```
(** Romain SADOK & Antoine MULLIER **)

theory TP89
imports Main (* "~~/src/HOL/Library/Code_Target_Nat" *)
begin

(*
quickcheck_params [size=6,tester=narrowing,timeout=120]
nitpick_params [timeout=120]
*)

type_synonym transid= "nat*nat*nat"

datatype message=
  Pay transid nat
| Ack transid nat
| Cancel transid

type_synonym transaction= "transid * nat"

(******************)
datatype etat = Validee | EnCours | Annulee | PrixProposer |PrixDemander

type_synonym trans= "etat * transid * nat (*offre*) * nat  (*demande *)"
type_synonym transBdd = "trans list"

fun equal:: "transid \<Rightarrow> transid \<Rightarrow> bool"
where
  " equal (c1,m1,idT1) (c2,m2,idT2) = (c1=c2 & m1=m2 & idT1=idT2)"

fun annuleOffre:: "transid \<Rightarrow> transBdd \<Rightarrow> transBdd"
where
"annuleOffre tId [] = [(Annulee, tId, 0, 0)]"|
"annuleOffre tId1 ((etat, tId2, offre, demande)#s) = (
    if (equal tId1 tId2) then (Annulee, tId2,offre, demande)#s
    else (etat, tId2, offre, demande)#(annuleOffre tId1 s))"


fun offreClient::"transid \<Rightarrow> nat(* montant proposé *) \<Rightarrow> transBdd \<Rightarrow> transBdd"
where
"offreClient tId x [] = (if (x > 0) then (PrixProposer, tId, x, 0)#[] else [])" |(* 0 \<rightarrow> pour l'instant aucune demande de la part du marchand *)
"offreClient tId1 x ((etat, tId2, pClient, pMarchand)#s) =  (if (equal tId1 tId2) then
(case etat of
  Validee \<Rightarrow> (etat, tId2, pClient, pMarchand)#s | (* aucune renégociation *)
  Annulee \<Rightarrow> (etat, tId2,pClient , pMarchand)#s | (* rien à faire *)
  EnCours \<Rightarrow> (if x > pClient then
                (if x \<ge> pMarchand (*prendre en compte la nouvelle offre ...*)
                   then (Validee, tId2, x, pMarchand)#s  (*...et la valider*)
                 else (etat, tId2, x, pMarchand)#s) (*...sans la valider*)
              else (etat, tId2, pClient, pMarchand)#s (*ignorer l'offre*)
              )|
  PrixProposer \<Rightarrow> (if (x > pClient) then
                    (etat, tId2, x, pMarchand)#s (*on edite le montant avec le nouveau (plus grand) *)
                  else (etat, tId2, pClient, pMarchand)#s (*on ne touche pas a l'offre*)
                  ) |
  PrixDemander \<Rightarrow> (if (x > 0) then
                      (if(x \<ge> pMarchand) (*prendre en compte la nouvelle offre ...*)
                          then (Validee, tId2, x, pMarchand)#s  (*...et la valider*)
                       else (EnCours, tId2, x, pMarchand)#s) (*...sans la valider*)
                  else (etat, tId2, pClient, pMarchand)#s (*ignorer l'offre*)
                       )
                  )
 else((etat, tId2, pClient, pMarchand)#(offreClient tId1 x s))
                 )"

fun offreMarchand::"transid \<Rightarrow> nat (* montant demander *) \<Rightarrow> transBdd \<Rightarrow> transBdd"
where
"offreMarchand tId x [] = (if (x > 0) then (PrixDemander, tId, 0, x)#[] else [])" | (* 0 \<rightarrow> pour l'instant rien de proposer et la demande est sup a zero *)
"offreMarchand tId1 x ((etat, tId2, pClient, pMarchand)#s) = (if (equal tId1 tId2)
then (case etat of
  Validee \<Rightarrow> (etat, tId2, pClient, pMarchand)#s |(* rien à faire *)
  Annulee \<Rightarrow> (etat, tId2, pClient, pMarchand)#s| (* rien à faire *)
  EnCours \<Rightarrow> (if x < pMarchand
              then
                (if(x \<le> pClient) (*prendre en compte la nouvelle offre ...*)
```



```
                    then (Validee, tId2, pClient, x)#s  (*...et la valider*)
