–

MODULE *LogSync*

EXTENDS TLC, $Naturals$, $Sequences$

CONSTANTS Key, $WatchClient$, nil

state is in-memory data
db is the same data but on the *db*
watch_chan is the receive channel for client

VARIABLES pc, $current\_key$, db,
    $state$, $state\_seq$, $next\_log$, $next\_seq$, $wait\_list$,
    $watch\_pc$, $watch\_keys$, $watch\_chan$, $watch\_seq$,
    $watch\_log\_index$, $watch\_state$, $watch\_local\_key$,
    $num\_client\_restart$, $num\_main\_restart$, num_delete_state

$main\_vars \triangleq \langle pc, current\_key, db \rangle$

$watch\_vars \triangleq \langle watch\_pc, watch\_keys, watch\_chan,$
    $watch\_seq, watch\_log\_index, watch\_state, watch\_local\_key \rangle$

$server\_vars \triangleq \langle state, state\_seq, next\_log, next\_seq, wait\_list \rangle$

$aux\_vars \triangleq \langle num\_client\_restart, num\_main\_restart, num\_delete\_state \rangle$

$vars \triangleq \langle main\_vars, server\_vars, watch\_vars, aux\_vars \rangle$

$max\_log\_size \triangleq 3$

$max\_client\_restart \triangleq 1$
$max\_main\_restart \triangleq 1$
$max\_delete\_state \triangleq 1$

$Status \triangleq \{\text{"Running"}, \text{"Completed"}, \text{"Gone"}\}$

$LogEntry \triangleq 20 \mathrel{..} 30$

$SeqMaxLen(S, n) \triangleq \textsc{union}\ \{[1 \mathrel{..} m \to S] : m \in 0 \mathrel{..} n\}$

$Info \triangleq [logs : Seq(LogEntry), status : Status]$

$NullInfo \triangleq Info \cup \{nil\}$

$NullKey \triangleq Key \cup \{nil\}$

$NullLogEntry \triangleq LogEntry \cup \{nil\}$

$Event \triangleq [$
    $type : \{\text{"AddLog"}, \text{"Finished"}, \text{"JobGone"}\},$
    $key : Key, line : NullLogEntry]$

$NullEvent \triangleq Event \cup \{nil\}$

$Channel \triangleq [status : \{\text{“Empty”}, \text{“Ready”}, \text{“Consumed”}\}, data : NullEvent]$

$StateSeq \triangleq 100 \,..\, 120$

$WatchState \triangleq \{\text{“Init”}, \text{“AddToWaitList”}, \text{“WaitOnChan”}, \text{“UpdateDB”}\}$

$TypeOK \triangleq$

$$
\begin{aligned}
&\land\ pc \in \{\text{“Init”}, \text{“PushJob”}\}\\
&\land\ current\_key \in NullKey\\
&\land\ db \in [Key \rightarrow NullInfo]\\[1ex]
&\land\ state \in [Key \rightarrow NullInfo]\\
&\land\ state\_seq \in [Key \rightarrow StateSeq]\\
&\land\ next\_log \in LogEntry\\
&\land\ next\_seq \in StateSeq\\
&\land\ wait\_list \in [Key \rightarrow \textsc{subset}\ WatchClient]\\[1ex]
&\land\ watch\_pc \in [WatchClient \rightarrow WatchState]\\
&\land\ watch\_keys \in [WatchClient \rightarrow \textsc{subset}\ Key]\\
&\land\ watch\_chan \in [WatchClient \rightarrow Channel]\\
&\land\ watch\_seq \in [WatchClient \rightarrow [Key \rightarrow StateSeq]]\\
&\land\ watch\_log\_index \in [WatchClient \rightarrow [Key \rightarrow Nat]]\\
&\land\ watch\_state \in [WatchClient \rightarrow [Key \rightarrow NullInfo]]\\
&\land\ watch\_local\_key \in [WatchClient \rightarrow NullKey]\\[1ex]
&\land\ num\_client\_restart \in 0 \,..\, max\_client\_restart\\
&\land\ num\_main\_restart \in 0 \,..\, max\_main\_restart\\
&\land\ num\_delete\_state \in 0 \,..\, max\_delete\_state
\end{aligned}
$$

