# Algebraic Approaches to Formal Analysis of the Mondex Electronic Purse System

**COE Symposium/VERITE, Mar. 7, 2007**

**Graduate School of Information Science, JAIST**
**Weiqiang Kong**

**Joint-work with Kazuhiro Ogata and Kokichi Futatsugi**

# Overview

- The Mondex electronic purse system.
- Specification and Verification using the OTS/CafeOBJ method.
- Falsification using the BOTS/Maude method.
- Related work and conclusion.

# Part 1:
# Mondex* Electronic Purse System

- A payment system that uses smartcards as electronic purses, which provides an alternative form of cash to physical notes and coins.

- Cards store monetary value as electronic information
  - Value can be (re)loaded from ATM or through phone lines;
  - Value can be transferred between cards via communication devices.

- No need of a central control for transactions as credit/debit cards do;
- Can make Card-to-Card transaction.
- …

* MasterCard International. Mondex. URL: *http://www.mondex.com*

# Communication Protocol of Mondex System

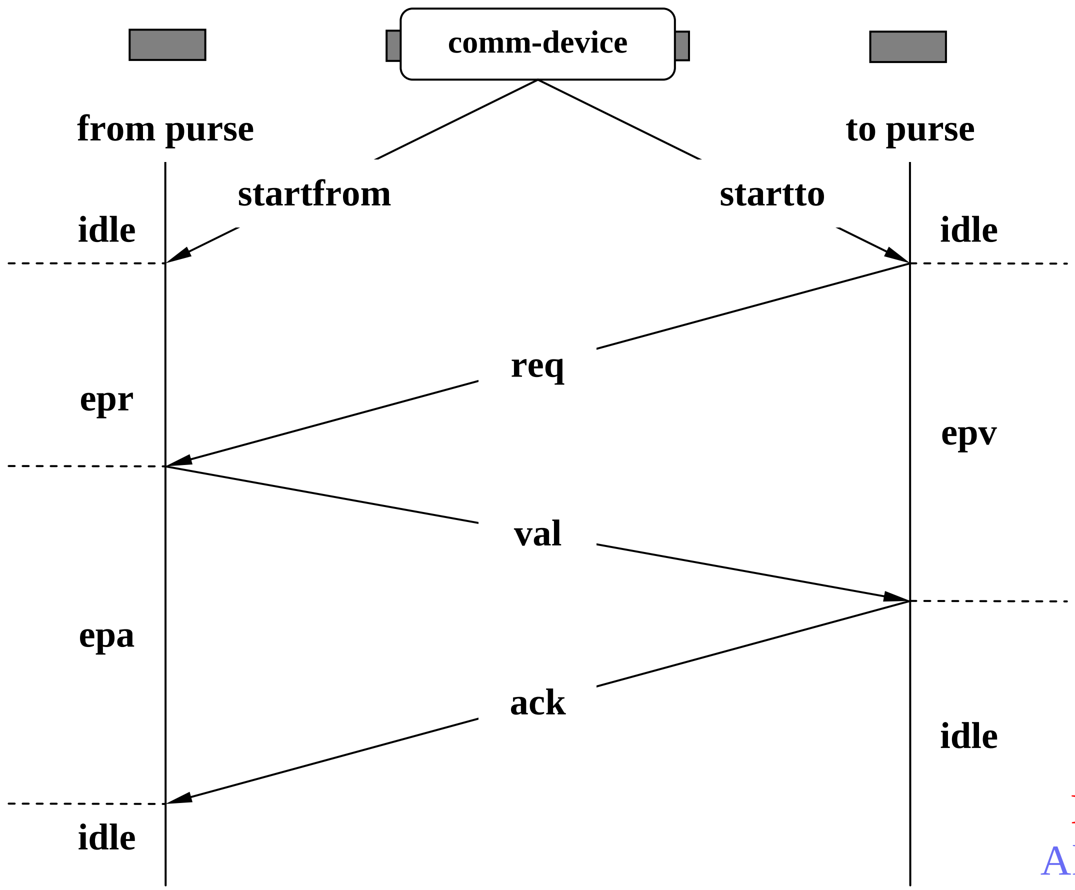



Figure adapted from
Alloy paper for Mondex

- **startfrom(toName,value,toSeq), startto(fromName,value,fromSeq),**
- **req(payDetail), val(payDetail), ack(payDetail)**
- **mk-pd(fromName,fromSeq,toName,toSeq,value)**

# Seems to be simple, But…

- Some security issues
  - The protocol can be stopped at any time;
  - A message can be lost and replayed;
  - A message can be read by any purse.

