Specification and Proof of the Mondex Electronic Purse

Chris George and Anne Haxthausen

United Nations University

International Institute for Software Technology

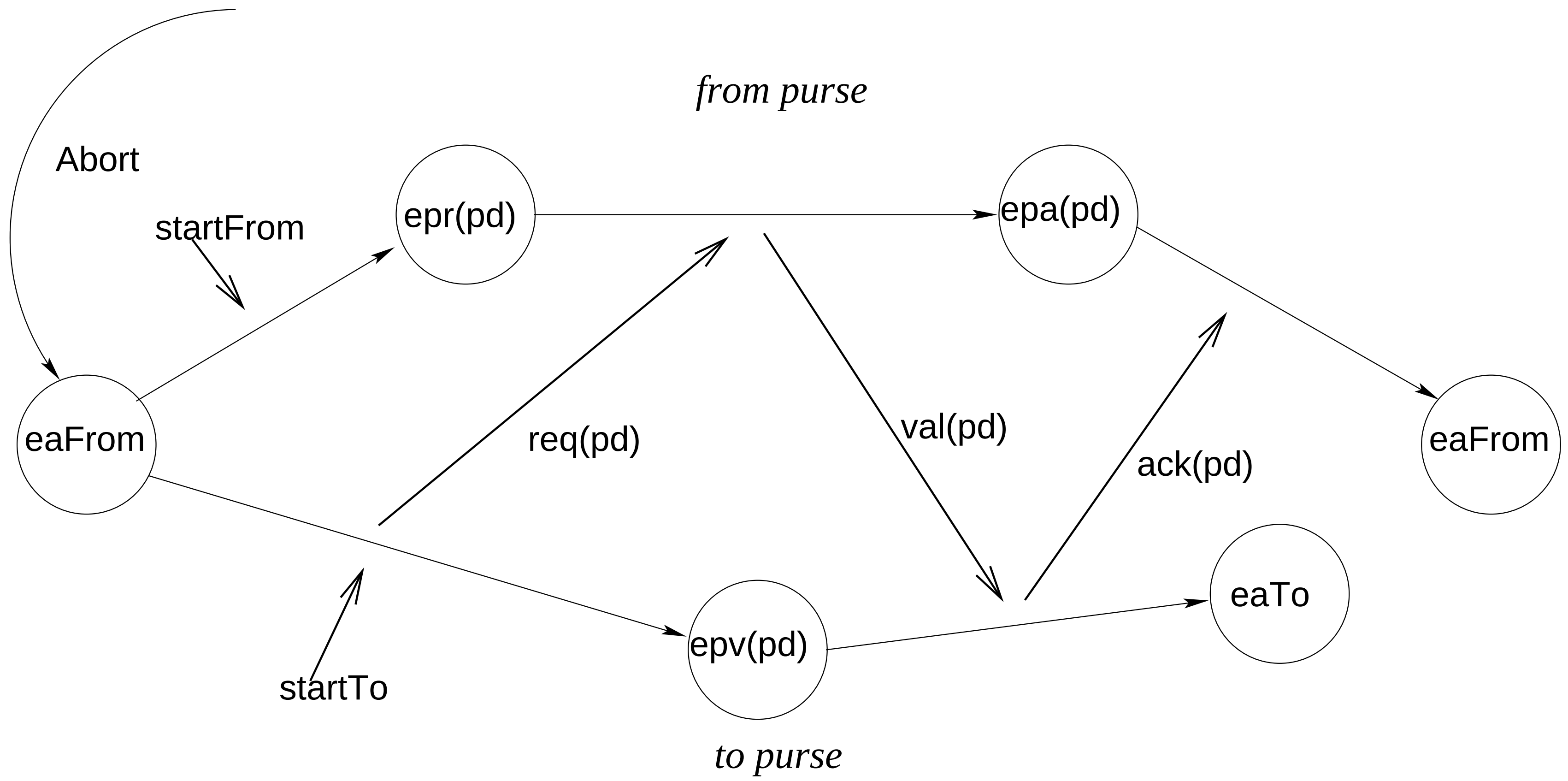
Macao SAR, China

and

Informatics and Mathematical Modelling

Technical University of Denmark

Lyngby, Denmark



Abort logs the pd in fromLogs if the purse is in state epa or in toLogs if it is in state epv

lost = toLogs inter (fromLogs union fromInEpa)

inTransit = (fromLogs union fromInEpa) inter toInEpv

sending val reduces the from purse's balance; sending ack increases the to purse's balance

