MODULE *StateTransfer*

This specification describes both the allowable behaviours and liveness
of a system that transfers dynamically-changing state from a *mutator*
to an observer via snapshot and delta messages.

EXTENDS *Naturals*, *Sequences*

Model checking constants

For bounded checking, our models provide the following
constant values.

The set of values from which a *mutator* might pick as its next state.
CONSTANT *PickableValues*

A value, not in *PickableValues*, that signifies an observer has not yet observed
a *mutator*-chosen value.
CONSTANT *Empty*
ASSUME $Empty \notin PickableValues$

The maximum number of messages that the (*FIFO*) network can hold at any time.
CONSTANT *NetworkCapacity*
ASSUME $NetworkCapacity \in Nat$

Model state

The evolving state of the specification comprises the three variables below.

The value last picked by the *mutator*.
VARIABLE *mutator*

The value (if any) last observed by the observer (since connecting).
VARIABLE *observer*

The queue of messages from the *mutator* to the observer.
It is emptied upon a disconnection.
VARIABLE *network*

$vars \triangleq \langle mutator, observer, network \rangle$

Types

Below we describe the structure of the types that the
variables above inhabit.

The set of possible mutations that the *mutator* may perform.

$$Mutations \triangleq [PickableValues \to PickableValues]$$

The set of possible snapshot messages the *mutator* may send to the observer. At the time of sending, it contains the *mutator*'s latest picked value, *i.e.*, its state.

$$SnapshotMessages \triangleq [type : \{\text{“snapshot”}\},\ value : PickableValues]$$

The set of possible delta messages the *mutator* may send to the observer. At the time of sending, it contains a function that relates the previous state to the current state. In a real implementation, the *mutator* might send a well-known event rather than a “mapping”.

$$DeltaMessages \triangleq [type : \{\text{“delta”}\},\ event : Mutations]$$

The set of messages the *mutator* may send to the observer.

$$Messages \triangleq SnapshotMessages \cup DeltaMessages$$

TypeOK is an invariant that *TLC* checks. It ensures that, in all states, our variables are assigned values of the types we expect.

$$\begin{aligned} TypeOK \triangleq\ & \\ & \land mutator \in PickableValues && \text{The } mutator \text{ state is a value it has picked.} \\ & \land observer \in (PickableValues \cup \{Empty\}) && \text{The observer state is a value that} \\ & && \text{it has observed or empty.} \\ & \land network \in Seq(Messages) && \text{The network is a queue of messages} \\ & && \text{from the } mutator \text{ to the observer.} \end{aligned}$$

Behaviour

In this section, we describe the behaviours of our system.

The system starts in an *Init* state, where the network and observer are empty, but the *mutator* has already picked a value.

$$\begin{aligned} Init \triangleq\ & \\ & \land mutator \in PickableValues \\ & \land observer = Empty \\ & \land network = \langle\rangle \end{aligned}$$

An action the *mutator* might take, if there is capacity on the network, is to send a snapshot of the value it last picked. In a real system, the observer may request a snapshot, as an optimisation.

$$\begin{aligned} TransferSnapshot \triangleq\ & \\ & \land Len(network) < NetworkCapacity \\ & \land network' = Append(network, [type \mapsto \text{“snapshot”},\ value \mapsto mutator]) \\ & \land \text{UNCHANGED } \langle mutator, observer\rangle \end{aligned}$$

Another action the *mutator* might take, if there is capacity on the network, is to pick a new value and send an event to the observer that describes the transition.

$TransferDelta \triangleq$
$\land\ Len(network) < NetworkCapacity$
$\land\ \exists\, event \in Mutations :$
$\quad \land$ DOMAIN $event = \{mutator\}$
$\quad \land\ mutator' = event[mutator]$
$\quad \land\ network' = Append(network, [type \mapsto$ "delta", $event \mapsto event])$
$\quad \land$ UNCHANGED $\langle observer \rangle$

Note:
I wonder the specification would be more general/applicable if the *mutator*'s actions of "picking a new value" and "sending a delta" were not tied together. For example, that might allow the specification to describe systems employing delta conflation strategies.

The only action to observer can take, when it receives a message over the network, is to accumulate state.

$Accumulate \triangleq$
$\land\ Len(network) > 0$
$\land\ network' = Tail(network)$
$\land\ \lor\ \land\ Head(network).type =$ "snapshot" A new value
$\quad\quad \land\ observer' = Head(network).value$
$\quad \lor\ \land\ Head(network).type =$ "delta" An applicable change in value
$\quad\quad \land\ observer \in$ DOMAIN $Head(network).event$
$\quad\quad \land\ observer' = Head(network).event[observer]$
$\quad \lor\ \land\ Head(network).type =$ "delta" An inapplicable change in value
$\quad\quad \land\ observer = Empty$
$\quad\quad \land$ UNCHANGED $\langle observer \rangle$
$\land$ UNCHANGED $\langle mutator \rangle$

Occasionally, the network might disconnect the *mutator* from the observer and lose all its in-flight messages. The observer detects this drop in connection and resets its state to *Empty* to avoid processing changes after a gap.

$Disconnect \triangleq$
$\land\ network' = \langle\rangle$
$\land\ observer' = Empty$
$\land$ UNCHANGED $\langle mutator \rangle$

After an *Init* state, the system can transition to another state via any of these actions.

$Next \triangleq$
$\lor\ TransferSnapshot$
$\lor\ TransferDelta$
$\lor\ Accumulate$

$\lor$ *Disconnect*

Together the *Init* states and *Next* action define the allowed behaviours of our system.

$Spec \triangleq$

$\land$ *Init*

$\land \Box[Next]_{vars}$

FairSpec adds assumptions about the fairness of certain actions to the behaviours of *Spec*, thus limiting the behaviours to those that satisfy the *StateTransfers* property.

$FairSpec \triangleq$

$\land$ *Spec*	Must be a behaviour that satisfies *Spec*.
$\land \mathrm{WF}_{vars}(TransferDelta)$	Any behaviour where *TransferDelta* is constantly enabled must eventually perform it.
$\land \mathrm{WF}_{vars}(TransferSnapshot)$	Any behaviour where *TransferSnapshot* is constantly enabled must eventually perform it.
$\land \mathrm{SF}_{vars}(Accumulate)$	Any behaviour where *Accumulate* is repeatedly (but not necessarily constantly) enabled must eventually perform it.

Properties

This property says that the observer evenutally witnesses any value that the *mutator* picks.

$StateTransfers \triangleq \Box(\forall\, v \in PickableValues : mutator = v \leadsto observer = v)$

Note: I'm surprised the model checker says that the behaviours of *FairSpec* satisfy this property, as I can imagine some infinite behaviours that start with the prefix:

- *TransferDelta* (*i.e.*, pick a new value)
- *Disconnect* (*i.e.*, lose value)
- *TransferDelta* (*i.e.*, pick a new value)
- ...