The consensus problem requires a set of processes to choose a single value. This module specifies
the problem by specifying exactly what the requirements are for choosing a value.

EXTENDS Naturals, FiniteSets, FiniteSetTheorems, TLAPS

We let the constant parameter Value be the set of all values that can be chosen.

CONSTANT Value

We now specify the safety property of consensus as a trivial algorithm that describes the allowed
behaviors of a consensus algorithm. It uses the variable chosen to represent the set of all chosen
values. The algorithm is trivial; it allows only behaviors that contain a single state-change in
which the variable chosen is changed from its initial value {} to the value {v} for an arbitrary
value v in Value. The algorithm itself does not specify any fairness properties, so it also allows a
behavior in which chosen is not changed. We could use a translator option to have the translation
include a fairness requirement, but we don’t bother because it is easy enough to add it by hand
to the safety specification that the translator produces.

A real specification of consensus would also include additional variables and actions. In particular,
it would have Propose actions in which clients propose values and Learn actions in which clients
learn what value has been chosen. It would allow only a proposed value to be chosen. However,
the interesting part of a consensus algorithm is the choosing of a single value. We therefore restrict
our attention to that aspect of consensus algorithms. In practice, given the algorithm for choosing
a value, it is obvious how to implement the Propose and Learn actions.

For convenience, we define the macro Choose() that describes the action of changing the value of
chosen from {} to {v}, for a nondeterministically chosen v in the set Value. (There is little reason
to encapsulate such a simple action in a macro; however our other specs are easier to read when
written with such macros, so we start using them now.) The when statement can be executed only
when its condition, chosen = {}, is true. Hence, at most one Choose() action can be performed
in any execution. The with statement executes its body for a nondeterministically chosen v in
Value. Execution of this statement is enabled only if Value is non-empty—something we do not
assume at this point because it is not required for the safety part of consensus, which is satisfied
if no value is chosen.

We put the Choose() action inside a while statement that loops forever. Of course, only a single
Choose() action can be executed. The algorithm stops after executing a Choose() action. Techni-
cally, the algorithm deadlocks after executing a Choose() action because control is at a statement
whose execution is never enabled. Formally, termination is simply deadlock that we want to hap-
pen. We could just as well have omitted the while and let the algorithm terminate. However,
adding the while loop makes the TLA+ representation of the algorithm a tiny bit simpler.

```
--algorithm Consensus

variable chosen = {} ;

macro Choose( ) when chosen = {} ;

with ( v ∈ Value ) { chosen := {v} } 

{ lbl: while ( TRUE ) { Choose() } 

}
```

The PlusCal translator writes the TLA+ translation of this algorithm below. The formula Spec is
the TLA+ specification described by the algorithm’s code. For now, you should just understand
its two subformulas Init and Next. Formula Init is the initial predicate and describes all possible
initial states of an execution. Formula Next is the next-state relation; it describes the possible
state changes (changes of the values of variables), where unprimed variables represent their values
in the old state and primed variables represent their values in the new state.
We now prove the safety property that at most one value is chosen. We first define the type-correctness invariant $TypeOK$, and then define $Inv$ to be the inductive invariant that asserts $TypeOK$ and that the cardinality of the set $chosen$ is at most 1. We then prove that, in any behavior satisfying the safety specification $Spec$, the invariant $Inv$ is true in all states. This means that at most one value is chosen in any behavior.

$TypeOK \triangleq \land chosen \subseteq Value 
\land IsFiniteSet(chosen)$

$Inv \triangleq \land TypeOK 
\land Cardinality(chosen) \leq 1$

We now prove that $Inv$ is an invariant, meaning that it is true in every state in every behavior. Before trying to prove it, we should first use TLC to check that it is true. It’s hardly worth bothering to either check or prove the obvious fact that $Inv$ is an invariant, but it’s a nice tiny exercise. Model checking is instantaneous when $Value$ is set to any small finite set.

To understand the following proof, you need to understand the formula $Spec$, which equals

$Init \land \Box[Next]vars$

where $vars$ is the tuple $(chosen, pc)$ of all variables. It is a temporal formula satisfied by a behavior iff the behavior starts in a state satisfying $Init$ and such that each step (sequence of states) satisfies $[Next]vars$, which equals

$Next \lor (vars' = vars)$

Thus, each step satisfies either $Next$ (so it is a step allowed by the next-state relation) or it is a “stuttering step” that leaves all the variables unchanged. The reason why a spec must allow stuttering steps will become apparent when we prove that a consensus algorithm satisfies this specification of consensus.

The following lemma asserts that $Inv$ is an inductive invariant of the next-state action $Next$. It is the key step in proving that $Inv$ is an invariant of (true in every behavior allowed by) specification $Spec$.

**Lemma InductiveInvariance**

$Inv \land [Next]vars \Rightarrow Inv'$
We now define LiveSpec to be the algorithm’s specification with the added fairness condition of weak fairness of the next-state relation, which asserts that execution does not stop if some action is enabled. The temporal formula Success asserts that some value is eventually chosen. Below, we prove that LiveSpec implies Success. This means that, in every behavior satisfying LiveSpec, some value is chosen.

\[
\text{LiveSpec} \triangleq \text{Spec} \land \text{WF}_{\text{vars}}(\text{Next})
\]
\[
\text{Success} \triangleq \Diamond (\text{chosen} \neq \{\})
\]

For liveness, we need to assume that there exists at least one value.