The Problem

1. Specify the protocol in detail
2. Prove that each operation satisfies two conditions on the before (1) and after (2) states:
   (a) *NoValueCreation*:
   $\text{inPurses}_2 + \text{inTransit}_2 \leq \text{inPurses}_1 + \text{inTransit}_1$
   (b) *AllValueAccounted*:
   $\text{inPurses}_2 + \text{inTransit}_2 + \text{lost}_2 = \text{inPurses}_1 + \text{inTransit}_1 + \text{lost}_1$

We will call such an operation *correct*.

The RAISE Approach

3 levels of specification:

1. Abstract: a problem in accounting. No purses; no messages; just three “bottom line” values and some abstract *correct* operations that transfer money between them.

2. Middle: abstract purses and concrete operations. No details of the mechanisms that preserve the (asserted) invariant. Prove that each operation is *correct*.

3. Concrete: full details of the protocol. Prove that each operation implements its middle version.

Abstract Specification

4 abstract operations

TransferLeft

$$\begin{aligned}&\text{inPurses}_2 = \text{inPurses}_1 - \text{valu(m)} \wedge\\&(\text{lost}_2 = \text{lost}_1 \wedge \text{inTransit}_2 = \text{inTransit}_1 + \text{valu(m)} \vee\\&\ \text{lost}_2 = \text{lost}_1 + \text{valu(m)} \wedge \text{inTransit}_2 = \text{inTransit}_1)\end{aligned}$$

TransferRight

$$\begin{aligned}&\text{inPurses}_2 = \text{inPurses}_1 + \text{valu(m)} \wedge\\&\text{lost}_2 = \text{lost}_1 \wedge\\&\text{inTransit}_2 = \text{inTransit}_1 - \text{valu(m)}\end{aligned}$$

Abort

$$\exists\, \mathsf{v} : \mathbf{Nat} \bullet$$
$$\mathsf{inPurses}_2 = \mathsf{inPurses}_1 \wedge$$
$$\mathsf{lost}_2 = \mathsf{lost}_1 + \mathsf{v} \wedge$$
$$\mathsf{inTransit}_2 = \mathsf{inTransit}_1 - \mathsf{v}$$

No_op

$$\mathsf{inPurses}_2 = \mathsf{inPurses}_1 \wedge$$
$$\mathsf{lost}_2 = \mathsf{lost}_1 \wedge$$
$$\mathsf{inTransit}_2 = \mathsf{inTransit}_1$$

No_op is really a special case of Abort.

It is easy to prove these 4 operations are *correct*.

Correct combinations 1

```
compose : (Pre $\times$ Op) $\times$ (Pre $\times$ Op) $\rightarrow$ Pre $\times$ Op
compose((p1, op1), (p2, op2)) $\equiv$
  (p1,
   $\lambda$ (n, m, w) : T.Name $\times$ T.Message $\times$ World $\bullet$
          **let** w1 = op1(n, m, w) **in**
            **if** $\exists$ n1 : T.Name, m1 : T.Message $\bullet$ p2(n1, m1, w1)
            **then**
              **let** (n1, m1) : T.Name $\times$ T.Message $\bullet$ p2(n1, m1, w1)
              **in** op2(n1, m1, w1) **end**
            **else** w1 **end**
          **end**
  )
```

Correct combinations 2

sequence : (Pre $\times$ Op) $\times$ (Pre $\times$ Op) $\rightarrow$ Pre $\times$ Op
sequence((p1, op1), (p2, op2)) $\equiv$
  (λ (n, m, w) : T.Name $\times$ T.Message $\times$ World •
      p1(n, m, w) $\wedge$ p2(n, m, op1(n, m, w)),
  λ (n, m, w) : T.Name $\times$ T.Message $\times$ World •
      op2(n, m, op1(n, m, w)))

Easy to prove compose and sequence preserve correctness.

sequence useful as some operations defined for convenience as sequence(Abort, Op) or sequence(Op, Abort).

