RAPPORT DU TP03 IA01 : Conduite **d'expertise d'un SE d'ordre 0+**

SE D'AIDE A L'EVALUATION DU RISQUE DE CONDUITE

Réalisé par :

Ide Flore KENMOGNE FOKAM
Diallo Mohamed MADIOU

Sommaire :

I- DESCRIPTION DU CONTEXTE ET BUT DU SE

L'une des principales causes de décès est un accident routier, qui est du aux défaillances du conducteur (entre 90% à 95%), du véhicule et de l'environnement (routes, piétons, signalisations). Ceci est la cause principale de l'émergence des systèmes embarqués d'aide à la conduite permettant de PRÉVOIR et de COMPENSER une DÉFAILLANCE. L'objectif de notre système sera donc d'évaluer le niveau de risque en temps réel associé à chaque contexte de conduite intégrant l'ensemble des paramètres importants du système Conducteur-Véhicule-Environnement(C-V-E). Le niveau de risque ainsi obtenu pourra être utilisé dans des applications d'aide à la conduite etc.

II- DESCRIPTION DES CONNAISSANCES NÉCESSAIRES

Le système prendra en entré une situation de conduite qui est constituée de : l'état du conducteur (l'âge son permis et le kilométrage parcouru), l'état du véhicule (la vitesse actuelle, l'accélération tangentielle de celui-ci) et l'état de l'environnement (la vitesse limite autorisée, l'état de la route et l'accélération normale). Il est à noter que l'état de la route peut être soit neigeux, glissant (pendant ou après une pluie), ou sec (normal).
Notre base de règles sera établie de manière hiérarchique. Car il faudra tout d'abord déterminer les niveaux de risques liés :

- Au conducteur (niveau de défaillance lié au conducteur en fonction de son état),
- Au véhicule (niveau de défaillance lié au véhicule en fonction de son état et de la vitesse limite sur l'environnement actuel),
- A l'environnement (niveau de défaillance lié à l'environnement).

Ensuite, nous utiliserons les résultats obtenus plus haut pour déterminer le niveau de risque lié à la situation de conduite actuelle (C-V-E). Nous avons décidé, dans le but de limiter la perte d'information sur la situation de conduite, de faire varier la valeur du risque de la situation entre 1 et 10. Donc à chaque situation C-V-E sera attribuée une note sur 10 de manière à ce que la note 1 correspond à la situation la moins risquée possible alors celle 10 à la plus risquée. Tout ceci pour également faciliter la réutilisation du résultat par d'autres applications.

Référence: **Driving Risk Assessment with Belief Functions**
Jérémie Daniel, Jean-Philippe Lauffenburger, Sacha Bernet and Michel Basset

1- Base des règles

a) Règles permettant de déterminer le niveau de risque du conducteur

- Si **$permis <1 ans** et **$pratique <1000 km** alors **$niveau = Novice_inexpérimenté**
- Si **$permis <1 ans** et **$pratique >=1000 km** alors **$niveau = Novice_expérimenté**
- Si **$permis >=1 ans** et **$permis < 5ans** et **$pratique < 10000** alors **$niveau = conducteur-inexpérimenté**
- Si **$permis >=5 ans** et **$pratique <10000 km** alors **$niveau = conducteur-inexpérimenté**

- Si **$permis <5 ans** et **$pratique >=10000 km $pratique <100000 km** alors**$niveau = conducteur expérimenté**
- Si **$permis >5 ans** et **$pratique >10000 km** et **$pratique < 100000** alors **$niveau = conducteur experimenté**
- **Si $permis >5 ans** et **$pratique >100000 km** alors **$niveau = conducteur très expérimenté**

b) Règles permettant de determiner le niveau de risque du véhicule

- Si **$acceleration_tangentielle > 4** alors **$niveau = renversement_élevé**
- Si **$acceleration_tangentielle> 2** ou **$acceleration_tangentielle < =4** et **vitesse > vitesse_limite** alors **$niveau = dérapage_élevé**
- Si **$acceleration_tangentielle> 2** ou **$acceleration_tangentielle < =4** et **vitesse <=vitesse_limite** alors**$niveau = dérapage_moyen**
- Si **$acceleration_tangentielle>=0** ou **$acceleration_tangentielle< =2** et **vitesse > vitesse_limite** alors **$niveau = dérapage_moyen**
- Si **$acceleration_tangentielle>=0** ou **$acceleration_tangentielle < =2** et **vitesse <= vitesse_limite** alors **$niveau = peu_risqué**

c) Règles permettant de determiner le niveau de risque de l'environnement

- Si **$etat_route = neige** alors **$niveau = très_risqué**
- Si **$etat_route = glissant** alors **$niveau = très_risqué**
- Si **$etat_route = sèche** et **$acceleration_normale > 4** alors **$niveau = très_risqué**
- Si **$etat_route = sèche** et **$acceleration_tangentielle >2** ou **$acceleration_tangentielle <= 4** alors **$niveau = moyennement_risqué**
- Si **$etat_route = sèche** et **$acceleration_tangentielle >=0** ou **$acceleration_tangentielle <= 2** alors **$niveau = peu_risqué**

d) Règles permettant de déterminer le niveau de risque de la situation de conduite

