DOI 10.1007/s00165-007-0059-y
BCS © 2007
Formal Aspects of Computing (2008) 20: 117–139

Formal Aspects of Computing

Mechanising Mondex with Z/Eves

Leo Freitas and Jim Woodcock

Department of Computer Science, University of York, Heslington, York YO10 5DD, UK.
E-mail: leo@cs.york.ac.uk; jim@cs.york.ac.uk

Abstract. We describe our experiences in mechanising the specification, refinement, and proof of the *Mondex Electronic Purse* using the Z/Eves theorem prover. We took a conservative approach and mechanised the original LaTeX sources without changing their technical content, except to correct errors. We found problems in the original specification and some missing invariants in the refinements. Based on these experiences, we present novel and detailed guidance on how to drive Z/Eves successfully. The work contributes to the Repository for the Verified Software Grand Challenge.

Keywords: Correctness; Electronic finance; Grand challenges; Grand Challenge in Verified Software; Mondex; Refinement; Security; Smart cards; Software archaeology; Theorem proving; Verification; Verified Software Repository; Z/Eves; Z notation

1. Introduction

The Mondex case study is a year-long pilot project launched in January 2006 as part of the International Grand Challenge in Verified Software [JOW06, QPQURL]. The case study demonstrates how research groups can collaborate and compete in scientific experiments to generate artefacts to populate the Grand Challenge's Verified Software Repository [BHW06]. The objective is to verify a key property of the Mondex smart card in order to assess the current state of proof mechanisation.

Mondex [MonURL] is an electronic purse hosted on a smart card and developed about ten years ago to the high-assurance standard ITSEC Level E6 [ITS91] by a consortium led by NatWest, a UK high-street bank. Purses interact using a communications device, and strong guarantees are needed that transactions are secure in spite of power failures and mischievous attacks. These guarantees ensure that electronic cash cannot be counterfeited, although transactions are completely distributed. There is no centralised control: all security measures are locally implemented, with no real-time external audit logging or monitoring.

A security flaw in Mondex would be a serious matter, so any claim about the correctness of its behaviour required careful justification. Logica, a commercial software house, with assistance from the University of Oxford, used the Z notation [Spi92, WoD96] for the development process [SCW00, SCW98, CSW02]. Formal models were constructed of the system and its abstract security policy, and conducted hand-written proofs that the system design possesses the required security properties. The abstract security policy specification is about 20 pages of Z; the concrete specification (an n-step protocol) is about 60 pages of Z; the verification, suitable for external evaluation, is about 200 pages of proofs; and the derivation of new refinement rules is about 100 pages [SCW00, CSW02].

The original proof was carefully structured for understanding, something much appreciated by the Mondex case study groups. The original proof was vital in successfully getting the required certification. It was also useful

Correspondence and offprint requests to: J. Woodcock, E-mail: jim@cs.york.ac.uk; leo@cs.york.ac.uk

in finding and evaluating different models. The original team made an important modelling discovery, which led to an abstraction that gave the precise security property and invariants that explain why the protocol is secure. The resulting proof revealed a bug in the implementation of a secondary protocol, explained what had gone wrong, and produced a convincing counterexample that the protocol was flawed. This led to an insight to change the design to correct the problem. Third-party evaluators also found a bug: an undischarged assumption in the hand-written proofs.

A commercially sanitised version of the documentation of the Mondex development is publicly available [SCW00]. It contains the Z specifications of security properties; the abstract specification; the intermediate-level design; the concrete design; and rigorous correctness proofs of security and conformance. Originally, there was absolutely no question of mechanising proofs: it was believed that the extra cost would far outweigh the benefit of greater assurance in this case. The feeling that mechanising such a large proof *cost-effectively* was beyond the state of the art ten years ago gives us two sharply focused questions. (a) Was that really true then?, (b) is it true now?

Seven groups came together to collaborate and compete. The teams were:

Alloy	[Jac06a, Jac06b]	Daniel Jackson/Tahina Ramananandro	MIT
Event-B	[MAV05]	Michael Butler	SOUTHAMPTON
OCL	[UMLURL]	Martin Gogolla	BREMEN
PerfectDeveloper	[Cro04]	David Crocker	ESCHER LTD
π-calculus	[Mil99]	Cliff Jones/Ken Pierce	NEWCASTLE
Raise	[DGJ+02]	Chris George/Anne Haxthausen	MACAO/DTU
Z	[Spi92, WoD96]	Leo Freitas/Jim Woodcock	YORK

We all agreed to work for one year, without funding. Meanwhile, separately and silently, a group led by Gerhard Schellhorn at the University of Augsburg began work using KIV and ASMs [SGH+06b, SGH+06a].

Two distinct approaches emerged amongst the seven. The *Archaeologists* made as few changes as possible to the original documentation. They reasoned that models should not be changed just to make verification easier, because how else would we know that our results had anything to do with the original specification? The *Technologists* wanted to use the best proof technology now available. Since these new tools do not work for Z, they had two choices: translate existing models into new languages; or create new models better suited to their languages in new tools.

We spend the rest of this paper describing our archaeological experiment in mechanising the proof of Mondex in Z/Eves [Saa97a]. We assume throughout this paper that the reader is thoroughly familiar with the Z notation ([WoD96] is an introductory textbook). As a side effect of our work, we discovered undocumented techniques for driving Z/Eves, and we present these in Sect. 2. We discuss our formalisation in Sect. 3, including the fidelity of our mechanisation, suggestions for improvement of the existing models, and the completeness of our work. Section 4 contains a list of the problems, omissions, and errors that we found. Benchmarks might be useful in calibrating tools and comparing experiments, and we have collected some of these in Sect. 5. Finally, in Sect. 6, we present a few conclusions, some suggestions for further work, and call on interested colleagues to join in similar experiments on Mondex and other challenges in the future.

2. Driving Z/Eves

Here we provide general information about proving theorems in Z/Eves. It is useful to understand how the theorem prover works, in order to explain how to handle a specification as big and complex as Mondex. Some understanding of Z/Eves, or at least experience with theorem-proving in Z, is expected in this section.

2.1. Ability and usage directives

The level of automation in Z/Eves can be fine tuned by selecting "abilities" (boolean enabling conditions) and "usages" (scope control) [Saa97a]. Usages may be assumption (grule), forward (frule), and rewriting (rule), and they define the scope of definitions and theorems with respect to available transformation tactics. These directives must be chosen carefully in a theory to prevent the prover taking wrong turns while transforming goals. Although some guidelines exist for the proper selection of usages, it is difficult to predict the appropriate ability values without experience with the theory being proved and the prover itself.

2.2. Housekeeping rules

Rewriting rules can be used for fine-grained automation, since they are used by the specialised tactics that rewrite the goal: rewrite, reduce, prove, prove by reduce, and their corresponding trivial and normalised forms. Assumption rules can be used for coarse-grained automation, commonly needed for non-maximal type consistency checks that often appear in proofs, since they are used by every tactic that rewrites the goal. Forward rules are normally used to expose implicit facts about schema components and invariants, without expanding schema definitions. This is particularly valuable when there are complex schema expressions, because we can surgically guide specific aspects of the goal without cluttering the proof with a mass of unrelated assumptions from included schemas. The resulting proofs are easier to conduct, understand, and amend. For instance, for

$$S \mathrel{\widehat=} [x : \mathbb{N};\ s : \operatorname{seq} \mathbb{N} \mid x = \#s]$$

the following forward rules could be added

theorem frule *fSSMaxType*
$\forall S \bullet s \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

theorem frule *fSSInv*
$\forall S \bullet x = \#s$

giving access to simple properties of S required by almost all subsequent proofs.

On the other hand, there are theorems for situations where either automation is not necessary, or where a syntactic issue restricts the declaration of theorems as rules [Saa97a, p.41]. Another reason for some theorems not to be declared as rules arises when the tautology-elimination engine automatically removes recently applied rewriting rules during transformation tactics, hence the important information just introduced might be removed from the assumptions before being used. Therefore, it is desirable to have theorems that are not involved in automation schemes, and the rule of thumb is: theorems that are not applied often or are not general enough should not be considered as rewriting rules. Bear in mind that although a large number of rewriting rules provides higher levels of automation, the side effect is the degradation of performance. As the number of rewriting rules available increases, it takes longer to check whether their conditional sub-formulae patterns match. There is also a higher chance of clashes among multiple choices of such matches.

2.3. Non-maximal generic types

The abstraction provided by generic types allows specifications to be simpler, smaller, and reusable. Nevertheless, it also introduces quite awkward and unexpected problems if not used carefully. There is a specific idiom to be followed in Z/Eves in order to use generic types properly: whenever generic actual-parameters are needed for theorem instantiations, the maximal type ought to be given in order to avoid side-condition proof obligations. Without the appropriate instantiation, it is easy to end up with an obviously *true* goal the prover cannot automatically discharge. For example, suppose one needs to instantiate the following Z/Eves toolkit theorem manually

theorem domSeq $[X]$
$\forall s : \operatorname{seq} X \bullet \operatorname{dom} s = 1 \mathinner{..} \# s$

where the generic type X is to be instantiated with the natural numbers $\mathbb{N}$ using the proof command

use domSeq[$\mathbb{N}$][s := *mySeq*];

The use command expects a mandatory theorem name, and optional generic actual-parameters and universally quantified variables, where type inference and automatic name association is attempted if none of these parameters are given. In this example, the hypothesis added to the goal carries the generic actual-parameters as in

$$s \in \operatorname{seq} \mathbb{N} \Rightarrow \operatorname{dom}[\mathbb{Z}, \mathbb{N}]\, s = 1 \mathinner{..} \#[\mathbb{Z} \times \mathbb{N}]\, s$$

which turns out to be hard to discharge automatically. Even though usually provable, discharging such side-conditions is difficult, time consuming, and completely unrelated to the desired goal under concern. For our example, the appropriate instantiation of X is its maximal type value of the integers $\mathbb{Z}$, which eliminates the need to instantiate every generic formal-parameter in the theorem's expressions. That is because types inferred by Z/Eves (and used while resolving implicit generic actual-parameters) are maximal.