Middle Specification: Purses: types and observers

type

  PurseBase,

  Purse $= \{|$ p : PurseBase • isPurse(p) $|\}$

value

  balance : PurseBase $\rightarrow$ **Nat**,

  pdAuth : PurseBase $\rightarrow$ T.PayDetails,

  status : PurseBase $\rightarrow$ T.Status,

  name : PurseBase $\rightarrow$ T.Name

Middle Specification: Purses: axioms for Req

```
[ balance_req ]
  ∀ m : T.Message, p : Purse •
    canReq(m, p) ⇒
      balance(req(m, p)) = balance(p) − T.valu(pdAuth(p)),
[ pdAuth_req ]
  ∀ m : T.Message, p : Purse •
    canReq(m, p) ⇒ pdAuth(req(m, p)) = pdAuth(p),
[ status_req ]
  ∀ m : T.Message, p : Purse •
    canReq(m, p) ⇒ status(req(m, p)) = T.epa,
[ name_req ]
  ∀ m : T.Message, p : Purse •
    canReq(m, p) ⇒ name(req(m, p)) = name(p),
```

Middle Specification: World: observers

value

  purses : World → PursesMap,

  toLogs : World → T.PayDetails-**set**,

  fromLogs : World → T.PayDetails-**set**,

  ether : World → T.Message-**set**,

  visible : World → T.Message-**set**

Middle Specification: World: invariant

```
axiom
  [ isWorldAxiom ]
    ∀ w : World, p : P.Purse •
      p ∈ rng purses(w) ⇒
        (P.status(p) = T.epr ⇒
          P.pdAuth(p) ∉ fromInEpa(w) ∧
          P.pdAuth(p) ∉ fromLogs(w) ∧
          (T.req(P.pdAuth(p)) ∈ ether(w) ⇒
            P.pdAuth(p) ∈ toInEpv(w) ∧ P.pdAuth(p) ∉ toLogs(w) ∨
            P.pdAuth(p) ∈ toLogs(w) ∧ P.pdAuth(p) ∉ toInEpv(w))) ∧
        ... ∧
      visible(w) ⊆ ether(w)
```

Middle Specification: World: axioms

$$
\begin{array}{l}
[\,\mathrm{purses\_req}\,] \\
\quad \forall\ \mathrm{n} : \mathrm{T.Name},\ \mathrm{m} : \mathrm{T.Message},\ \mathrm{w} : \mathrm{World} \bullet \\
\qquad \mathrm{canReq(n, m, w)} \Rightarrow \\
\qquad\quad \mathrm{purses(req(n, m, w))} = \\
\qquad\qquad \textbf{let}\ \mathrm{(p1, m1)} = \mathrm{P.req(m, purses(w)(n))}\ \textbf{in}\ \mathrm{purses(w)} \dagger [\,\mathrm{n} \mapsto \mathrm{p1}\,]\ \textbf{end}, \\
[\,\mathrm{toLogs\_req}\,] \\
\quad \forall\ \mathrm{n} : \mathrm{T.Name},\ \mathrm{m} : \mathrm{T.Message},\ \mathrm{w} : \mathrm{World} \bullet \\
\qquad \mathrm{canReq(n, m, w)} \Rightarrow \\
\qquad\quad \mathrm{toLogs(req(n, m, w))} = \mathrm{toLogs(w)}, \\
[\,\mathrm{fromLogs\_req}\,] \\
\quad \forall\ \mathrm{n} : \mathrm{T.Name},\ \mathrm{m} : \mathrm{T.Message},\ \mathrm{w} : \mathrm{World} \bullet \\
\qquad \mathrm{canReq(n, m, w)} \Rightarrow \\
\qquad\quad \mathrm{fromLogs(req(n, m, w))} = \mathrm{fromLogs(w)},
\end{array}
$$