- Two desired security properties
  - No value may be created in the system,
  - All value should be counted in the system (no value is lost).

# A Chosen Case Study for Grand Challenge 6 (GC6)

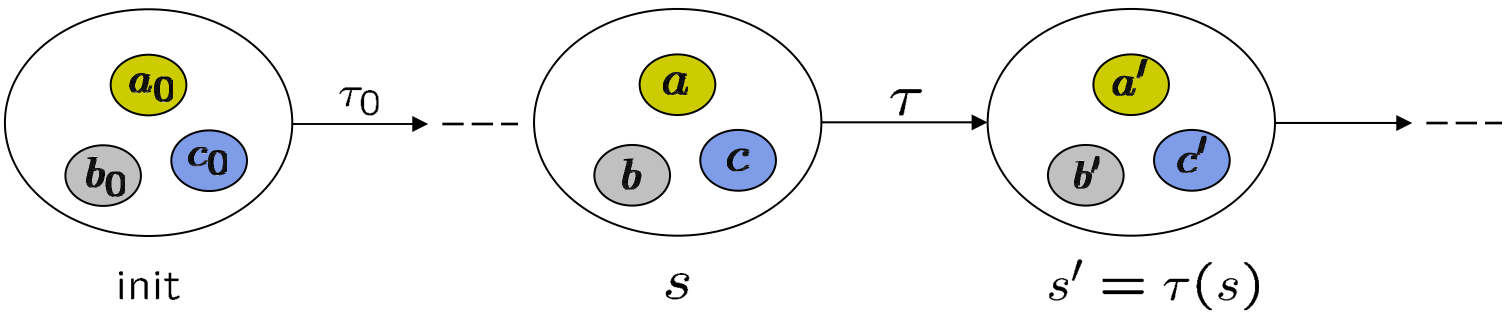
- Mondex was originally specified and manually proved by Z method. [240 pages for Spec & Proof, additional 54 pages for refinement theory]
  - An abstract model: atomic transaction. Easily proved properties hold.
  - A concrete model: transaction using protocol. Prove that it is a refinement of the abstract model.

- Mondex was chosen (Jan. 2006) as a main case study for a GC6 (dependable software evolution) project:
  - To see what the current state-of-the-art is in mechanizing the specification, refinement, and proof. (Ideally aim for full automation.)

- Several follow-up work
  - KIV, Alloy, RAISE and Event-B etc.

# Part 2:
# The OTS/CafeOBJ Formalism – Modeling & Spec.

- Observational Transition System (OTS)




- Specification of OTS in CafeOBJ (called OTS/CafeOBJ specification)
  - States are characterized by return values (observed values) of observers

    eq $o(\texttt{init}, x_1, \ldots, x_m) = f(x_1, \ldots, x_m)$ .

  - State transitions are characterized by changes of return values of observers

    ceq $o(\underline{\tau(S, y_1, \ldots, y_n)}, x_1, \ldots, x_m)$
    $= e\text{-}\tau(S, y_1, \ldots, y_n, x_1, \ldots, x_m)$ if $c\text{-}\tau(S, y_1, \ldots, y_n)$ .

    ***Successor state of S wrt $\tau$***

# Basic Data Types used in the OTS Modeling

- Purse. Constructor mk-purse

(1) Name (2) Previous Bal (3) Current Bal (4) Seqnum
(5**)** Status: idle | epr | epv | epa
(6) Paydetail: mk-pd(fromName, fromSeq, toName, toSeq, value)
(7) Exlog: a list of payment details of failed transactions.

- Message. Constructors startfrom, startto, req, val, ack

startfrom(N:Name, V:Bal, S:Seqnum), startto(N:Name, V:Bal, S:Seqnum)
req(P:Paydetail), val(P:Paydetail), ack(P:Paydetail)