2.4. Automation for schemas and bindings

Z/Eves adds theorems for each schema: for S, we get an assumption rule defining the maximal type of the set of bindings for S and a forward rule to infer the types of each schema component.[1]

theorem grule $S\$declaration$
$S \in \mathbb{P}\, (\!|x : \mathbb{Z};\ s : \mathbb{P}(\mathbb{Z} \times \mathbb{Z})|\!)$

theorem frule $S\$declarationPart$
$S \Rightarrow x \in \mathbb{N} \land s \in \operatorname{seq} \mathbb{N}$

Rules to reason about θ-expressions are also added, and we describe them in order of relevance for automation.

- θ and binding expressions

  theorem rule S$thetaInSet
  $\theta S \in (\!|x : x';\ s : s'|\!) \Leftrightarrow x \in x' \land s \in s'$

- θ-expression as schema inclusion

  theorem rule S$thetaMember
  $\theta S \in S \Leftrightarrow S$

- θ-expression equality, which is useful for Ξ inclusions

  theorem rule S$thetasEqual
  $\theta S = \theta S' \Leftrightarrow x = x' \land s = s'$

- θ binding selection expression (for each schema component)

  theorem rule S$select$x
  $\theta S.x = x$

- θ-expression as variables of type S [2]

  theorem disabled rule S$member
  $x\$ \in S \Leftrightarrow (\exists S \bullet x\$ = \theta S)$

- θ-expression as variables of the binding type of S

  theorem disabled rule S$inSet
  $x\$ \in (\!|x : x';\ s : s'|\!) \Leftrightarrow (\exists x : x';\ s : s' \bullet x\$ = \theta S)$

- the meaning of the set of bindings of S

  theorem disabled rule S$setInPowerSet
  $(\!|x : x;\ s : s|\!) \in \mathbb{P}\, (\!|x : x';\ s : s'|\!) \Leftrightarrow x \in \mathbb{P}\, x' \land s \in \mathbb{P}\, s' \lor x = \{\} \lor s = \{\}$

Although this gives a powerful automation toolkit for schema mechanisation, when schema components are quite complex, such as layered inclusions or components with binding type, one might still need to expose implicit facts about schemas. For instance, the following frule is needed in Mondex for discharging goals involving *StartFromPurseEafromOkay* [SCW00, p.30], where some of its components refer to bindings of the *CounterPartyDetails* schema.

theorem frule fCounterPartyDetailsValueType
$x \in CounterPartyDetails \Rightarrow x.value \in \mathbb{N}$

Luckily, with the above toolkit, such theorems are often proved quite trivially.

[1] The name *S$declaration* is an extended identifier (the dollar-sign separates the identifier from its extension); such names are generated by Z/EVES to avoid name clashes.

[2] A command modifier can be used to enable or disable a theorem or definition. Disabled commands are not used automatically, but must be invoked explicitly.

2.5. Free-type rules

We usually need the implicit fact that the constructors are injective when we reason about a free type. For example, for the free type *List*, *cons* is injective:

$$List ::= nil \mid cons\langle\!\langle \mathbb{Z} \times List \rangle\!\rangle$$

theorem grule *gConsPInjType*
$cons \in (\mathbb{N} \times List) \rightarrowtail List$

To prove this rule, we often need to provide intermediate theorems for expressions that appear as side-conditions on goals. These are similar theorems stating that *cons* is an injection, a partial function, a relation, or even a set of pairs of its maximal type.

In the *Mondex* case study, free-type constructors appear frequently in the specification of the protocol messages, with schema bindings as parameters to give an extra level of complexity. For instance, the schema *ValidStartFrom* refers to the expression $startFrom^{\sim} m?$, where the result is a *CounterPartyDetails* schema. Thus, we need not only the injectivity theorems, but also forward rules about the binding type of the *CounterPartyDetails* schema as

theorem frule fCounterPartyDetailsMember
$x \in CounterPartyDetails \Rightarrow x \in \langle\!| name : NAME;\ nextSeqNo : \mathbb{N};\ value : \mathbb{N} |\!\rangle$

theorem frule fCounterPartyDetailsInSetMember
$x \in \langle\!| name : NAME;\ nextSeqNo : \mathbb{N};\ value : \mathbb{N} |\!\rangle \Rightarrow x \in CounterPartyDetails$

Thanks to the automation for θ and binding expressions provided by Z/Eves, these forward rules are relatively easy to prove by expanding the schema definition.

2.6. Rules for axiomatically declared functions

The successful automation of proofs involving functions requires additional rules about the function's (maximal) type, the type of its result, and about its totality. Depending on the structure of the function type and which toolkit theorems are used, we often need more than one type theorem, and we must predict these in advance for the best automation. Fortunately, there is a pattern to these rules. In a function f declared as

$$\frac{f : \operatorname{seq}(\mathbb{F}\ \mathbb{N}) \rightarrow \operatorname{seq}\ \mathbb{N}}{\cdots}$$

we might need to add type information as assumptions (*grules*), such as

$$f \in \mathbb{P}(\mathbb{P}(\mathbb{Z} \times \mathbb{P}\ \mathbb{Z}) \times \mathbb{P}(\mathbb{Z} \times \mathbb{Z}))$$

which is often required in applying rules, or

$$f \in \operatorname{seq}(\mathbb{P}\ \mathbb{Z}) \leftrightarrow \operatorname{seq}\ \mathbb{Z}$$

which might appear when relational definitions from the toolkit are used.

When using a free type we usually need to prove a theorem about the injectivity of its constructors. It gets more complicated when the free type refers to a schema binding, as is the case with the *MESSAGE* free-type in Mondex [SCW00, p.26].

$$MESSAGE ::= startFrom\langle\!\langle CounterPartyDetails \rangle\!\rangle \mid \cdots$$

In Mondex, we have expressions such as

$$cpd \in CounterPartyDetails \land m? \in MESSAGE \land cpd = startFrom^{\sim} m?$$

To discharge the consistency checks about this expression, the injectivity theorem is given with the maximal type of *CounterParyDetails* as

theorem grule gStartFromInjType
$startFrom \in \langle\!| name : NAME;\ value : \mathbb{N};\ nextSeqNo : \mathbb{N} |\!\rangle \rightarrowtail MESSAGE$

It establishes that the *startFrom* is an injection between *CounterPartyDetails* and *MESSAGE*. Furthermore, to prove this injectivity theorem, we need auxiliary lemmas about functional and relational types of *startFrom*, as well as extra (maximal) type rules for *CounterPartyDetails*, presented above.

2.7. Schema invariants and preconditions

While proving theorems involving schema inclusion, it is often necessary to expose particular elements and predicates of the state invariant. Nevertheless, it is not productive to expand the inclusion, as this leads to lengthy and complex predicates. Thus, in order to surgically expose schema components, we need to introduce forward (frule) rules. For instance, let us define a state schema S as

$$S \mathrel{\widehat=} [\, x : \mathbb{N};\ s : \mathrm{seq}\ \mathbb{N} \mid x \geqslant \# s \land s \neq \langle\rangle \,]$$

and an operation over this state as

$$Op \mathrel{\widehat=} [\, \Delta S;\ i? : \mathbb{N} \mid x' = x + 1 \land s' = s \frown \langle i? \rangle \,]$$

Due to the nature of the predicates involved in the operations, as well as the theorems we might be proving about them (such as precondition calculation or refinement simulation), we might want to expose parts of the state invariant in the middle of a proof without expanding S. This careful control is necessary to avoid the hypothesis and goal explosion problem, but it should also be as automatic as possible to avoid the need for micro-management of the proof. In order to achieve all this, we introduce forward rules, such as

theorem frule *fSSInv*1
$\forall S \bullet s \neq \langle\rangle$

theorem frule *fSSInv*2
$\forall S \bullet x \geqslant \# s$

They are trivially proved by expanding S; nevertheless, they allow us to conclude the invariants of S without expansion, provided that S appears as part of our original goal.

As occurred before for functions and free types, we might also need to expose the (maximal) type of the schema components, hence theorems like

theorem frule fSSMaxType
$\forall S \bullet s \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$

could be defined. The rationale of when such theorems should be introduced depends on the kind of goals that appear in proofs about the schema in question.

During the refinement proofs in Mondex, it is often mentioned that particular elements of *BetweenWorld* should be used. To do that without expanding the schema, one just needs to add an frule as

theorem frule fBetweenWorldMaybeLostExpansion
$\forall BetweenWorld \bullet maybeLost = (fromInEpa \cup fromLogged) \cap toInEpv$

which is again trivially true from the definition of *BetweenWorld* [SCW00, p.42].

A strategy is needed for the proof of the preconditions of complex promoted operations in Mondex. Additional lemmas are needed for the precondition of the promoted operation, and although these lemmas cannot be used directly in the promoted precondition proofs, they define how to instantiate the quantifiers in the promoted precondition. This strategy turned out to be very effective.