Concrete Specification: Purse: types

```
type
  PurseBase ::
    balance : Nat
    exLog : PayDetails-set ↔ change_log
    name : Name
    nextSeqNo : Nat
    pdAuth : PayDetails
    status : Status,
  Purse = {| p : PurseBase • isPurse(p) |}
```

Concrete Specification: Purse: operation

```
value
  req : Message $\times$ Purse $\overset{\sim}{\rightarrow}$ Purse $\times$ T.Message
  req(m, p) $\equiv$
    let pd = pdAuth(p), bal = balance(p) $-$ valu(pd) in
      (mk_PurseBase(
          bal, exLog(p), name(p), nextSeqNo(p), pd, epa),
       T.val(pd))
    end
  pre canReq(m, p),
```

Concrete Specification: World: types

```
type
  World = {| w : WorldBase • isWorld(w) |},
  WorldBase ::
    purses : PursesMap
    ether : Message-set
    visible : Message-set
    archive : LogBook,
```

Concrete Specification: World: invariant 1

$$
\begin{array}{l}
\text{isWorld} : \text{WorldBase} \rightarrow \textbf{Bool} \\
\text{isWorld(w)} \equiv \\
\quad (\forall\ \text{n} : \text{Name} \bullet \\
\qquad \text{n} \in \textbf{dom}\ \text{archive(w)} \Rightarrow \text{n} \in \textbf{dom}\ \text{purses(w))} \wedge \\
\quad (\forall\ \text{pd} : \text{PayDetails} \bullet \\
\qquad \text{to(pd)} \in \text{purses(w)} \wedge \\
\qquad \text{pdAuth(purses(w)(to(pd)))} = \text{pd} \Rightarrow \\
\qquad\quad \text{status(purses(w)(to(pd)))} \notin \{\text{epr, epa}\}) \wedge \\
\quad (\forall\ \text{pd} : \text{PayDetails} \bullet \\
\qquad \text{ffrom(pd)} \in \text{purses(w)} \wedge \\
\qquad \text{pdAuth(purses(w)(ffrom(pd)))} = \text{pd} \Rightarrow \\
\qquad\quad \text{status(purses(w)(ffrom(pd)))} \neq \text{epv)} \wedge
\end{array}
$$

Concrete Specification: World: invariant 2

$$
\begin{array}{l}
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{req(pd)} \in \mathrm{ether(w)} \Rightarrow \\
\qquad \mathrm{to(pd)} \in \mathrm{purses(w)} \wedge \\
\qquad \mathrm{toSeqNo(pd)} < \mathrm{nextSeqNo(purses(w)(to(pd))))} \wedge \\
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{val(pd)} \in \mathrm{ether(w)} \Rightarrow \\
\qquad \mathrm{to(pd)} \in \mathrm{purses(w)} \wedge \mathrm{ffrom(pd)} \in \mathrm{purses(w)} \wedge \\
\qquad \mathrm{toSeqNo(pd)} < \mathrm{nextSeqNo(purses(w)(to(pd)))} \wedge \\
\qquad \mathrm{fromSeqNo(pd)} < \mathrm{nextSeqNo(purses(w)(ffrom(pd))))} \wedge \\
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{ack(pd)} \in \mathrm{ether(w)} \Rightarrow \\
\qquad \mathrm{to(pd)} \in \mathrm{purses(w)} \wedge \mathrm{ffrom(pd)} \in \mathrm{purses(w)} \wedge \\
\qquad \ldots
\end{array}
$$

