serial run. In the (c) part, there may be other non-trivial protocol instantiations going on that are interleaved with this one, and these should be considered. In the (b) part, if any of the agents involved, at the point of the $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$ step, has not yet reached the first use of or has already passed the last use of its state, then it may be, at this point of the protocol instantiation, involved in other activities unrelated to the

Fig. 6. Timelines for three agents engaged in a protocol, shading the first-to-last uses of their state spaces.

present protocol instantiation. In such a case, propagating a τ_A function to the $\langle w_{d-1}, k_d, A_d, q_d, w_d\rangle$ step is in general invalid.

Fig. 6 illustrates the situation. Three agents A_1, A_2, A_3 engage in a protocol, with time going left to right, and with the first use to last use of the agent subspaces shaded grey; propagating τ_A functions into the unshaded areas is not permissible.

Thus in a non-serial run, protocol instantiations generally have 'ragged starts' and 'ragged ends' —the shaded part of Fig. 6— allowing different protocol instantiations to overlap in time, even when they involve some of the same agents. On the other hand, in a serial run, protocol instantiations have 'clean starts' and 'clean ends' —the whole rectangular area in Fig. 6— so all involved agents are in the right state at the beginning of the rectangle, and in the right state at the end of it. This 'clean' formulation was extremely useful in Section 3, where it led directly to the straightforward definition of the past and future retrieve relations R^P and R^F in terms of the equally clean outer retrieve relation R.

The clean separation of concerns engendered by template *Atomic* and *Protocol* transition relations, distinct from system runs, permits the gap between serial and non-serial system runs to be closed in a number of ways. Perhaps the simplest of these keeps *Atomic* and *Protocol* unchanged, and just reinterprets the various retrieve relations in a manner appropriate for non-serial runs. This is the approach we will take.

Definition 6.3. Let *Atomic* and *Protocol* be as in Section 3, let $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ be a maximal path through *Protocol*, and let R^1 be as in (25) with respect to a given SA. Let V_{A_d} be an agent subspace of the state space of *Protocol* and v_{A_d} a typical element of V_{A_d}. Then the fragmented retrieve relation $\widehat{R^1_{A_d}}$ is defined by:

$$\widehat{R^1_{A_d}}(u, v_{A_d}) \equiv \exists\, v_{\overline{A_d}} \bullet R^1(u, \langle v_{A_d}, v_{\overline{A_d}}\rangle) \wedge \left(\begin{array}{c} v_{A_d} \text{ occurs no earlier than the first use of } V_{A_d} \text{ and no} \\ \text{later than the last use of } V_{A_d}, \text{ along } MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle} \end{array} \right) \tag{29}$$

where $v_{\overline{A_d}}$ refers to the protocol state in all agent subspaces other than V_{A_d}, and the dependence of $\widehat{R^1_{A_d}}$ on $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$ and the SA is understood.

We see that whereas R is stipulated (in Assumptions 3.2.1) to be a partial function, the various $\widehat{R^1_{A_d}}$ evidently are not. Given the $\widehat{R^1_{A_d}}$, we can recover the original R^1 as follows.

Proposition 6.4. Let *Atomic*, *Protocol*, $MPath_{\langle\alpha,\beta,\ldots,\gamma\rangle}$, a given SA, and the various $\widehat{R^1_{A_d}}$ be as previously. Then:

$$R^1(u, \langle v_{A_{d1}}, v_{A_{d2}}, v_{A_{d3}}, \ldots\rangle) \equiv \widehat{R^1_{A_{d1}}}(u, \hat{v}_{A_{d1}}) \wedge \widehat{R^1_{A_{d2}}}(u, \hat{v}_{A_{d2}}) \wedge \widehat{R^1_{A_{d3}}}(u, \hat{v}_{A_{d3}}) \wedge \ldots \tag{30}$$

where $d1, d2, d3, \ldots$ ranges over all the agents involved in the protocol and

$$\hat{v}_{A_{d1}} = \begin{cases} v_{\text{first},A_{d1}} & \text{where } v_{\text{first},A_{d1}} \text{ is the before-state of the first use of } V_{A_{d1}} \\ & \quad \text{if } v_{A_{d1}} \text{ occurs in a step which precedes the first use of } V_{A_{d1}} \\ v_{\text{last},A_{d1}} & \text{where } v_{\text{last},A_{d1}} \text{ is the after-state of the last use of } V_{A_{d1}} \\ & \quad \text{if } v_{A_{d1}} \text{ occurs in a step which follows the last use of } V_{A_{d1}} \\ v_{A_{d1}} & \text{otherwise} \end{cases}$$

$$\hat{v}_{A_{d2}} = \begin{cases} v_{\text{first},A_{d2}} & \text{where } v_{\text{first},A_{d2}} \text{ is the before-state of the first use of } V_{A_{d2}} \text{ if } \ldots \text{ etc.} \\ v_{\text{last},A_{d2}} & \text{where } v_{\text{last},A_{d2}} \text{ is the after-state of the last use of } V_{A_{d2}} \text{ if } \ldots \text{ etc.} \\ v_{A_{d2}} & \text{otherwise} \end{cases}$$

etc. (31)

Proof. A straightforward calculation that extends the various $\widehat{R^1_{A_d}}$ into the 'white bits' of Fig. 6. □

While Proposition 6.4 is quite general, in practice we can often do better if the abstract state can be partitioned along agent lines, like the protocol state. In that case one can quantify out the irrelevant abstract state (as well as the irrelevant protocol state) and the $\widehat{R^1_{A_d}}$ relations can become functions. Mondex is an instance of this, where the From and To balances from the protocol state survive into (and are distinguishable at) the abstract level.

We observe that the more narrowly defined fragmented retrieve relations reflect more accurately the way that a community of agents might instantiate a family of isolated protocols in an interleaved manner, consistent with the event structure picture given earlier, and how this description correspondingly obscures the relationship with the refinement of atomic action to finegrained protocol.

With the tools that we have now developed, we can make a connection between an arbitrary system run and the collection of protocols that it instantiates. Again, there are a number of detailed ways to do this. Our approach is aimed at a good correspondence with Remark 6.2.