2.8. Z idioms

Non-maximal generic types instantiation

Unfortunately, using maximal types as generic actual-parameters is not always possible. This happens whenever non-maximal types are referenced in rewriting rules, or the user requires information about a non-maximal type property. For instance, a theorem about the result of partial function application could be given as

theorem applyInRanPfun[A,B]
$\forall f : A \nrightarrow B \bullet \forall a : \mathrm{dom}\ f \bullet f\, a \in \mathrm{ran}\ f \land f\, a \in B$

Usually, one has a function f on non-maximal types A and B, say seq $\mathbb{N}$ and $\mathbb{N}_1$, respectively. Whenever the information about strictly positive numbers is needed, the generic actual for B needs to be (the non-maximal type) $\mathbb{N}_1$ instead of (the maximal type) $\mathbb{Z}$. Under these circumstances, to avoid the problems with generic actual-parameters around expressions (see Sect. 2.3), the trick is to universally quantify the generic non-maximal types of interest (A and B) as power sets of the corresponding maximal type. Then, the actual theorem can come next with the quantified (non-maximal) types (X and Y) given as

theorem applyInRanPfun $[X, Y]$
$\forall A : \mathbb{P}\, X;\ B : \mathbb{P}\, Y \bullet \forall f : A \pfun B \bullet \forall a : \mathrm{dom}\ f \bullet f\, a \in \mathrm{ran}\ f \land f\, a \in B$

Nevertheless, this trick often creates the kind of syntactic restrictions that forbids a theorem to be a rewriting rule (see Sect. 2.2), hence automation is affected and an explicit command with both maximal and desired non-maximal types must be used

use applyInRanPfun[$\mathbb{P}(\mathbb{Z} \times \mathbb{Z}), \mathbb{Z}$][A := seq $\mathbb{N}$, B := $\mathbb{N}_1$, f := *myFunction*, a := *myArg*];

One might argue that these technicalities increase the complexity of the theorem usage and automation, due to explicit necessity of instantiations and impossibility to declare it as a rewriting rule. The important point is that the solution is effective in removing side-conditions with expressions involving non-maximal generic types as mentioned in Sect. 2.2, hence the compromise in automation introduced is well-balanced with the simplicity of the goals to be proved.[3] For Mondex, this situation happens in the proof of one-point-mu below, and in the proofs involving finiteness of *PayDetails* (see Sect. 3.1).

One-point-mu

It is notoriously hard to reason about definite descriptions in Z, mainly because the notion is primitive to the language: it is not easy to eliminate an arbitrary μ-term. There are three very specific automation rules in Z/Eves: two symmetric rules concerning equality with a μ-term; and a third rule concerning a definite description drawn from a singleton set. In each rule there is a candidate value for the expression, and we can call them one-point rules, by analogy with predicate calculus. Definite description is used in Mondex to build bindings with components defined pointwise, and we devised another one-point rule to eliminate a μ-term in favour of an explicit binding: a θ-term with substitutions for each component. That is, to replace $(\mu S \mid x = e \land \cdots \land z = g)$ by $\theta S[x := e, \ldots, z := g]$. The special Z/Eves assignment in a θ-expression is a shorthand for schema substitution with expressions rather than names:

$\theta S[x := e] = (\mathbf{let}\ x == e \bullet S)$

For example, here is such a theorem involving the schema *PayDetails*:

theorem rule rStartFromMuPayDetailsValue
$\forall name : NAME;\ nextSeqNo : \mathbb{N};\ cpd : CounterPartyDetails \mid name \neq cpd.name \bullet$
$(\mu PayDetails \mid from = name \land to = cpd.name \land value = cpd.value \land$
$fromSeqNo = nextSeqNo \land toSeqNo = cpd.nextSeqNo)$
$= \theta PayDetails[from := name, to := cpd.name, value := cpd.value,$
$fromSeqNo := nextSeqNo, toSeqNo := cpd.nextSeqNo]$

This is useful because Z/Eves has better automation support for bindings, and because it is an automatic rewriting rule, equality substitution takes place fully automatically. This is also useful in proving the next three rules, which are necessary for every proof involving definite descriptions of this sort. First, μ *PayDetails* maximal type in *StartFromPurseEafromOkay*:

theorem rule rStartFromMuPayDetailsMaxType
$\forall name : NAME;\ nextSeqNo : \mathbb{N};\ cpd : CounterPartyDetails \mid name \neq cpd.name \bullet$
$(\mu m : \{PayDetails \mid from = name \land to = cpd.name \land value = cpd.value \land$
$fromSeqNo = nextSeqNo \land toSeqNo = cpd.nextSeqNo\})$
$\in \langle\!| from : NAME;\ fromSeqNo : \mathbb{Z};\ to : NAME;\ toSeqNo : \mathbb{Z};\ value : \mathbb{Z} |\!\rangle$

[3] A full discussion of the difficulties inherent in generic formals would require a full paper in itself.

Second, μ *PayDetails* non-maximal type in *StartFromPurseEafromOkay*:

theorem rule rStartFromMuPayDetailsType
$\forall name : NAME;\ nextSeqNo : \mathbb{N};\ cpd : CounterPartyDetails \mid name \neq cpd.name \bullet$
$(\mu m : \{PayDetails \mid from = name \land to = cpd.name \land value = cpd.value \land$
$fromSeqNo = nextSeqNo \land toSeqNo = cpd.nextSeqNo\}) \in PayDetails$

Third, μ *PayDetails from* purse type in *StartFromPurseEafromOkay*:

theorem rule rStartFromMuPayDetailsFromType
$\forall name : NAME;\ nextSeqNo : \mathbb{N};\ cpd : CounterPartyDetails \mid name \neq cpd.name \bullet$
$(\mu m : \{PayDetails \mid from = name \land to = cpd.name \land value = cpd.value \land$
$fromSeqNo = nextSeqNo \land toSeqNo = cpd.nextSeqNo\}).from \in NAME$

Finally, four more rules are added to establish the θ-μ *PayDetails* expressions equivalence for *to* purses, with the same shape but *name* $=$ *to*. For these proofs, the knowledge about non-maximal generic actual-parameters discussed above (see Sect. 2.3) was essential.

Keep declarations non-finite

There is very limited automation for reasoning about finite sets in Z/Eves, and so the best advice is to avoid doing so unnecessarily. A useful tip is not to declare a set as finite, but rather to give its finiteness as a property. This is important because the side-conditions and rules available in the automation toolkit are always with respect to the *maximal type*. For example, instead of declaring *abAuthPurse* as a finite function

$$AbWorld \mathrel{\widehat=} [abAuthPurse : NAME \twoheadrightarrow AbPurse]$$

declare it as a function and constrain it to be finite:

$$AbWorld \mathrel{\widehat=} [abAuthPurse : NAME \pfun AbPurse \mid abAuthPurse \in NAME \twoheadrightarrow AbPurse]$$

In this way, some finiteness domain checks are avoided.

Avoid binding selection on free-type constructor results

In schema *BetwInitIn* [SCW00, p.52], the use of $(req^\sim m?).from$ directly is a bad idea for Z/Eves automation, as it incurs rather complex lemmas involving the functionality of the inverse function. Instead, one could simply declare a variable to hold such value with something like

$$BetwInitIn \mathrel{\widehat=} [\cdots;\ x : PayDetails \mid x = req^\sim m? \land \cdots \land \cdots x.from \cdots]$$

Z/Eves would then know the type of this expression, the main problem appearing in the domain check. Nevertheless, for the sake of keeping to the original as much as possible, we left it unchanged.

2.9. Extending the Z toolkit

Functional overriding

Although there are useful rules for relational overriding, there are no rules specifically for functional overriding. This operator plays a central role in updating the state in the abstract specification and security model. We added three simple rules in the proofs from [SCW00, Chap.8].

theorem rule rPFunElement $[X, Y]$
$\forall f : X \pfun Y;\ x : X;\ y : Y \mid x \in \mathrm{dom}\ f \land y = f\ x \bullet (x, y) \in f$

theorem rule rPFunSubsetOplusRel $[X, Y]$
$\forall f, g : X \pfun Y \mid g \subseteq f \bullet f \oplus g = f \oplus (\mathrm{dom}\ g \lhd f)$

theorem lPFunSubsetOplusUnitRel $[X, Y]$
$\forall f : X \pfun Y;\ x : X;\ y : Y \mid x \in \mathrm{dom}\ f \land y = f\ x \bullet f = f \oplus \{(x \mapsto y)\}$

The first theorem is useful for partial functions in general to retrieve an actual tuple represented by some function. Theorem *rPFunSubsetOplusRel* is useful in automating proofs involving functional overriding ($\oplus$) of known members ($g \subseteq f$). The last theorem is the one useful for Mondex; it says that for known members, function overriding is idempotent.