Concrete Specification: World: invariant 3

$$
\begin{array}{l}
(\forall\ \text{pd} : \text{PayDetails} \bullet \\
\quad \text{pd} \in \text{fromLogs(w)} \Rightarrow \\
\quad\quad \text{req(pd)} \in \text{ether(w)} \wedge \\
\quad\quad \text{fromSeqNo(pd)} < \text{nextSeqNo(purses(w)(ffrom(pd)))} \wedge \\
\quad\quad \text{(status(purses(w)(ffrom(pd)))} \in \{\text{epr, epa}\} \Rightarrow \\
\quad\quad\quad \text{fromSeqNo(pd)} < \text{fromSeqNo(pdAuth(purses(w)(ffrom(pd))))))} \wedge \\
(\forall\ \text{pd} : \text{PayDetails} \bullet \\
\quad \text{pd} \in \text{toLogs(w)} \Rightarrow \\
\quad\quad \text{req(pd)} \in \text{ether(w)} \wedge \\
\quad\quad \text{ack(pd)} \notin \text{ether(w)} \wedge \\
\quad\quad \text{(status(purses(w)(to(pd)))} \in \{\text{epv, eaTo}\} \Rightarrow \\
\quad\quad\quad \text{toSeqNo(pd)} < \text{toSeqNo(pdAuth(purses(w)(to(pd))))))} \wedge
\end{array}
$$

Concrete Specification: World: invariant 4

```
(∀ pd : PayDetails •
    ffrom(pd) ∈ purses(w) ∧
    status(purses(w)(ffrom(pd))) = epa ⇒
      req(pdAuth(purses(w)(ffrom(pd)))) ∈ ether(w)) ∧
(∀ pd : PayDetails •
    ffrom(pd) ∈ purses(w) ∧
    status(purses(w)(ffrom(pd))) = epr ⇒
      val(pdAuth(purses(w)(ffrom(pd)))) ∉
        ether(w) ∧
      ack(pdAuth(purses(w)(ffrom(pd)))) ∉
        ether(w)) ∧
```

Concrete Specification: World: invariant 5

$$
\begin{array}{l}
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{to(pd)} \in \mathrm{purses(w)} \wedge \\
\quad \mathrm{status(purses(w)(to(pd)))} = \mathrm{epv} \Rightarrow \\
\qquad \mathrm{req(pdAuth(purses(w)(to(pd))))} \in \mathrm{ether(w)} \wedge \\
\qquad \mathrm{ack(pdAuth(purses(w)(to(pd))))} \notin \mathrm{ether(w))} \wedge \\
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{req(pd)} \in \mathrm{ether(w)} \wedge \mathrm{ack(pd)} \notin \mathrm{ether(w)} \Rightarrow \\
\qquad \mathrm{(pd} \in \mathrm{toInEpv(w)} \vee \mathrm{pd} \in \mathrm{toLogs(w)))} \wedge \\
(\forall\ \mathrm{pd} : \mathrm{PayDetails} \bullet \\
\quad \mathrm{val(pd)} \in \mathrm{ether(w)} \wedge \mathrm{pd} \in \mathrm{toInEpv(w)} \Rightarrow \\
\qquad \mathrm{pd} \in \mathrm{fromInEpa(w)} \vee \mathrm{pd} \in \mathrm{fromLogs(w))} \wedge
\end{array}
$$

Concrete Specification: World: invariant 6

$$
\begin{array}{l}
(\forall\ \mathrm{pd} : \mathrm{PayDetails},\ \mathrm{n} : \mathrm{Name} \bullet \\
\quad \mathrm{exceptionLogResult(n, pd)} \in \mathrm{ether(w)} \Rightarrow \\
\quad\quad \mathrm{n} \in \mathbf{dom}\ \mathrm{allLogs(w)} \wedge \mathrm{pd} \in \mathrm{allLogs(w)(n))} \wedge \\
(\forall\ \mathrm{pds} : \mathrm{PayDetailsSet1},\ \mathrm{n} : \mathrm{Name} \bullet \\
\quad \mathrm{exceptionLogClear(n, image(pds))} \in \mathrm{ether(w)} \Rightarrow \\
\quad\quad \mathrm{n} \in \mathbf{dom}\ \mathrm{archive(w)} \wedge \mathrm{pds} \subseteq \mathrm{archive(w)(n))} \wedge \\
(\forall\ \mathrm{m} : \mathrm{Message} \bullet \\
\quad \mathrm{m} \in \mathrm{visible(w)} \Rightarrow \mathrm{m} \in \mathrm{ether(w))}
\end{array}
$$

