MODULE *RiverCrossing*

This specification describes a history of the universe where the rules of the "Wolf, goat, and cabbage" puzzle are enforced. As taken from Wikipedia (https://*en.wikipedia.org*/wiki/Wolf,_*goat_and_cabbage_problem*):

> Once upon a time a farmer went to a market and purchased a wolf, a goat, and a cabbage. On his way home, the farmer came to the bank of a river and rented a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases: the wolf, the goat, or the cabbage.
>
> If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage.
>
> The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact. How did he do it?

This specification includes the following invariants:

- *TypeOK*: Type invariant that ensures the river banks are represented by the actors on them.
- *NothingEatenNow*: Invariant that ensures the wolf doesn't eat the goat and the goat doesn't eat the cabbage.
- *SolutionInvariant*: An invariant that we know does not hold, because this puzzle has a solution. By adding it as an invariant to a model, TLC will provide an error trace that provides a solution to the puzzle. Set the number of TLC worker threads to 1 to see the shortest solution.

EXTENDS TLC Pull in *PrintT* for debugging

State

The riverbanks the farmer starts from and finishes at.

VARIABLES $riverbankstart$, $riverbankfinish$

All the actors in this puzzle.

$Actors \triangleq \{\text{"farmer"}, \text{"wolf"}, \text{"goat"}, \text{"cabbage"}\}$

Riverbanks are represented by the actors on them. The boat, though critical to the human perception of how the farmer transports the items, does not play any role in the specification itself and is therefore omitted.

$$TypeOK \triangleq \begin{array}{l} \land\ riverbankstart \subseteq Actors \\ \land\ riverbankfinish \subseteq Actors \end{array}$$

Operations

This was useful for debugging purposes as I wrote the initial specification of this system. Include as the last conjunct in an expression to minimize the number of things printed.

$PrintVariables \triangleq PrintT(\langle \text{"Start was"}, riverbankstart, \text{" and is now "}, riverbankstart',$

"Finish was", $riverbankfinish$, " and is now ", $riverbankfinish'\rangle)$

Items are only safe when the wolf isn't left alone with the goat and the goat isn't left alone with the cabbage.

$$\begin{aligned} ItemsAreSafe(riverbank) \triangleq\ & \land riverbank \neq \{\text{"wolf"}, \text{"goat"}\} \\ & \land riverbank \neq \{\text{"goat"}, \text{"cabbage"}\} \end{aligned}$$

Ensure things don't get eaten on either bank, either in current state or next (primed). Use *NothingEatenNow* in a *TLC* model to ensure this safety property holds for this specification.

$$\begin{aligned} NothingEatenNow &\triangleq ItemsAreSafe(riverbankstart) \land ItemsAreSafe(riverbankfinish) \\ NothingEatenNext &\triangleq ItemsAreSafe(riverbankstart') \land ItemsAreSafe(riverbankfinish') \\ NothingGetsEaten &\triangleq NothingEatenNow \land NothingEatenNext \end{aligned}$$

Move an item from one bank to the other, maintaining safety.

$MoveItem(item, start, finish) \triangleq$

The farmer finds himself on a given *start* riverbank

$\land$ "farmer" $\in start$

The item also finds itself on a given *start* riverbank

$\land item \in start$

He leaves *start* with himself and one *item*

$\land start' = start \setminus \{item, \text{"farmer"}\}$

He immediately takes that *item* and himself to *finish* riverbank

$\land finish' = finish \cup \{item, \text{"farmer"}\}$

Everything survives or we don't do it

$\land NothingGetsEaten$

The farmer may travel to another bank without taking anything with him.

$Go(start, finish) \triangleq$

$\land$ "farmer" $\in start$
$\land start' = start \setminus \{\text{"farmer"}\}$
$\land finish' = finish \cup \{\text{"farmer"}\}$

If he does that, we still need to ensure nothing gets eaten.

$GoEmptyHanded \triangleq$

$\land \lor Go(riverbankstart, riverbankfinish)$
$\lor Go(riverbankfinish, riverbankstart)$
$\land NothingGetsEaten$

Spec

We begin with all items on *riverbankstart* and nothing on *riverbankfinish*.

$$\begin{aligned} Init \triangleq\ & \land riverbankstart = Actors \\ & \land riverbankfinish = \{\} \end{aligned}$$

We can move items from one bank to the other, maintaining the safety invariant that nothing gets eaten.

$$
\begin{aligned}
Next \triangleq\ & \lor \exists\, item \in riverbankstart \setminus \{\text{“farmer”}\} : \\
& \qquad MoveItem(item,\ riverbankstart,\ riverbankfinish) \\
& \lor \exists\, item \in riverbankfinish \setminus \{\text{“farmer”}\} : \\
& \qquad MoveItem(item,\ riverbankfinish,\ riverbankstart) \\
& \lor GoEmptyHanded
\end{aligned}
$$

Temporal formula that is the specification of the system. If true, then both the *Init* state is valid and the *Next* state is always valid for every step in every behavior following the initial state. This includes behaviors that contain stuttering steps, or steps in the history of the universe wherein neither *riverbankstart* nor *riverbankfinish* change at all.

$$Spec \triangleq Init \land \Box[Next]_{\langle riverbankstart,\ riverbankfinish \rangle}$$

See a solution to this puzzle by setting this as an invariant in the model.

$$SolutionInvariant \triangleq riverbankfinish \neq Actors$$

---

* Modification History
* Last modified *Fri Oct* 25 09:51:59 *EDT* 2019 by *daniel*
* Last modified *Thu Nov* 16 15:56:03 *EST* 2017 by *dgregoire*
* Created *Tue Nov* 14 15:18:30 *EST* 2017 by *dgregoire*