AFFIRM Progress

Lee Pike Benjamin Jones Galois Inc.

February 11, 2015

Tower to SAL

Ideas on generating logical specifications (SAL) from an architectural DSL (Tower)

Premise: We want to generate models of a system that is specified in our DSL using abstractions appropriate to the domain of fault tolerant distributed systems.

Example

Consider a toy example system:

- one node labeled “A”
- A’s state consists of one integer variable
- a typed input channel to A, “rx”, carrying integers
- A updates its state by adding each received integer to it

Toy Example Specified in Tower

The node A is represented by a monitor that contains a handler listening to the input channel. The handler calls an `update` function upon receiving a message.

```
monitor "A" $ do

  st <- state "st"   -- local state
  store st 0         -- initialization

  handler rx "rx" $ do  -- handle channel "rx"
    callback (\m -> update m st)
```

Toy Example (continued..)

The update function specifies the details of A's state transition.

```
update m st = do
  m' <- deref m          -- dereference msg
  st' <- deref st        -- dereference current state
  store st (st' + m')    -- add msg to state and store resul
```

SAL Model

To generate a SAL model from the Tower code:

- generate a SAL MODULE for each monitor node
- map monitor state variables to SAL module LOCAL variables
- map channel inputs, clocks, and signals to module INPUTs
- map channel outputs to module OUTPUTs
- generate a TRANSITION from the asynchronous composition of the handlers

Toy Example in SAL

SAL module definition is straightforward. The `new?` input is added in order to model message received events.

```
monitorA: MODULE =
  INPUT   new? : BOOLEAN
  INPUT   rx   : INTEGER
  LOCAL   st   : INTEGER
  INITIALIZATION
    st = 0
  {- TRANSITION ... -}
END
```

State Transition

```
TRANSITION
  [
    new? --> st' = st + rx;
             new?' = FALSE
  []
    ELSE -->
  ]
```

Update Abstracted

Programmer annotates the state machines:

```
callback $ \m ->
  requires (0 <=? 0) $
  ensures (\r -> st <=? r) $
    update m st = {- original update code -}
```

SAL Transition Abstracted

```
TRANSITION
  [
    new? AND rx >= 0
         --> st' IN { x : INTEGER | x >= st };
             new?' = FALSE
  []
    new? AND rx < 0
         --> st' IN INTEGER;   -- undefined behavior?
  []
    ELSE -->
  ]
```

Concrete Steps

Short-term plans for implementing the ideas we've presented:

- *Implement SAL syntax in Haskell* and an embedded language of constructors and combinators for generating native SAL syntax `https://github.com/benjaminfjones/sal-lang`
- *Map Tower to SAL* using the requires/ensures framework to abstract state machine details
- Explore using *fault annotations* on channels
- Explore using the *synchronous observer model* for specifying system properties