Additional support for automating finiteness

The need to prove that a set is finite arises most often from proving properties involving the cardinality operator. Here are some extra theorems to help reason about set sizes. First, smaller sets have smaller sizes.

theorem sizeOfPSubset $[X]$
$\forall T : \mathbb{F}\ X \mid S \subset T \bullet 0 \leqslant \#S < \#T$

Next, we have the maximal type of finite set cardinality, which is useful for discharging their side-conditions and type checks from proofs.

theorem disabled grule cardType $[X]$
$\forall x : \mathbb{F}\ X \bullet \#x \in \mathbb{Z}$

The next two theorems reason about the relationship between set membership and finite set cardinality.

theorem disabled rule cardDiffIsSmaller $[X]$
$\forall S : \mathbb{F}\ X;\ x : X \mid x \in S \bullet (\#(S \setminus \{x\}) < \#S)$

theorem disabled rule cardDiffLower $[X]$
$\forall S : \mathbb{F}\ X;\ x : X \mid x \in S \bullet (0 \leqslant \#(S \setminus \{x\}))$

Finite relations have finite domains and ranges.

theorem finRelHasFinDom $[X, Y]$
$\forall R : X \leftrightarrow Y \mid R \in \mathbb{F}(X \times Y) \bullet \mathrm{dom}\ R \in \mathbb{F}\ X$

theorem finRelHasFinRan $[X, Y]$
$\forall R : X \leftrightarrow Y \mid R \in \mathbb{F}(X \times Y) \bullet \mathrm{ran}\ R \in \mathbb{F}\ Y$

We also use a variation on the last rule that abstracts the sets involved. Similar rules apply to sequences.

theorem grule seqIsFinite $[X]$
$\forall s : \mathrm{seq}\ X \bullet s \in \mathbb{F}(\mathbb{Z} \times X)$

theorem seqHasFinRan $[X]$
$\forall s : \mathrm{seq}\ X \bullet \mathrm{ran}\ s \in \mathbb{F}\ X$

theorem seqHasFinRan2 $[X]$
$\forall A : \mathbb{P}\ X \bullet \forall s : \mathrm{seq}\ A \bullet \mathrm{ran}\ s \in \mathbb{F}\ A$

theorem iseqHasFinRan $[X]$
$\forall s : \mathrm{iseq}\ X \bullet \mathrm{ran}\ s \in \mathbb{F}\ X$

theorem iseqHasFinRan2 $[X]$
$\forall A : \mathbb{P}\ X \bullet \forall s : \mathrm{iseq}\ A \bullet \mathrm{ran}\ s \in \mathbb{F}\ A$

There are other mathematical datatypes in the toolkit that are not used in Mondex (for example, bags), but they could be handled in the same way.

Regarding automation, it is important to provide weakening rules for common (compound) expressions involving finiteness. This enables the rewriting engine to automatically apply for finite sets ($\mathbb{F}\,X$) other toolkit rules available for infinite sets ($\mathbb{P}\,X$).

theorem rule rFiniteMember $[X]$
$A \in \mathbb{F}\ X \Rightarrow A \in \mathbb{P}\ X$

Although the Z/Eves toolkit has a weakening rule for finite cross products, the version below seems to give higher levels of automation

theorem rule rCrossFinite
$A \times B \in \mathbb{F}(C \times D) \Leftrightarrow A = \{\} \lor B = \{\} \lor (A \in \mathbb{F}\, C \land B \in \mathbb{F}\, D)$

Finally, to complete the basic weakening rules, we prove a theorem establishing that a subset of a finite set is also finite. Generic types were avoided and implicitly (universally quantified) variables were used, in order to simplify the possible side-condition to be generated.

theorem lFinsetSubset
$X \in \mathbb{P}\, Y \land Y \in \mathbb{F}\, Z \Rightarrow X \in \mathbb{F}\, Z$

Unfortunately, because of the conjunction in the hypothesis (one of the Z/Eves syntactic restrictions for rules mentioned in Sect. 2.2), this theorem cannot be given as a rule.

Alternative finiteness definition

As mentioned above, reasoning about finiteness is difficult; there are four reasons for this. (a) Proofs about finiteness often require reasoning about total functions, injections, bijections, and set cardinality. (b) Pointwise instantiation is needed. (c) There is a lack of automation rules and toolkit theorems about finiteness. (d) Low-level rewriting for set membership is not restricted to finite sets. The definition for the finite set ($\mathbb{F}\, X$) is

$$\mathbb{F}\, X \;==\; \{ S : \mathbb{P}\, X \mid \exists n : \mathbb{N} \bullet \exists f : 1 \mathinner{..} n \rightarrow S \bullet \operatorname{ran} f = S \}$$

and for finite set cardinality

$$[X]$$
$$\# : \mathbb{F} \rightarrow \mathbb{N}$$
$$\forall S : \mathbb{F}\, X \bullet \exists f : 1 \mathinner{..} (\#S) \rightarrowtail S \bullet true$$

An alternative, inductive definition is given in the ISO Z Standard [ISO02].

$$\mathbb{F}_{new}\, X \;==\; \bigcap \{A : \mathbb{P}(\mathbb{P}\, X) \mid \{\} \in A \land (\forall a : A;\; x : X \bullet a \cup \{x\} \in A)\}$$

This is useful because there is no need to deal with instantiations, and it gives a better pattern for proofs involving the bijection present in cardinality.

3. Formalisation

3.1. Fidelity

Our formalisation is a carbon copy of *Oxford Monograph PRG-126*, [SCW00] except in two respects: finiteness of *PayDetails*, and theorems about *totalAbBalance*.

Suitable definition for NAME

In Mondex, *NAME*s are mostly used for characterising payment details between two distinct purses. Nevertheless, nothing about the structure of *NAME* is given. Therefore, in order for the hidden finiteness assumptions used throughout the monograph [SCW00] to hold, we must introduce that *NAME*s are unique and finite. Luckily, defining this is fairly obvious and abstract, and could be given as

$$NAMES \;==\; \mathbb{F}\, NAME$$

and then changing *NAME* for *NAMES* wherever necessary. Nevertheless, this does not ensure *NAMES* are unique. Furthermore, in [SCW00, p.78], the existentially quantified equivalence of the retrieve relation partition for the abstract and between worlds is provable only if there exists at least two different names. Taking this into account, let us (axiomatically) define at least two distinct known names.

$$name1, name2 : NAME$$
$$\neg\ name1 = name2$$

Next, we ensure that the definition of *NAMES* we are about to introduce is consistent. These consistency checks for axiomatic definitions are required in order to avoid introducing inconsistent axioms. In Z, that means ensuring that there exists a model for what is being axiomatically specified. In Z/Eves, although domain checks ensure well-definedness of function application, nothing is given for inconsistent specifications [Saa97a, p.22–24]. Thus, throughout our formalisation of Mondex, we have included (existential) consistency theorems for axiomatic definitions. For the definition of *NAMES*, consistency relies on the following theorem:

theorem tNamesConsistency
$$\exists NAMES : \mathbb{F}_1\ NAME \bullet (\exists n1, n2 : NAMES \bullet \neg\ n1 = n2) \land (\forall n1 : NAMES \bullet \exists n2 : NAMES \bullet \neg\ n1 = n2)$$

After proving this theorem, we can then safely axiomatically define *NAMES* as

$$NAMES : \mathbb{F}_1\ NAME$$
$$\exists n1, n2 : NAMES \bullet \neg\ n1 = n2$$
$$\forall n1 : NAMES \bullet \exists n2 : NAMES \bullet \neg\ n1 = n2$$

This structured model of *NAME*s can represent the finiteness assumptions made throughout the monograph [SCW00].

Finite definition for PayDetails

The definition of *PayDetails* [SCW00, p.24] required modification in order to make it finite. In the original monograph [SCW00] this is a bug, which leads to many claims about finiteness being unprovable (see 4.3 below). For instance, one of those finiteness assumptions comes from conjectures such as

$$PayDetails\ \widehat{=}\ [from, to : NAME;\ value, fromSeqNo, toSeqNo : \mathbb{N} \mid from \neq to]$$
$$name \in NAME \vdash \{pd : PayDetails \mid pd.from = name\} \in \mathbb{F}\ PayDetails$$

Because this set comprehension expression represents a set of *PayDetails* bindings, although the *from* purse is fixed to a *name*, we could have an infinite number of *to* purses this *from* purse is associated with. Even if we used our structured (finite) *NAMES* definition, we could still pay (infinitely) many possible *value*s. Thus, this and other assumed finiteness properties (e.g. *AuxWorld* [SCW00, p.39], *BetwFinState* [SCW00, p.53], etc.) cannot be proved throughout the monograph.

In order to make *PayDetails* schema bindings finite as required, we must also add a finite space of numbers by defining a non-empty range over $\mathbb{N}$, named *NAT*, which is bounded by a *MAX_NAT* value, and are axiomatically defined below as

$$MAX\_NAT : \mathbb{N}$$

$$NAT == 0\,..\,MAX\_NAT$$

As before, we have also included corresponding consistency theorems.

theorem tNATBoundaryConsistency
$$\exists MAX\_NAT : \mathbb{N} \bullet true$$

We do not need a consistency theorem for *NAT*, since it is defined using an abbreviation. Of course, this does not mean that it is always defined, since its value may be specified using improper expressions, such as definite descriptions that do not uniquely define an object, or a partial function applied outside its domain. However, the possibility of improper expressions will be dealt with by Z/Eves generating appropriate domain checks. Finally, the (finite) definition of the *PayDetails* schema is given as

$$PayDetails\ \widehat{=}\ [from, to : NAMES;\ value, fromSeqNo, toSeqNo : NAT \mid from \neq to]$$

using both bounded names *NAMES* and *NAT* values.

Inappropriate definition of totalAbBalance

The auxiliary toolkit definitions from [SCW00, App.D] present difficulties for mechanisation in Z/Eves because they require reasoning about finiteness and cardinality; they also require witnesses in their instantiation. For instance,

$$
\begin{array}{|l}
totalAbBalance : (NAME \twoheadrightarrow AbPurse) \to \mathbb{N} \\
\hline
totalAbBalance\, \emptyset = 0 \\
\forall w : (NAME \twoheadrightarrow AbPurse);\ n : NAME;\ AbPurse \mid n \notin \mathrm{dom}\ w \bullet \\
\quad totalAbBalance(\{n \mapsto \theta AbPurse\} \cup w) = balance + totalAbBalance\, w
\end{array}
$$

In order to use the inductive case of *totalAbBalance*, we need to partition the argument into two parts, the first being a singleton. Without a witness for this element, this rule is useless for automation.

The problem with *totalAbBalance* appears to be that we are missing a principle of induction over finite sets to match the style of recursion in the definition. It would have been possible to craft a new induction rule from the principle provided by Z/Eves. However, using that induction rule would still have been difficult.