$consumed\_chan \triangleq [status \mapsto \text{“Consumed”}, data \mapsto nil]$

$Init \triangleq$

$$
\begin{aligned}
&\land pc = \text{“Init”}\\
&\land current\_key = nil\\
&\land db = [k \in Key \mapsto nil]\\[1ex]
&\land state = [k \in Key \mapsto nil]\\
&\land state\_seq = [k \in Key \mapsto 100]\\
&\land next\_log = 20\\
&\land wait\_list = [k \in Key \mapsto \{\}]\\
&\land next\_seq = 100\\[1ex]
&\land watch\_pc = [c \in WatchClient \mapsto \text{“Init”}]\\
&\land watch\_keys = [c \in WatchClient \mapsto \{\}]\\
&\land watch\_chan = [c \in WatchClient \mapsto consumed\_chan]\\
&\land watch\_seq = [c \in WatchClient \mapsto [k \in Key \mapsto 100]]
\end{aligned}
$$

```
    $\land\ watch\_log\_index = [c \in WatchClient \mapsto [k \in Key \mapsto 0]]$
    $\land\ watch\_state = [c \in WatchClient \mapsto [k \in Key \mapsto nil]]$
    $\land\ watch\_local\_key = [c \in WatchClient \mapsto nil]$

    $\land\ num\_client\_restart = 0$
    $\land\ num\_main\_restart = 0$
    $\land\ num\_delete\_state\ \ = 0$

$newJob \triangleq [logs \mapsto \langle\rangle,\ status \mapsto \text{“Running”}]$

$AddDBJob(k) \triangleq$
    $\land\ pc = \text{“Init”}$
    $\land\ db[k] = nil$
    $\land\ pc' = \text{“PushJob”}$
    $\land\ current\_key' = k$
    $\land\ db' = [db \text{ EXCEPT } ![k] = newJob]$
    $\land\ \text{UNCHANGED } server\_vars$
    $\land\ \text{UNCHANGED } watch\_vars$
    $\land\ \text{UNCHANGED } aux\_vars$

$updateStateSeq(k) \triangleq$
    $\land\ next\_seq' = next\_seq + 1$
    $\land\ state\_seq' = [state\_seq \text{ EXCEPT } ![k] = next\_seq']$

$PushJob \triangleq$
    $\land\ pc = \text{“PushJob”}$
    $\land\ pc' = \text{“Init”}$
    $\land\ current\_key' = nil$
    $\land\ state' = [state \text{ EXCEPT } ![current\_key] = db[current\_key]]$
    $\land\ \text{UNCHANGED } \langle next\_seq,\ state\_seq\rangle$
    $\land\ \text{UNCHANGED } wait\_list$
    $\land\ \text{UNCHANGED } db$
    $\land\ \text{UNCHANGED } next\_log$
    $\land\ \text{UNCHANGED } watch\_vars$
    $\land\ \text{UNCHANGED } aux\_vars$

$canPushKeyToClient(k,\ c,\ old\_watch\_ch) \triangleq$
    $\land\ old\_watch\_ch[c].status = \text{“Empty”}$
    $\land\ c \in wait\_list'[k]$
    $\land\ watch\_seq[c][k] < state\_seq'[k]$

$pushToClientChan(k,\ c,\ old\_watch\_ch) \triangleq$
    LET
        $last\_index \triangleq watch\_log\_index[c][k]$
    $state\_index \triangleq Len(state'[k].logs)$

    $new\_line \triangleq state'[k].logs[last\_index + 1]$

    $add\_event \triangleq [$
        $type \mapsto$ "AddLog",
        $key \mapsto k,$
        $line \mapsto new\_line]$

    $finished\_or\_gone \triangleq$
        IF $state'[k].status =$ "Gone"
            THEN "JobGone"
            ELSE "Finished"

    $finish\_event \triangleq [$
        $type \mapsto finished\_or\_gone,$
        $key \mapsto k,$
        $line \mapsto nil]$

    $is\_running \triangleq state'[k].status =$ "Running"

    $add\_log\_cond \triangleq is\_running \lor last\_index < state\_index$

    $update\_seq\_cond \triangleq$
        IF $last\_index = state\_index$
            THEN TRUE
            ELSE IF $last\_index + 1 = state\_index \land is\_running$
                THEN TRUE
                ELSE FALSE

    $new\_event \triangleq$
        IF $add\_log\_cond$
            THEN $add\_event$
            ELSE $finish\_event$

    $new\_state \triangleq [status \mapsto$ "Ready", $data \mapsto new\_event]$
IN