- Ether. Constructors **nil**, **_,_**

Predicates and Operations: _/in_, empty?, get, top

# Specification of the OTS Model

- Observers and transitions of the OTS model

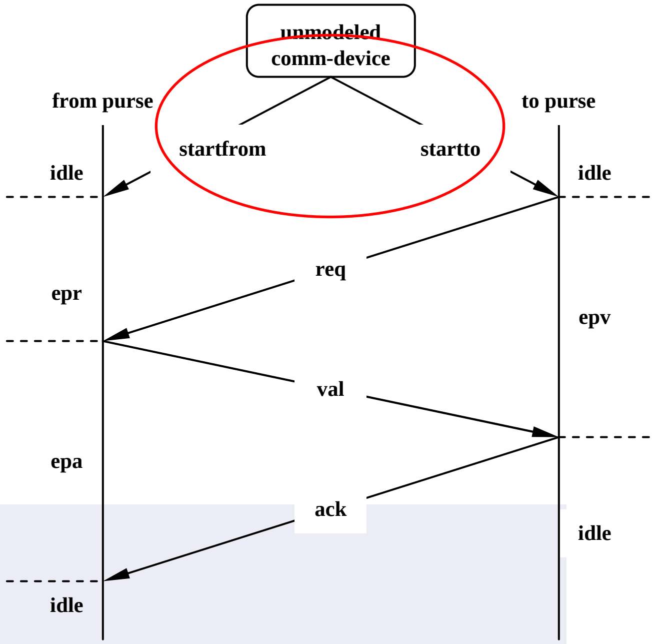
```
purse  :  Sys Name -> Purse
ether  :  Sys -> Ether
```

```
startpay      :  Sys Name Name Bal -> Sys
recstartfrom  :  Sys Name Message -> Sys
recstartto    :  Sys Name Message -> Sys
recreq        :  Sys Name Message -> Sys
recval        :  Sys Name Message -> Sys
recack        :  Sys Name Message -> Sys
abort         :  Sys Name -> Sys
drop          :  Sys -> Sys
duplicate     :  Sys -> Sys
```

- Any initial state

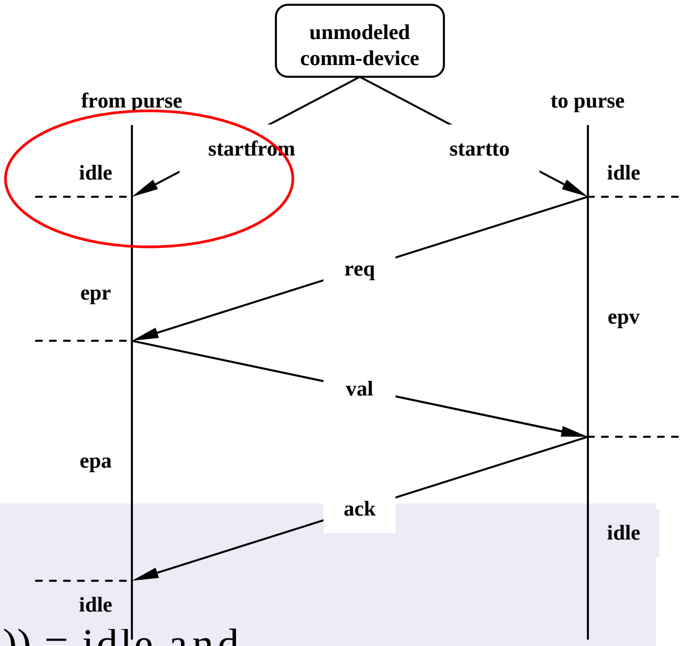
```
eq purse(init,Q) = mk-purse(Q, ibal(Q,seedval), ibal(Q,seedval),
                            inum(Q,seednum), idle, none, emptyexlog) .
eq ether(init) = nil .
```

# Transition startpay



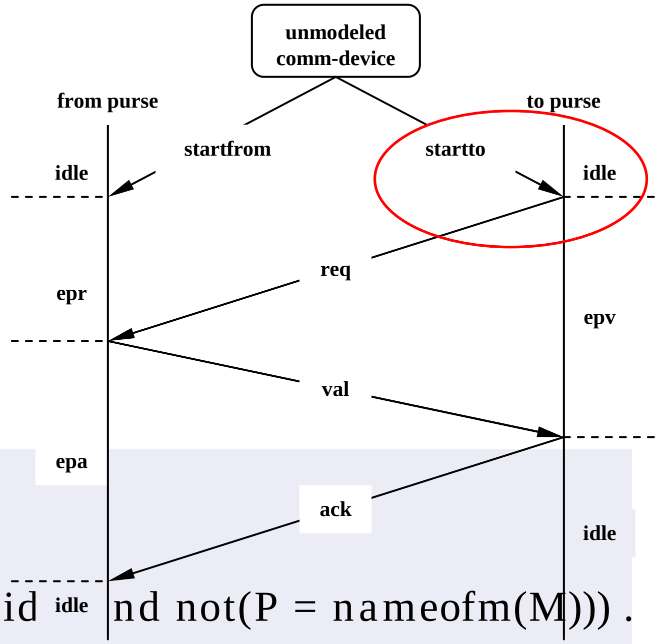

```
-- Effective condition of transition startpay
op c-startpay : Sys Name Name Bal -> Bool
eq c-startpay(S,P1,P2,V)
   = (sta(purse(S,P1)) = idle and sta(purse(S,P2)) = idle and not(P1 = P2)) .

-- Equations defining the execution of transition startpay
ceq purse(startpay(S,P1,P2,V),Q) = purse(S,Q)        if c-startpay(S,P1,P2,V) .
ceq ether(startpay(S,P1,P2,V))
    = startfrom(P2,V,seq(purse(S,P2))),
      startto(P1,V,seq(purse(S,P1))),ether(S)         if c-startpay(S,P1,P2,V) .
ceq startpay(S,P1,P2,V) = S                           if not c-startpay(S,P1,P2,V) .
```

