```
MODULE optishopylist

EXTENDS TLC, Integers, FiniteSets, Sequences, Reals

CONSTANTS PRODUCTS, APPS, IDs, GATEAPPS
ASSUME Cardinality(APPS) > 0

PT ≜ INSTANCE PT

set ++ item ≜ set ∪ {item}
set −− item ≜ set \ {item}
```

In shoppying list, the product is in fact the identifier. Any item could have an information for how much of a product one wants to buy (not relevant in this specification).

```
ShopyItems       ≜ [id : PRODUCTS, bought : BOOLEAN ]

ADD_ACTION ≜ "add"
RM_ACTION ≜ "rm"
SET_BOUGHT_ACTION ≜ "set_bought"
REQ_SYNC_ACTION ≜ "req_sync"
RESP_SYNC_ACTION ≜ "resp_sync"
END_SYNC_ACTION ≜ "end_sync"
```

Actions is the set of all possible actions in the system.

```
Actions ≜ {
    ADD_ACTION,
    RM_ACTION,
    SET_BOUGHT_ACTION,
    REQ_SYNC_ACTION,
    RESP_SYNC_ACTION,
    END_SYNC_ACTION
}

SyncActions ≜ {REQ_SYNC_ACTION, RESP_SYNC_ACTION}
```

SyncMsgs is the set of all possible messages sent for synchronisation of shopy lists.

```
SyncMsgs ≜
    [id : IDs,
     app : APPS,
     list : SUBSET ShopyItems,
     mergedList : SUBSET ShopyItems,
     type : SyncActions]
```

Messages sent for joining the network.

```
JoinRespMsgs ≜
    [app : APPS,
     knownHosts : PT!SeqOf(APPS, Cardinality(APPS))]
```

$JoinReqMsgs \triangleq$
    $[app : APPS]$

Messages sent for notifications about new joiners in the network.

$JoinNotifMsgs \triangleq$
    $[app : APPS]$

The spec now depicts a shopping-list *app* where the server *app* manages several users and hence multiple lists of items that synch eventually.

The list contains unique items, thus we use a set.

```
  --algorithm  $OptiShopyList$

variable
    whether an app is a gate
    $isGate = [a \in APPS \mapsto a \in GATEAPPS]$,
    one shopping list for all APPS
    $shopyList = [a \in APPS \mapsto \{\}]$,
    sync shopyList requests/responses
    $syncReqQueue = [a \in APPS \mapsto \langle\rangle]$,
    $syncRespQueue = [a \in APPS \mapsto \langle\rangle]$,
    join to network requests/responses
    $joinReqQueue = [a \in APPS \mapsto \langle\rangle]$,
    $joinRespQueue = [a \in APPS \mapsto \langle\rangle]$,
    new joiner notifications
    $newJoinerNotif = [a \in APPS \mapsto \langle\rangle]$,
    set of taken IDs
    $takenIDs = \{\}$ ;

define
    A couple of helpers for shopy-list items

    $NewShopyItem(list) \triangleq$
        $[id \mapsto (\text{CHOOSE } x \in PRODUCTS : \neg\exists\, i \in list : x = i.id)$,
         $bought \mapsto \text{FALSE}]$

    $ExistingShopyItem(list) \triangleq \text{CHOOSE } x \in list : \text{TRUE}$

    $ExistingNotBoughtShopyItem(list) \triangleq \text{CHOOSE } x \in list : x.bought = \text{FALSE}$

    Helpers for Sync messages request/response.

    $NewSyncMsg(id, a, l, ml, t) \triangleq$
        $[id \mapsto id$,
         $app \mapsto a$,
         $list \mapsto l$,
         $mergedList \mapsto ml$,
         type ↦ t]

    NewSyncReqMsg(a, l, ml, t) ≜
         NewSyncMsg(
             (CHOOSE i ∈ IDs : ∀ ti ∈ takenIDs : i = ti),
             a, l, ml, t
         )

    NewSyncReq(app) ≜
         NewSyncReqMsg(app, shopyList[app], {}, REQ_SYNC_ACTION)

    NewSyncResp(app, mergeResult, id) ≜
         NewSyncMsg(id, app, shopyList[app], mergeResult, RESP_SYNC_ACTION)
```