    $\land\ watch\_chan' = [old\_watch\_ch$ EXCEPT $![c] = new\_state]$
    $\land\ watch\_log\_index' = [watch\_log\_index$ EXCEPT $![c][k] = last\_index + 1]$
    $\land$ IF $update\_seq\_cond$
        THEN $watch\_seq' = [watch\_seq$ EXCEPT $![c][k] = state\_seq'[k]]$
        ELSE UNCHANGED $watch\_seq$


$pushToClientOrDoNothing(c,\ old\_watch\_ch) \triangleq$
LET
    $doNothing \triangleq$
        $\land\ \forall k \in Key : \neg canPushKeyToClient(k,\ c,\ old\_watch\_ch)$
```

```
            $\land$ $watch\_chan' = old\_watch\_ch$
            $\land$ UNCHANGED $\langle watch\_seq, watch\_log\_index \rangle$
    IN
    $\lor$ $\exists k \in Key$ :
        $\land$ $canPushKeyToClient(k, c, old\_watch\_ch)$
        $\land$ $pushToClientChan(k, c, old\_watch\_ch)$
    $\lor$ $doNothing$

$pushKeyOrDoNothing(k) \triangleq$
    LET
        $doPush \triangleq$
            $\exists c \in WatchClient$ :
                $\land$ $canPushKeyToClient(k, c, watch\_chan)$
                $\land$ $pushToClientChan(k, c, watch\_chan)$

        $doNothing \triangleq$
            $\land$ $\forall c \in WatchClient : \neg canPushKeyToClient(k, c, watch\_chan)$
            $\land$ UNCHANGED $\langle watch\_chan, watch\_seq, watch\_log\_index \rangle$
    IN
        $doPush \lor doNothing$

$ProduceLog(k) \triangleq$
    $\land$ $state[k] \neq nil$
    $\land$ $state[k].status =$ "Running"
    $\land$ $Len(state[k].logs) < max\_log\_size$

    $\land$ $next\_log' = next\_log + 1$
    $\land$ $state' = [state$ EXCEPT $![k].logs = Append(@, next\_log')]$

    $\land$ $updateStateSeq(k)$

    $\land$ UNCHANGED $wait\_list$
    $\land$ $pushKeyOrDoNothing(k)$

    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $\langle watch\_pc, watch\_keys, watch\_state, watch\_local\_key \rangle$
    $\land$ UNCHANGED $aux\_vars$

$FinishJob(k) \triangleq$
    $\land$ $state[k] \neq nil$
    $\land$ $state[k].status =$ "Running"
    $\land$ $state' = [state$ EXCEPT $![k].status =$ "Completed"$]$
    $\land$ $updateStateSeq(k)$

    $\land$ UNCHANGED $wait\_list$
    $\land$ $pushKeyOrDoNothing(k)$
    $\land$ UNCHANGED $next\_log$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $\langle watch\_pc, watch\_keys, watch\_state, watch\_local\_key \rangle$
    $\land$ UNCHANGED $aux\_vars$

$new\_chan \triangleq [status \mapsto$ "Empty", $data \mapsto nil]$

$NewWatchChan(c) \triangleq$
    LET
        $new\_watch\_ch \triangleq [watch\_chan$ EXCEPT $![c] = new\_chan]$
    IN
        $\land watch\_pc[c] =$ "Init"
        $\land watch\_pc' = [watch\_pc$ EXCEPT $![c] =$ "WaitOnChan"$]$

        $\land$ UNCHANGED $server\_vars$
        $\land pushToClientOrDoNothing(c, new\_watch\_ch)$

        $\land$ UNCHANGED $\langle watch\_keys, watch\_state, watch\_local\_key \rangle$
        $\land$ UNCHANGED $main\_vars$
        $\land$ UNCHANGED $aux\_vars$

$active\_keys \triangleq$
    LET
        $db\_set \triangleq \{k \in Key : db[k] \neq nil \land db[k].status =$ "Running"$\}$
    IN
        $db\_set \setminus \{current\_key\}$

$UpdateWatchKeys(c) \triangleq$
    $\land watch\_keys[c] \neq active\_keys$
    $\land watch\_keys' = [watch\_keys$ EXCEPT $![c] = active\_keys]$
    $\land$ UNCHANGED $\langle watch\_pc, watch\_chan, watch\_seq, watch\_log\_index, watch\_state \rangle$
    $\land$ UNCHANGED $watch\_local\_key$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $server\_vars$
    $\land$ UNCHANGED $aux\_vars$