Nous résumerons les règles de cette partie dans le tableau suivant qui donne qui présente les différentes combinaisons des résultats du conducteur, du véhicule et de l'environnement obtenu plus haut ainsi que le niveau de risque associé à chaque triplet.

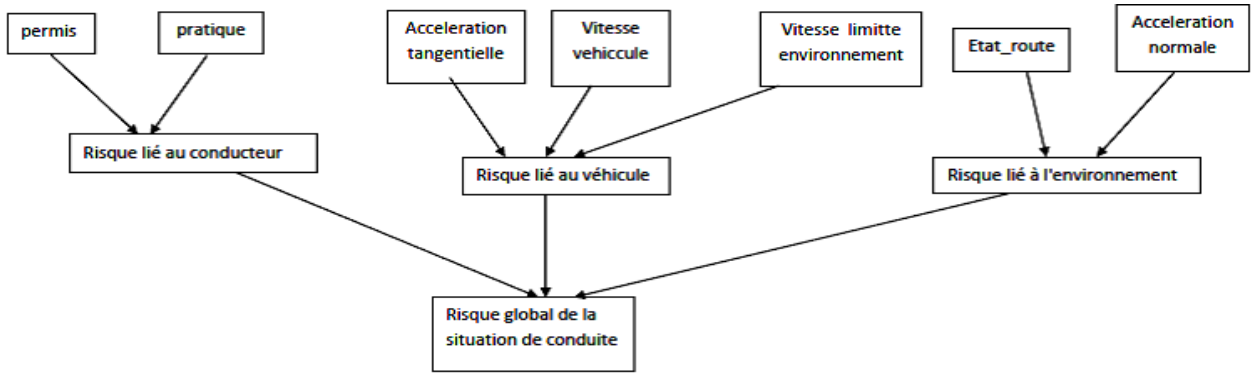
Note : Chaque triplet représente une situation de conduite bien précise et la note 10 équivaut à la situation la plus risquée et la note 1 équivaut à la moins risquée possible donc, la meilleure comme nous l'avons dit plus haut.

État Conducteur	**État Véhicule**	**État Environnement**	**Note**
Novice inexpérimenté	Risque de renversement élevé	environnement très risqué	**10**
		environnement moyennement risqué	**10**
		environnement peu risqué	**10**
	risque de dérapage élevé	environnement très risqué	**10**
		environnement moyennement risqué	**9**
		environnement peu risqué	**9**
	risque de dérapage	environnement très risqué	**9**
		environnement moyennement risqué	**9**
		environnement peu risqué	**9**
	peu risqué	environnement très risqué	**8**
		environnement moyennement risqué	**6**
		environnement peu risqué	**5**
	Risque de renversement élevé	environnement très risqué	**10**
		environnement moyennement risqué	**9**
		environnement peu risqué	**9**
		environnement très risqué	**9**

Novice expérimenté	risque de dérapage élevé	environnement moyennement risqué	**8**
		environnement peu risqué	**8**
	risque de dérapage	environnement très risqué	**8**
		environnement moyennement risqué	**7**
		environnement peu risqué	**6**
	peu risqué	environnement très risqué	**6**
		environnement moyennement risqué	**5**
		environnement peu risqué	**4**
Conducteur inexpérimenté	Risque de renversement élevé	environnement très risqué	**10**
		environnement moyennement risqué	**8**
		environnement peu risqué	**8**
	Risque de dérapage élevé	environnement très risqué	**8**
		environnement moyennement risqué	**7**
		environnement peu risqué	**7**
	risque de dérapage	environnement très risqué	**7**
		environnement moyennement risqué	**7**
		environnement peu risqué	**7**
	peu risqué	environnement très risqué	**5**
		environnement moyennement risqué	**4**
		environnement peu risqué	**3**
Conducteur expérimenté	Risque de renversement élevé	environnement très risqué	**10**
		environnement moyennement risqué	**7**
		environnement peu risqué	**6**
	risque de dérapage élevé	environnement très risqué	**7**
		environnement moyennement risqué	**6**
		environnement peu risqué	**5**
	risque de dérapage	environnement très risqué	**6**
		environnement moyennement risqué	**5**
		environnement peu risqué	**4**
	peu risqué	environnement très risqué	**4**
		environnement moyennement risqué	**3**
		environnement peu risqué	**2**
Conducteur très expérimenté	Risque de renversement élevé	environnement très risqué	**10**
		environnement moyennement risqué	**6**
		environnement peu risqué	**5**
	risque de dérapage élevé	environnement très risqué	**5**
		environnement moyennement risqué	**4**
		environnement peu risqué	**4**
	risque de dérapage	environnement très risqué	**5**
		environnement moyennement risqué	**4**
		environnement peu risqué	**3**
	peu risqué	environnement très risqué	**3**
		environnement moyennement risqué	**2**
		environnement peu risqué	**1**