Helpers for the decentralized network features.

```
NewJoinReqMsg(app) ≜ [app ↦ app]

NewJoinRespMsg(app, hosts) ≜
    [app ↦ app,
     knownHosts ↦ hosts]

GateApps ≜ {a ∈ APPS : isGate[a]}

NewJoinerNotifReq(app) ≜ [app ↦ app]

PickGossipFriends(app, knownApps) ≜
    LET Opposit ≜
            PT!Index(knownApps, app) + (Len(knownApps) ÷ 2) − (Len(knownApps)%2)

        PreviousIndex(i) ≜
            IF Len(knownApps) < 3
             THEN 1
             ELSE IF i = 1 THEN Len(knownApps) ELSE i − 1

        NextIndex(i) ≜
            IF Len(knownApps) < 3
             THEN Len(knownApps)
             ELSE IF i = Len(knownApps) THEN 1 ELSE i + 1
    IN  {knownApps[PreviousIndex(Opposit)], knownApps[NextIndex(Opposit)]}
```

Not used, it's an example of how we'd keep the ordering of the responsed *knownApps* sequence on joining. If it's used, it should be with small sequences to not generate enormous sets.

```
MergeKnownApps(apps1, apps2) ≜
    LET AppSeq(n) ≜ PT!SeqOf(APPS, n)

        Contains(appSeq, appItem) ≜
            Cardinality(PT!Matching(appSeq, appItem)) > 0
            $f[args \in AppSeq(Len(apps1))$
                        $\times AppSeq(Len(apps2))$
                        $\times AppSeq(Len(apps1) + Len(apps2))] \triangleq$
                LET $l1 \triangleq args[1]$
                    $l2 \triangleq args[2]$
                    $acc \triangleq args[3]$
                    $PickFromL1 \triangleq f[\langle$
                        $Tail(l1),$
                        $l2,$
                        $Append(acc, Head(l1))\rangle]$
                    $SkipOneL1 \triangleq f[\langle$
                        $Tail(l1),$
                        $l2,$
                        $acc\rangle]$
                    $PickFromL2 \triangleq f[\langle$
                        $l1,$
                        $Tail(l2),$
                        $Append(acc, Head(l2))\rangle]$
                    $SkipOneL2 \triangleq f[\langle$
                        $l1,$
                        $Tail(l2),$
                        $acc\rangle]$
                IN
                    IF $Len(l2) = 0$
                     THEN IF $Len(l1) = 0$
                            THEN $acc$
                            ELSE IF $Contains(acc, Head(l1))$ THEN $SkipOneL1$ ELSE $PickFromL1$
                     ELSE IF $Head(l1) \neq Head(l2)$
                            THEN IF $Contains(acc, Head(l2))$ THEN $SkipOneL2$ ELSE $PickFromL2$
                            ELSE IF $Contains(acc, Head(l1))$ THEN $SkipOneL1$ ELSE $PickFromL1$
        IN $f[\langle apps1, apps2, \langle\rangle\rangle]$
end define ;

macro $Notify(gossipFriends, newJoiner)$
begin
    with $a \in gossipFriends$
     do
        $newJoinerNotif[a] := Append(newJoinerNotif[a], NewJoinerNotifReq(app))$ ;
    end with ;
end macro ;
```

This proccess represents a shopy-list running in one of the several network clients.

Since Opti-shopylist is a decentralized program, the user creates a network of connected instances of Opti-shopylist.