$updateServerWaitList(c) \triangleq$
    LET
        $old\_set(k) \triangleq wait\_list[k]$
        $new\_set(k) \triangleq$
            IF $k \in watch\_keys[c]$
                THEN $old\_set(k) \cup \{c\}$
                ELSE $old\_set(k) \setminus \{c\}$
    IN
        $wait\_list' = [k \in Key \mapsto new\_set(k)]$
```

```
$serverWatchClientKeys(c) \triangleq \{k \in Key : c \in wait\_list[k]\}$

$createPlaceHolderStateForWaitList \triangleq$
  LET
      $in\_wait\_list(k) \triangleq wait\_list'[k] \neq \{\}$

      $keysWithNilState \triangleq$
          $\{k \in Key : in\_wait\_list(k) \land state[k] = nil\}$

      $new\_state\_fn(k) \triangleq$
          IF $k \in keysWithNilState$
              THEN $[logs \mapsto \langle\rangle, status \mapsto \text{"Gone"}]$
              ELSE $state[k]$

      $new\_seq\_fn(k) \triangleq$
          IF $k \in keysWithNilState$
              THEN $next\_seq'$
              ELSE $state\_seq[k]$

      $update\_state \triangleq$
          $\land next\_seq' = next\_seq + 1$
          $\land state' = [k \in Key \mapsto new\_state\_fn(k)]$
          $\land state\_seq' = [k \in Key \mapsto new\_seq\_fn(k)]$

      $do\_nothing \triangleq$
          UNCHANGED $\langle state, next\_seq, state\_seq\rangle$
  IN
      IF $keysWithNilState \neq \{\}$
          THEN $update\_state$
          ELSE $do\_nothing$


$AddToWaitList(c) \triangleq$
    $\land watch\_keys[c] \neq serverWatchClientKeys(c)$
    $\land updateServerWaitList(c)$

    $\land createPlaceHolderStateForWaitList$
    $\land pushToClientOrDoNothing(c, watch\_chan)$

    $\land$ UNCHANGED $\langle watch\_pc, watch\_keys, watch\_state, watch\_local\_key\rangle$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $next\_log$
    $\land$ UNCHANGED $aux\_vars$


$updateStateFromChan(c) \triangleq$
  LET
      $k \triangleq watch\_chan[c].data.key$
      $type \triangleq watch\_chan[c].data.type$
```

```
        $log\_line \triangleq watch\_chan[c].data.line$

        $old\_state \triangleq watch\_state[c][k]$

        $old\_logs \triangleq$
            IF $old\_state = nil$
                THEN $\langle\rangle$
                ELSE $old\_state.logs$

        $new\_state \triangleq$
            $[logs \mapsto Append(old\_logs, log\_line), status \mapsto$ "Running"$]$

        $do\_add\_log \triangleq$
            $\land\ watch\_state' = [$
                    $watch\_state$ EXCEPT $![c][k] = new\_state]$
            $\land$ UNCHANGED $watch\_local\_key$
            $\land\ watch\_pc' = [watch\_pc$ EXCEPT $![c] =$ "Init"$]$

        $new\_status \triangleq$
            IF $type =$ "JobGone"
                THEN "Gone"
                ELSE "Completed"

        $do\_complete \triangleq$
            $\land\ watch\_state' = [$
                $watch\_state$ EXCEPT
                    $![c][k] = [logs \mapsto old\_logs, status \mapsto new\_status]]$
            $\land\ watch\_local\_key' = [watch\_local\_key$ EXCEPT $![c] = k]$
            $\land\ watch\_pc' = [watch\_pc$ EXCEPT $![c] =$ "UpdateDB"$]$
    IN
        IF $type =$ "AddLog"
            THEN $do\_add\_log$
            ELSE $do\_complete$

$ConsumeWatchChan(c) \triangleq$
    $\land\ watch\_pc[c] =$ "WaitOnChan"
    $\land\ watch\_chan[c].status =$ "Ready"

    $\land\ watch\_chan' = [$
            $watch\_chan$ EXCEPT
                $![c].status =$ "Consumed",
                $![c].data = nil]$

    $\land\ updateStateFromChan(c)$