2- Arbre de déduction du système

Nous pouvons également modéliser les règles de notre système sous forme d'arbre. Dans cette représentation, les nœuds sont les tests effectués, les branches sont les valeurs à tester et les feuilles les actions résultantes (niveau associé). Cela nous permet d'effectuer certains tests uniquement sous certaines conditions de valeurs d'autres tests.



3- Jeux d'essais

Nous avons pu recenser plusieurs situations de conduite et nous pourrons utiliser celles-ci pour tester le fonctionnement du système. En effet dans chacune de ces situations, nous avons un même conducteur dans des environnements différents et des paramètres différents, nous verrons le changement de risque au cours de sa conduite à des instants différents.

Temps	**conducteur**		**véhicule**		**environnement**		
	permis	pratique	vitesse	accélération tangentielle	etat route	acceleration_normale	vitesse_limite
0	40	25 000	70	2	Seche	2,22	110
1			70	4	Glissant	3,26	60
2			50	1	Neige	1,13	110
0	0 (<1)	1 000	70	2	Seche	2,22	110
1			70	4	Glissant	3,26	60
2			50	1	Neige	1,13	110
0	3	10 000	70	2	Seche	2,22	110
1			70	4	Glissant	3,26	60
2			50	1	Neige	1,13	110

III- PROGRAMMATION ET TEST DU SE

1- Description et justification de la représentation lisp choisie

a) Représentation d'un fait

Pour représenter un fait, nous avons utilisé la représentation suivante :
(nom_fait (attribut valeur) (attribut valeur) . . . (attribut valeur))

Nous avons choisi cette représentation car il nous fallait distinguer chaque élément d'une situation de conduite en entré afin de pourvoir faciliter les traitements qui suivrons.

Exemple : les faits initiaux seront représentés comme suit :

```
'(conducteur (permis valeur) (pratique valeur))
'(vehicule (vitesse valeur) (acc_tangentielle valeur))
'(environnement (vitesse_limite valeur) (etat_route valeur) (acc_normale valeur))
```

b) Représentation de la base des faits

La base de faits est un ensemble de faits : c'est une liste de fait.
(fait1 fait2 fait3 . . . faitn)

Dans notre situation, la base des faits initiaux est de cette forme:

```
(setq  base_fait   '((conducteur (permis valeur) (pratique valeur))
(vehicule (vitesse valeur) (acc_tangentielle valeur))
(environnement (vitesse_limitte valeur) (etat_route valeur) (acc_normale valeur)))
```

c) Représentation d'une règle

Une règle dans notre système est sous forme de a-liste dans laquelle la première liste représente le niveau où on se trouve, le dernier représente la conclusion de la règle et celles

intermédiaires la liste des prémisses de la règle. Nous représentons donc ces règles comme suit:

((niveau valeur) (attribut1 opérateur valeur1) (attribut2 opérateur valeur2) . . . (attribut3 opérateur valeur3) (conclusion valeur))

Par exemple : quelques règles peuvent être représentées comme suit:

- **((niveau 0) (permis < 1) (pratique >= 1000) (conducteur novice_experimente))**
- **((niveau 0) (permis >= 1) (permis < 5) (pratique < 10000) (conducteur conducteur_inexperimente))**
- **((niveau 0) (etat_route eql seche) (acc_normale > 4) (environnement tres_risque))**
- **((niveau 0) (acc_tangentielle <= 4) (acc_tangentielle > 2) (vitesse <= vitesse_limitte) (vehicule derapage))**
- **((niveau 1) (conducteur eql novice_inexperimente) (vehicule eql renversement_eleve) (niveau_risque 10))**
- **((niveau 1) (conducteur eql conducteur_inexperimente) (vehicule eql derapage) (niveau_risque 7))**
- **((niveau 1) (conducteur eql conducteur_experimente) (vehicule eql peu_risque) (environnement peu_risque) (niveau_risque 2))**

d) Représentation de la base de règles

La base de règles de notre système est une liste contenant toutes les règles permettant de déterminer le niveau de risque (niveau 0 et niveau1). Donc c'est une liste dont chaque élément est une a-liste qui représente une règle.