We assume that every client *app* has only one shopy-list.

```
fair process $ClientApp \in APPS$
variables
    $joined = \text{FALSE}$,
    $gossipFriends = \{\}$,
    $knownApps = \langle self \rangle$ ;

begin $AppLoop$:
    while TRUE do

        Below are actions to manage the connection to the network.
        either
             SEND JOIN REQUEST
            with $a \in (GateApps -- self)$
             do
                $joinReqQueue[a] := Append(joinReqQueue[a], NewJoinReqMsg(self))$ ;
            end with ;
        or
             RESPOND TO JOIN REQUEST
            if $isGate[self]$ then
                await $joinReqQueue[self] \neq \langle\rangle$ ;
                with $joinRequest = Head(joinReqQueue[self])$,
                        $updatedKnownApps = Append(knownApps, joinRequest.app)$
                 do

                    $joinRespQueue[joinRequest.app] := Append($
                        $joinRespQueue[joinRequest.app]$,
                        $NewJoinRespMsg(self, SelectSeq(knownApps,$
                            $\text{LAMBDA } app : app \neq joinRequest.app)))$ ;

                    if $joinRequest.app \notin PT!Range(knownApps)$
                     then
                        $knownApps := updatedKnownApps$ ;
                        $gossipFriends := PickGossipFriends(self, Tail(updatedKnownApps))$ ;
                    end if ;

                    $joinReqQueue[self] := Tail(joinReqQueue[self])$ ;

                    $joined := \text{TRUE}$ ;
                end with ;
            end if ;
        or
             RECEIVE JOIN RESPONSE
            await $joinRespQueue[self] \neq \langle\rangle$ ;
            with $joinResponse\ \ \ = Head(joinRespQueue[self])$,
                 $newKnownApps = PT!Range(joinResponse.knownHosts) \setminus PT!Range(knownApps)$
             do
                $gossipFriends := PickGossipFriends(self, joinResponse.knownHosts)$ ;

                $knownApps := knownApps \circ PT!OrderSet(newKnownApps)$ ;

                $joinRespQueue[self] := Tail(joinRespQueue[self])$ ;

                $joined :=$ TRUE ;
            end with ;
```

Following are the actions applying to the shopy-list managed by the *app*.

```
or
     ADD
    await $Cardinality(shopyList[self]) < Cardinality(PRODUCTS)$ ;
    $shopyList[self] := shopyList[self] \mathbin{++} NewShopyItem(shopyList[self])$ ;
or
     REMOVE
    await $shopyList[self] \neq \{\}$ ;
    $shopyList[self] := shopyList[self] -- ExistingShopyItem(shopyList[self])$ ;
or
     ITEM HAS BEEN BOUGHT
    await $shopyList[self] \neq \{\}$ ;
    await $\exists\, item \in shopyList[self] : \neg item.bought$ ;
    with $modifiedItem = ExistingNotBoughtShopyItem(shopyList[self])$
     do
        $shopyList[self] := shopyList[self] -- modifiedItem \mathbin{++} [modifiedItem$ EXCEPT $!.bought =$ TRUE
    end with ;
```