    $\land$ UNCHANGED $\langle watch\_keys, watch\_seq, watch\_log\_index\rangle$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $server\_vars$
    $\land$ UNCHANGED $aux\_vars$

$UpdateDB(c) \triangleq$
    LET
        $k \triangleq watch\_local\_key[c]$
    IN
        $\land watch\_pc[c] = $ "UpdateDB"
        $\land watch\_pc' = [watch\_pc$ EXCEPT $![c] = $ "Init"$]$
        $\land db' = [db$ EXCEPT $![k] = watch\_state[c][k]]$
        $\land watch\_local\_key' = [watch\_local\_key$ EXCEPT $![c] = nil]$
        $\land$ UNCHANGED $\langle watch\_keys, watch\_chan, watch\_seq \rangle$
        $\land$ UNCHANGED $\langle watch\_log\_index, watch\_state \rangle$
        $\land$ UNCHANGED $server\_vars$
        $\land$ UNCHANGED $\langle pc, current\_key \rangle$
        $\land$ UNCHANGED $aux\_vars$

$ClientRestart(c) \triangleq$
    $\land num\_client\_restart < max\_client\_restart$
    $\land num\_client\_restart' = num\_client\_restart + 1$
    $\land watch\_chan' = [watch\_chan$ EXCEPT $![c] = consumed\_chan]$
    $\land watch\_keys' = [watch\_keys$ EXCEPT $![c] = \{\}]$
    $\land watch\_local\_key' = [watch\_local\_key$ EXCEPT $![c] = nil]$
    $\land watch\_log\_index' = [watch\_log\_index$ EXCEPT $![c] = [k \in Key \mapsto 0]]$
    $\land watch\_seq' = [watch\_seq$ EXCEPT $![c] = [k \in Key \mapsto 100]]$
    $\land watch\_state' = [watch\_state$ EXCEPT $![c] = [k \in Key \mapsto nil]]$
    $\land watch\_pc' = [watch\_pc$ EXCEPT $![c] = $ "Init"$]$
    $\land$ UNCHANGED $server\_vars$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $\langle num\_main\_restart, num\_delete\_state \rangle$

$MainRestart \triangleq$
    $\land num\_main\_restart < max\_main\_restart$
    $\land num\_main\_restart' = num\_main\_restart + 1$
    $\land current\_key' = nil$
    $\land pc' = $ "Init"
    $\land$ UNCHANGED $db$
    $\land$ UNCHANGED $\langle num\_client\_restart, num\_delete\_state \rangle$
    $\land$ UNCHANGED $server\_vars$
    $\land$ UNCHANGED $watch\_vars$

$DeleteRandomKeyInState(k) \triangleq$
    $\land num\_delete\_state < max\_delete\_state$
    $\land num\_delete\_state' = num\_delete\_state + 1$
    $\land\ state[k] \neq nil$

    $\land\ state' = [state \text{ EXCEPT } ![k] = nil]$
    $\land\ state\_seq' = [state\_seq \text{ EXCEPT } ![k] = 100]$
    $\land\ wait\_list' = [wait\_list \text{ EXCEPT } ![k] = \{\}]$

    $\land$ UNCHANGED $\langle next\_log, next\_seq \rangle$
    $\land$ UNCHANGED $\langle num\_client\_restart, num\_main\_restart \rangle$
    $\land$ UNCHANGED $main\_vars$
    $\land$ UNCHANGED $watch\_vars$


$statusIsFinished(st) \triangleq$
    $\lor\ st =$ "Completed"
    $\lor\ st =$ "Gone"

$TerminateCond \triangleq$
    $\land\ \forall k \in Key : db[k] \neq nil \land statusIsFinished(db[k].status)$
    $\land\ \forall k \in Key : state[k] \neq nil \Rightarrow statusIsFinished(state[k].status)$
    $\land\ \forall c \in WatchClient :$
        $\land\ watch\_pc[c] =$ "WaitOnChan"
        $\land\ watch\_keys[c] = active\_keys$
        $\land\ watch\_keys[c] = serverWatchClientKeys(c)$
        $\land\ watch\_chan[c].status =$ "Empty"

$Terminated \triangleq$
    $\land\ TerminateCond$
    $\land$ UNCHANGED $vars$


$Next \triangleq$
    $\lor\ \exists k \in Key :$
        $\lor\ AddDBJob(k)$
        $\lor\ ProduceLog(k)$
        $\lor\ FinishJob(k)$
        $\lor\ DeleteRandomKeyInState(k)$
    $\lor\ PushJob$