2- Description du moteur d'inférence

a- Chainage avant en profondeur d'abord

Nous avons choisi d'utiliser ce type d'algorithme pour notre moteur d'inférence car il facilite la recherche de la solution de la situation de conduite. Car comparé à celui en largeur, il est moins couteux en complexité.

- Fonctions de services

- Fonction permettant étant donné une a-liste et le nom d'une propriété, de retourner sa la valeur de cette propriété.

```
(defun getv (a_liste prop)
  (cdr (assoc prop a_liste)))

(defun get_value (a_liste valeur)
  (let ((real_val (car (getv a_liste valeur))))  ;;;extrait la liste contenant la propriété dans la a-liste
  (cond ((integerp valeur) valeur)               ;;;si la propiete est un entier la retourne
        ((null real_val ) valeur)                ;;;si la valeur extraite est nil retourne le nom de la propriété
        (t real_val))))                          ;;;; sinon retourne la valeur réelle extraite
```

Cette fonction procède comme suit : à l'aide de la fonction **getv** recherche dans la a-liste passée en paramètre la liste contenant la propriété et récupère sa valeur à l'aide de **getv**. Ensuite elle teste si la propriété était un entier, alors le retourne; sinon si sa valeur n'est pas trouvée, retourne le nom de la propriété elle-même autrement retourne la valeur du extraire initialement.

- cette fonction extrait dans un ensemble de règle bien déterminé un sous ensemble de règles qui sont utiles pour les éléments de la liste passée en paramètre. Son but principal est de limiter les parcours effectués dans la base de règles.

```
(defun get_regle_utile (liste regle)
  (let ((element (car liste)) regle_utile)  ;;extrait le nom du fait
    (loop
```

```
(let* ((sous_regle (cdr (pop regle)))          ;;parcours de la liste des règles passée en paramètre
       (type (car (last_element  sous_regle))))  ;;extrait le car de la conclusion pour connaitre l'element dont la regle donne le niveau de risque
       (if (eql element type)
          (setq regle_utile (cons  sous_regle regle_utile)))  ;;si ce type est égale aunom du fait alors la regle est selectionnée
    (unless regle (return-from get_regle_utile regle_utile))))))     ;;retourne la liste des regles selectionnées
```

- fonction qui récupère le dernier élément d'une a-liste passée en paramètre.

```
(defun last_element (a_liste)
  (car (last a_liste)))
```

- fonction qui récupère tous les noms des propriétés de la a-liste et les retourne dans une liste. Pour cela, elle utilise **mapcar** qui renvoie la liste constitué de ces-derniers.

```
(defun get_premier_element (a_liste)
  (mapcar #'car a_liste))
```

- fonction qui récupère le niveau d'une règle

```
(defun get_niveau (regle)
  (cadr (assoc 'niveau regle)))
```

- fonction qui verifie que deux variables sont de même type.

```
(defun check_type (var1 var2)
  (cond ((and (symbolp var1) (symbolp var2)))
        ((and (integerp var1) (integerp var2)))
        (t nil)))
```

- fonction recupère la vitesse limite dans l'environnement qui permettra plutard la mettre dans les paramètres du véhicule en vue d'évaluer son niveau de risque.

```
(defun take_base_fait (base)
  (let (result)
    (dolist (x base)
      (when (setq result (assoc 'vitesse_limite (cdr x) )))) result))
```

- fonction qui étant donnée une variable récupère sa valeur dans la base des faits donc récupère le fait correspondant à cette variable. Cette fonction parcours la base de faits à la recherche de la variable en testant à chaque fois la valeur du **car** à variable. Il est à noter que, comme la vitesse limite est un paramètre de l'environnement mais permet de décider du niveau de risque du véhicule, alors pour ce fait cette fonction doit extraire de l'environnement la liste de **vitesse_limite** et la transfère dans le véhicule pour continuer.

```
(defun get_variable (base_fait variable)
  (let ((var (take_base_fait base_fait)) ;;; on recupere le couple (vitesse_limite val) dans la base des faits
        result)
    (dolist (x base_fait)
              (if (eql (car x) variable)   ;; parcours de la base de faits
                 (setq result x)))
   (cond ((eql variable 'conducteur) result)
         ((eql variable  'vehicule) (cons variable (cons var (cdr result))))  ;;transfert de vitesse_limitte dans véhicule venant du fait environnement
         (t (cons variable (remove var (cdr result)))))))  ;; si on veut extraire l'environnement , on y enleve la vitesse limitte.
```

Exécution :

```
CG-USER(62): (get_variable  '((conducteur (permis 4) (pratique 10000 )) (vehicule  (acc_tangentielle 3) (vitesse 60))
                    (environnement (etat_route seche) (vitesse_limite 50) (acc_normale 2)))  'conducteur )
CG-USER(63):
 (CONDUCTEUR (PERMIS 4) (PRATIQUE 10000))
```

- Fonction permettant de déterminer le niveau de risque d'un fait

- Cette fonction permet à partir d'un fait et de la base de règle correspondante, de déterminer le niveau de risqué de ce fait. Pour mieux comprendre son fonctionnement, l'algorithme suivant présente son fonctionnement.