We kept this definition in our formalisation, but did not prove the theorems from [SCW00, Sect.2.4], which are related to *totalAbBalance* and the other auxiliary function from [SCW00, Appendix D]. Moreover, the proofs given in [SCW00, Sect.2.4] are informal, as the suggested instantiations are not possible as mentioned in the text.

3.2. Suggestion of improvement

Alternative definition for totalAbBalance

Although there is little automation for finite functions in general, there is good automation for sequences. An alternative definition for *totalAbBalance* using sequences and induction avoids both finiteness and instantiation problems by relying on the rich toolkit theorems for sequences. Summation over sequences is very simple and is inductively defined as

$$
\begin{array}{|l}
sum : \mathrm{seq}\, \mathbb{Z} \to \mathbb{Z} \\
\hline
sum\, \langle\rangle = 0 \\
\forall n : \mathbb{Z} \bullet sum\, \langle n \rangle = n \\
\forall s, t : \mathrm{seq}\, \mathbb{Z} \bullet sum\, (s \frown t) = sum\, s + sum\, t
\end{array}
$$

An inductive update over sequences is given by

$$
\begin{array}{|l}
update : \mathrm{seq}\, \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \to \mathrm{seq}\, \mathbb{Z} \\
\hline
\forall i, n : \mathbb{Z} \bullet update(\langle\rangle, i, n) = \langle\rangle \\
\forall i, x, n : \mathbb{Z} \bullet update(\langle x \rangle, i, n) = \textbf{if } i = 1 \textbf{ then } \langle n \rangle \textbf{ else } \langle x \rangle \\
\forall s, t : \mathrm{seq}\, \mathbb{Z};\ i, n : \mathbb{Z} \bullet update((s \frown t), i, n) = \\
\quad \textbf{if } i \in \mathrm{dom}\, s \textbf{ then } update(s, i, n) \frown t \\
\quad \textbf{else if } i - \#s \in \mathrm{dom}\, t \textbf{ then } s \frown update(t, (i - \#s), n) \\
\quad \textbf{else } s \frown t
\end{array}
$$

The effect an update has is given in terms of the changing sum:

theorem tSumUpdate
$\forall s : \mathrm{seq}\, \mathbb{Z};\ i, n : \mathbb{Z} \mid i \in \mathrm{dom}\, s \bullet$
$\quad sum(update(s, i, n)) = sum\, s - s\, i + n$

theorem rule rSumPos
$\forall s : \mathrm{seq}\, \mathbb{N} \bullet sum\, s \in \mathbb{N}$

These definitions are much simpler than the ones using finite functions.

Table 1. Name convention for structured precondition proofs of $\mathcal{B}$-world operations

	Ch	Local operation	∃	PS	Description
(i)	4	*StartFromPurseEafromOkay*	4	20	successful *startFrom* over *ConPurse*
(ii)	4	*StartFromPurseOkay*	38	115	(i) with recovery from abortion via *AbortPurseOkay*
(iii)	5	*StartFromEafromOkay*	15	6	(i) promoted in $\mathcal{B}$ via ΦBOp
(iv)	8	*StartFromOkay*	21	16	(ii) promoted in $\mathcal{B}$ via ΦBOp
(v)	5	*StartFrom*	15	6	(ii) promoted in $\mathcal{B}$ via ΦBOp with *Abort* and *Ignore* disjuncts

Better structuring of precondition proofs

The precondition proofs in [SCW00, Chap. 8] are understated: they mention that various operations have trivial preconditions because they are disjoined with *Ignore*, a schema with a very easy proof of totality. [4] This is not a very satisfying analysis. Nevertheless, if one tries to calculate the preconditions in Z/Eves without relying on *Ignore*, they turn out to be quite challenging, and the informal proof is of little help. The reasons for this are as follows.

- Z/Eves does not support compositional precondition calculation (as described in [WoD96]), so there is no way to mimic the informal proof without doing a lot of calculation.
- There are many variables to instantiate, and so a lot of ingenuity is required to find appropriate values. This cannot be automated.
- The proof needs to be structured with auxiliary lemmas, one for each conjunct of the formulas; this mainly affects the one on page 47, *StartFromEafromOkay*.
- The precondition proof of *StartFromEafromOkay* is missing; it requires the precondition proof of various other parts of the specification.
- Several lemmas are needed for layering precondition proofs to deal with promoted operations and appropriate instantiations.
- The hand-written proofs avoid analysis of promoted operations.
- Other lemmas are stated but not used.
- No informal explanation is given for harder proofs.

In our formalisation using Z/Eves, we structure the precondition proof of the *StartFrom* operation without the *Ignore* disjunct and the *Abort* composition as *StartFromEafromOkay*, as well as their promoted versions. As the names can be quite confusing, we summarise them in Table 1. The chapters refer to where in the original monograph [SCW00] the additional operations are included; unless mentioned, the operations are disjoined with neither *Abort* nor *Ignore*; and the number of existentially quantified variables (∃), and proof steps (**PS**) on each proof is also given. It may seem that, in the end, it does not matter which way one chooses to structure the precondition proofs, as long as they are properly calculated and mention the operations under concern. On the contrary, a naïve attempt at proving such complex preconditions can result in a laborious, tedious, repetitive, and time-consuming effort.

For instance, in the precondition calculation for (iii) and (iv) in Table 1, we structure the proof script so that side-condition type-checks from *BetweenWorld* and *Auxworld* are separated. In the *BetweenWorld* proof 36 more variables were instantiated with a total of 250 proof steps, where 16 of those 36 variables and 117 of those 250 proof steps where related to the enclosed *AuxWorld*. That is, without this structure, the proof effort for this operation would increase in at least 36 more instantiations and 250 more proof steps. The same layered strategy is applied to *StartTo*, *ReadExceptionLog*, *ClearExceptionLog*, and *AuthoriseExLogClear* operations, hence the profit in less proof-effort is even higher. Although this structure is not simple, it makes the rather complex precondition proofs much easier: it enables a reproducible, general proof strategy for all operations reusing the *BetweenWorld* side-condition type check proofs, hence actually reducing the total number of quantified instantiations for the *StartFrom* operation from 165 ($4 + 38 + (15 \times 2) + 21 + (36 \times 2)$) to 93 ($4 + 38 + (15 \times 2) + 21$), and total number of proof steps for from 663 ($20 + 115 + 6 + 16 + 6 + (250 \times 2)$) to 163 ($20 + 115 + 6 + 16 + 6$).

[4] An overview of Mondex and its formal specification is also published in this issue of *FACJ*[WSC+08].

The other $\mathcal{B}$-world operations (e.g. *Ignore*, *Increase*, *Abort*, *Req*, *Val*, *Ack*, and *Archive*) require a simpler structure involving only (iv) and (v), whereas the framing schema ΦBOp uses the same proof strategy for (iv). Thus, the proof steps for these operations are quite similar and massively (8 times) reused. The strategy is similarly applied for the calculation of (13) $\mathcal{C}$-world operations preconditions.

When the strategy is applied to all 24 operations (and 2 framing schemas) from both the $\mathcal{B}$- and $\mathcal{C}$-worlds, the number of proof steps for these precondition proofs is considerably reduced, around 5.65 times from a possible 6, 753 to an actual 1, 197 proof steps. These estimated numbers are calculated taking into account that we know that although the precondition proofs are large, they are repetitive. We consider 4, 771 (70.65%) of those 6, 753, to come from the side-conditions involving *BetweenWorld* (13-ΦBOp-related$\times$ $250 = 3,250$) and *ConWorld* (13-ΦCOp-related$\times$ $117 = 1,521$). The recovery from abortion involves 1, 150 (17.03%) steps from both $\mathcal{B}$- ($5 \times 115 = 575$) and $\mathcal{C}$-worlds ($5 \times 115 = 575$). The other remaining 832 (12.32%) proof steps come from the actual precondition theorems from both $\mathcal{B}$- and $\mathcal{C}$-worlds. These 832 steps are divided amongst the 10 operations (10-i/ii/iv/v$\times$ $48 = 480$) requiring recovery from abortion (e.g. *StartFrom*, *CStartFrom*, etc.) and involving (i)–(v), the 2 framing schemas, and the other 14 operations ((2 + 14)-iv/v$\times$ $22 = 352$) requiring disjunction with *Abort* (*CAbort*) and *Ignore* (*CIgnore*) and involving (iv)–(v) only.

Overall, [SCW00, Chap. 8] contributes 2, 999 (66%) in the (4, 544) total number of proof steps (see Table 4 in Sect. 5). From these, a total of 1, 197 steps come from precondition proofs for $\mathcal{B}$- and $\mathcal{C}$-worlds. The $\mathcal{A}$-world contributes 1, 688 steps, whereas housekeeping rules and extended toolkit functionality covers the remaining 114 proof steps. This structuring decision seriously reduced the whole proof effort from a possible total of 8, 298 ($4,544-2,999+6,753$) to 4, 544 proof steps (i.e. around 1.83 times overall improvement). Furthermore, following the tabular precondition presentation style from [WoD96, p.231], we have also collected the tables of calculated preconditions for all operations from $\mathcal{A}$-, $\mathcal{B}$- and $\mathcal{C}$-worlds.

Clearer proof explanation

As the promoted proofs for *Eafrom* operations cannot rely on disjunction with *Ignore*, the whole proof is much more complex, as we need to tackle the entire operation itself. Because of the various quantifiers and schema inclusions, it is quite hard to figure out what the appropriate instantiations would be, hence we have difficult precondition proofs to discharge. We decided to break them down into the various schema inclusion parts, so that the appropriate instantiations are clearly understood.