Below actions manage the synchronization of the list.

```
or
     SEND SYNC REQUEST
    with $a \in (PT!Range(knownApps) -- self)$
     do
        $syncReqQueue[a] := Append(syncReqQueue[a], NewSyncReq(self))$ ;
    end with ;
or
     RCV SYNC REQUEST
    await $syncReqQueue[self] \neq \langle\rangle$ ;
    with $syncRequest = Head(syncReqQueue[self])$,
         $mergeResult = shopyList[self] \cup syncRequest.list$,
         $newResp = NewSyncResp(self, mergeResult, syncRequest.id)$
     do
```

```
                syncReqQueue[self] := Tail(syncReqQueue[self]) ;
                 merge from request app
                shopyList[self] := mergeResult ;
                syncRespQueue[syncRequest.app] := Append(syncRespQueue[syncRequest.app], newResp) ;
            end with ;
        or
             RCV SYNC RESPONSE
            await syncRespQueue[self] $\neq \langle\rangle$ ;
            with syncResponse = Head(syncRespQueue[self]),
                 mergeResult = shopyList[self] $\cup$ syncResponse.list
             do

                shopyList[self] := mergeResult ;
                syncRespQueue[self] := Tail(syncRespQueue[self]) ;
            end with ;
        end either ;
    end while ;
end process ;

end algorithm  ;
 BEGIN TRANSLATION (chksum(pcal) = "a4dd4f8d" $\land$ chksum(tla) = "e6dbda73")
VARIABLES isGate, shopyList, syncReqQueue, syncRespQueue, joinReqQueue,
          joinRespQueue, newJoinerNotif, takenIDs

 define statement
NewShopyItem(list) $\triangleq$
    [id     $\mapsto$ (CHOOSE $x \in$ PRODUCTS : $\neg\exists\, i \in$ list : $x = i.id$),
     bought $\mapsto$ FALSE]

ExistingShopyItem(list) $\triangleq$ CHOOSE $x \in$ list : TRUE

ExistingNotBoughtShopyItem(list) $\triangleq$ CHOOSE $x \in$ list : $x.bought$ = FALSE



NewSyncMsg(id, a, l, ml, t) $\triangleq$
    [id $\mapsto$ id,
     app $\mapsto$ a,
     list $\mapsto$ l,
     mergedList $\mapsto$ ml,
     type $\mapsto$ t]

NewSyncReqMsg(a, l, ml, t) $\triangleq$
    NewSyncMsg(
        (CHOOSE $i \in$ IDs : $\forall\, ti \in$ takenIDs : $i = ti$),
        a, l, ml, t
```

```
  )

NewSyncReq(app) ≜
    NewSyncReqMsg(app, shopyList[app], {}, REQ_SYNC_ACTION)

NewSyncResp(app, mergeResult, id) ≜
    NewSyncMsg(id, app, shopyList[app], mergeResult, RESP_SYNC_ACTION)



NewJoinReqMsg(app) ≜ [app ↦ app]

NewJoinRespMsg(app, hosts) ≜
    [app ↦ app,
     knownHosts ↦ hosts]

GateApps ≜ {a ∈ APPS : isGate[a]}

NewJoinerNotifReq(app) ≜ [app ↦ app]

PickGossipFriends(app, knownApps) ≜
    LET Opposit ≜
                PT!Index(knownApps, app) + (Len(knownApps) ÷ 2) − (Len(knownApps)%2)

        PreviousIndex(i) ≜
            IF Len(knownApps) < 3
             THEN 1
             ELSE IF i = 1 THEN Len(knownApps) ELSE i − 1

        NextIndex(i) ≜
            IF Len(knownApps) < 3
             THEN Len(knownApps)
             ELSE IF i = Len(knownApps) THEN 1 ELSE i + 1
    IN  {knownApps[PreviousIndex(Opposit)], knownApps[NextIndex(Opposit)]}



MergeKnownApps(apps1, apps2) ≜
    LET AppSeq(n) ≜ PT!SeqOf(APPS, n)

        Contains(appSeq, appItem) ≜
            Cardinality(PT!Matching(appSeq, appItem)) > 0

        f[args ∈ AppSeq(Len(apps1))
                  × AppSeq(Len(apps2))
                  × AppSeq(Len(apps1) + Len(apps2))] ≜
        LET $l1 \triangleq args[1]$
            $l2 \triangleq args[2]$