Algorithme det_niveau (fait base_regle)
Debut
var_a_verifier ← l'ensemble des attributs à vérifier venant du fait
regle_concernee ← l'ensemble des règles applicables pour toutes les **variables** de **var_a_verifier**
niveau← niveau de **base_regle**

- Si la **vitesse limitte** dans **var_a_verifier** alors **l'y enlever**
- Pour tout **V** dans **var_a_verifier** faire
  Regle_candidates ← **extraire** les regles candidates de **V** dans les **regle_concernee**
  - Pour tout **R** dans **Regles_candidates**
    Resultat ← verifier la règle **R**
    Si **resultat non null** alors **retourner resultat**
    Finpour
  Finpour
  Retourner Null

Fin

```
(defun det_niveau (liste sous_base_regle)
 ;;on confronte la liste a la base des regles pour determiner le niveau de la liste \
 ce niveau sera de la forme  (attribut valeur);;
 (let ((variable_a_verifiee (get_premier_element (cdr liste)))
       (regle_concernee (get_regle_utile liste sous_base_regle)))
  (if (member 'vitesse_limite variable_a_verifiee)
    (setq variable_a_verifiee (remove 'vitesse_limite variable_a_verifiee)))

  (dolist (x variable_a_verifiee)
          (let ((regle_candidate (take_regle_candidate  x (cdr liste) regle_concernee)))
       (loop
     (let ((regle (pop regle_candidate)) resultat)
   (setq resultat (check_regle (cdr liste) regle)) ;;; verifier la validité de la regle candidate
   (when resultat (return-from det_niveau resultat)))
     (unless regle_candidate (return nil)))))))
```

- Fonctions de service pour **det_niveau**

- fonction règle candidate

Cette fonction prend en paramètre une **variable**, une **a_liste**, le **niveau** et la **base de regle** et renvoie une liste qui contient l'ensemble des règles candidates c'est-a-dire les règles pour lesquelles la condition de la variable est satisfaite. Pour cela, on parcourt toute la base des règles passée en paramètre et pour chaque règle si un élément de cette règle est vérifié, on ajoute la règle à la liste des règles candidates. Cette fonction a un traitement presque similaire que ce soit le niveau 0 ou le niveau 1: la seule différence vient du fait que dans les règles de niveau 0 il faudra utiliser un **funcall** afin de tester la valeur venant de la règle et la valeur venant du fait passé en paramètre en appliquant la fonction (opérateur) présent dans la règle. Alors qu'au niveau 1 il faudra juste tester l'égalité des deux valeurs.

```
(defun take_regle_candidate (variable a_liste  regle)
 ;;pour chaque regle de regle si le couple (variable valeur) de la a_liste  est vérifié dans la regle \
cette dernière est une regle candidate
```

```
 (let ((val_variable (get_value a_liste variable)) liste_regle_candidate)
  (cond ((null variable) nil)
        (t
         (loop
          (unless regle (return-from take_regle_candidate liste_regle_candidate))  ;; a la fin du parcout  on
retourne la  liste de regles candidates
           (let ((element_regle (pop regle)) cible)
            (setq cible (getv element_regle variable))
            (when cible
  ;; recherche de la valeur de la variable dans la regle et la comparer avec sa valeur dans la a_liste

           (let ((fn (car cible))
                 (valeur (get_value a_liste (cadr cible))))
                (if (funcall fn val_variable  valeur)
                  (setq liste_regle_candidate (cons element_regle  liste_regle_candidate)))))
           ))))))
```

- fonction qui permet d'extraire la liste des règles du niveau passé en paramètre

```
(defun  extrait_niveau (niveau)
 (let (result)
  (dolist (x *base_regles*)
   (let ((var (cadr (assoc 'niveau x))))
    (when (= var niveau)
     (setq result (cons x result)))))
  result))
```