The definition of *StartFromEafromOkay* is given in [SCW00, Sect.5.6.1] as

$$StartFromEafromOkay \mathrel{\widehat=} \exists \Delta ConPurse \bullet \Phi BOp \land StartFromPurseEafromOkay$$

Thus, at first, we want to prove the precondition of *StartFromPurseEafromOkay* alone. Because of the way these promoted operations are defined, we need to include additional automation lemmas even before this first precondition. More precisely, the use of pointwise definite descriptions in the definition of *StartFromPurseEaFromOkay* makes automation hard.

The hand-written proofs in [SCW00] are nicely written, and we have found this informal help very useful: the proofs are thoroughly explained, especially in later chapters.

Minor mistakes

There are some minor mistakes in the text. First, there is a mistaken LaTeX mark-up involving a subscript and a stroke [SCW00, Chaps. 7 and 10], which in fact revealed a surprise in the Z standard [ISO02]. Consider the following quantified predicate: "$\exists x'_1, x'_1 : \mathbb{N} \bullet x'_1 \neq x'_1$". Two surprises: first, it parses and is type correct; and second, it evaluates to *true*. The two variables in the predicate are actually different, even though they are typeset identically, as can be seen from the LaTeX source used:

```
\exists x_1', x'_1: \nat @ x_1' \neq x'_1
```

This confusion is present in the Mondex source text: a variable is marked up in both ways, introducing an undeclared variable and a failed type-check.

An operation and a theorem are both called *AbOp* [SCW00, p.63, Chap. 8], causing the type checker to fail, and the proofs in Chap. 10 refer to lemmas in [SCW00, Appendix C] without proper referencing.

3.3. How complete is it?

At the time of writing, the following parts of Mondex [SCW00] have been mechanically verified using Z/Eves:

- Models
  - $\mathcal{A}$ model [Chap. 3]
  - $\mathcal{B}$ model: purse, world, init., final. [Chaps. 4, 5, 6]
  - $\mathcal{C}$ model [Chap. 7]
  - applicability proofs [Chap. 8]
- Refinement: $\mathcal{A}$ to $\mathcal{B}$
  - refinement rules as Z/Eves theorems [Chap. 9]
  - retrieve definitions [Chap. 10]
  - $\mathcal{A}$ to $\mathcal{B}$ initialisation [Chap. 11]
  - $\mathcal{A}$ to $\mathcal{B}$ finalisation [Chap. 12]
  - $\mathcal{A}$ to $\mathcal{B}$ applicability [Chap. 13]
  - $\mathcal{A}$ to $\mathcal{B}$ lemmas for backward simulation [Chap. 14, Appendix C]
  - $\mathcal{A}$ to $\mathcal{B}$ backward simulation correctness (partially) [Chaps. 15–23]
- Refinement: $\mathcal{B}$ to $\mathcal{C}$
  - refinement rules as Z/Eves theorems [Chap. 25]
  - retrieve definitions [Chap. 26]
  - $\mathcal{B}$ to $\mathcal{C}$ init., final., applicability [Chap. 27]
  - $\mathcal{B}$ to $\mathcal{C}$ lemmas for forward simulation (partially) [Chap. 28, App. C]
  - $\mathcal{B}$ to $\mathcal{C}$ backward simulation correctness (partially) [Chap. 29]
- Security properties [Chap. 2]
  - all definitions
  - original proofs in Sect. 2.4 contain informal arguments
  - *totalAbBalance* is inadequate for mechanisation [Appendix D]
  - mechanisable using suggested model of sequences

Chapters 1 and 30, and Appendixes *A* and *B* only contain textual information without formal material. It is expected that the remaining parts from both correctness refinement proofs (Chaps. 14–24 and 28–29) will be mechanised shortly. They require a reorganisation of the original proof strategy that has been partially done, but still require some minor fine-tuning. The security property proofs need to be tackled once an agreeable version for *totalAbBalance* is chosen, as discussed in Sect. 3.1. As we want to keep changes to a minimum, instead of using a variation with sequences (see Sect. 3.2), we pursued a modified version of the original *totalAbBalance* that keeps the structure and data types, yet exploits our proof strategy for finiteness of schema bindings (see Sect. 4.2) extending it for inductive proofs.

4. Problems found

4.1. Are after purses authentic?

Mondex has a state invariant that requires all abstract purses involved in a transaction to be authentic: *abAuthPurse*. This invariant is not required in the after-state of the operations *AbTransferOkayTD* and *AbTransferLostTD*

[SCW00, p.20], although it should be. We demonstrated the necessity of the after-invariant by showing that the original definitions can lead to a state with inauthentic purses.

4.2. Discharging PayDetails finiteness assumptions

Most (implicit) finiteness assumptions in the monograph [SCW00] are related to *PayDetails* being finite, as in the conjecture about authentic *from* purses in *AuxWorld*

$$AuxWorld \vdash authenticFrom \in \mathbb{F}\ PayDetails$$

Besides the technical difficulties in finiteness proofs already mentioned, because we are dealing with a finite set of bindings, the problem complexity increases even further. In here, we present a general solution to prove finiteness properties related to schema bindings by showing a proof that set of *PayDetails* bindings is finite. That means we need some extra lemmas about the finiteness of particular sets we use to instantiate the general proof strategy for *PayDetails* bindings.

The general proof strategy is to use functions to compare the sizes of sets. As the range of a function is never larger than its domain, hence a finite domain implies a finite range, and the problem boils down to finding such finite (function) domain. At this point, we can use the Z/Eves toolkit theorem below

theorem finiteFunction $[X, Y]$
$\forall f : X \pfun Y \bullet \mathrm{dom}\, f \in \mathbb{F}\ X \land ran f \in \mathbb{F}\ Y \land \#(\mathrm{ran}\, f) \leqslant \#(\mathrm{dom}\, f) = \# f$

That is, to show a set S is finite, find some surjective function $f \in T \pfun S$ for a known finite set T. To use this strategy with *PayDetails* for S, we need to establish a basic theory for *PayDetais* finiteness first. This theory can be easily generalised for other finiteness proofs involving schema bindings. As we have already included finite types in *PayDetails* (see Sect. 3.1), it is now possible to try proving a more basic conjecture: the set of bindings *PayDetails* represent is itself finite

$$? \vdash PayDetails \in \mathbb{F}\ \langle\!|from : NAME;\ to : NAME;\ value : \mathbb{Z};\ fromSeqNo : \mathbb{Z};\ toSeqNo : \mathbb{Z}|\!\rangle$$

To prove this conjecture we use the general proof strategy above. Firstly, we need to establish some facts about *PayDetails* in general. Let us define the space under concern as the cross product of *PayDetails* as

$$PayDetailsSpace == (NAMES \times NAMES) \times (NAT \times (NAT \times NAT))$$

which contains tuples like

$$((from, to), (value, (fromSeqNo, toSeqNo)))$$

As we have bounded the types (see Sect. 3.1), we know the cross product is finite (see Theorem rCrossFinite in Sect. 2.9). Next, we format the set of *PayDetails* bindings as tuples of the above form by establishing a relation between these tuples with the corresponding *PayDetails* binding

$$PayDetailsBindings == \{ PayDetails \bullet (((from, to), (value, (fromSeqNo, toSeqNo))), \theta PayDetails) \}$$

where we want to prove that the domain of this relation is finite and within the space of *PayDetails*

theorem lPayDetailsSpaceFiniteMaxType
$PayDetailsSpace \in \mathbb{F}((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z})))$

theorem lPayDetailsBindingsSubsetPayDetailsSpace
$\mathrm{dom}\ PayDetailsBindings \in \mathbb{P}\ PayDetailsSpace$

We use this domain as our (finite) set T above. The format of the last theorem is a Z/Eves idiom better suited for automation and is the same as

$$\mathrm{dom}\ PayDetailsBindings \subseteq PayDetailsSpace$$

With these two definitions, two theorems, and the extended toolkit theorem for finiteness introduced above (see Sect. 2.9), we can prove that our choice for the set T in our proof strategy is finite.

theorem lPayDetailsBindingsFiniteMaxType
$\mathrm{dom}\ PayDetailsBindings \in \mathbb{F}((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z})))$

where the proof script with appropriate non-maximal generic types instantiation (see Sect. 2.8) is given as

proof [*lPayDetailsBindingsFiniteMaxType*]
use lFinsetSubset [
$X := \text{dom}\ PayDetailsBindings$,
$Y := PayDetailsSpace$,
$Z := ((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z})))$] ;
use lPayDetailsBindingsSubsetPayDetailsSpace;
use lPayDetailsSpaceFiniteMaxType;
simplify;

□

With our choice for T and its finiteness proved, to use the finiteFunction toolkit theorem, we need to add additional facts about our choice of surjection as

$$PayDetailsBindings \in PayDetailsSpace \twoheadrightarrow PayDetails.$$

as further type information lemmas about the (finite) functionality of *PayDetailsBindings*

theorem grule gPayDetailsBindingsPfun
$PayDetailsBindings \in ((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z}))) \pfun PayDetails$

theorem grule gPayDetailsBindingsFfun
$PayDetailsBindings \in ((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z}))) \finfun PayDetails$

which are easy to prove because *PayDetailsBindings* is functional by construction and *PayDetailsSpace* is finite. The grule usage ensures this information is coarsely used by every transformation tactic. To make it a surjection, we also need to prove that

theorem rule lPayDetailsBindinsRange
$\text{ran}\ PayDetailsBindings = PayDetails$

which is again trivial by construction. Finally, we can state the conjecture as a theorem

theorem grule lPayDetailsIsFinite
$PayDetails \in \mathbb{F}\ (\!|from : NAME;\ fromSeqNo : \mathbb{Z};\ to : NAME;\ toSeqNo : \mathbb{Z};\ value : \mathbb{Z}|\!)$

and because of the proof strategy and adequate housekeeping rule usage (see Sect. 2.2) provided and adequate (maximal) generic actual-parameters instantiation (see Sect. 2.8), the proof script is now straightforward.