            $acc \triangleq args[3]$
            $PickFromL1 \triangleq f[\langle$
                $Tail(l1),$
                $l2,$
                $Append(acc, Head(l1))\rangle]$
            $SkipOneL1 \triangleq f[\langle$
                $Tail(l1),$
                $l2,$
                $acc\rangle]$
            $PickFromL2 \triangleq f[\langle$
                $l1,$
                $Tail(l2),$
                $Append(acc, Head(l2))\rangle]$
            $SkipOneL2 \triangleq f[\langle$
                $l1,$
                $Tail(l2),$
                $acc\rangle]$
        IN
            IF $Len(l2) = 0$
             THEN IF $Len(l1) = 0$
                    THEN $acc$
                    ELSE IF $Contains(acc, Head(l1))$ THEN $SkipOneL1$ ELSE $PickFromL1$
             ELSE IF $Head(l1) \neq Head(l2)$
                    THEN IF $Contains(acc, Head(l2))$ THEN $SkipOneL2$ ELSE $PickFromL2$
                    ELSE IF $Contains(acc, Head(l1))$ THEN $SkipOneL1$ ELSE $PickFromL1$
    IN  $f[\langle apps1, apps2, \langle\rangle\rangle]$

VARIABLES $joined, gossipFriends, knownApps$

$vars \triangleq \langle isGate, shopyList, syncReqQueue, syncRespQueue, joinReqQueue,$
         $joinRespQueue, newJoinerNotif, takenIDs, joined, gossipFriends,$
         $knownApps\rangle$

$ProcSet \triangleq (APPS)$

$Init \triangleq$  Global variables
        $\land isGate = [a \in APPS \mapsto a \in GATEAPPS]$
        $\land shopyList = [a \in APPS \mapsto \{\}]$
        $\land syncReqQueue = [a \in APPS \mapsto \langle\rangle]$
        $\land syncRespQueue = [a \in APPS \mapsto \langle\rangle]$
        ∧ joinReqQueue = [a ∈ APPS ↦ ⟨⟩]
        ∧ joinRespQueue = [a ∈ APPS ↦ ⟨⟩]
        ∧ newJoinerNotif = [a ∈ APPS ↦ ⟨⟩]
        ∧ takenIDs = {}
        Process ClientApp
        ∧ joined = [self ∈ APPS ↦ FALSE]
        ∧ gossipFriends = [self ∈ APPS ↦ {}]
        ∧ knownApps = [self ∈ APPS ↦ ⟨self⟩]

ClientApp(self) ≜ ∧ ∨ ∧ ∃ a ∈ (GateApps −− self) :
                            joinReqQueue′ = [joinReqQueue EXCEPT ![a] = Append(joinReqQueue[a], NewJ
                      ∧ UNCHANGED ⟨shopyList, syncReqQueue, syncRespQueue, joinRespQueue, joined,
                    ∨ ∧ IF isGate[self]
                           THEN ∧ joinReqQueue[self] ≠ ⟨⟩
                                ∧ LET joinRequest ≜ Head(joinReqQueue[self])IN
                                     LET updatedKnownApps ≜ Append(knownApps[self], joinRequest.a
                                       ∧ joinRespQueue′ = [joinRespQueue EXCEPT ![joinRequest.app] =



                                       ∧ IF joinRequest.app ∉ PT!Range(knownApps[self])
                                             THEN ∧ knownApps′ = [knownApps EXCEPT ![self] = upda
                                                  ∧ gossipFriends′ = [gossipFriends EXCEPT ![self] = P
                                             ELSE ∧ TRUE
                                                  ∧ UNCHANGED ⟨gossipFriends,
                                                               knownApps⟩
                                       ∧ joinReqQueue′ = [joinReqQueue EXCEPT ![self] = Tail(joinReq
                                       ∧ joined′ = [joined EXCEPT ![self] = TRUE]
                           ELSE ∧ TRUE
                                ∧ UNCHANGED ⟨joinReqQueue,
