MODULE *GeneralDieHard*

In the movie Die Hard 3, the heros must obtain exactly 4 gallons of water using a 5 gallon jug, a 3 gallon jug, and a water faucet. Our goal: to get *TLC* to solve the problem for us.

First, we write a spec that describes all allowable behaviors of our heros.

EXTENDS *Integers*

This statement imports the definitions of the ordinary operators on integers, such as $+$.

This statement defines the sizes of the jugs as well as the target amount of water to defuse the bomb.

CONSTANT *BigJug*, The capacity of the bigger jug
SmallJug, The capacity of the smaller jug
Goal The target amount of water

We next declare the specification's variables.

VARIABLES *big*, The number of gallons of water in the 5 gallon jug.
small The number of gallons of water in the 3 gallon jug.

Now we define of the initial predicate, that specifies the initial values of the variables. I like to name this predicate *Init*, but the name doesn't matter.

Note: TLA+ uses the convention that a list of formulas bulleted by $\land$ or $\lor$ denotes the conjunction or disjunction of those formulas. Indentation of subitems is significant, allowing one to eliminate lots of parentheses. This makes a large formula much easier to read. However, it does mean that you have to be careful with your indentation.

$$Init \triangleq \begin{array}{l} \land\ big \quad\ = 0 \\ \land\ small = 0 \end{array}$$

Now we define the actions that our hero can perform. There are three things they can do:

- Pour water from the faucet into a jug.
- Pour water from a jug onto the ground.
- Pour water from one jug into another

We now consider the first two. Since the jugs are not calibrated, partially filling or partially emptying a jug accomplishes nothing. So, the first two possibilities yield the following four possible actions.

$$FillSmall \triangleq \begin{array}{l} \land\ small' = SmallJug \\ \land\ big' \quad\ = big \end{array}$$

$$FillBig \triangleq \begin{array}{l} \land\ big' \quad\ = BigJug \\ \land\ small' = small \end{array}$$

$$EmptySmall \triangleq \begin{array}{l} \land\ small' = 0 \\ \land\ big' \quad\ = big \end{array}$$

$$EmptyBig \triangleq \begin{array}{l} \land\ big' \quad\ = 0 \\ \land\ small' = small \end{array}$$

We now consider pouring water from one jug into another. Again, since the jugs are not callibrated, when pouring from jug A to jug B, it makes sense only to either fill B or empty A. And there's no point in emptying A if this will cause B to overflow, since that could be accomplished by the two actions of first filling B and then emptying A. So, pouring water from A to B leaves B with the lesser of (i) the water contained in both jugs and (ii) the volume of B. To express this mathematically, we first define $Min(m, n)$ to equal the minimum of the numbers m and n.

$$Min(m, n) \triangleq \text{IF } m < n \text{ THEN } m \text{ ELSE } n$$

Now we define the last two pouring actions. From the observation above, these definitions should be clear.

$$\begin{array}{l} SmallToBig \triangleq \text{LET } poured \triangleq Min(big + small, BigJug) - big \\ \qquad\qquad\qquad\qquad\qquad \text{The amount of water poured.} \\ \qquad\qquad\qquad\quad \text{IN } \land big' = big + poured \\ \qquad\qquad\qquad\qquad\quad \land small' = small - poured \end{array}$$

$$\begin{array}{l} BigToSmall \triangleq \\ \quad \text{LET } poured \triangleq Min(big + small, SmallJug) - small \\ \quad \text{IN } \land big' = big - poured \\ \qquad\quad \land small' = small + poured \end{array}$$

We define the next-state relation, which I like to call *Next*. A *Next* step is a step of one of the six actions defined above. Hence, *Next* is the disjunction of those actions.

$$\begin{array}{ll} Next \triangleq & \lor FillSmall \\ & \lor FillBig \\ & \lor EmptySmall \\ & \lor EmptyBig \\ & \lor SmallToBig \\ & \lor BigToSmall \end{array}$$

We now define *TypeOK* to be the type invariant, asserting that the value of each variable is an element of the appropriate set. A type invariant like this is not part of the specification, but it's generally a good idea to include it because it helps the reader understand the spec. Moreover, having *TLC* check that it is an invariant of the spec catches errors that, in a typed language, are caught by type checking.

$$\begin{array}{ll} TypeOK \triangleq & \land small \in 0 \,..\, SmallJug \\ & \land big \in 0 \,..\, BigJug \end{array}$$

Remember that our heros must measure out 4 gallons of water. Obviously, those 4 gallons must be in the 5 gallon jug. So, they have solved their problem when they reach a state with $big = 4$. We find a solution by having *TLC* check if big # 4 is an invariant, which will cause it to print out an "error trace" consisting of a behavior ending in a states where big # 4 is false–that is, ending in a state in which $big = 4$. Such a behavior is the desired solution. (Because *TLC* uses a breadth-first search, it will find the shortest solution.)