MODULE *TowersOfHanoi*

```
***************************************************************************
 TOWERS OF HANOI is a classical Puzzle Game. It consists of three Rods on Top of which
 Disks with various diameters can be stacked. In the beginning all disks are stacked with their
 order having decreasing diameter from bottom to top. The Puzzles idea is to move that stack,
 persisting the order to the far right rod.

     =          |          |
    ___        |          |
   _____        |          |
  _______        |          |
     |          |          |

 Number of moves requied is 2^n − 1, where n is the number of disks

 Legal Moves :
  − Move one Disk at a time
  − ∀ Move : take upper disk from one stack, place it on top of another stack
  − Disks can not be placed on top of a smaller disk
***************************************************************************
EXTENDS Naturals, Sequences, TLC
```

```
FlattenSeq(seqs) ≜
   From TLA+ CommunityModules SequencesExt
  IF Len(seqs) = 0 THEN seqs ELSE
    LET flatten[i ∈ 1 .. Len(seqs)] ≜
          IF i = 1 THEN seqs[i] ELSE flatten[i − 1] ∘ seqs[i]
    IN  flatten[Len(seqs)]
```

```
CONSTANT NumberOfDisks
ASSUME NumberOfDisks ∈ Nat

CorrectTower[disk ∈ 1 .. NumberOfDisks] ≜ disk
InitialPuzzle[tower ∈ 1 .. 3] ≜
    IF tower = 1
         THEN CorrectTower
     ELSE ⟨⟩

 --algorithm towersOfHanoi{
     built from TLA+ solution i already finished
   variables
       towers = InitialPuzzle,
       towersDomain = DOMAIN towers ;

   define {
       TopElement(tower) ≜ Head(tower)
```

```
        $TowerWithoutTopElement(tower) \triangleq Tail(tower)$
        $TowerIsEmpty(tower) \triangleq tower = \langle\rangle$
        $TowerIsNotEmpty(tower) \triangleq tower \neq \langle\rangle$
        $TowerWithNewTopElement(top, tower) \triangleq \langle top\rangle \circ tower$
        $WouldPlaceSmallerOnLargerDisk(fromTower, toTower) \triangleq TopElement(toTower) > TopElement(fro$
        $GameRulesFollowed(fromTower, toTower) \triangleq TowerIsEmpty(toTower) \lor WouldPlaceSmallerOnLarge$
     }

    procedure $checkValidityAndMove($ $from = 1, to = 1$ )
        variables
            $fromTower = towers[from]$,
            $toTower = towers[to]$ ; {
        if ( $TowerIsNotEmpty(fromTower) \land GameRulesFollowed(fromTower, toTower)$ ) {
             call $move(from, to)$ ;
         } ;
        return ;
     }

    procedure $move($ $from = 1, to = 1$ )
        variables
            $fromTower = towers[from]$,
            $toTower = towers[to]$ ; {
        $towers := [towers$ EXCEPT
                    $![from] = TowerWithoutTopElement(fromTower)$,
                    $![to] = TowerWithNewTopElement(TopElement(fromTower), toTower)]$ ;
        return ;
     }

    {
        while (TRUE){
            with ($origin \in towersDomain$ ; $target \in towersDomain$){
                call $checkValidityAndMove(origin, target)$ ;
            }
        }
    }
 }
```

BEGIN TRANSLATION ($chksum(pcal) =$ "66abdfd5" $\land$ $chksum(tla) =$ "16bf5e86")

Procedure variable *fromTower* of procedure *checkValidityAndMove* at line 54 col 13 changed to *fromTower_*

Procedure variable *toTower* of procedure *checkValidityAndMove* at line 55 col 13 changed to *toTower_*

Parameter from of procedure *checkValidityAndMove* at line 52 col 36 changed to *from_*

Parameter to of procedure *checkValidityAndMove* at line 52 col 43 changed to *to_*

VARIABLES $towers, towersDomain, pc, stack$

define statement

$TopElement(tower) \triangleq Head(tower)$

$TowerWithoutTopElement(tower) \triangleq Tail(tower)$

$TowerIsEmpty(tower) \triangleq tower = \langle\rangle$
$TowerIsNotEmpty(tower) \triangleq tower \neq \langle\rangle$
$TowerWithNewTopElement(top, tower) \triangleq \langle top \rangle \circ tower$
$WouldPlaceSmallerOnLargerDisk(fromTower, toTower) \triangleq TopElement(toTower) > TopElement(fromTower$
$GameRulesFollowed(fromTower, toTower) \triangleq TowerIsEmpty(toTower) \lor WouldPlaceSmallerOnLargerDisk(f$