Definition 6.5. Let $\mathcal{T}$ be a(n arbitrary) run of a collection of protocol instantiations by a set of agents. We say a system agent subspace W_d is busy (in a given state occurence $\overline{w}$ of $\mathcal{T}$) iff: $\overline{w}$ either is or occurs after the before-state of the first use of W_d by its agent A_d, and either is or occurs before the after-state of the last use of W_d by A_d, during some instantiation of some protocol during $\mathcal{T}$. If *Protocol-g* is the protocol being instantiated by agent A_d and its colleagues, we say that W_d is busy while instantiating *Protocol-g*. Otherwise we say W_d is non-busy (in state occurence $\overline{w}$ of $\mathcal{T}$).

Proposition 6.6. Let $\mathcal{T}$ be a run of a collection of protocol instantiations by a set of agents. Then for each state $\overline{w}$ of $\mathcal{T}$, $\overline{w}$ is related to the abstract states of the set of protocol instantiations —while instantiating which, the relevant agent subspace is busy in $\overline{w}$— via the retrieve relation:

$$S(\langle u_{g1}, u_{g2} \ldots \rangle, \overline{w}) \equiv$$
$$\bigwedge_{\substack{\text{subspace } W_d \text{ is busy in } \overline{w} \text{ while} \\ \text{instantiating } \textit{Protocol-g}}} \left(\widehat{R^1_{A_d,\textit{Protocol-g}}}(u_g, v_t) \wedge \tau^{-1}_{A_d,\textit{Protocol-g}}(v_t, w_d) \right) \tag{32}$$

where $g1, g2 \ldots$ ranges over all the protocols whose agent subspaces are busy in $\overline{w}$, and both $\widehat{R^1_{A_d}}$ and τ have acquired an extra '*Protocol-g*' index to label all the protocols that are being instantiated at $\overline{w}$.

Proof. This is the obvious adaptation of Proposition 6.1 to a situation in which more than one protocol might be being instantiated at a given state $\overline{w}$, and for a given protocol instantiation, not all of the agents involved may be busy in $\overline{w}$ while instantiating it. □

Definition 6.7. A transaction prefix is like a transaction, but such that the step map τ_S of the instantiation is only defined on a proper prefix of the maximal path of the protocol being instantiated. A transaction prefix is deadlocked iff there is a system state $\overline{w}$ in the run, at or beyond the last step in the range of τ_S, such that there is no possible extension of the run after $\overline{w}$, for which τ_S can be extended to an instantiation of a longer prefix of a maximal path of the protocol.

Theorem 6.8. Let there be a collection of agents, each with its own agent subspace, and a collection of atomic actions, each refined to a protocol as above. Suppose that:

1. The system as a whole executes a run.
2. Each agent only executes steps that instantiate protocol steps.
3. Each agent's steps extend a prefix of the instantiation of a maximal path through a protocol.
4. If, in a state of the system run an agent subspace V_A is busy, then agent A can only execute steps that extend a maximal path through the protocol with whose instantiation agent A is busy.

Then:

(a) Each state of the system run $\overline{w}$, is related to the abstract before- or after- states of the atomic actions being instantiated in $\overline{w}$, via the run-specific retrieve relations S of (32), with corresponding results for I/O.

(b) If a transaction prefix is deadlocked, then there is some agent subspace needed by the transaction which remains permanently busy (with another instantiation), or there is some external input needed by the transaction which never arrives, or both, ('permanently' referring to any extension of the run beyond the point at which deadlock can be established).

(c) If the system run is fair —i.e. if every transaction prefix that is not deadlocked eventually progresses (and thus eventually completes the instantiation of a maximal path)— then the transactions that run to completion can be

serialised as in Theorem 5.10, and the transactions that do not run to completion (i.e. the deadlocked transaction prefixes) can be relegated to the end of the serialised run (and in the case of infinite runs, to indefinitely far into the future).

Proof. Part (a) amounts to the invariant of a straightforward inductive process, which follows readily from our earlier calculations. Thus the initial state of a run is related only to the before-state of the first step of the first protocol being instantiated, providing the base case. For the inductive argument, consider that the next (run) step is one of the following: (i) the first step of a protocol instantiation; (ii) a non-first step of a protocol instantiation which makes a previously non-busy agent subspace busy; (iii) a non-last step of a protocol instantiation which makes a previously busy agent subspace non-busy; (iv) the last step of a protocol instantiation; (v) an internal step of a protocol instantiation, i.e. a step not covered by one of the preceding cases.

For (i), the agent subspace required must be non-busy prior to the run step, since pre-emption is prevented by 4. The run step makes it busy. The *S* relation is amplified by mapping the before-state of that agent subspace to the before-state of the atomic action being instantiated, and the after-state of that agent subspace to either the before-state of the atomic action again (if the run step does not instantiate the SA step), or the after-state of the atomic action (if the run step does instantiate the SA step). Both possibilities are catered for in the big conjunction in (32).

For (ii), the argument is similar, although other agent subspaces are already busy with the transaction. Depending on whether the run step instantiates: (α) a step before the SA step, (β) the SA step itself, (γ) a step after the SA step, yields a three way case split for mapping the before- and after- states of the run step to the before- and after- states of the atomic action.

For (iii), the argument is the converse; thus the step completes, making the relevant agent subspace non-busy. There is a three way case split for before- and after- states as in case (ii).

For (iv), the argument is the converse of case (i), with a two way case split for before- and after- states.

For (v), the relevant agent subspace is busy with the transaction, and remains so. There is a three way case split for before- and after- states.

Regarding the I/O, Definition 5.4.5 stipulates that the inputs and outputs of a protocol path and its instantiation are identical. Therefore the protocol/system I/O relationship is trivial, and the atomic/protocol I/O relationship has been taken care of in Theorem 3.8.