16 conjuncts which must be proved as invariant for 11 operations!!

The Argument for Correctness

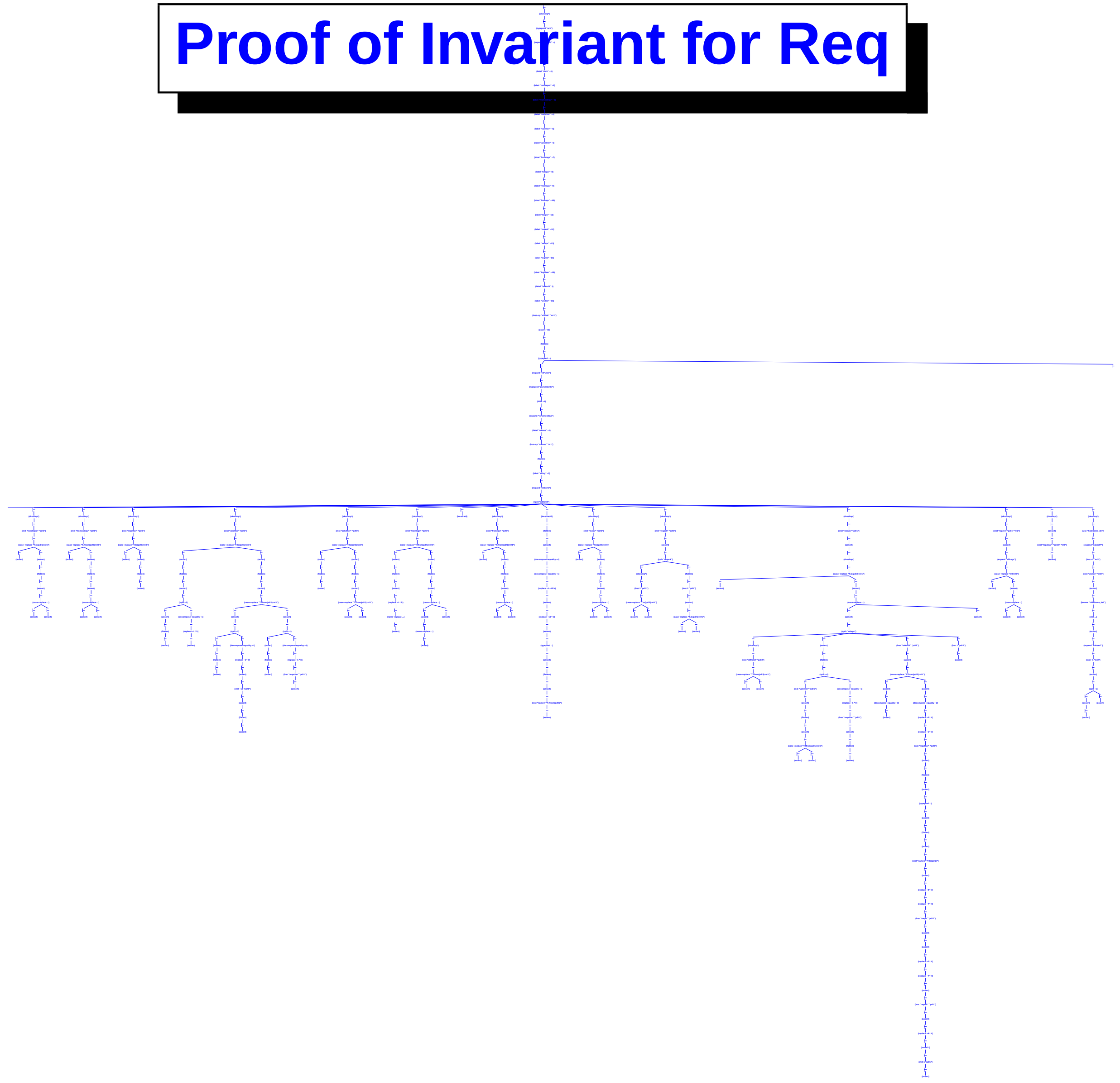
1. LeftTransfer, RightTransfer, Abort and No_op are *correct*; Composition and sequence preserve correctness.
2. Each abstract operation is a LeftTransfer, RightTransfer, Abort or No_op.
3. Each abstract operation is refined by its concrete operation.