# Transition recstartfrom



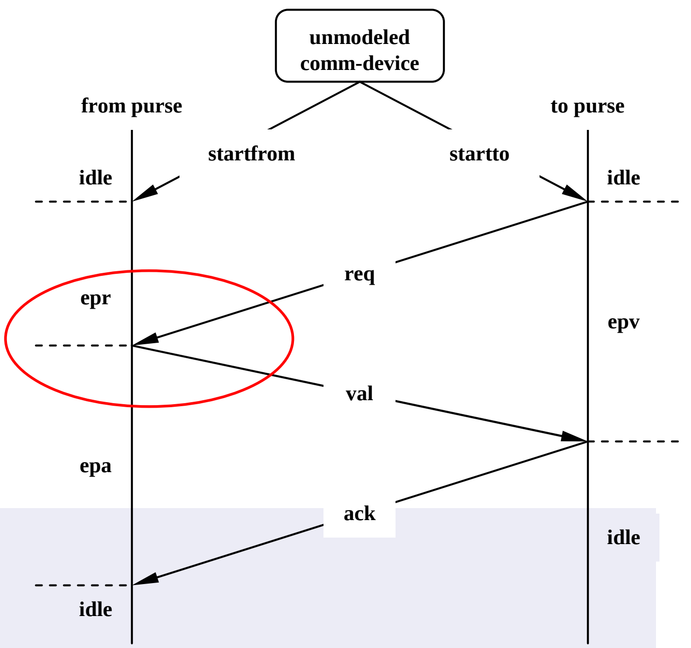

```
op c-recstartfrom : Sys Name Message -> Bool
eq c-recstartfrom(S,P,M)
   = (M /in ether(S) and isstartfrom(M) and sta(purse(S,P)) = idle and
      not(P = nameofm(M)) and valueofm(M) <= bal(purse(S,P))) .
--
ceq purse(recstartfrom(S,P,M),Q) =
    mk-purse(Q,
             (if (P = Q) then bal(purse(S,Q)) else pbal(purse(S,Q)) fi),
             bal(purse(S,Q)),
             (if (P = Q) then nextseqnum(seq(purse(S,Q))) else seq(purse(S,Q)) fi),
             (if (P = Q) then epr else sta(purse(S,Q)) fi),
             (if (P = Q) then
                   mk-pd(Q,seq(purse(S,Q)),nameofm(M),seqofm(M),valueofm(M))
                   else pay(purse(S,Q)) fi),
              log(purse(S,Q)))                  if c-recstartfrom(S,P,M) .
ceq ether(recstartfrom(S,P,M)) = ether(S)       if c-recstartfrom(S,P,M) .
ceq recstartfrom(S,P,M) = S                 if not c-recstartfrom(S,P,M) .
```

