MODULE *Commons*

LOCAL INSTANCE *Naturals*
LOCAL INSTANCE *FiniteSets*
LOCAL INSTANCE *Sequences*

LOCAL $Identity(x) \triangleq x$
LOCAL $Choose(S) \triangleq$ CHOOSE $x \in S :$ TRUE
LOCAL $IsEven(x) \triangleq x\%2 = 0$
$Max(S) \triangleq$ CHOOSE $x \in S : \forall y \in S : x \geq y$

Three different conflict relations. We identify the relation to use through the configuration files. We verify each property with all three.

Use the message's identifier, where the evens conflict with evens and odds with odds. This relationship has a partial ordering.

$IdConflict(m, n) \triangleq IsEven(m.id) = IsEven(n.id)$

All messages conflict in this relationship. The executions with this conflict relation are equivalent to the Atomic Multicast.

$AlwaysConflict(m, n) \triangleq$ TRUE

There is no conflict in this relationship. The executions with this conflict relation are equivalent to the Reliable Multicast.

$NeverConflict(m, n) \triangleq$ FALSE

We use multiple procedures provided by the TLA$^+$ community.
Most of the procedures are used locally to create the messages.

From Community Modules

LOCAL $IsInjective(f) \triangleq$

A function is injective iff it maps each element in its domain to a distinct element.

This definition is overridden by *TLC* in the *Java* class *SequencesExt*. The operator is overridden by the *Java* method with the same name.

$\forall a, b \in$ DOMAIN $f : f[a] = f[b] \Rightarrow a = b$

From Community Modules

LOCAL $SetToSeq(S) \triangleq$

Convert a set to some sequence that contains all the elements of the set exactly once, and contains no other elements.

CHOOSE $f \in [1\,..\,Cardinality(S) \rightarrow S] : IsInjective(f)$

From Community Modules

LOCAL $SetToSeqs(S) \triangleq$

Convert the set S to a set containing all sequences containing the elements of S exactly once and no other elements. Example:

$SetToSeqs(\{\}), \{\langle\rangle\}$
$SetToSeqs(\{\text{"}t\text{"}, \text{"}l\text{"}\}) = \{\langle\text{"}t\text{"}, \text{"}l\text{"}\rangle, \langle\text{"}l\text{"}, \text{"}t\text{"}\rangle\}$

LET $D \triangleq 1\,..\,Cardinality(S)$
IN $\{f \in [D \rightarrow S] : \forall\, i, j \in D : i \neq j \Rightarrow f[i] \neq f[j]\}$

From Community Modules

LOCAL $SetToAllKPermutations(S) \triangleq$

Convert the set S to a set containing all k-permutations of elements of S for $k \in 0\,..\,Cardinality(S)$. Example:

$SetToAllKPermutations(\{\}) = \{\langle\rangle\}$
$SetToAllKPermutations(\{\text{"}a\text{"}\}) = \{\langle\rangle, \langle\text{"}a\text{"}\rangle\}$
$SetToAllKPermutations(\{\text{"}a\text{"}, \text{"}b\text{"}\}) =$
$\{\langle\rangle, \langle\text{"}a\text{"}\rangle, \langle\text{"}b\text{"}\rangle, \langle\text{"}a\text{"}, \text{"}b\text{"}\rangle, \langle\text{"}b\text{"}, \text{"}a\text{"}\rangle\}$

UNION $\{SetToSeqs(s) : s \in$ SUBSET $S\}$

From Community Modules

LOCAL $MapThenFoldSet(op(\_, \_), base, f(\_), choose(\_), S) \triangleq$

Starting from base, apply *op* to $f(x)$, for all $x \in S$, by choosing the set elements with *choose* . If there are multiple ways for choosing an element, *op* should be associative and commutative. Otherwise, the result may depend on the concrete implementation of *choose* .

FoldSet, a simpler version for sets is contained in *FiniteSetsEx*. *FoldFunction*, a simpler version for functions is contained in Functions. *FoldSequence*, a simpler version for sequences is contained in *SequencesExt*.