Part (b) rests on the observation that in our system model, the only 'resources' that a step might need (and whose absence can therefore disable it) are the availability of a needed input, and the availability of the agent subspace in which the needed state change is to take place. Thus suppose that a transaction prefix is deadlocked. Then, by definition, the next step of any extension must be disabled. So in the case of each possible extension, either an input is unavailable, or the agent subspace is unavailable (or both). In the former case, since all prior steps of the transaction prefix have already been instantiated, producing all required internal outputs (which are assumed to reach their destinations in a finite amount of time), any needed input of a disabled step must be external. In the latter case, the agent needed to execute the disabled step must be unavailable. This can only be because that agent is permanently busy with some other transaction, by 4, since if there was ever a point in the future at which it became non-busy, a run extension could be constructed that elected to instantiate the hitherto blocked step at that point.

Part (c) rests on the observation that no completed transaction τ can depend on a deadlocked transaction prefix — in the sense of τ needing either an external output, or a state value in an agent subspace, that is produced by the deadlocked transaction prefix. This is because all transactions and transaction prefixes are 2PXDD-normal. Since the deadlocked transaction prefix is blocked, it cannot have yet acquired all the resources it needs to complete. Therefore, by the 2PXDD-normal property, it cannot have started to produce any external outputs, and all its agent subspaces remain busy forever by the 'no pre-emption' property above, 4. Since τ completes, it cannot have needed any of these resources.

Now consider the serialisation process of Theorem 5.10. By the preceding, deadlocked transaction prefixes occur as leaves of the original dependency DAG Dep_0. Since the serialisation process 'consumes' transactions from the roots, the deadlocked transaction prefixes eventually all appear as isolated nodes of successive dependency DAGs Dep_n. Once a deadlocked transaction prefix is in this condition, it can be ignored by the remainder of the serialisation, and thus get relegated to the end of a finite serialised run, or indefinitely far into the future in an infinite run. □

The provisions of Theorem 6.8 bring the global perspective of Theorem 5.10, which deals exclusively with fully instantiated maximal paths, closer to the practical world, in which agents, though constrained to execute only instantiations of protocol steps, do so in ignorance of global information, and therefore risk deadlock. The latter is defined in very simple terms in our models as just the absence of needed input, or non-availability of needed state (given the assumption of non-pre-emption), this being a precise expression of the sense in which the instantiations of our protocols are *isolated*.

It is clear that we can expect that the assumptions of Theorem 6.8 will be rather easy to check in practical examples. Moreover, once one knows that agents only ever execute instantiations of protocol steps, the 'simple forwards induction' alluded to in Remark 6.2 amounts to the observation that all system steps are in the range of appropriate τ_S maps.

We have now reached the point, beyond which we do not elaborate our modelling to a greater level of detail than we have already achieved. This is mainly because the relevant facts may be brought about in many different ways. Thus, at one extreme, system agents may be true independent processing entities, each with its own memory, physically isolated from the rest of the world (as happens in Mondex). At the other extreme, system agents and their memories may be abstractions (such as threads) maintained by an OS scheduling process, all within a single processor. Many variations in between can obviously be envisaged. Each situation will support its own reasons for establishing the needed facts that our general formalism relies on. Thus every real world application of the general formalism will require a little 'glue theory' to connect its specific details to the general provisions. This is entirely analogous to the way that 'glue logic' is often needed to connect up off-the-shelf hardware components, or the way that 'glue code' is needed to connect up off-the-shelf software components.

7. Mondex and its Refinements

In this section we reflect on the Mondex protocol, and the extent to which its refinement possibilities correspond to the preceding theory. We consider a number of specific refinements. Some of these were constructed as contributions to the Verification Grand Challenge [JOW06, Woo06, WB07], for which the mechnical verification of Mondex constituted the first major case study. The VGC work is surveyed in the special issue [JWe08]. Others among our refinements were constructed independently, and we also consider some refinements that do not correspond to any refinement explored in the literature at all, but which, on the basis of the theory elaborated above, are evidently perfectly possible. There are a number of points to be borne in mind.

First of all, our theory has been couched in terms of single transitions (which is less cluttered), whereas Mondex is couched in terms of Z *operations* [Spi92, DB01, ISO02]. The distinction is the same as the one discussed in Section 4 between the generic event structure and its replication in the detailed computational structure by all the permitted values of the generically omitted state. Therefore when we say below that such and such an operation is synchronised with such and such an atomic action, we are referring in bulk to all the transitions of the operation being suitably synchronised with appropriate instantiations of the atomic action.

Secondly, we will restrict our attention for now to runs of the protocol which commence with the two *Start* operations, *StartFrom* and *StartTo*, in either order, (returning to other possibilities at the end of this section). Referring to Fig. 2, this means that after the two *Start* operations, the protocol, which is henceforth serial (as is obvious from the causal dependencies of the *req*, *val* and *ack* messages), executes some prefix of the *Req-Val-Ack* sequence of operations. If it does not complete that sequence, each purse that still has elements of the *Req-Val-Ack* sequence left to do, performs an *Abort* operation (which replaces the first such unperformed *Req-Val-Ack* operation left on that purse's agenda), completing the protocol abnormally. Note however that unlike the *Req-Val-Ack* operations which are causally constrained by the *req*, *val*, *ack* messages, *Abort* operations are not causally constrained and can occur at any time. Every variation in the order of performing the protocol's operations when *Abort* events are involved, causes a branching of the computation tree structure, and leads overall, to quite a complex protocol computation tree. All of this concurs with the possibilities offered in the event structure of Fig. 4.

Thirdly, while the original Mondex refinement is structured as two refinement stages (A model to B model, then B model to C model), the separation of concerns that this embodies is different to the one in our general two stage architecture. In particular, there is no separation of protocol-local reasoning from global-embedding reasoning in the original Mondex development, making the Mondex inter-model invariants quite complicated compared with ours. Of course the comparison is not completely fair, since our architecture did not write down the atomic system level view explicitly, as pointed out in the previous section — so a fairer comparison would cast the approach of this paper as a three stage architecture.