VARIABLES $from\_, to\_, fromTower\_, toTower\_, from, to, fromTower, toTower$

```
$vars \triangleq \langle towers, towersDomain, pc, stack, from\_, to\_, fromTower\_, toTower\_,$
         $from, to, fromTower, toTower \rangle$

$Init \triangleq$  Global variables
         $\land towers = InitialPuzzle$
         $\land towersDomain = \text{DOMAIN } towers$
          Procedure $checkValidityAndMove$
         $\land from\_ = 1$
         $\land to\_ = 1$
         $\land fromTower\_ = towers[from\_]$
         $\land toTower\_ = towers[to\_]$
          Procedure move
         $\land from = 1$
         $\land to = 1$
         $\land fromTower = towers[from]$
         $\land toTower = towers[to]$
         $\land stack = \langle\rangle$
         $\land pc = \text{"Lbl\_4"}$

$Lbl\_1 \triangleq$ $\land pc = \text{"Lbl\_1"}$
          $\land$ IF $TowerIsNotEmpty(fromTower\_) \land GameRulesFollowed(fromTower\_, toTower\_)$
                THEN $\land$ $\land from' = from\_$
                          $\land stack' = \langle[procedure \mapsto \text{"move"},$
                                      $pc \mapsto \text{"Lbl\_2"},$
                                      $fromTower \mapsto fromTower,$
                                      $toTower \mapsto toTower,$
                                      $from \mapsto from,$
                                      $to \mapsto to]\rangle$
                                      $\circ\ stack$
                          $\land to' = to\_$
                       $\land fromTower' = towers[from']$
                       $\land toTower' = towers[to']$
                       $\land pc' = \text{"Lbl\_3"}$
                ELSE $\land pc' = \text{"Lbl\_2"}$
                       $\land$ UNCHANGED $\langle stack, from, to, fromTower, toTower \rangle$
          $\land$ UNCHANGED $\langle towers, towersDomain, from\_, to\_, fromTower\_,$
                           $toTower\_ \rangle$
```

$$
\begin{array}{lll}
Lbl\_2 & \triangleq & \land\ pc = \text{“Lbl\_2”} \\
 & & \land\ pc' = Head(stack).pc \\
 & & \land\ fromTower\_' = Head(stack).fromTower\_ \\
 & & \land\ toTower\_' = Head(stack).toTower\_ \\
 & & \land\ from\_' = Head(stack).from\_ \\
 & & \land\ to\_' = Head(stack).to\_ \\
 & & \land\ stack' = Tail(stack) \\
 & & \land\ \text{UNCHANGED}\ \langle towers,\ towersDomain,\ from,\ to,\ fromTower,\ toTower \rangle
\end{array}
$$

$$checkValidityAndMove \triangleq Lbl\_1 \lor Lbl\_2$$

$$
\begin{array}{lll}
Lbl\_3 & \triangleq & \land\ pc = \text{“Lbl\_3”} \\
 & & \land\ towers' = [towers\ \text{EXCEPT} \\
 & & \qquad\qquad\quad ![from] = TowerWithoutTopElement(fromTower), \\
 & & \qquad\qquad\quad ![to] = TowerWithNewTopElement(TopElement(fromTower),\ toTower)] \\
 & & \land\ pc' = Head(stack).pc \\
 & & \land\ fromTower' = Head(stack).fromTower \\
 & & \land\ toTower' = Head(stack).toTower \\
 & & \land\ from' = Head(stack).from \\
 & & \land\ to' = Head(stack).to \\
 & & \land\ stack' = Tail(stack) \\
 & & \land\ \text{UNCHANGED}\ \langle towersDomain,\ from\_,\ to\_,\ fromTower\_,\ toTower\_ \rangle
\end{array}
$$

$$move \triangleq Lbl\_3$$

$$
\begin{array}{lll}
Lbl\_4 & \triangleq & \land\ pc = \text{“Lbl\_4”} \\
 & & \land\ \exists\, origin \in towersDomain : \\
 & & \quad \exists\, target \in towersDomain : \\
 & & \qquad \land\ \land\ from\_' = origin \\
 & & \qquad\quad \land\ stack' = \langle [\begin{array}{lll} procedure & \mapsto & \text{“checkValidityAndMove”}, \\ pc & \mapsto & \text{“Lbl\_4”}, \\ fromTower\_ & \mapsto & fromTower\_, \\ toTower\_ & \mapsto & toTower\_, \\ from\_ & \mapsto & from\_, \\ to\_ & \mapsto & to\_] \rangle \end{array} \\
 & & \qquad\qquad\qquad\qquad \circ\ stack \\
 & & \qquad\quad \land\ to\_' = target \\
 & & \qquad \land\ fromTower\_' = towers[from\_'] \\
 & & \qquad \land\ toTower\_' = towers[to\_'] \\
 & & \qquad \land\ pc' = \text{“Lbl\_1”} \\
 & & \land\ \text{UNCHANGED}\ \langle towers,\ towersDomain,\ from,\ to,\ fromTower,\ toTower \rangle
\end{array}
$$

$$Next \triangleq checkValidityAndMove \lor move \lor Lbl\_4$$

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

END TRANSLATION

```
$IsSorted(tower) \triangleq$
  $\forall\, i, j \in 1 \,..\, Len(tower) :$
    $i < j \Rightarrow tower[i] \leq tower[j]$

$OnlyContainsAllowedNumberOfDisks \triangleq$
    $Len(FlattenSeq(towers)) = 5$

$TypeOK \triangleq$
    $\land\; \forall\, tower \in towersDomain :$
          LET $towerToCheck \triangleq towers[tower]$
          IN  $\land\, IsSorted(towerToCheck)$
              $\land\, Len(towerToCheck) \leq NumberOfDisks$
    $\land\, OnlyContainsAllowedNumberOfDisks$

THEOREM $Spec \Rightarrow \Box\, TypeOK$

$InvariantOrElseFinished \triangleq$
    $towers[3] \neq CorrectTower$
```

* Modification History
* Last modified *Thu Mar* 21 09:13:36 *CET* 2024 by *jeujeus*
* Created *Wed Mar* 20 09:29:51 *CET* 2024 by *jeujeus*