Example:

$MapThenFoldSet($LAMBDA $x, y : x \cup y,$
$\{\},$
LAMBDA $x : \{\{x\}\},$
LAMBDA $set :$ CHOOSE $x \in set :$ TRUE,
$\{1, 2\})$

$= \{\{1\}, \{2\}\}$

$$
\begin{array}{l}
\text{LET } iter[s \in \text{SUBSET } S] \triangleq \\
\qquad \text{IF } s = \{\} \text{ THEN } base \\
\qquad \text{ELSE LET } x \triangleq choose(s) \\
\qquad\qquad \text{IN } op(f(x), iter[s \setminus \{x\}]) \\
\text{IN } iter[S]
\end{array}
$$

From Community Modules

$\text{LOCAL } ToSet(s) \triangleq$

The image of the given sequence s. $Cardinality(ToSet(s)) \leq Len(s)$ see https://en.wikipedia.org/wiki/Image_(mathematics)

$\{s[i] : i \in \text{DOMAIN } s\}$

From Community Modules

$ReplaceAt(s, i, e) \triangleq$

Replaces the element at position i with the element e.

$[s \text{ EXCEPT } ![i] = e]$

$\text{LOCAL } Originator(G, P) \triangleq \langle Choose(G), Choose(P) \rangle$

Initialize the message structure we use to check the algorithm.

$$
\begin{array}{l}
CreateMessages(nmessage, G, P) \triangleq \\
\qquad \{[id \mapsto m, d \mapsto G, o \mapsto Originator(G, P)] : m \in 1 \,..\, nmessage\}
\end{array}
$$

Create all possible different possibilities in the initial ordering. Since we replaced the combination of Reliable Multicast + Atomic Broadcast with multiple uses of Atomic Broadcast, messages can have distinct orders across groups. We force this distinction.

$$
\begin{array}{l}
CreatePossibleMessages(S) \triangleq \\
\qquad \text{LET } M \triangleq SetToAllKPermutations(S) \\
\qquad \text{IN } MapThenFoldSet(\\
\qquad\qquad\qquad \text{LAMBDA } x, y : \langle x \rangle \circ y, \\
\qquad\qquad\qquad \langle\rangle, \\
\qquad\qquad\qquad Identity, \\
\qquad\qquad\qquad Choose, \\
\qquad\qquad\qquad \{m \in M : Len(m) = Cardinality(S)\})
\end{array}
$$

We create the tuple with the message, state, and timestamp.

$\text{LOCAL } InitialMessage(m) \triangleq \langle m, \text{“S0”}, 0 \rangle$

A totally ordered message buffer.

$TotallyOrdered(F) \triangleq$
$\quad [x \in \text{DOMAIN } F \mapsto InitialMessage(F[x])]$

Creates a partially ordered buffer from the sequence using the given predicate to identify conflicts between messages.

$\text{LOCAL } ExistsConflict(x, S, Op(\_, \_)) \triangleq$
$\quad \exists d \in ToSet(S) :$
$\quad\quad \exists \langle n, s, ts \rangle \in d : Op(x, n)$

$PartiallyOrdered(F, Op(\_, \_)) \triangleq$
$\quad MapThenFoldSet($
$\quad\quad \text{LAMBDA } x, y :$
$\quad\quad\quad \text{IF } Len(y) = 0 \lor ExistsConflict(x, y, Op)$
$\quad\quad\quad\quad \text{THEN } \langle \{InitialMessage(x)\} \rangle \circ y$
$\quad\quad\quad \text{ELSE } \langle y[1] \cup \{InitialMessage(x)\} \rangle,$
$\quad\quad\quad \langle \rangle,$
$\quad\quad\quad Identity,$
$\quad\quad\quad Choose,$
$\quad\quad\quad ToSet(F))$

We enumerate the entries in the given set.

$Enumerate(base, E) \triangleq$
$\quad \text{LET } f \triangleq SetToSeq(E) \text{ IN } \quad \{\langle base + i, f[i] \rangle : i \in \text{DOMAIN } f\}$