7.1. The Original Mondex Refinement [SCW00]

In [SCW00], the refinement is constructed to synchronise with the atomic description as early as possible, given the assumptions above. Thus the atomic action is synchronised with the *Req* operation, which refines both *AbTransferOK* and *AbTransferLost*. Since the protocol still has plenty of opportunity to fail after the *Req* operation, the *Req* operation

itself does not fix the outcome, so the refinement, achieved on the basis of a global inductive proof, has to be a backward one. We can visualise to some extent the substructure of Fig. 3 that forces a backward simulation (referred to at the end of Section 3), from Fig. 2, if we add an edge from *Req* to an *Abort*, as an alternative to the message towards *Val*, since the two abstract outcomes are already available at the end of the *Req* operation. Furthermore, since for a failing transaction the protocol has already angelically chosen to refine *AbTransferLost*, the *Abort* operation(s), which actually signal the failure at the protocol level, all refine *AbIgnore* (which is Mondex-speak for an abstract skip).

7.2. The Refinement of Banach et al. [BPJS07]

In [BPJS07], amongst other things, a synchronisation with the atomic description that occured late was sought, in order to try to get a forward simulation.[9] The natural operation to refine *AbTransferOK* to is *Val*, since that is the moment that the money safely arrives at the recipient. However, if the refinement of *AbTransferOK* is 'obvious,' then the refinement of *AbTransferLost* is less so. The subtlety lies within the *Abort* operation. The deeper structure of the Mondex protocol implies that if only one *Abort* occurs in a transaction, it is harmless, and such an *Abort* can refine *AbIgnore*. Only if two *Abort* operations occur for a transaction, each while its respective purse is in a critical state, has the transaction failed non-trivially, whereupon the transaction needs to refine *AbTransferLost*. This leads to the decomposition of the *Abort* operation into cases, depending on the precise role of the operation in the transaction. In the formalism of this paper, the *Abort* operation of Mondex corresponds to a collection of events which occur at different places in the computation tree of the protocol, and are thus distinguishable.

The case analysis is interesting. The distinction between benign and non-benign instances of *Abort* is made on the basis of a purse's local state (specifically, on whether the purse is in state *epv* or *epa* (non-benign), or in some other state (benign)). However, since two *Abort*s make one *AbTransferLost*, we can only refine *AbTransferLost* to one of the pair — and it has to be the second of the pair, since if only one *Abort* in a critical state happens, then it turns out to be benign nonetheless. In [BPJS07] *non-local* state information is used to distinguish the first non-benign *Abort* from the second, and the first is then made to refine *AbIgnore* while the second refines *AbTransferLost*.

7.3. The Refinement of Schellhorn et al. [SGH$^+$07]

In [SGH$^+$07] we have the second mechanized verification of Mondex using the KIV theorem prover [RSSB98]. While the first [SGHR06b] used the original backward simulation and data refinement, the second uses abstract state machines (ASMs, [Gur95], [BS03]) together with ASM refinement and generalized forward simulations [Sch01].

The refinement, like [BPJS07], synchronizes successful transfers by having *Val* implement *AbTransferOK*. But it chooses to synchronize failed transfers at the earliest point possible. This gives two cases for the *Req* operation, which is the point where the From purse sends money. In the first, the To purse is still ready to receive the money, in which case *Req* implements *AbIgnore*. But if the To purse has already aborted then the second case applies, and *Req* implements *AbTransferLost*.[10] Instead of having two cases (as in [BPJS07]) in which the *Abort* operation implements *AbTransferLost*, the design of [SGH$^+$07] leaves only one: the case where the To purse aborts in *epv* after money has been sent.

The different choices for the synchronisation points was one motivation for us to study the general possibilities here. Another one was to provide a general formalization of using past and future simulation relations (R^P and R^F). Instances of such relations with a schematic encoding into Dynamic Logic are not only used in the case study [SGH$^+$07] but also in earlier work. Future simulation relations occur in the correctness proof of ASM refinement [Sch01]. Past simulation relations are used in coupled refinement [DW03] as noted in [Sch05].

7.4. The Refinements of Haxthausen, George et al. [HGS06]

The two refinements of [HGS06] use the RAISE specification language [The92]. They are another mechanized verification of Mondex using the theorem prover PVS [ORS92]. This case study is slightly out of scope of our theory, since

[9] Looking forward to some extent to the specific results of the present paper —which show that the essentials of a protocol can be understood by discussing the protagonists in isolation— the discussion in [BPJS07] was restricted to a world of just two purses, a single From purse and a single To purse.

[10] This differs from [BPJS07], where the *Abort* of the From purse that is bound to happen in this situation implements *AbTransferLost*.

it does not start with atomic actions, but with a two step protocol: the first step (called *TransferLeft*) is a send operation, which nondeterministically chooses between a success and failure, and we call the two cases *SendOK* and *SendFail*. After *SendOK*, there are again two possibilities: receiving may succeed or fail. For symmetry, we call these operations *ReceiveOK* and *ReceiveFail*, [HGS06] calls them *TransferRight* and *Abort*. Already, the splitting of transactions at the abstract level into send and receive, allows us to keep the balances of abstract and concrete level in perfect synchrony, as is required by RAISE refinement. The two refinements implement *TransferLeft* with *Req* and *ReceiveOK* with *Val*.

To compare the synchronisation points with our proofs, we have to add an additional refinement of the original abstract Mondex level to the abstract RAISE level. The refinement would have to implement *AbTransferOK* by the sequence *SendOK*;*ReceiveOK*. *AbTransferLost* would be implemented by both *SendFail* and *SendOK*;*ReceiveFail*. Because *SendOK* is ON, a forward simulation proof would have to synchronize with the last operation of every sequence. Composing the resulting simulation relation with the existing refinements, we find that the synchronization is the one used in [SGH$^+$07].

7.5. The Refinements of Butler and Yadav

These refinements develop a Mondex-like money transfer protocol using the B4free tool [B4f]. In accordance with the Event-B [AH06] methodology, the protocol is developed in many small, but easily mechanically provable refinement steps, the simulations being forward simulations. The strategy decomposes the abstract events to facilitate separate refinement of distinct pieces to distinct protocol level operations. Aside from that, it is similar to that of [BPJS07] in that failing transfers are refined by *Abort*s.

Note that with the exception of the original (backward) one, the preceding refinements are all forward simulations when viewed at the individual protocol instance level (cf. Corollary 3.10). As such, and particularly when they are based on $(1,1)$ refinements, they all readily extend to forward simulation refinements of full system runs — just as the original $(1,1)$ backward simulation readily extended to a backward simulation refinement for full system runs.