# Transition recstartto



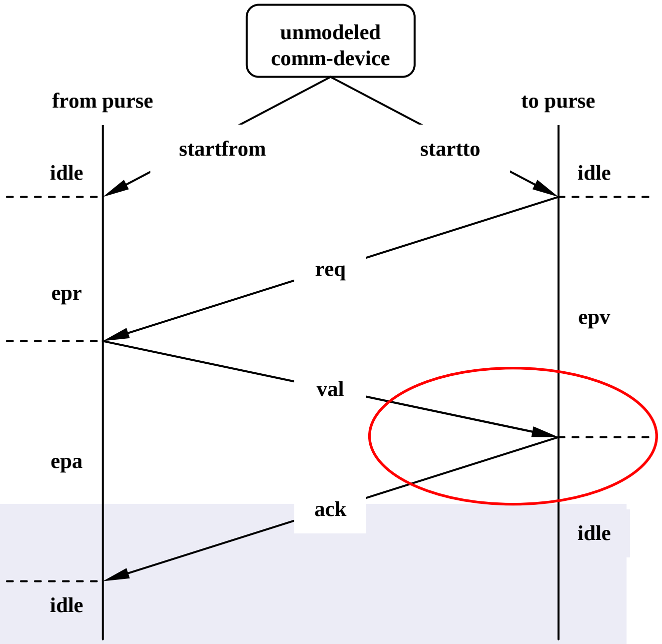

```
op c-recstartto : Sys Name Message -> Bool
eq c-recstartto(S,P,M)
    = (M /in ether(S) and isstartto(M) and sta(purse(S,P)) = idle and not(P = nameofm(M))) .
--
ceq purse(recstartto(S,P,M),Q) =
      mk-purse(Q,
                    (if (P = Q) then bal(purse(S,Q)) else pbal(purse(S,Q)) fi),
                    bal(purse(S,Q)),
                    (if (P = Q) then nextseqnum(seq(purse(S,Q))) else seq(purse(S,Q)) fi),
                    (if (P = Q) then epv else sta(purse(S,Q)) fi),
                    (if (P = Q) then
                              mk-pd(nameofm(M),seqofm(M),Q,seq(purse(S,Q)),valueofm(M))
                              else pay(purse(S,Q)) fi),
                    log(purse(S,Q)))                         if c-recstartto(S,P,M) .
ceq ether(recstartto(S,P,M)) =
            req(mk-pd(nameofm(M),seqofm(M),P,seq(purse(S,P)),valueofm(M))),ether(S)
                                                             if c-recstartto(S,P,M) .
ceq recstartto(S,P,M) = S                                 if not c-recstartto(S,P,M) .
```

# Transition recreq



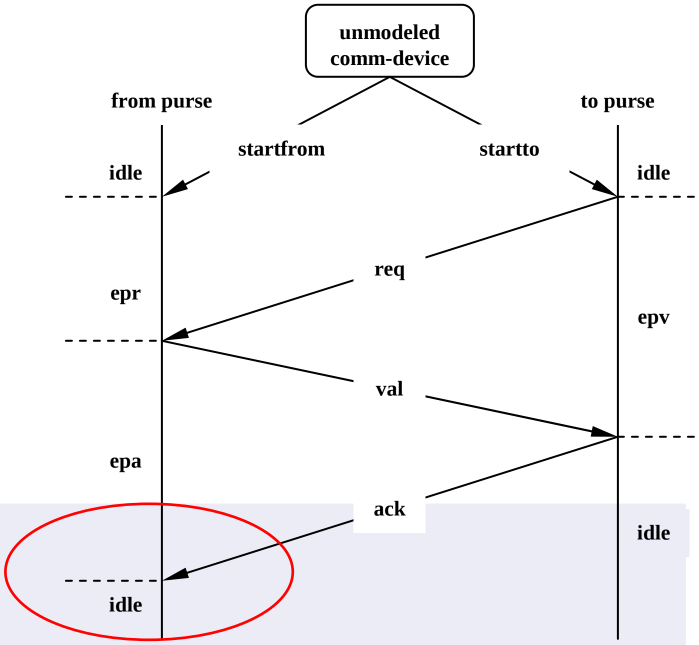

```
op c-recreq : Sys Name Message -> Bool
eq c-recreq(S,P,M)
    = (M /in ether(S) and isreq(M) and sta(purse(S,P)) = epr and
       pay(purse(S,P)) = pdofm(M)) .
--
ceq purse(recreq(S,P,M),Q) =
      mk-purse(Q,pbal(purse(S,Q)),
                    (if (P = Q) then (bal(purse(S,Q)) - value(pdofm(M)))
                                  else bal(purse(S,Q)) fi),
                    seq(purse(S,Q)),
                    (if (P = Q) then epa else sta(purse(S,Q)) fi),
                    pay(purse(S,Q)),log(purse(S,Q)))              if c-recreq(S,P,M) .
ceq ether(recreq(S,P,M)) = val(pdofm(M)),ether(S)                 if c-recreq(S,P,M) .
ceq recreq(S,P,M) = S                                         if not c-recreq(S,P,M) .
```