                  else (etat, tId2, pClient, x)#s) (*...sans la valider*)
              else (etat, tId2, pClient, pMarchand)#s (*ignorer l'offre*)
             )|
  PrixProposer \<Rightarrow> (if x \<ge> 0 then
                    (if pClient \<ge> x (*prendre en compte la nouvelle offre ...*)
                        then (Validee, tId2, pClient, x)#s  (*...et la valider*)
                     else (EnCours, tId2, pClient, x)#s (*...sans la valider*)
                    )
                  else (etat, tId2, pClient, pMarchand)#s (*ignorer l'offre*)
                  ) |
  PrixDemander \<Rightarrow> (if ((x < pMarchand) \<and> (x > 0))
                    then (etat, tId2, pClient, x)#s (* On accèpte la nouvelle demande si elle est inf à l'ancienne... *)
                  else (etat, tId2, pClient, pMarchand)#s (* ...Sinon on l'ignore  *)
                       )
                  )
else (etat,tId2, pClient, pMarchand)#(offreMarchand tId1 x s)) "



(*** Traitement de message ***)
fun traiterMessage::"message \<Rightarrow> transBdd \<Rightarrow> transBdd"
where
"traiterMessage (Pay tId x) bdd = (offreClient tId x bdd)" |
"traiterMessage (Ack tId x) bdd = (offreMarchand tId x bdd)" |
"traiterMessage (Cancel tId) bdd = (annuleOffre tId bdd)"
(* ################################################## *)

(* Reste à construire la liste des transactions validées *)
fun export::"transBdd \<Rightarrow> transaction list"
where
"export [] = []"|
"export ((Validee, tId, pClient, pMarchand)#s) = ((tId,pClient)#(export s))"|
"export ((_,tId,pClient,pMarchand)#s) = (export s)"

(* ################################################## *)

(*** traiter une liste de message ***)
fun reverse::"'a list \<Rightarrow> 'a list"
where
"reverse [] = []" |
"reverse (x#xs) = ((reverse xs)@[x])"

fun tM_aux::"message list \<Rightarrow> transBdd"
where
"tM_aux [] = []" |
"tM_aux (m#s) = (traiterMessage m (tM_aux s))"

fun traiterMessageList::"message list \<Rightarrow> transBdd"
where
"traiterMessageList l= (tM_aux (reverse l))"
(* ################################### *)

(*************** Methodes utilisé pour les lemmes ************************)

fun getTidInTransBdd :: "transid \<Rightarrow> transBdd \<Rightarrow> trans"
where
"getTidInTransBdd tId1 [] = (EnCours,tId1,0,0) "| (* # Pas encore sur # *)
"getTidInTransBdd tId1 ((etat, tId2, pClient, pMarchand)#s) =
  (if (equal tId1 tId2)
    then (etat, tId2, pClient, pMarchand)
    else (getTidInTransBdd tId1 s))"

(* Vérifie si une transaction est dans la transBdd  *)
fun tidIsInTransBdd :: "transid \<Rightarrow> transBdd \<Rightarrow> bool"
where
"tidIsInTransBdd _ [] = False" |
"tidIsInTransBdd tId1 ((etat,tId2, pClient, pMarchand)#s) =
  (if (equal tId1 tId2)
    then True
    else((tidIsInTransBdd tId1 s)))"
(* ############################################## *)

(******** Lemmes **********)
lemma lemme1 : "(List.member (export (traiterMessageList l)) (tid, val)) \<longrightarrow> (val > 0)"
(*quickcheck [tester=narrowing, timeout=400, size=8]*)
nitpick [timeout=1]
sorry
```

```
154
155 fun memberTrans::"transaction list \<Rightarrow> transaction \<Rightarrow> bool"
156 where
157 "memberTrans [] m = False"|
158 "memberTrans ((trId,_)#s) (trId2,p) = ((trId = trId2)\<or> (memberTrans s (trId2,p)))"
159
160 fun transactionUnicite::"transaction list \<Rightarrow> bool"
161 where
162 "transactionUnicite [] = True"|