ASSUME ValueNonempty \(\triangleq\) Value \(\neq\) \{\}

TLAPS does not yet reason about enabled formulas. Therefore, we must omit all proofs that involve enabled formulas. To perform as much of the proof as possible, as much as possible we restrict the use of an enabled expression to a step asserting that it equals its definition. Enabled A is true of a state s iff there is a state t such that the step s \(\rightarrow\) t satisfies A. It follows from this semantic definition that Enabled A equals the formula obtained by

1. Expanding all definitions of defined symbols in A until all primes are priming variables.
2. For each primed variable, replacing every instance of that primed variable by a new symbol (the same symbol for each primed variable).
3. Existentially quantifying over those new symbols.

**Lemma EnabledDef** \(\triangleq\)

\[
\text{TypeOK} \Rightarrow \\
\left((\text{Enabled} \langle \text{Next} \rangle_{\text{vars}}) \equiv (\text{chosen} = \{\})\right)
\]

\(\langle 1 \rangle\) DEFINE \(E \triangleq\)

\[
\exists \text{chosenp} : \\
\text{chosen} = \{\} \\
\forall v \in \text{Value} : \text{chosenp} = \{v\} \\
\neg((\text{chosenp}) = (\text{chosen}))
\]
Here is our proof that $\textsf{Livespec}$ implies $\textsf{Success}$. It uses the standard TLA proof rules. For example $\textsf{RuleWF1}$ is defined in the $\textsf{TLAPS}$ module to be the rule $\textsf{WF1}$ discussed in

\[ \textsf{AUTHOR} = "\text{Leslie Lamport}", \]
\[ \textsf{TITLE} = "\text{The Temporal Logic of Actions}", \]
\[ \textsf{JOURNAL} = \textsf{toplas}, \]
\[ \textsf{volume} = 16, \]
\[ \textsf{number} = 3, \]
\[ \textsf{YEAR} = 1994, \]
\[ \textsf{month} = \text{may}, \]
\[ \textsf{PAGES} = "872--923" \]

$\textsf{PTL}$ stands for propositional temporal logic reasoning. We expect that, when $\textsf{TLAPS}$ handles temporal reasoning, it will use a decision procedure for $\textsf{PTL}$.

**THEOREM** $\textsf{LiveSpec} \Rightarrow \textsf{Success}$

1. $\Box \text{Inv} \land \Box[\text{Next}]_{\text{vars}} \land \text{WF}_{\text{vars}}(\text{Next}) \Rightarrow (\text{chosen} = \{\} \Rightarrow \text{chosen} \neq \{\})$

2. $\text{DEFINE } P \triangleq \text{chosen} = \{\} \quad Q \triangleq \text{chosen} \neq \{\}$

3. $\text{SUFFICES } \Box[\text{Next}]_{\text{vars}} \land \text{WF}_{\text{vars}}(\text{Next}) \Rightarrow ((\text{Inv} \land P) \Rightarrow Q)$

4. $\text{BY } \textsf{PTL}$

5. $\text{(Inv} \land P) \land [\text{Next}]_{\text{vars}} \Rightarrow ((\text{Inv} \land P') \lor Q')$

6. $\text{BY } \textsf{InductiveInvariance}$

7. $\text{(Inv} \land P) \land (\text{Next})_{\text{vars}} \Rightarrow Q'$

8. $\text{BY } \textsf{Def Inv, Next, vars}$

9. $\text{(Inv} \land P) \Rightarrow \text{ENABLED} (\text{Next})_{\text{vars}}$

10. $\text{BY } \textsf{EnabledDef Def Inv}$

11. $\text{HIDE } \textsf{Def P, Q}$

12. $\text{QED}$

BY (2)2, (2)3, (2)4, $\textsf{PTL}$

13. $\text{(chosen} = \{\} \Rightarrow \text{chosen} \neq \{\}) \Rightarrow ((\text{chosen} = \{\}) \Rightarrow \Diamond (\text{chosen} \neq \{\}))$

14. $\text{BY } \textsf{PTL}$

15. $\text{QED}$

BY $\textsf{Indvariance, (1)1, (1)2, PTL Def LiveSpec, Spec, Init, Success}$

---

The following theorem is used in the refinement proof in module $\textsf{VoteProof}$. 

4
THEOREM \textit{LiveSpecEquals} \triangleq \\
\textit{LiveSpec} \equiv \textit{Spec} \land (\Box \Diamond (\text{Next})_{\text{vars}} \lor \Box \Diamond (\text{chosen} \neq \{\})) \\
(1)1. \land \textit{Spec} \equiv \textit{Spec} \land \Box \textit{TypeOK} \\
\land \textit{LiveSpec} \equiv \textit{LiveSpec} \land \Box \textit{TypeOK} \\
\text{by Invariance, PTL def LiveSpec, Inv} \\
(1)2. (\text{chosen} \neq \{\}) \equiv \neg(\text{chosen} = \{\}) \\
\text{OBlivial} \\
(1)3. \Box \textit{TypeOK} \Rightarrow ((\Box \Diamond \neg \text{ENABLED} (\text{Next})_{\text{vars}}) \equiv \Box \Diamond (\text{chosen} \neq \{\})) \\
\text{by (1)2, EnabledDef, PTL} \\
(1)4. \text{QED} \\
\text{by (1)1, (1)3, PTL def LiveSpec}

\* Modification History
\* Last modified Mon May 11 18:36:27 CEST 2020 by merz
\* Last modified Mon Aug 18 15:00:45 CEST 2014 by tomar
\* Last modified Mon Aug 18 14:58:57 CEST 2014 by tomar
\* Last modified Tue Feb 14 13:35:49 PST 2012 by lamport
\* Last modified Mon Feb 07 14:46:59 PST 2011 by lamport