7.6. The Refinements of Schellhorn and Banach [SB08]

In [SB08] the authors developed a new refinement strategy for Mondex based round the essential requirements that the Mondex protocol addresses, in contrast to other approaches which were driven primarily by the technical demands of one refinement technique or another. The refinement of [SB08] has the same initial and final specification as the ASM refinement in [SGH$^+$07]. It is broken down into three separate refinements for the three essential concepts inherent in the Mondex protocol:

- Implementing money transfer by sending messages over a lossy transport medium.
- Checking messages to be fresh to protect against replay attacks.
- Implementing a challenge-response system needed to ensure freshness using sequence numbers and a fourth message.

The first refinement implements atomic money transfer using a protocol on the second level that consists of *req*, *val* and *ack* messages only. It ensures uniqueness of protocol runs by using abstract transaction identifiers (*tid*s, inspired by [BPJS07]). The *StartTo* and *StartFrom* steps of the final protocol shown in Fig. 2 are added only in the last refinement which implements *tid*s by pairs of sequence numbers.

Since the first refinement is the main non-atomic refinement of the development it is natural to ask whether our theory is flexible enough to view this refinement as an instance of the theory developed in this paper.

The answer is a cautious yes, although some small adjustments are necessary, which we now describe. First, the operations of the second level do not use input and output explicitly. Messages sent from a sending purse are directly placed into an *inbox* of incoming messages of the intended receiver.

To view the ASM as having operations $St(v,j,p,v')$ with a common input/output type for j and p, this type would have to consist of paired message contents and intended receivers. A valid protocol run would have to satisfy the constraint that each output p is used at most once in a later step, only by the intended receiver, for which it forms the input j. This is clearly straightforward enough to arrange.

As a side remark, matching the third level to operations $St(v,j,p,v')$ is easier, since the second refinement replaces *inbox*es with a global ether of sent messages. Thereby replay attacks become possible, and the third level adds appropriate checks to prevent them.

A second niggle when matching the approach of this paper to the first refinement of [SB08] is the use of transaction identifiers. When the first *req* message of a protocol run is generated in the *StartTo* step, it is equipped with a transaction identifier *tid*, which is chosen to be fresh relative to a global set *tidset* of already used ones. Since our present approach does not consider global state (except for the implicit set of messages sent so far) this could be encoded by using external inputs for the *StartTo* operation, each containing its needed fresh *tid*. Uniqueness would result in the constraint that valid protocol runs never reuse these external *tid* inputs.[11]

Apart from these adjustments, the proof technique we have developed here can be informally seen to apply: the synchronization points are the same as in [SGH$^+$07] (late synchronization using simulation relations R^P that look into the past).

The verification of the first refinement in [SB08] follows the results of Section 5. In particular, it uses a *protocol-local* invariant that is similar to formula (32): the conjunction over all instances *Protocol-g* becomes a quantifier over all *tid*s that have been used so far. These transaction identifiers also implicitly fix the mapping τ. The quantification over busy subspaces W_d considers the two spaces of the From and To purses. Using *protocol-local* invariants is a result of using the results of this paper. The older [SGH$^+$07] used more complex purse-local invariants.

7.7. Other Possibilities

Our general theory shows that even more possibilities than have been discussed above are actually possible. For example, the refinement of [BPJS07] could have chosen to refine *AbTransferOK* to *Ack* instead of *Val*, since *Ack* occurs as the last operation of a successful transaction. However, since in general there is a possibility that a transaction succeeds but that the *ack* message is lost, causing the *Ack* operation to be replaced by an *Abort* (which as it turns out is harmless), we infer that in such a refinement there would be a case in which *AbTransferOK* would have to be refined by *Abort*!

An alternative to the preceding is to synchronise right at the beginning, with the first (or second) *Start* event — and there are plenty of hybrid cases, combining aspects from several of the described or suggested refinements arising from the rich structure of the protocol computation tree. We leave the curious reader to work out such scenarios for himself.

7.8. The Non-2-Phase Fragments

In discussing the preceding refinements, we have always assumed that the two *Start* operations are performed first. But it could happen that one purse *Start*s and immediately afterwards *Abort*s, before the second purse has *Start*ed. This spoils the 2P property since the first purse has relinquished its use of its local state before the second purse has claimed its first use. In such a case, either purse may engage in other transactions, changing the local state, after the first purse's *Abort* and before the second purse's *Start*.

A remaining possibility is that only one purse *Start*s, and the other purse merely *Abort*s (as explicitly permitted in the event structure of Fig. 4), or indeed does nothing (a possibility allowed for in the definitions of [SCW00] though not shown in Fig. 4). In such a case, even if the other purse's *Abort* happens after the (inevitable) *Abort* of the first purse, it is arguable that the protocol is nevertheless 2P, since the other purse's use of its state amounts to no more than skip. Even if one does not accept this argument, it is evident that the breakdown of the 2P property is rather mild.

Dealing formally with such situations requires an extension of our theory, which will be discussed in detail elsewhere [BS]. Note though, that even if these situations are not serialisable via the standard 2P technique, the fact that we have $(1, 1)$ refinements of the protocol, guarantees nonetheless that these 'rogue' interleavings preserve atomic semantics.

8. Mechanical Verification

To gain assurance in the relatively informal account of protocol theory given above, some mechanical verification has been undertaken, using the KIV theorem prover. As well as supporting the preceding theory, this constitutes an interesting exercise in formal verification in its own right.

[11] As an alternative to such a constraint, which falls slightly outside of our general framework, we could explicitly introduce a *tid* server agent whose 'protocol' consisted of the doling out of fresh *tid*s as external outputs, with our default I/O assumption of reliable point-to-point delivery upholding the freshness and absence of duplication requirements.

KIV [RSSB98] is an interactive theorem prover for many-sorted higher-order logic. There are several extensions to this logic (Dynamic Logic, Temporal Logic and a logic for Java programs), but they are not used here. Structured algebraic specifications can be built from elementary theories using the standard operators (similar to CASL [CoF04]): union, enrichment, renaming and actualization of parametric specifications. Theorem proving uses sequent calculus.