proof [*lPayDetailsIsFinite*]
use finiteFunction[
$((NAME \times NAME) \times (\mathbb{Z} \times (\mathbb{Z} \times \mathbb{Z})))$,
$(\!|\ from : NAME;\ fromSeqNo : \mathbb{Z};\ to : NAME;\ toSeqNo : \mathbb{Z};\ value : \mathbb{Z}\ |\!)$]
$[f := PayDetailsBindings]$;
rewrite;

□

This concludes the example of the given proof strategy as a set of lemmas to be (automatically) used in the proof of the implicit finiteness conjectures in Mondex. For better automation, we just add two further rewriting rules that mimic the shape of most goals involved

theorem rule lPayDetailsFromPurseNameFinite
$\forall name : NAMES \bullet \{ pd : PayDetails \mid pd.from = name \} \in \mathbb{F}\ PayDetails$

theorem rule lPayDetailsFromPurseDomainFinite
$\forall purse : NAMES \pfun ConPurse \bullet$
$\{ pd : PayDetails \mid pd.from \in \text{dom}\ purse \} \in \mathbb{F}\ PayDetails$

With this in place, proving the initial conjecture about *authenticFrom* in *AuxWorld* and many other similar ones is now trivial.

theorem frule fAuxWorldAuthenticFromIsFinite
$\forall AuxWorld \bullet authenticFrom \in \mathbb{F}\ PayDetails$

As mentioned in Sect. 2.7, we provide it as a forward rule so that there is no need to expand *AuxWorld* in (the many) goals where such finiteness properties need to be discharged as side-conditions.

4.3. Four missing properties of *BetweenWorld*

BetweenWorld is inconsistent when *val* or *ack* messages are handled. For property $B3$ [SCW00, p.42], the original purse is missing additional information about the authenticity of *val* messages in the *ether* for *to* and *from* purses. This does not apply to *rel* messages, as property $B1$ shows. We include the new properties in the predicate part to have a uniform signature across *BetweenWorld* properties.

$$\forall pd : PayDetails \mid val\, pd \in ether \bullet pd \in authenticTo$$
$$\forall pd : PayDetails \mid val\, pd \in ether \bullet pd \in authenticFrom$$

Similarly, for property $B4$, we need to include the authenticity of *ack* messages in the *ether* for *to* and *from* purses.

$$\forall pd : PayDetails \mid ack\, pd \in ether \bullet pd \in authenticTo$$
$$\forall pd : PayDetails \mid ack\, pd \in ether \bullet pd \in authenticFrom$$

For $B5$ nothing else is needed because the *from* purses in *fromLogged* are already authentic, from the definition in *AuxWorld*. Similarly, for $B6$, $B7$, and $B8$, the *to* purses in *toLogged* and *from* purses in *fromLogged* are already authentic.

4.4. Retrieve relations

The proof of the first refinement [SCW00, Chap.10] reaches the following goal:

$$RabCl \Rightarrow (\exists pdThis : PayDetails \bullet true)$$

which requires that at least one *PayDetails* schema exists. For this to be true, we must have at least two different names for *from* and *to* components. Of course this is a reasonable assumption, but it has not been made explicitly in the original specification. This problem also appears in lemmas from [SCW00, Chap.14,Appendix C]. The structured (finite) version of *NAMES* given above (see Sect. 3.1) solves this problem.

5. Benchmarks

By October 2006, around 90% of Mondex had been verified, discharging the verification conditions of 199 definitions. Tables 2, 3, and 4 present some statistics from the proof work, which we explain in detail.

There are four kinds of conjectures that we needed to prove about Mondex: additional rules for language constructs and mathematical toolkit definitions, such as finiteness and functional overriding; theorems stating axiomatic consistency and verification conditions for the correctness of refinements; lemmas used to structure theorems; and Z/Eves domain checks, which ensure that partial functions are applied within their domains and that definite descriptions are proper. Most of the additional rules arise from automating the type-checking of free-types and schemas, as described above.

We have proved 318 verification conditions to do with consistency and correctness properties. The 141 automation-related proofs come from the 64 (19 + 20 + 25) Z/Eves rules and the 77 new lemmas. There are two parts to proving the consistency of a Z specification of a system as an abstract datatype: first, the existence of a model for the specification; and second, the non-triviality of that datatype.

Is there a model for the definitions? This question asks whether the specification is satisfiable (are the definitions contradictory)? Notice from the table that most paragraphs analysed so far are used to define schemas. Any contradictions in the declarations and constraints of a schema result in the definition of an empty set of bindings, so there is always a model for a schema, albeit perhaps a trivial one. So the problem becomes one of finding carriers for the two given sets that can lead to existence proofs for the free-type and axiomatic definitions. A total of 15 proofs will be needed, and 6 are finished. Z/Eves has a further requirement for consistency: it generates *domain checks* to guarantee the definedness of expressions involving partial functions; it generates these checks for every definition involving such functions, even if the definition is never subsequently used.

Table 2. Mondex statistics I: summary of paragraphs, theorems, and automation

Paragraph type	
Original given sets	2
Original free types	4
Original axiomatic definitions	3
Original schemas	106
Original number of paragraphs	115
New axiomatic definitions	12
New schemas	72
Number of new paragraphs	84
Total number of paragraphs	199
Verification conditions	
Original lemmas	5
Original theorems	19
Number of original proofs	24
New Z/Eves rules	64
New lemmas	77
New theorems	81
New domain checks	66
Axiomatic def. consistency	6
Number of new proofs	294
Total number of proofs	318

Automation	grule	frule	rule	lemmas	Total
Free types	14	0	5	0	19
Schemas/bindings	0	13	4	30	47
μ-θ-expressions	0	0	5	3	8
Extended toolkit	0	0	5	1	6
Finiteness	3	3	5	13	24
Structured names	2	4	1	4	11
Precondition proofs	0	0	0	26	26
Total	19	20	25	77	141

Does the specification define a non-trivial abstract datatype? This question is addressed by proving state initialisation and operation precondition theorems.

An interesting metric for a proof is the number of interactions required for its successful completion by a mechanical theorem prover, but this metric can be misleading. A proof with few interactions is rather like a successful attempt at push-button model-checking: it does not reveal what has gone on behind the scenes to get to this stage. Our archaeological approach to Mondex meant that we have not restructured the specification to make mechanisation easier: like Peter Lely's portrait of Oliver Cromwell, we have taken Mondex, warts and all.

About 67% of our proof steps are *trivial*, relying on the automation we included for Z/Eves to discharge verification conditions. Another 23% rely on an *intermediate* level of skill: this involves understanding how the proof is going, often using repetitive steps from previous proofs or on knowledge of how Z/Eves works internally. The final 10% rely on *creative* steps requiring domain knowledge, such as instantiating existential variables (see Table 3).

We conducted an experiment summarised in Table 5. It consisted of parsing, type-checking, and executing the proof of the various Mondex specification parts in the latest Z/Eves version 2.3.1 `emacs` interface on a Toshiba Portégé M400 Tablet PC released in March 2006, which we call `Zarathustra`. It has a dual Pentium T2400 CPU, running at 1.83 GHz with 2 GB of physical RAM and 4 GB of virtual RAM under Windows XP SP2 with 60% CPU load (100% capacity in one CPU and 20% capacity in the other CPU). We include the time taken by each part together with the amount of virtual memory (in MB). In order to make a fairer judgment, we decided to build a time-machine and call it the `Tardis`. We managed to get a top-quality PC from June 1995, the date when the original Mondex work was accomplished, and we ran the same experiment on Z/Eves version 2.1 (1995) with 100% CPU load. This PC has an Intel ATx486 CPU, running at 160 MHz with 80 MB of physical RAM and 190 MB of virtual RAM under Windows NT4. In Table 5, we also include the execution times overall improvement. Unfortunately, during the refinement proofs running on `Tardis`, oddly, we had a power cut and could not accurately collect the amount of time and memory used. As the results previously to the power cut

Table 3. Mondex statistics II: detailed proof steps per complexity

Trivial	Steps		Overall
`Invoke`	803	26.41%	17.67%
`Prenex`	278	9.14%	6.12%
`Cases`	92	3.03%	2.02%
`Next`	351	11.54%	7.72%
`Simplify`	137	4.51%	3.01%
`Rewrite`	540	17.76%	11.88%
`Reduce`	52	1.71%	1.14%
`Prove by rewrite`	444	14.60%	9.77%
`Prove by reduce`	57	1.87%	1.25%
`Rearrange`	201	6.61%	4.42%
`Instantiate`	86	2.83%	1.89%
Total	3,041	100%	66.92%
Intermediate			
`Invoke`	76	7.31%	1.67%
`Simplify`	44	4.23%	0.97%
`Rewrite`	111	10.68%	2.44%
`Reduce`	10	0.96%	0.22%
`rearrange`	11	1.06%	0.24%
`Instantiate`	178	17.13%	3.92%
`Split`	15	1.44%	0.33%
`Apply`	272	26.18%	5.99%
`Use`	230	22.14%	5.06%
`Equality substitute`	10	0.96%	0.22%
`Trivial rewrite`	11	1.06%	0.24%
`With enabled/disabled`	70	6.74%	1.54%
`With normalization`	1	0.10%	0.02%
Total	1,039	100%	22.87%
Creative			
`Invoke`	6	1.29%	0.13%
`Simplify`	10	2.16%	0.22%
`Rewrite`	32	6.90%	0.70%
`Reduce`	8	1.72%	0.18%
`Rearrange`	11	2.37%	0.24%
`Instantiate`	150	32.33%	3.30%
`Split`	60	12.93%	1.32%
`Apply`	40	8.62%	0.88%
`Use`	30	6.47%	0.66%
`Equality substitute`	117	25.22%	2.57%
Total	464	100%	10.21%
Overall	4,544	–	100%