# Transition recval



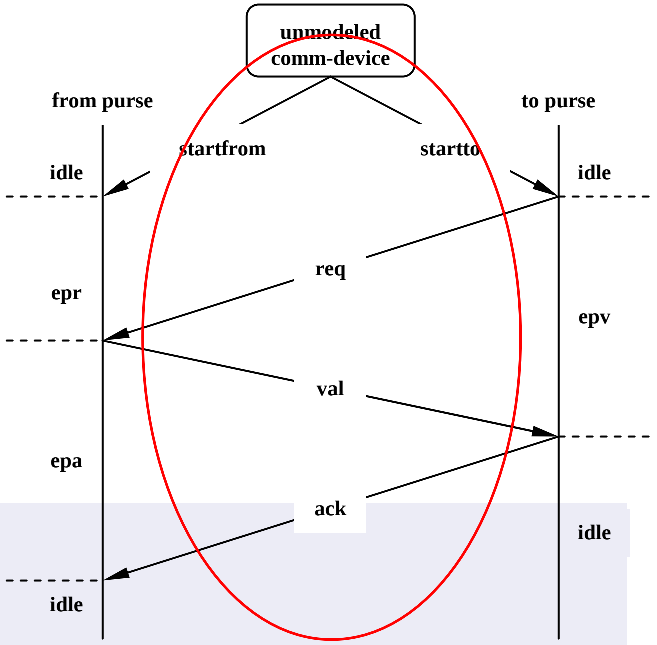

```
op c-recval : Sys Name Message -> Bool
eq c-recval(S,P,M)
    = (M /in ether(S) and isval(M) and sta(purse(S,P)) = epv and
       pay(purse(S,P)) = pdofm(M)) .
--
ceq purse(recval(S,P,M),Q) =
      mk-purse(Q,pbal(purse(S,Q)),
                   (if (P = Q) then (bal(purse(S,Q)) + value(pdofm(M)))
                              else bal(purse(S,Q)) fi),
                   seq(purse(S,Q)),
                   (if (P = Q) then idle else sta(purse(S,Q)) fi),
                   pay(purse(S,Q)), log(purse(S,Q)))          if c-recval(S,P,M) .
ceq ether(recval(S,P,M)) = ack(pdofm(M)),ether(S)             if c-recval(S,P,M) .
ceq recval(S,P,M) = S                                     if not c-recval(S,P,M) .
```

# Transition recack



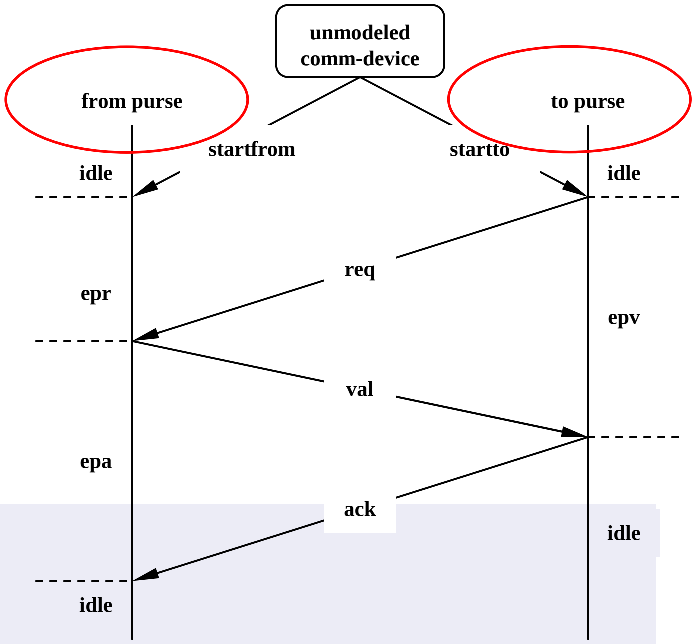

```
op c-recack : Sys Name Message -> Bool
eq c-recack(S,P,M)
    = (M /in ether(S) and isack(M) and sta(purse(S,P)) = epa and
      pay(purse(S,P)) = pdofm(M)) .
--
ceq purse(recack(S,P,M),Q) =
      mk-purse(Q,pbal(purse(S,Q)),bal(purse(S,Q)),seq(purse(S,Q)),
                  (if (P = Q) then idle else sta(purse(S,Q)) fi),
                  pay(purse(S,Q)),log(purse(S,Q)))            if c-recack(S,P,M) .
ceq ether(recack(S,P,M)) = ether(S)                          if c-recack(S,P,M) .
ceq recack(S,P,M) = S                                    if not c-recack(S,P,M) .
```