As a first step towards a formalized theory of protocols, KIV specifications and proofs have been developed for the isolated protocols of Section 3. The results are available on the Web [KIV07]. Checking theorems with KIV led to small improvements which are already incorporated in Section 3, so in this section we only discuss a few topics, which are relevant when transferring pencil-and-paper proofs to an interactive theorem prover, and we give a lemma used in Theorem 3.8, that shows a modularization of the proof.

When formalizing the notion of execution paths a first difficulty is of course that no 'three dots notation' is available in formal specifications. Instead a free data type has been defined in KIV. Using Z notation this data type can be written as:

$$path ::= mkV\langle\langle V\rangle\rangle \mid mkpa\langle\langle V \times J \times P \times path\rangle\rangle \tag{33}$$

A number of operations are needed for paths. $\#pa$ is the number of steps of path pa, its nth node is $pa[n]$ for $0 \leq n \leq \#pa$, and its first and last nodes are $pa.first := pa[0]$ and $pa.last := pa[\#pa]$. The concatenation $pa + pa'$ of two paths pa and pa' is defined when $pa'.first = pa.last$. We also need the first n steps $pa\ to\ n$ (written infix) of a path, and the rest $pa\ from\ n$. $inputs(pa)$ and $outputs(pa)$ are the inputs resp. outputs done on a path. Finally, $Step(pa,n) \in V \times J \times P \times V$ is the n'th step of pa. A predicate $Path(pa)$ is defined recursively, which holds, iff every step satisfies some $St(\rho)(Step(pa,n))$. An argument ρ from some index type CIx replaces the subscript in St_ρ; the (higher-order) type of St being:

$$St : CIx \to V \times J \times P \times V \to bool \tag{34}$$

To give formal definitions of $FPath$ (3), $BPath$ and $MPath$, two unspecified predicates $init$ and $final$ characterizing initial and final states are used. Around 40 lemmas are proved over this theory and used as rewrite rules to get some basic automation for the main proofs.

The definition of $Protocol$ (cf. (6)) becomes:

$$Protocol(v,js,ps,v') == \exists\, pa \bullet MPath(pa) \wedge inputs(pa) = js \wedge outputs(pa) = ps \tag{35}$$

A synchronization assignment is defined as a function $SA : path \to nat$. The idea is that the synchronization step of a path is $Step(pa, SA(pa))$. Function SA is specified by two constraints:

$$MPath(pa) \Rightarrow SA(pa) < \#pa \tag{36}$$

$$MPath(pa) \wedge MPath(pa') \wedge n \leq \#pa \wedge m \leq \#pa' \wedge pa[n] = pa'[m] \Rightarrow (SA(pa) < n \Leftrightarrow SA(pa') < m) \tag{37}$$

The first axiom should be obvious, the second is a consistency condition: for two maximal paths, which have a state in common, the synchronization point must either be before that node in both paths, or both synchronization steps must follow the common node. Based on this definition we can characterize the steps of a maximal path to be the disjoint union of FS, BS and SA steps. As an example, the n'th step of path pa is a forward skip step iff $FS(pa,n)$ holds:

$$FS(pa,n) == MPath(pa) \wedge (n < SA(pa) \vee SA(pa) < n < \#pa \wedge OD(pa\ to\ SA(pa)) \tag{38}$$

where

$$OD(pa) == FPath(pa) \wedge \forall\, pa1, pa2 \bullet MPath(pa + pa1) \wedge MPath(pa + pa2) \Rightarrow pa1.last = pa2.last \tag{39}$$

As Lemmas for Theorem 3.8 and Corollary 3.9 we then prove that all steps can be simulated forwards and backwards, the only exception being BS steps, which can only be simulated backwards:

$$\text{BS-BW} : MPath(pa) \wedge BS(pa,n) \wedge R^1(u,v') \wedge pa[n] = v \wedge pa[n+1] = v' \Rightarrow R^1(u,v) \tag{40}$$

$$\text{FS-FW} : FS(pa,n) \wedge R^1(u,v) \wedge pa[n] = v \wedge pa[n+1] = v' \Rightarrow R^1(u,v') \tag{41}$$

$$\begin{aligned}\text{SA-FW} : {} & MPath(pa) \wedge R^1(u,v) \wedge Step(pa, SA(pa)) = (v,j,p,v') \\ & \Rightarrow \exists\, u', i, o, k \bullet At(k)(u,i,o,u') \wedge R^1(u',v') \wedge Input1(i,j) \wedge Output1(o,p)\end{aligned} \tag{42}$$

$$\begin{aligned}\text{SA-BW} : {} & MPath(pa) \wedge R^1(u',v') \wedge Step(pa, SA(pa)) = (v,j,p,v') \\ & \Rightarrow \exists\, u, i, o, k . At(k)(u,i,o,u') \wedge R^1(u,v) \wedge Input1(i,j) \wedge Output1(o,p)\end{aligned} \tag{43}$$

$$\text{FS-BW} : MPath(pa) \wedge FS(pa,n) \wedge R^1(u,v') \wedge pa[n] = v \wedge pa[n+1] = v' \Rightarrow R^1(u,v) \tag{44}$$

$$\text{BS-BW} : \mathit{MPath}(pa) \wedge R^1(u,v') \wedge pa[n] = v \wedge pa[n+1] = v' \Rightarrow R^1(u,v) \tag{45}$$

The proof of the last two lemmas requires Ass. 3.2.2, the others do not. The lemmas are independent of Ass. 3.1.3 which require all concrete states to be reachable. Based on the characterization of steps on paths, we can now define a global characterization of steps:

$$\mathit{BS}(v,j,p,v') == \exists\, pa, n \bullet \mathit{BS}(pa,n) \wedge \mathit{Step}(pa,n) = (v,j,p,v') \tag{46}$$

$$\mathit{FS}(v,j,p,v') == \neg \mathit{BS}(v,j,p,v') \wedge \exists\, pa, n \bullet \mathit{FS}(pa,n) \wedge \mathit{Step}(pa,n) = (v,j,p,v') \tag{47}$$

$$\mathit{SA}(v,j,p,v') == \exists\, pa.\mathit{MPath}(pa) \wedge \mathit{Step}(pa, \mathit{SA}(pa)) = (v,j,p,v') \tag{48}$$