                                             joinRespQueue, joined,
                                             gossipFriends, knownApps⟩
                      ∧ UNCHANGED ⟨shopyList, syncReqQueue, syncRespQueue⟩
                    ∨ ∧ joinRespQueue[self] ≠ ⟨⟩
                      ∧ LET joinResponse ≜ Head(joinRespQueue[self])IN
                            LET newKnownApps ≜ PT!Range(joinResponse.knownHosts) \ PT!Range(kno
                              ∧ gossipFriends′ = [gossipFriends EXCEPT ![self] = PickGossipFriends(self, j
                              ∧ knownApps′ = [knownApps EXCEPT ![self] = knownApps[self] ∘ PT!OrderS
                              ∧ joinRespQueue′ = [joinRespQueue EXCEPT ![self] = Tail(joinRespQueue[se
                              ∧ joined′ = [joined EXCEPT ![self] = TRUE]
                      ∧ UNCHANGED ⟨shopyList, syncReqQueue, syncRespQueue, joinReqQueue⟩
                    ∨ ∧ Cardinality(shopyList[self]) < Cardinality(PRODUCTS)
                      ∧ shopyList′ = [shopyList EXCEPT ![self] = shopyList[self] ++ NewShopyItem(shop
                      ∧ UNCHANGED ⟨syncReqQueue, syncRespQueue, joinReqQueue, joinRespQueue, join
                    ∨ ∧ shopyList[self] ≠ {}
```

```
                ∧ shopyList′ = [shopyList EXCEPT ![self] = shopyList[self] −− ExistingShopyItem(s
                ∧ UNCHANGED ⟨syncReqQueue, syncRespQueue, joinReqQueue, joinRespQueue, joi
              ∨ ∧ shopyList[self] ≠ {}
                ∧ ∃ item ∈ shopyList[self] : ¬item.bought
                ∧ LET modifiedItem ≜ ExistingNotBoughtShopyItem(shopyList[self])IN
                    shopyList′ = [shopyList EXCEPT ![self] = shopyList[self] −− modifiedItem ++ [r
                ∧ UNCHANGED ⟨syncReqQueue, syncRespQueue, joinReqQueue, joinRespQueue, joi
              ∨ ∧ ∃ a ∈ (PT!Range(knownApps[self]) −− self) :
                    syncReqQueue′ = [syncReqQueue EXCEPT ![a] = Append(syncReqQueue[a], Neu
                ∧ UNCHANGED ⟨shopyList, syncRespQueue, joinReqQueue, joinRespQueue, joined, g
              ∨ ∧ syncReqQueue[self] ≠ ⟨⟩
                ∧ LET syncRequest ≜ Head(syncReqQueue[self])IN
                    LET mergeResult ≜ shopyList[self] ∪ syncRequest.listIN
                      LET newResp ≜ NewSyncResp(self, mergeResult, syncRequest.id)IN
                        ∧ syncReqQueue′ = [syncReqQueue EXCEPT ![self] = Tail(syncReqQueue[se
                        ∧ shopyList′ = [shopyList EXCEPT ![self] = mergeResult]
                        ∧ syncRespQueue′ = [syncRespQueue EXCEPT ![syncRequest.app] = Appenc
                ∧ UNCHANGED ⟨joinReqQueue, joinRespQueue, joined, gossipFriends, knownApps⟩
              ∨ ∧ syncRespQueue[self] ≠ ⟨⟩
                ∧ LET syncResponse ≜ Head(syncRespQueue[self])IN
                    LET mergeResult ≜ shopyList[self] ∪ syncResponse.listIN
                      ∧ shopyList′ = [shopyList EXCEPT ![self] = mergeResult]
                      ∧ syncRespQueue′ = [syncRespQueue EXCEPT ![self] = Tail(syncRespQueue[
                ∧ UNCHANGED ⟨syncReqQueue, joinReqQueue, joinRespQueue, joined, gossipFriend
            ∧ UNCHANGED ⟨isGate, newJoinerNotif, takenIDs⟩

Next ≜ (∃ self ∈ APPS : ClientApp(self))

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ ∀ self ∈ APPS : WF_vars(ClientApp(self))

END TRANSLATION