Easy!

Perhaps ...

- This is the 10th version of the specification, which is 2200 lines of RSL in 13 files.
- There are 366 proofs, perhaps half proved automatically.
- A typical invariant proof for the concrete specification is about 300 prover commands (recall there are 11 of these proofs).
- Other unpleasant proofs were that the concrete invariant implied the abstract one (150 prover commands), and that some sets defined by comprehension are finite.

Proof of Invariant for Req



Automation?

- The biggest problem is identifying the invariant.
  - Too strong: helps with proofs of refinement but can't be proved.
  - Too weak: easier to prove but refinement proofs fail.
- This problem has many large proofs with similar structure: tactics are worth developing.
  - The perfect tactic is very hard to write.
  - A tactic that does all the setting up of standard hypotheses, names them, and does the basic case analysis and tries to discharge the results can be very useful.
- One incautious `grind` generated (eventually) 1580 subgoals!

Did We Capture the Requirements Correctly?

- There may be many subtle points in 2200 lines of RSL!
- In the Z specification, for example, the description of a complete transfer is only informally stated, but seems to be unimplementable, because it requires you to know in advance a property of Abort that is underspecified (and perhaps nondeterministic): possible increase in nextSeqNo.
- Is there an “**axiom false**” somewhere?

Are our tools correct?

We rely on

- Translator from RSL to PVS
- PVS proof engine

Comparison with Z Approach

Z specification has 3 levels:

1. Abstract model A has purse operations AbTransferOK, AbTransferLost, and AbIgnore. Specification is that these are *correct*.
2. Between model B has concrete purses, concrete operations, concrete world containing collection of purses (with operations defined by promoting the purse operations), ether of messages, and archive of logs. Invariant much like our concrete one. Use of “backward” proof rules to show that sequences of concrete operations simulate sequences of abstract operations.
3. Concrete model C has similar structure to B but no state invariant. Loss of messages introduced. Proof that C refines B done using “forward” proof rules.

Remarks on Z Approach

- Refinement is relational.
- A set of operations in B refines a set of operations in A if the operations (seen as state transformers) in B simulate the corresponding operations in A.
- Simulation is defined in terms of a retrieve relation between the A and B states: seen through this retrieve relation the states reachable in B should be a subset of those reachable in A.
- We get a subset relation because of underspecification (seen as nondeterminism in Z): if an abstract operation in A can finish in any of states S1, S2 or S3 and the concrete one can finish in S2, say, then this is OK.

RAISE approach

Refinement in RAISE is different from Z: based on classes; gives substitutivity.

But in fact very similar. We effectively show

- Req implements TransferLeft
- Val implements TransferRight
- StartFrom, StartTo, Abort, Readlog, ClearLog implement Abort
- Ack, Increase, AuthoriseClear, Archive implement No_op

Further work

- Drawing general conclusions on modelling and proving such systems
- Seeing what can be done with model checking
  - What are useful abstractions?
  - Finding a sequence forming a complete transfer
  - Checking candidate invariant clauses for falsity!
- Can we improve the automation?