- fonction qui permet de valider une règle donc en fonction d'un fait teste si la règle est validée par application des valeurs du fait à la règle.

```
(defun check_regle (a_liste regle)
   ;;une regle est vrai si tous les elements qui le compose sont vraie pour celà parcourir tous les elements de la
regle et verifier la valider

 (let ((conclusion  (last_element regle)) liste)
      (setq liste (remove conclusion regle :test #' equal))
  (loop
(unless liste (retur-from check_regle conclusion))

    (let ((element (pop liste)) valeur_aliste)
          (setq valeur_aliste (getv a_liste (car element)))
          (when valeur_aliste
            (let ((fn (cadr  element))
                  (valeur_regle (get_value a_liste (last_element element))))
           (setq res (funcall fn (car valeur_aliste) valeur_regle))
          (if (not res)
                  (return-from check_regle nil) ))))))  ;;;  si presence d'un attribut qui est faux donc cette regle
          n'est pas valide
```

- fonction qui permet d'afficher le niveau de risque global de la situation de conduite

```
(defun afficher_conclusion (conclusion)
 (setq lis '(**********************************************************************************))
 (let ((risk (cadr conclusion)))
  (format t "~& ~{~s~}"lis)
  (cond
   ((or (= 9 risk) (= 10 risk)) (format t "~%Votre niveau de risque est:~a~% Vous êtes dans une situation
extrêmement risquée" risk))
   ((or (= 7 risk) (=  8 risk)) (format t "~%Votre niveau de risque est :~a~% vous êtes dans une situation
très risquée" risk))
```

```
    ((or (= 5 risk) (= 6 risk)) (format t "~%Votre niveau de risque est :~a~% vous êtes dans une situation
moyennement risquée" risk))
    (t  (format t "~%Votre niveau de risque est :~a~% vous êtes dans une situation peu risquée" risk)))
   (print conclusion))
  (format t "~& ~{~s~}"lis))
```

- fonction principale

Elle détermine les niveaux de risque du conducteur, du véhicule et de l'environnement ceci à l'aide de la fonction "**det_niveau**". A partir de ces niveaux, forme un nouveau fait, l'ajoute à la base de faits et rappelle la fonction "**det_niveau**" avec ici les règles de la base de règles correspondantes au niveau 1. Ensuite retourne le niveau de risque global de la situation.

```
(defun explore (base_fait)
  (let* ((regle_0 (extrait_niveau 0))
        (regle_1 (extrait_niveau 1))
       (var_conducteur  (get_variable base_fait 'conducteur))
       (var_vehicule (get_variable base_fait 'vehicule))
       (var_environnement (get_variable base_fait 'environnement))
       (niveau_conducteur (det_niveau var_conducteur regle_0))
       (niveau_vehicule (det_niveau var_vehicule regle_0))
       (niveau_environnement (det_niveau var_environnement regle_0)))

   (setq liste `(,niveau_conducteur ,niveau_vehicule ,niveau_environnement))
   (princ liste)
   (dolist (x liste)
     (when (not x) (error "il n'y a pas de solution pour votre cas")))
   (setq conclusion_niveau_0 (cons 'niveau_risque liste))
   (setq base_fait (cons conclusion_niveau_0 base_fait))
    (setq conclusion (det_niveau conclusion_niveau_0 regle_1))
     (if conclusion
       (afficher_conclusion  conclusion)
    (error "aucune conclusion convenable a votre situtation n'a été trouvé"))
   ))
```

b- Chainage arrière

Ici, nous avons pu pour un niveau de risque passé en paramètre, lister les règles ayant probablement conduits à ce niveau de risque. Ensuite avec plus de données de l'utilisateur, affiner notre reconstitution pour donner des informations plus précises et plus pointus sur la situation ayant conduit à ce dernier (faits initiaux). L'algorithme de notre moteur est la suivant:

- Fonction principale qui récupère un niveau et permet de reconstituer les conditions initiales. Elle procède comme suit:
  - a partir du niveau de risque, recherche les faits qui sont responsables
  - ensuite, affine les règles de la base des règles du niveau 0 pour être plus précis sur les faits recherchés
  - enfin retourne les faits ainsi reconstitués.

```
(defun get_fait (niveau )
;; a partir du niveau de risque , on recherche les faits qui sont responsables

  (let ((regle_init (backtrack niveau))
      (liste (question)) regle but)

;; on trie les regles de la base des regles du niveau 0 pour etre plus precis sur les faits

   (setq regle (mapcar #'(lambda (x)
```

```
                (take_regle_candidate x liste regle_init))
            (get_premier_element liste)))

  (dolist (sous_regle   regle) ;;; les reponses aux questions posées nous permets de reduire les faits de la base_fait
    (when (setq conclusion (check_regle liste (car sous_regle)))
      (setq but (append but  `(,conclusion ,(pop liste))))) ;;on verifie si les questions posées sont suffisantes
  (print but)
  (cond ((base_fait_est_precis? but))
        (t (print "les conditions des faits initiaux les plus probables sont:")
           but))))
```