NoDuplicates(seq) ≜
    ∀ i, j ∈ DOMAIN seq :
        i ≠ j ⇒ seq[i] ≠ seq[j]

JoinedApps(joinedApps) ≜ {j ∈ APPS : joinedApps[j]}

CountGossipOf(app, gossips, joinedApps) ≜
    PT!ReduceSet(
        LAMBDA a, acc : acc + (IF app ∈ gossips[a] ∧ joinedApps[a]
                                  THEN 1 ELSE 0),
        APPS, 0)

AverageGossipOf(gossips, joinedApps) ≜
    PT!ReduceSet(
        $\textsc{lambda}\ a, acc : acc + CountGossipOf(a, gossips, joinedApps),$
        $APPS, 0)$
    $\div$
    $Cardinality(JoinedApps(joinedApps))$

$ExistsRoute(from, to, \_gossipFriends) \triangleq$
    $\textsc{let}\ f[\langle app, visited\rangle \in APPS \times \textsc{subset}\ APPS] \triangleq$
            $to \in \_gossipFriends[app]$
            $\lor \exists\, a \in (\_gossipFriends[app] \setminus visited) : f[a, visited +\!+\ app]$
    $\textsc{in}\quad from = to \lor f[\langle from, \{\}\rangle]$

$TypeOK \triangleq$
    $\land\quad \forall\, a \in APPS :$
        Checking on variables' domains.
        $\land\ shopyList[a] \subseteq ShopyItems$
        $\land\ PT!Range(syncReqQueue[a]) \subseteq SyncMsgs$
        $\land\ PT!Range(syncRespQueue[a]) \subseteq SyncMsgs$
         The queue for join requests is only for gate apps
        $\land\ \textsc{if}\ isGate[a]$
            $\textsc{then}\ PT!Range(joinReqQueue[a]) \subseteq JoinReqMsgs$
                  $\land \forall\, req \in PT!Range(joinReqQueue[a]) : req.app \neq a$
            $\textsc{else}\ joinReqQueue[a] = \langle\rangle$
        $\land\ PT!Range(joinRespQueue[a]) \subseteq JoinRespMsgs$
        $\land\ PT!Range(newJoinerNotif[a]) \subseteq JoinNotifMsgs$
         $knownApps$ is a collection of unique, ordered apps.
        $\land\ PT!Range(knownApps[a]) \subseteq APPS$
        $\land\ NoDuplicates(knownApps[a])$
         Apps don't gossip themselves
        $\land\ gossipFriends[a] \subseteq (APPS -\!-\ a)$
         Invariant when we're connected or not.
        $\land\ gossipFriends[a]\ \neq \{\}$
           $\equiv knownApps[a] \neq \langle a\rangle$
        Debug breakpoint for all apps 'a'.
  \* a sync response has been sent
  $\land\ (syncRespQueue[a] = \langle\rangle$
     \* no shopy lists are empty
     $\lor\ shopyList[a] = \{\})$
    $\land \forall\, ja \in JoinedApps(joined) : CountGossipOf(ja, gossipFriends, joined) = 0 \lor ($
        $\land\ CountGossipOf(ja, gossipFriends, joined) \geq AverageGossipOf(gossipFriends, joined) - 1$
        $\land\ CountGossipOf(ja, gossipFriends, joined) \leq AverageGossipOf(gossipFriends, joined) + 1$
      $)$
    $\land\ takenIDs \subseteq IDs$

$Liveness \triangleq$
    At some point, someone has joined and gossips have been assigned.
```

$\land \Diamond(\forall\, ja \in \mathit{JoinedApps}(\mathit{joined}) : \mathit{joined}[ja] \land \mathit{CountGossipOf}(ja,\ \mathit{gossipFriends},\ \mathit{joined}) > 0)$

There's a route from every other connected *app* to a joined *app*.

$$\begin{aligned}
&\land \forall\, a \in \mathit{APPS} : \\
&\qquad \mathit{joined}[a] \\
&\qquad \leadsto \forall\, a2 \in \{ j \in (\mathit{APPS} -- a) : \mathit{joined}[j] \} : \\
&\qquad\qquad \mathit{ExistsRoute}(a2,\ a,\ \mathit{gossipFriends})
\end{aligned}$$

* Modification History
* Last modified *Wed Mar* 17 13:38:18 *CET* 2021 by *davd*
* Created *Tue Mar* 02 12:33:43 *CET* 2021 by *davd*