Modeling and Verifying Mondex in PD

Progress and Pitfalls

ESCHER TECHNOLOGIES

What I said last time…

ESCHER TECHNOLOGIES

Our goals in this project

- Learn more about how to best represent system-level specifications in PD
- Understand the limitations of the PD prover in this context, and overcome them
- Achieve a fully automatic proof of the Mondex case study
- Use what we have learned to make PD more usable for system-level verification

The top-level plan

- Translate the concrete model from Z to PD
- Adapt the refinement steps as needed
  - Make them more suitable for PD
- Work on the unproven verification conditions
  - Where necessary, improve the prover and/or provide hints
- Generate code for the purse and other components
  - If time, write a simple Java framework in which to exercise them

What I learned at the last meeting…

- The Z refinement proof is structured the way it is in order to handle concurrency
  - The refinement relation maps the entire abstract transaction to the first step of the concrete protocol, which can only be done with a non-deterministic refinement relation
  - …so this can't be mapped directly to PD, which requires a deterministic retrieve function
- The original Z project was to model and verify an existing concrete implementation
  - So there is no good reason for us not to use a different approach that is more natural for PD

A more natural approach for PD

- In a concurrent system, don't try to refine an atomic abstract operation to a sequence of concrete operations
- Recognise that transactions are non-atomic and re-formulate the security properties accordingly
- In particular, we need to account for value that has been debited from a sending purse but not been credited to the intended recipient
  - If the recipient is still expecting it, it is 'in transit'
  - If the recipient has recorded the transaction in its exception log, it is 'lost'

Restating the security properties

- For any atomic operation:
  - totalBalance' + totalInTransit' ≤ totalBalance + totalInTransit
  - totalBalance' + totalInTransit' + totalLost' = totalBalance + totalInTransit + totalLost
- Always:
  - totalInTransit ≥ 0
- At initialisation and after finalisation:
  - totalInTransit = 0
  - …the properties reduce to the ones in the Z abstract specification
  - [do we really need finalisation with this approach?]

Modeling the concrete purse

- We have modelled the concrete purse in PD apart from clearing the exception logs
  - They stay on the purse for now
- We initially followed the Z model except as follows:
  - Removed state **eaTo** (it doesn't appear to serve any useful purpose – left over from the original full model?)
  - Removed non-determinacy when increasing sequence numbers (lets us generate code and may slightly simplify the proofs)
  - Split the exception log into separate 'from' and 'to' logs
  - Expanded some schemas, e.g. AckPurseOK

Modeling the World

- We have modelled the concrete world, except for the global exception logs
  - The authorised purses are a **map of** (Name -> Purse)
  - Messages are represented by a type hierarchy
- Ether constraints were translated from the Z
  - We split some of them into several simpler constraints
  - B13 not needed as all collections in PD are finite
  - B16 not needed as subsumed by B10c and B12b
  - 19 class invariants in all

Ether invariants B3 and B4

- We had to add additional invariants to ensure the main invariants were well-formed, e.g. for B3:

// Additional invariants to satisfy the preconditions of B3a and B3b

forall m::ether.valMessages
:- m.pd.toIdent in authPurse,

forall m::ether.valMessages
:- m.pd.fromIdent in authPurse,

// Invariants B3a, B3b: there are no future val messages

forall m::ether.valMessages
:- m.pd.toSeq < authPurse[m.pd.toIdent].nextSeq,

forall m::ether.valMessages
:- m.pd.fromSeq < authPurse[m.pd.fromIdent].fromSeq;

Current state of proof

- 213 verification conditions are generated
  - 106 in class *World*, 106 in class *Purse*, 1 in class *Message*
- 191 are automatically proved (~90%)
- 22 are not proved (~10%)
  - All are in class *World*
  - Most are VCs that an ether constraint is preserved
- Some prover limitations were exposed
  - Failures to use applicable proof rules due to over-aggressive optimisation (now fixed)
  - Failure to find a unification that depends on an equality

What does this tell us?

- System-level VCs are not like component-level VCs
  - Different in character
  - Harder to prove?
- The PD prover needs more work to handle them well
  - Mondex has exposed some sources of prover incompleteness that we haven't seen in software verification problems
  - Fully-automated provers may be less likely to succeed in this area
  - - but we haven't given up yet!

Where next?

- Refactor the problem to help the prover and pinpoint the problems
  - Replace the 2 public methods *receiveStartMessage* and *receiveTransferMessage* of class *Purse* by 5 new methods, one for each message type
- Improve the prover unification-with-equality algorithm
  - We believe this contributes to 1/3 to 1/2 of the proof failures
- Add extra assertions to help the prover with the security properties
  - State what happens to each of *totalBalance*, *totalLost* and *totalInTransit* when a purse processes a given message type

Statistics

- Total size of specification: ~ 550 lines
  - Including comments and layout
- Time per proof run: ~ 6 hours
  - Timeout was set to 16 minutes per VC…
  - …but longest time taken for a successful proof was 5½ minutes
- Total time I spent on this project: ~ 60 hours