# Transitions drop and duplicate




```
-- transition drop
op c-drop : Sys -> Bool
eq c-drop(S) = not empty?(ether(S)) .
--
ceq purse(drop(S),Q) = purse(S,Q)       if c-drop(S) .
ceq ether(drop(S)) = get(ether(S))       if c-drop(S) .
ceq drop(S) = S                          if not c-drop(S) .


-- transition duplicate
op c-duplicate : Sys -> Bool
eq c-duplicate(S) = not empty?(ether(S)) .
--
ceq purse(duplicate(S),Q) = purse(S,Q)              if c-duplicate(S) .
ceq ether(duplicate(S)) = top(ether(S)),ether(S)    if c-duplicate(S) .
ceq duplicate(S) = S                                if not c-duplicate(S) .
```

# Transition abort




```
eq purse(abort(S,P),Q) =
    mk-purse(Q,pbal(purse(S,Q)),bal(purse(S,Q)),
                (if (P = Q) then nextseqnum(seq(purse(S,Q)))
                            else seq(purse(S,Q)) fi),
                (if (P = Q) then idle else sta(purse(S,Q)) fi),
                pay(purse(S,Q)),
                (if (P = Q) then
                            (if ((sta(purse(S,Q)) = epa) or (sta(purse(S,Q)) = epv))
                                 then pay(purse(S,Q)) @ log(purse(S,Q)) else log(purse(S,Q)) fi)
                            else log(purse(S,Q)) fi)) .
eq ether(abort(S,P)) = ether(S) .
```

# Desired Security Properties – Property 1

- No value may be created in the system:
- Two different purses that have **same payment details** and in status **idle**:
  - No transaction ever happens for each of them (pay details are none),
  - A transaction between them just finished, normally or abnormally does not matter.

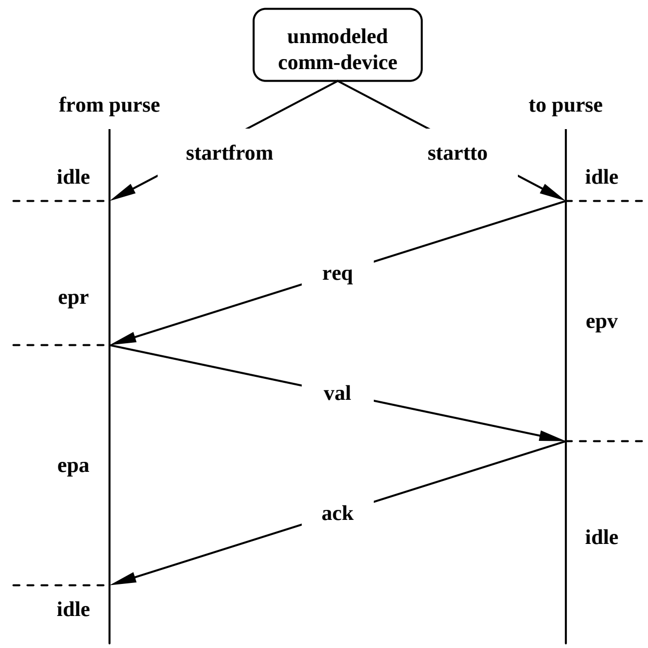
```
eq inv100(S,P1,P2) =
    ((sta(purse(S,P1)) = idle and sta(purse(S,P2)) = idle and
      pay(purse(S,P1)) = pay(purse(S,P2)) and not(P1 = P2))
     implies
     ((bal(purse(S,P1)) + bal(purse(S,P2))) <= (pbal(purse(S,P1)) + pbal(purse(S,P2))))) .
```

```
eq inv340(S,P1,P2) =
    ((pay(purse(S,P1)) = pay(purse(S,P2)) and not(P1 = P2))
     implies
     ((bal(purse(S,P1)) + bal(purse(S,P2))) <= (pbal(purse(S,P1)) + pbal(purse(S,P2))))) .
```

# How to Express Property 2?

- All value should be counted in the system (no value is lost).
- Two different purses that have **same payment details** and in status **idle**:
  - No transaction ever happens for each of them (pay details are none),
  - A transaction between them just finished, normally or abnormally does not matter.