Table 4. Mondex statistics III: proof steps summary per Chapter & complexity

Chapter	Proof steps		Mondex part	Total	
3	203	4.47%			
4	102	2.24%			
5	373	8.21%			
6	75	1.65%			
7	4	0.09%	Specification	757	16.66%
8	2,999	66.00%	Preconditions	2,999	66.00%
10	233	5.13%			
11	12	0.26%			
12	50	1.10%			
15–24	100	2.20%			
27	22	0.48%			
28–29	371	8.16%	Refinement	788	17.34%
Total	4,544	100%		4,544	100%

Table 5. Experiment benchmark: `Tardis` × `Zarathustra`

Mondex part	`Tardis`		`Zarathustra`		Improv.
TUI v2.1	Time	Mem	Time	Mem	Time-fold
Specification (Chaps.3–7)	2:25:18	1.8	06:56	55.72	21 ×
Preconditions (Chap. 8)	24:01:10	58.0	2:51:08	81.68	8 ×
Refinement (Chaps.14–29)	power cut!		35:26	5.10	–
TUI v2.3					
Specification (Chaps.3–7)	1:42:30	24.94	02:57	27.39	35 ×
Preconditions (Chap.8)	30:21:31	107.71	49:59	147.25	35 ×
Refinement (Chaps.14–29)	1:00:00	0.37	10:29	25.62	6 ×

Table 6. Mondex specification workload in Z/Eves

Task	Days		
Specification and domain checks	7	17.07%	
Precondition proofs	10	24.39%	
Backward simulation problem	3	7.32%	
Forward simulation strategy	5	12.20%	
Forward simulation proof	2	4.88%	
Induction over union (*sumValue*)	1	2.44%	
Finiteness of *PayDetails*	5	12.20%	
`Tardis` × `Zarathustra`	3	7.32%	
LaTeX document	5	12.20%	
Total	41	100%	Around 8 weeks

were close to those for `Zarathustra`, we decided not to run it again for `Tardis`, as this would imply re-running the whole experiment with the `Tardis`.

Version 2.1 of the Z/Eves proof system (1996) relies on a single-threaded Allegro LISP engine with user-controlled garbage-collection settings. Version 2.3.1 (2004) has a compiled LISP proof-engine with multi-threading (i.e. allows multiple tasks) and hyper-threading (i.e. exploits dual processors) support, which leads to much faster times and less memory-eager proof executions. It is interesting to note that this technology could have been exploited in 1996, but this took place only when users requested it in 2004.

Our work took around eight weeks to mechanise the contents of the Mondex monograph [SCW00] with Z/Eves, which includes additional effort studying the problem, planning the mechanisation, and extending the Z toolkit. We summarise the workload in Table 6. In particular, the finiteness strategy took quite a lot of effort, which we hope to amortise over future work. In total, this effort was spread across 6 months (May–October 2006).

6. Conclusions

We have conducted an experiment to find out how the proof of the Mondex electronic purse can be automated. Perhaps the most surprising result from this experiment is that half of it can be accomplished in an order of magnitude less effort than it took to conduct the original Mondex work. This has been possible due to the existence of good models with their precise invariants. In other words, we should not see this experiment as saying that the original work could be slashed to one-tenth of its effort, but rather that a mechanical proof could be added for an extra 10% of the overall effort. Since the theorem prover that we used was available in the same release 10 years ago, this contradicts the popular opinion that it would not be cost-effective.

Our work discovered some unknown bugs as the payback for our efforts: the missing properties of *BetweenWorld* reported in this paper allow six kinds of operations to involve inauthentic purses. Schema *PayDetails* not being finite affects mostly everywhere in *Betw* and *Conc*. The work is also useful in providing good insight in driving the Z/Eves prover, which is definitely helpful for future exercises.

Our work can act as a reference model for those not using Z. If they suspect a bug in Mondex, then we can check to see if this behaviour is genuine, or simply an artefact of translation or remodelling. Similarly, we can check to see if our colleagues also find the same bugs.

The Mondex case study shows that the verification community is willing to undertake competitive and collaborative projects—and that there is some value in doing this. A challenge now is to find the most effective way of curating all our results in the Verified Software Repository.[5]

Acknowledgements

First, we warmly acknowledge our collaboration with Mark Saaltink: he helped our work in many ways, both directly and indirectly. We would like to thank the other members of the Mondex club, especially those who have been working directly on the verification effort. This includes Michael Butler, David Crocker, Chris George, Martin Gogolla, Anne Haxthausen, Cliff Jones, Ken Pierce, Tahina Ramananandro, Gerhard Schellhorn, and Divakar Yadav. Juan Bicarregui, Paul Boca, Jonathan Bowen, and Jim Davies have all attended our workshops and given valuable criticism and feedback. Peter Cooper found the Tardis for us. Tony Hoare inspired the work as part of the Grand Challenge, and Cliff Jones and Peter O'Hearn guided our efforts as members of the steering committee for Grand Challenge 6 on *Dependable Systems Evolution*. The work on Z/Eves and Mondex was presented at Dagstuhl Seminar 06281 on *The Challenge of Software Verification* in July 2006, and as a keynote talk at the *International Colloquium on Theoretical Aspects of Computing* in Tunis in 2006. The authors are grateful for the financial support of QinetiQ Malvern (the *Circus-based Development* project) and the EPSRC (*VSR-net: A network for the Verified Software Repository*). The work of the Mondex club started from a smaller collaboration with Daniel Jackson and others on modelling and verifying a hotel room-key system, now recorded in the Alloy book [Jac06a]. Finally, we should thank Susan Stepney and David Cooper for their original work that got us all going.

References

[BHW06] Bicarregui J, Hoare T, Woodcock J (2006) The verified software repository: a step towards the verifying compiler. FACJ 18(2):143–151

[CSW02] Cooper D, Stepney S, Woodcock J (2002) Derivation of Z Refinement Proof Rules. Technical Report YCS-2002-347, University of York

[Cro04] Crocker D (2004) Safe object-oriented software: the verified design-by-contract paradigm. In: Redmill F, Anderson T (eds) Practical elements of safety: proceedings of the 12th safety-critical systems symposium. Springer, Heidelberg

[DGJ+02] Hung DV, George C, Janowski T, Moore R (eds) (2002) Specification Case Studies in RAISE. FACIT (Formal Approaches to Computing and Information Technology) series. Springer, Heidelberg

[ISO02] Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics. ISO/IEC 13568:2002(E) 2002

[ITS91] ITSEC. Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria. Document COM(90) 314, Version 1.2. Commission of the European Communities (1991)

[Jac06a] Jackson D (2006) Software Abstractions: Logic, Language, and Analysis pp 350. The MIT, Cambridge

[Jac06b] Jackson D (2006) Dependable software by design. Scientific American. June 2006

[JOW06] Jones C, O'Hearn P, Woodcock J (2006) Verified software: a Grand Challenge. IEEE Comput 39(4):93–95

[MAV05] Métayer C, Abrial J-R, Voisin L (2005) Event-B Language. Project IST-511599 RODIN Rigorous Open Development Environment for Complex Systems. RODIN Deliverable 3.2 Public Document. 31st May 2005 rodin.cs.ncl.ac.uk

[Mil99] Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge

[MonURL] Mondex smart cards. www.mondex.com

[QPQURL] The QPQ Deductive Software Repository. qpq.csl.sri.com

[Saa97a] Saaltink M (1997) The Z/EVES User's Guide. ORA Canada

[SCW98] Stepney S, Cooper D, Woodcock J (1998) More powerful Z data refinement: pushing the state of the art in industrial refinement. ZUM '98. Berlin, Germany. LNCS, vol 1493. Springer, Heidelberg, pp 284–307

[SCW00] Stepney S, Cooper D, Woodcock J (2000) An electronic purse: specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory. July 2000

[SGH+06a] Schellhorn G, Grandy H, Haneberg D, Reif W (2006) The Mondex challenge: machine checked proofs for an electronic purse. In: Misra J et al (eds) FM 2006: formal methods, 14th international symposium on formal methods, Hamilton, Canada, August 21–27, 2006. Springer, Heidelberg, pp 16–31

[SGH+06b] Schellhorn G, Grandy H, Haneberg D, Reif W (2006) The Mondex challenge: machine checked proofs for an electronic purse. Technical Report. Institute of Computer Science, University of Augsburg

[Spi92] Spivey JM (1992) The Z Notation: a reference manual, 2nd edn. Prentice Hall International Series in Computer Science, Englewood Cliffs, pp 150

[UMLURL] UML 2.0 OCL Specification. OMG Adopted Specification ptc/03-10-14 2004

[5] http://vsr.sourceforge.net

[WoD96] Woodcock J, Davies J (1996) Using Z: Specification, Refinement, and Proof. Prentice Hall International Series in Computer Science 1996. pp 391. The complete text is available for free download from: www.usingz.com
[WSC+08] Woodcock J, Stepney S, Cooper D, Clark J, Jacob J (2008) The certification of the Mondex electronic purse to ITSEC Level E6. Formal Aspects Comput J 20(1) (this issue)

Received 19 January 2007
Accepted in revised form 9 November 2007 by C. B. Jones
Published online 8 December 2007