Note that a step which is an FS step on one path and a BS step on another, must be classified as a BS step, since it is the successor of an OD step on *some* path. The three classes of steps are proven to be disjoint, and provided all states are reachable every step $\mathit{St}(\rho)(v,j,p,v')$ falls into one of the three classes. This allows us to prove Theorem 3.8 formally. As an example, the definition of Clause 4 of Theorem 3.8 is proven formally as:

$$R^1(u',v') \wedge \mathit{BS}(v,j,p,v') \Rightarrow R^1(u',v) \tag{49}$$

using Lemma (45). The four clauses (44), (43) and (45) together imply Corollary 3.9. We also prove that forward simulation is always possible by choosing the synchronization step as the last step of every maximal path:

$$\begin{aligned}&(\forall\, pa \bullet \mathit{MPath}(pa) \Rightarrow \mathit{SA}(pa) = \#pa - 1) \wedge \mathit{St}(\rho)(v,j,p,v') \wedge R^1(u,v)\\ &\quad \Rightarrow \exists\, u' \bullet R^1(u',v') \wedge (u = u' \vee \exists\, i,o \bullet \mathit{Atomic}(u,i,o,u'))\end{aligned} \tag{50}$$

The KIV proofs for the theorems of Section 3 are relatively small compared to other KIV case studies (eg. the Mondex case study [SGHR06b, SGH+07] already mentioned). The tricky bit about them is mainly to get all the assumptions right for all the cases. As an example, the borderline case of an *MPath* consisting of a single node must be forbidden, since then constraint (36) is not satisfiable.

9. Conclusions, and Further Work

In the preceding sections we took the Mondex Electronic Purse —a prime example of a protocol enacted between a number of parties that was designed to achieve the effect of an atomic action— and we looked for a generalisation. We developed a refinement framework based on seeing the atomic action as a shallow computation tree and the protocol as a computation DAG, and saw that we could choose the way that the atomic action was synchronised with the protocol in a 'small diagram' refinement relatively freely. The properties of the choice, in particular how potential abstract outcomes were related to synchronisation points, was closely related to the prospects for forward and backward simulation at the small diagram level.

We then embedded this formulation of an isolated protocol run in a framework enabling different runs of perhaps different protocols to be interleaved in a natural way. When combined with a fairly standard 2-phase property, these system runs could be serialised, showing that the atomicity abstraction survives. While serial runs corresponded to the equivalent sequence of atomic actions in a relatively transparent way, the correspondence between non-serial runs and their atomic counterparts required the adaptation of the relationship between atomic actions their refining protocols. Specifically, in a serial framework, each protocol instantiation starts and ends cleanly, allowing the formulation of a simple functional atomic/protocol 'big step' retrieve relation R, which then allowed the 'small step' past and future retrieve relations R^P and R^F to be extracted straightforwardly. In a non-serial framework, the protocol instantiations' starts and ends are 'ragged' and the properties of R so useful for the refinement investigation get heavily obscured in the fragmented retrieve relations $\widehat{R^1_{A_d}}$ needed to accomodate the non-serial features (which are often non-functional even though R itself is). We regard the clarity brought about by the separation between atomic/protocol refinement concerns and protocol instantiation concerns in our approach, to be a major factor in its favour.

We then confronted the theory with various refinements for Mondex that have been created in the recent past, whether in direct connection with the Verification Grand Challenge [JOW06, Woo06, WB07] or otherwise, and showed that the flexibility regarding synchronisation points brought out by the general theory was well borne out in these various refinements.

Furthermore, the whole of this framework has been mechanically verified using the KIV theorem prover. To achieve this, some mild transliterations from the original version of Section 3 were needed to facilitate a better fit to KIV, and the process indeed resulted in some worthwhile improvements to the theoretical formulation, as already noted in Sect 8 — the detailed results are available on the web.

As noted at the end of Section 3, the approach advocated in this paper permits a different strategy for verifying protocols. In the new approach, the steps required to verify a specific protocol are, at bare minimum:

1. Choose the atomic action shallow computation tree.
2. Choose the protocol computation DAG and big-step *R*.
3. Confirm that a suitable synchronisation assignment exists.
4. Confirm that the protocol DAG is 2PXDD-normal.
5. Confirm that for any system run consisting exclusively of instantiations of protocol steps, the instantiations simulate those protocol steps suitably.

Compared with the list at the end of Section 3, the above is simpler, for the reason that the omitted steps are true generically. Only what is noted above is required to connect the specific details of a specific protocol to the generic facts. This dramatically cuts down the verification workload.

As also noted at the end of Section 3, the approach advocated in this paper amounts to a different notion of what it means for a protocol to be correct. However, as we also pointed out, by completing our construction with an abstract system level that instantiates a sequence of abstract atomic actions (obviously there are no awkward serialisation issues here) we can make a direct connection with the traditional approach, so we can have the best of both worlds.

Besides the gratifying way that all this worked out, a couple of further interesting questions suggest themselves. The first is, that although the majority of 'normal' Mondex transactions (including not only successful ones, but also ones that fail in a 'normal' kind of way) are 2-phase —and the modification of the protocol suggested by Schellhorn et al. in [SGH$^+$07] in order to design out the possibility of a certain kind of denial of service attack is 2-phase in its entirety— the original Mondex protocol has some (in practice rare, but in theory interesting) non-2-phase parts. These are not covered by the theory of this paper, and will be explored elsewhere [BS].

A second is, that Mondex is what we called an isolated protocol. That is to say, once the protocol has commenced, the parties engaging in it are fixed, and no intrusion by other agents is contemplated. (In practice, the Mondex purse's local state determines how much notice is taken of which messages from which agents — the options of ignoring any message, or of aborting the current protocol run are deliberately always enabled in Mondex.) Thus it is natural to ask how the theory develops for protocols having state that is genuinely shared between a number of agents, including cases where the number of agents is not necessarily determined at the start of the protocol, and cases where the boundary between those agents genuinely participating in the protocol and those not participating in the protocol is more diffuse. Again, the theory of this paper does not cover such situations, and this is another issue to be explored in [BS].