    $\lor\ \exists c \in WatchClient :$
        $\lor\ NewWatchChan(c)$
        $\lor\ UpdateWatchKeys(c)$
        $\lor\ AddToWaitList(c)$
        $\lor\ ConsumeWatchChan(c)$
        $\lor\ UpdateDB(c)$
        $\lor\ ClientRestart(c)$

    $\lor\ MainRestart$
    $\lor$ $\mathit{Terminated}$

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

$\mathit{FairSpec} \triangleq \mathit{Spec} \land \mathrm{WF}_{\mathit{vars}}(\mathit{Next})$

$\mathit{AlwaysTerminate} \triangleq \Diamond \mathit{TerminateCond}$

$\mathit{AllJobsMustBeFinished} \triangleq$
    $\mathit{TerminateCond} \Rightarrow$
        $\forall k \in \mathit{Key} : \mathit{db}[k] \neq \mathit{nil} \land \mathit{statusIsFinished}(\mathit{db}[k].\mathit{status})$

$\mathit{infoEqual}(\mathit{db\_val},\ \mathit{state\_val}) \triangleq$
    $\land\ \mathit{db\_val}.\mathit{status} \in \{\text{“Completed”},\ \text{“Gone”}\}$
    $\land\ \mathit{state\_val}.\mathit{status} \in \{\text{“Completed”},\ \text{“Gone”}\}$

    $\land\ \mathit{state\_val}.\mathit{status} = \text{“Completed”} \Rightarrow \mathit{db\_val}.\mathit{logs} = \mathit{state\_val}.\mathit{logs}$
    $\land\ \mathit{state\_val}.\mathit{status} = \text{“Completed”} \Rightarrow \mathit{db\_val}.\mathit{status} = \text{“Completed”}$

$\mathit{DBShouldSameAsMem} \triangleq$
    $\mathit{TerminateCond} \Rightarrow$
        $\forall k \in \mathit{Key} : \mathit{state}[k] \neq \mathit{nil} \Rightarrow \mathit{infoEqual}(\mathit{db}[k],\ \mathit{state}[k])$

$\mathit{DBShouldSameAsMemWhenNoRestart} \triangleq$
    LET
        $\mathit{cond} \triangleq$
            $\land\ \mathit{TerminateCond}$
            $\land\ \mathit{num\_main\_restart} = 0$
            $\land\ \mathit{num\_delete\_state}\ \ = 0$
    IN
        $\mathit{cond} \Rightarrow \forall k \in \mathit{Key} : \mathit{state}[k] = \mathit{db}[k] \land \mathit{db}[k].\mathit{status} = \text{“Completed”}$

$\mathit{StateAlwaysMatchWaitList} \triangleq$
    $\forall k \in \mathit{Key} :$
       $\mathit{wait\_list}[k] \neq \{\} \Rightarrow \mathit{state}[k] \neq \mathit{nil}$

$\mathit{StateAlwaysMatchSeq} \triangleq$
    $\forall k \in \mathit{Key} :$
       $\mathit{state}[k] = \mathit{nil} \Rightarrow \mathit{state\_seq}[k] = 100$

$\mathit{channelInitByClient}(c) \triangleq$
    $\land\ \mathit{watch\_chan}[c].\mathit{status} = \text{“Consumed”}$
    $\land\ \mathit{watch\_chan}[c].\mathit{data} = \mathit{nil}$
```

```
$channelInit \triangleq \forall c \in WatchClient : channelInitByClient(c)$

$channelNextByClient(c) \triangleq$
    $\lor \land watch\_chan[c].status = \text{"Empty"}$
       $\land watch\_chan'[c].status = \text{"Ready"}$
       $\land watch\_chan'[c].data \neq nil$

    $\lor \land watch\_chan[c].status = \text{"Consumed"}$
       $\land watch\_chan'[c].status = \text{"Empty"}$
       $\land watch\_chan'[c].data = nil$

    $\lor \land watch\_chan[c].status = \text{"Consumed"}$
       $\land watch\_chan'[c].status = \text{"Ready"}$
       $\land watch\_chan'[c].data \neq nil$

    $\lor \land \lor watch\_chan[c].status = \text{"Ready"}$
          $\lor watch\_chan[c].status = \text{"Empty"}$
       $\land watch\_chan'[c].status = \text{"Consumed"}$
       $\land watch\_chan'[c].data = nil$

$channelNextActions \triangleq \exists c \in WatchClient : channelNextByClient(c)$

$ChannelSpec \triangleq$
    $channelInit \land \Box[channelNextActions]_{watch\_chan}$

$Sym \triangleq Permutations(Key)$
```