163 "transactionUnicite ((trId,p)#s) = ((memberTrans s (trId,p)) \<or> (transactionUnicite s))"
164
165 lemma lemme2:" (transactionUnicite(export(traiterMessageList msgList)))"
166 (*quickcheck [size=6,tester=narrowing,timeout=120] *)
167 nitpick [timeout=1]
168 sorry
169
170 lemma lemme3 :"(bdd=(traiterMessageList (l@[(Cancel tid)]))) \<longrightarrow> (let (e, _, _
    , _)=(getTidInTransBdd tid bdd) in (e=Annulee))"
171 (*quickcheck [tester=narrowing, timeout=200, size=5] *)
172 nitpick [timeout=1]
173 sorry
174                                     (*@l2 on ajoute d'autres messages pour si tid est defini
    tivement annuler *)
175 lemma lemme4 :"(bdd=(traiterMessageList (l@[(Cancel tid)]@l2)) \<and> (tidIsInTransBdd tid b
    dd)) \<longrightarrow> (let (e, _, _, _)=(getTidInTransBdd tid bdd) in e = Annulee)"
176 (*quickcheck [tester=narrowing, timeout=200, size=4]*)
177 nitpick [timeout=1]
178 sorry
179
180 lemma lemme5 : "let bdd=(traiterMessageList l) in  (((List.member l (Pay tid m)) \<and> (Lis
    t.member l (Ack tid n)))\<and> (m>0) \<and> (m\<ge>n) \<and> (\<not>(List.member l (Cancel t
    id))) )
181                         \<longrightarrow> (let (e, _, _, _)=(getTidInTransBdd tid bdd) in e =
     Validee)"
182 (*quickcheck [tester=narrowing, timeout=200, size=6] *)
183 nitpick [timeout=1]
184 sorry
185
186 lemma lemme6 :"\<exists>m.((bdd=traiterMessageList l) \<and>  (List.member (export bdd) (tid
    , m))\<longrightarrow>(\<exists>n.(List.member l (Pay tid m))\<and>(List.member l (Ack tid n
    )) \<and>(m\<ge>n)))"
187 (*quickcheck [tester=narrowing, timeout=200, size=6] *)
188 nitpick [timeout=1]
189 sorry
190
191
192 lemma lemme7Client : "((bdd=(traiterMessageList ([(Pay tid m1)]@[(Pay tid m2)])))\<and>(m1>m
    2)\<and>(m2>0)) \<longrightarrow> (let (_, _, m, _) = (getTidInTransBdd tid bdd) in (m=m1))"
193 (*quickcheck [tester=narrowing, timeout=200, size=4]*)
194 nitpick [timeout=1]
195 sorry
196
197
198 lemma lemme7Marchand : "((bdd=(traiterMessageList ([(Ack tid m1)]@[(Ack tid m2)])))\<and>(m1
    <m2)\<and>(m1>0) ) \<longrightarrow> (let (_, _, _, m) = (getTidInTransBdd tid bdd) in (m=m1
    ))"
199 (*quickcheck [tester=narrowing, timeout=200, size=4] *)
200 nitpick [timeout=1]
201 sorry
202
                (* Une fois la transaction validee celle ci ne peut etre changer "le montant r
    este inchanger" sauf elle peut etre annulee d'où le 'ou'*)
203 lemma lemme8 : "((let bdd=(traiterMessageList l) in (List.member bdd (Validee, tid, m, offre
    )) \<longrightarrow> (let bdd2=(traiterMessage mess bdd) in ((List.member bdd2 (Validee, tid
    , m, offre)) \<or> (List.member bdd2 (Annulee, tid, m, offre))))))"
204 (*quickcheck [tester=narrowing, timeout=200, size=6]*)
205 nitpick [timeout=1]
206 sorry
207
208
209 lemma lemme9 : "(let bdd=(traiterMessageList l) in (let lTransValid=(export bdd) in
210                     (List.member lTransValid (tid,M)) \<longrightarrow> (let (_, _, mc, _)=(ge
    tTidInTransBdd tid bdd) in (M=mc))))"
211 (*quickcheck [tester=narrowing, timeout=200, size=6] *)
212 nitpick [timeout=1]
213 sorry
214
215
216 (* ##########################################################################################
217   ##########################################################################################
```



```
    *)
218
219 (* ----- Exportation en Scala (Isabelle 2014) -------*)
220
221 (* Directive d'exportation *)
222 (*export_code export traiterMessage in Scala*)
223
224
225
226 end
```