MODULE *Consensus*

This is an very abstract specification of the consensus problem, in which a set of processes must choose a single value. We abstract away even the processes. We specify the simple requirement that at most one value is chosen by describing the set of all chosen values. The naive requirement is that this set never contains more than one value, which is an invariance property. But a little thought shows that this requirement allows a value to be chosen then unchosen, and then another value to be chosen. So we specify that the set of chosen values is initially empty, it can be set to a single value, and then it can never change.

We are ignoring liveness, so we do not any requirement that a value is eventually chosen.

EXTENDS *Naturals*, *FiniteSets*

Imports standard modules that define operators of arithmetic on natural numbers and the *Cardinality* operator, where *Cardinality*(S) is the number of elements in the set S, if S is finite.

CONSTANT *Value*

The set of all values that can be chosen.

VARIABLE *chosen*

The set of all values that have been chosen.

The type-correctness invariant asserting the "type" of the variable 'chosen'. It isn't part of the spec itself–that is, the formula describing the possible sequence of values that 'chosen' can have in a behavior correct behavior of the system, but is an invariance property that the spec should satisfy.

$$\begin{aligned} TypeOK \triangleq\ & \land\ chosen \subseteq Value \\ & \land\ IsFiniteSet(chosen) \end{aligned}$$

The initial predicate describing the possible initial state of 'chosen'.

$$Init \triangleq chosen = \{\}$$

The next-state relation describing how 'chosen' can change from one step to the next. Note that is enabled (equals true for some next value *chosen′* of choseen) if and only if chosen equals the empty set.

$$\begin{aligned} Next \triangleq\ & \land\ chosen = \{\} \\ & \land\ \exists\, v \in Value : chosen' = \{v\} \end{aligned}$$

The TLA+ temporal formula that is the spec.

$$Spec \triangleq Init \land \Box[Next]_{chosen}$$

The specification should imply the safety property that 'chosen' can contain at most one value in any reachable state. This condition on the state is expressed by *Inv* defined here.

$$\begin{aligned} Inv \triangleq\ & \land\ TypeOK \\ & \land\ Cardinality(chosen) \leq 1 \end{aligned}$$

The following theorem asserts the desired safety propert. Its proof appears after the theorem. This proof is easily checked by the *TLAPS* prover.

THEOREM *Invariance* $\triangleq$ $Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

$\langle 1 \rangle 2.\ Inv \land [Next]_{chosen} \ \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED
  BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF *Spec*

If you are reading this specification in the *Toolbox* as you should be (either the source file or its pretty-printed version), then you have already opened the specification in the *Toolbox*. If not, you should do that now. Download and run the *Toolbox*. Open a new specification with this module file (*Consensus:tla*) as the root file, using the default specification name, which is the name of the root file. Along with the module, this will install a *TLC* model named 3*Values*. Open that model. You will see that the model specifies three things:

- The specification is formula *Spec*.
- Three unspecified constants a, b, and c (called model values) are substituted for the declared constants Values.
- *TLC* should check that formula *Inv* is an invariant (a formula true on all reachable states of he specification.

Run *TLC* on the model. For this tiny spec, this just takes perhaps a millisecond plus the couple of seconds that *TLC* needs to start and stop running any spec.

TLC's default setting to check for deadlock would cause it to report a deadlock because no action is possible after a value is chosen. We would say that the system terminated, but termination is just deadlock that we want to happen, and the model tells *TLC* that we want deadlock by disabling its check for it.