- Fonction qui récupère les valeurs des attributs de chaque liste de la a_liste qui compose la règle passée en paramètre

```
(defun take_valeur (liste)

 (let (result)
  (dolist (sous_liste liste result)
   (setq result (append result
                (mapcar #'(lambda(x) (last_element x)) sous_liste))))))
```

- Fonction qui recherche et sélectionne dans la base des règles ceux dont le niveau de risque correspond au niveau (paramètre de la fonction).

```
(defun regle_utile (niveau regle)
 (let (result)
  (loop
   (unless regle (return-from regle_utile result))
   (let* ((sous_regle (pop regle))
          (niveau_risque (last_element sous_regle)))
     (when (eql (cadr niveau_risque) niveau)
       ;;renvoie la liste des regles utile en supprimant la conclusion  (niveau_risque ) et le niveau  (niveau val)  de chaque regle
       (setq result (cons (cdr (remove niveau_risque sous_regle :test #' equal)) result) ))))))
```

- Fonction qui on vérifie si les réponses aux questions posées nous permettent d'obtenir une base de faits dont chaque attribut est unique

```
 (defun base_fait_est_precis? (liste)

 (let ((liste_sans_doublon (get_premier_element liste))
       liste_inter)
;;on recupere la liste de tous les attributs pour voir si un attribut est present plusieurs fois

   (dolist (x liste_sans_doublon liste)
     (if (member x liste_inter)            ;;;; si un attribut est present plusieurs les questions posées sont insuffisantes
         (return_from base_fait_est_precis? nil)
       (setq liste_inter (cons x liste_inter))))))
```

- Fonction qui pose des questions pour enrichir la base de faits.

```
(defun question ( )
 (format t "quelle est lâge de votre permis ?  ")
 (setq permis (read))
 (format t "quelle est l'etat de la route (seche, neige, glissant) ?  ")
 (setq  etat_route (read))
 (format t "quelle est l'acceleration tangentielle ?  ")
 (setq  acc_tangentielle (read))
```

```
(setq semi_base_fait `((permis ,permis) (etat_route ,etat_route) (acc_tangentielle ,acc_tangentielle))) )
```

- Cette fonction permet de récupérer l'ensemble des règles qui peuvent conduire à la valeur du niveau de risque (paramètre de la fonction).

```
(defun backtrack (niveau)

(let ((regle_1 (extrait_niveau 1))          ;;;;;;;; extraction des regles du niveau 1 et du niveau 0
      (regle_0 (extrait_niveau 0))
      regle_besoin liste_var_unique result conclusion regle)
  (setq regle_besoin (regle_utile niveau regle_1))      ;;;;;;extrait dans regle_1  les regles dont le niveau de risque  est niveau
  (setq liste_conclusion (take_valeur regle_besoin)) ;;;; on extrait les valeurs de chaque  premisses de chacune des  regles contenues dans regle_besoin
  (dolist (x liste_conclusion)
;;on extrait toutes les regles du  niveau_0 dont la valeur de la conclusion est contenue dans liste_conclusion
    (when (not (member x liste_var_unique))
     (setq liste_var_unique (cons x liste_var_unique))))
  (loop
    (unless liste_var_unique (return-from backtrack regle))
    (let* ((element (pop liste_var_unique))
        (conclusion (regle_utile  element regle_0)))
     (setq regle (append conclusion regle))))))
```

3- Test du moteur d'inférence

a) Test du chainage avant

Pour tester notre moteur d'inférence, nous allons tout d'abord représenter les situations du jeu de test donné plus haut conformément à notre représentation. Donc chaque cas sera représenté comme une base de faits et ensuite nous exécuterons ces situations et verons si le resultat obtenu est celui attendu. Sinon il nous permettra de determiner les limittes et les situations non gérées par notre système.

Cas1: (setq base_fait '((conducteur (permis 40) (pratique 25000))
(vehicule (vitesse 70) (acc_tangentielle 2))
(environnement (vitesse_limitte 110) (etat_route seche) (acc_normale 2.22))))

Exécution:

```
CG-USER(42): (NIVEAU_RISQUE (CONDUCTEUR CONDUCTEUR_EXPERIMENTE)
        (VEHICULE PEU_RISQUE)
        (ENVIRONNEMENT MOYENNEMENT_RISQUE))
Votre niveau de risque est :3
 vous êtes dans une situation peu risquée
(NIVEAU_RISQUE 3)
```

Cas2: (setq base_fait '((conducteur (permis 40) (pratique 25000))
(vehicule (vitesse 70) (acc_tangentielle 4))

(environnement (vitesse_limitte 60) (etat_route glissant) (acc_normale 3.26))))

Exécution:

CG-USER(54): (NIVEAU_RISQUE (CONDUCTEUR CONDUCTEUR_EXPERIMENTE)

(VEHICULE DERAPAGE_ELEVE) (ENVIRONNEMENT TRES_RISQUE))

Votre niveau de risque est :7

vous êtes dans une situation très risquée

(NIVEAU_RISQUE 7)

Pour ce les cas qui suivent, nous faisons varier les conditions de l'environnement et du véhicule mais le conducteur reste le même.