| from \ to | | abort | non-abort |
|---|---|---|---|
| abort | log | lost | No lost |
| abort | non-log | No lost | (impossible) |
| non-abort | | (impossible) | No lost |



# Desired Security Properties – Property 2

- All value should be counted in the system (no value is lost).

```
eq inv500(S,P1,P2) =
    ((sta(purse(S,P1)) = idle and sta(purse(S,P2)) = idle and
      pay(purse(S,P1)) = pay(purse(S,P2)) and not(P1 = P2))
      implies
      (if (pay(purse(S,P1)) /inexlog log(purse(S,P1)))
          and (pay(purse(S,P2)) /inexlog log(purse(S,P2)))
       then ((bal(purse(S,P1)) + bal(purse(S,P2)) + lost(pay(purse(S,P1)),log(purse(S,P1))))
                = (pbal(purse(S,P1)) + pbal(purse(S,P2))))
       else ((bal(purse(S,P1)) + bal(purse(S,P2)))
                = (pbal(purse(S,P1)) + pbal(purse(S,P2)))) fi)) .
```

# Part 3:
# Falsification of Desired Security Properties

- A way similar to Bounded Model Checking by employing Maude `search` command for finding counterexamples.

- Motivations:
  - Easier, more automatic than proof and informative counterexamples;
  - Before verification: provides certain degree's confidence;
  - During verification: filter out incorrect lemmas.

# A Sample Conditional Rewrite Rule

- Two purses p1 and p2 are considered, and bound is set to 9

```
crl[startpay_p1_p2_con]:
    (purse[p1] : PUR1)  (purse[p2] : PUR2)  (ether : ETH)  (steps : C)
    =>
    (purse[p1] : PUR1)  (purse[p2] : PUR2)
    (ether : (startfrom(p2, con, seq(PUR2)), startto(p1, con, seq(PUR1)),ETH))
    (steps : (C + 1))
if (sta(PUR1) = idle and sta(PUR2) = idle and not(p1 = p2) and C < bound) .
```

# Search Command for Property 1

```
search [1] in MONDEX :
init =>* (purse[P1] : PUR1)   (purse[P2] : PUR2)   S
         such that not(
                   (sta(PUR1) = idle and sta(PUR2) = idle and
                   pay(PUR1) = pay(PUR2) and not(name(P1) = name(P2)))
                   implies
                   ((bal(PUR1) + bal(PUR2)) <= (pbal(PUR1) + pbal(PUR2)))
                  ) .
```

- Two purses p1 and p2 are considered, and bound is set to 9

```
No solution.
states: 1725347 rewrites: 1304806394 in 8348686ms cpu (8579704ms real)
(156288 rewrites/second)

Costs about 2 hours on Jaist XT3 massively parallel processing system.
No response after 12 hours' running on my desktop (3.2 GHz, 2 GB RAM).
```

# Part 4:
# Related Work – Modeling and Verification

- RAISE and Alloy work is very similar to the Z work wrt. modeling.
- KIV work's ASM models modified the Z modeling in several aspects:
  - In general: operational style vs. relational style
  - In particular: merges status "eafrom" and "eato" into "idle"; removes ignore operation etc.
- Our work is inspired by KIV's ASM modeling method, but:
  - startfrom, startto messages need not to be always available.
  - No condition for abort. But KIV defined condition for it.
  - drop and duplicate are explicitly defined. But KIV uses "ether' $\subseteq$ ether", and does not model message replay explicitly.
- Verification: Directly proving invariants vs. Refinement proof
  - Share some exactly same and similar proof obligations.

# Related Work – Falsification

- In RAISE work, RSL specification is translated into SAL
  - Falsification within a finite reachable state space.
  - Falsification of refinement.
  - The possible loss of messages is not modeled.
  - Sequence numbers are in the range 0…3.
  - Besides, many changes to the ether.
- In Alloy work, Alloy-analyzer (model-finding using SAT-Solver)
  - Falsification within a finite scope (how many objects are used)
  - Falsification of refinement.
- We are able to use inductively defined data types, such as Ether.

## Conclusion:

- Show how Mondex can be analyzed using two algebraic approaches for both verification and falsification within a couple of weeks.

  - An alternative way of modeling of the Mondex system as an OTS,
  - An alternative way of expressing and verification of the security properties directly as invariants of the OTS,
  - An automatic way of falsification that may help in several aspects.

- Intruder purses are to be considered. After introducing a cryptographically secured communication protocol, prove that messages cannot be forged rather than assuming it.

Thanks!