References

[AH06] J.-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundamenta Informaticae*, 21, 2006.

[B4f] Clearsy. b4free tool home page. http://www.b4free.com.

[BC88] G. Boudol and I. Castellani. Concurrency and Atomicity. *Journal of Theoretical Computer Science*, 59:25–84, 1988.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[BN97] P. A. Bernstein and E. Newcomer. *Transaction Processing*. Morgan Kaufmann, 1997.

[Bou90] G. Boudol. Flow Event Structures and Flow Nets. In Guessarian, editor, *Semantics and Systems of Concurrent Processes, Proc. LITP-90*, pages 62–95. Springer LNCS 469, 1990.

[BPJS07] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the Purse: The Balance Enquiry Quandary, and Generalised and (1,1) Forward Refinements. *Fund. Inf.*, 77:29–69, 2007.

[BS] R. Banach and G. Schellhorn. Atomic Actions, and their Refinements to Not-so-Isolated Protocols. In preparation.

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

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

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

[DoTaI91] Department of Trade and Industry. Information Technology Security Evaluation Criteria, 1991. http://www.cesg.gov.uk/site/iacs/itsec/media/formal-docs/Itsec.pdf.

[DW03] J. Derrick and H. Wehrheim. Using Coupled Simulations in Non-atomic Refinement. In D. Bert, J. Bowen, S. King, and M. Walden, editors, *ZB 2003: Formal Specification and Development in Z and B*, pages 127–147. Springer LNCS 2651, 2003.

[GR93] J. Gray and A. Reuter. *Transaction Processing*. Morgan Kaufmann, 1993.

[Gur95] Yuri Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford Univ. Press, 1995.

[HGS06] A. E. Haxthausen, C. George, and M. Schütz. Specification and Proof of the Mondex Electronic Purse. In M. Reed C. Xin, Z. Liu, editor, *Proceedings of 1st Asian Working Conference on Verified Software, AWCVS'06, UNU-IIST Reports 348, Macau*, Nov. 2006.

[Int05] International Standards Organisation. Common criteria for information security evaluation, 2005. ISO 15408, v. 3.0 rev. 2.

[ISO02] ISO/IEC 13568. *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics: International Standard*, 2002. http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip.

[JOW06] C.B. Jones, P. O'Hearne, and J. Woodcock. Verified Software: A Grand Challenge. *IEEE Computer*, 39(4):93–95, April 2006.

[JWe08] C. Jones and J. Woodcock (eds.). FAC Special Issue on the Mondex Verification. *Formal Aspects of Computing*, 20(1):1–139, January 2008.

[KIV06] Web presentation of the Mondex case study in KIV, 2006. http://www.informatik.uni-augsburg.de/swt/projects/mondex.html.

[KIV07] Web presentation of isolated protocol refinement in KIV, 2007. http://www.informatik.uni-augsburg.de/swt/projects/Refinement/protocolrefine.html.

[LV93] N. A. Lynch and F. W. Vaandrager. Forward and Backward Simulations — Part I: Untimed Systems. Technical Report CS-R9313, C. W. I., 1993.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains. *Journal of Theoretical Computer Science*, 13:85–108, 1981.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *Automated Deduction - CADE-11. Proceedings*, LNAI 607, pages 748–752, Berlin, 1992. Saratoga Springs, NY, USA, Springer.

[PP95] G. Pinna and A. Poigne. On the nature of Events: Another Perspective in Concurrency. *Journal of Theoretical Computer Science*, 183:425–454, 1995.

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

[SB08] G. Schellhorn and R. Banach. A concept-driven construction of the mondex protocol using three refinements. In *Proceedings of ABZ 2008*, volume 5238 of *LNCS*, pages 57–70. Springer, 2008.

[Sch01] G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *Journal of Universal Computer Science (J.UCS)*, 7(11):952–979, 2001. http://www.jucs.org.

[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.

[SCW00] S. Stepney, D. Cooper, and J. Woodcock. An Electronic Purse: Specification, Refinement and Proof. Technical Report PRG-126, Oxford University Computing Laboratory, 2000.

[SGH+07] G. Schellhorn, H. Grandy, D. Haneberg, N. Moebius, and W. Reif. A Systematic Verification Approach for Mondex Electronic Purses using ASMs. In U. Glässer J.-R. Abrial, editor, *Proceedings of the Dagstuhl Seminar on Rigorous Methods for Software Construction and Analysis*, LNCS. Springer, 2007. (submitted, extended version available as [SGHR06a]).

[SGHR06a] G. Schellhorn, H. Grandy, D. Haneberg, and W. Reif. A Systematic Verification Approach for Mondex Electronic Purses using ASMs. Technical Report 27, Universität Augsburg, Fak. für Informatik, 2006. available at [KIV06].

[SGHR06b] G. Schellhorn, H. Grandy, D. Haneberg, and W. Reif. The Mondex Challenge: Machine Checked Proofs for an Electronic Purse. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *Proc. FM 2006*, volume 4085 of *LNCS*, pages 16–31. Springer, 2006.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, second edition, 1992.

[The92] The RAISE Language Group. *The RAISE Specification Language*. The BCS Practitioners Series. Prentice-Hall, 1992.

[WB07] J. Woodcock and R. Banach. The Verification Grand Challenge. *JUCS*, 13(5):661–668, May 2007.

[Win86] G. Winskel. Event Structures. In Brauer and Reisig and Rozenberg, editor, *Advances in Petri Nets*, pages 325–392. Springer LNCS 255, 1986.

[Win88] G. Winskel. An Introduction to Event Structures. In de Bakker, de Roever, and Rozenberg, editors, *Proc. REX Workshop*, pages 364–397. Springer LNCS 354, 1988.

[WN95] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science Volume 4 Semantic Modelling*, pages 1–148. Oxford Univ. Press, 1995.

[Woo06] J. Woodcock. First Steps in the The Verified Software Grand Challenge. *IEEE Computer*, 39(10):57–64, October 2006.

[WV02] G. Weikum and G. Vossen. *Transaction Processing*. Morgan Kaufmann, 2002.