Avec ces deux premiers cas nous remarquons que, en fonction de la variation de l'état de l'environnement et celui du véhicule le système donnera un risque différent donc il évalue bien en temps réel le niveau de risque de la situation.

Faisons maintenant varier le paramètre du conducteur en gardant ces de l'environnement et du véhicule statiques.

Cas3: (setq base_fait '((conducteur (permis 0) (pratique 1000))
(vehicule (acc_tangentielle 2) (vitesse 70))
(environnement (etat_route seche) (vitesse_limitte 110) (acc_normale 2.22))))

Exécution:

CG-USER(58): (NIVEAU_RISQUE (CONDUCTEUR NOVICE_EXPERIMENTE)

(VEHICULE PEU_RISQUE)

(ENVIRONNEMENT MOYENNEMENT_RISQUE))

Votre niveau de risque est :5

vous êtes dans une situation moyennement risquée

(NIVEAU_RISQUE 5)

Cas4: (setq base_fait '((conducteur (permis 0) (pratique 1000))
(vehicule (acc_tangentielle 2) (vitesse 70))
(environnement (etat_route seche) (vitesse_limitte 110) (acc_normale 2.22))))

Exécution:

CG-USER(60): (NIVEAU_RISQUE (CONDUCTEUR NOVICE_EXPERIMENTE)

(VEHICULE DERAPAGE_ELEVE) (ENVIRONNEMENT TRES_RISQUE))

Votre niveau de risque est :8

vous êtes dans une situation très risquée

(NIVEAU_RISQUE 8)

En somme nous pouvons voir que le système à chaque variation de la situation donne un niveau de risque précis permettant de limiter le risque d'accident. Donc le fait que ce dernier donne des résultats en temps réel permet de prévenir et de donner des warnings en fonction de la note de la situation. Nous avons pu également en fonction d'un niveau de risque donné

essayé de reconstituer les faits ou les causes probables de ce niveau de risque. Pour ce la, nous avons procédés par chainage arrière en largeur d'abord.

b) Test du chainage arrière

Pour ce faire, nous pouvons montrer ci-dessous quelques traces d'exécution du code.

Exécution1:

```
CG-USER(28): (get_fait 10)
quelle est lâge de votre permis ? 1
quelle est l'etat de la route (seche, neige, glissant ) ?glissant
quelle est l'acceleration tangentielle ?2
"****************resultat************************"
"les conditions initiales:  faits initiaux les plus probables sont:"
((((PERMIS >= 1) (PERMIS < 5) (PRATIQUE >= 10000)
  (PRATIQUE < 100000))
 ((PERMIS >= 1) (PERMIS < 5) (PRATIQUE < 10000)))
 (((ETAT_ROUTE EQL GLISSANT)))
 (((ACC_TANGENTIELLE <= 4) (ACC_TANGENTIELLE > 2)
  (VITESSE > VITESSE_LIMITE))))
```

Exécution1 :

```
CG-USER(29): (get_fait 1)
quelle est lâge de votre permis ? 6
quelle est l'etat de la route (seche, neige, glissant )?seche
quelle est l'acceleration tangentielle ?1
"****************resultat************************"
((PRATIQUE >= 100000) (PERMIS 6) (ACC_NORMALE >= 0)
 (ETAT_ROUTE SECHE) (VITESSE < VITESSE_LIMITE) (ACC_TANGENTIELLE 1))
```

Nous pouvons dire que, la reconstitution des conditions des faits initiaux d'un niveau de risque que fait notre système est assez symbolique mais exacte. Néanmoins il parvient à reconstituer une bonne partie de ceux-ci

IV- PROBLÈMES RENCONTRÉS

Le problème principal que nous avons rencontré était celui de la représentation des faits, des règles et de la base des règles.

- Car étant donné un fait il nous fallait pour le classer, savoir dans quelle classe il appartient afin d'avoir le niveau (0 ou 1) des règles qui lui sont associés.
- Pour une règles, il fallait parfois appliquer certains comparateurs pour les vérifier hors notre moteur d'inférence ne devrait pas faire de calcul, du coup il nous a fallu avoir deux types de représentation des prémisses d'une règle.
  - Pour celles du niveau 0 avec trois champs comme montré plus haut.
  - Pour celles du niveau 1 qui étaient sous la forme (attribut valeur).