Rapport TP3

IA01-A2020

Yicheng WANG et Hanlin WU

I - Introduction

Lors de notre TP numéro 3 , nous devions conduire une expertise permettant la conception d'un système expert d'inférence d'ordre 0+.

Le thème que nous avons choisi est la classification des corps célestes. Ce système expert permet à l'utilisateur de saisir diverses données d'un corps céleste inconnu (si une certaine donnée est inconnue, entrez "inconnu"), et de réaliser différentes manières de raisonner selon le choix de l'utilisateur, donc capable de confirmer le type spécifique de corps céleste.

II - Modélisation

A - DONNEE :

Le système expert demandera à l'utilisateur de saisir un total de 12 différentes données pour décrire un corps céleste inconnu :

Etat_Matiere : La structure physique et l'état des corps célestes

(Valeur posible : roche / glace / metal / hydrogene / helium / solide / gaz / plasma / degeneree)

Forme : Forme du corps céleste

(Valeur posible : sphere / irregulier)

Mode_Lumineux : Motifs lumineux des corps célestes

(Valeur posible : radiation / nil)

Orbite_Etoile : Indique si le corps céleste tourne autour d'une étoile

(Valeur posible : t / nil)

Orbite_Planete : Indique si le corps céleste tourne autour d'une planète

(Valeur posible : t / nil)

Reaction_nucleaire :Indique si le corps céleste subit une réaction de fusion nucléaire

(Valeur posible : normal / nil / pres_de_fin)

Masse : La masse du corps céleste (dans la masse du soleil, par exemple, la masse de la terre est de 3.0e-6, soit 0,000003 fois la masse du soleil)

(Valeur posible : un nombre ou un réel)

Vitesse_Rot : La vitesse de rotation du corps céleste (en secondes, c'est-à-dire le temps nécessaire au corps céleste pour tourner une fois)

(Valeur posible : un nombre ou un réel)

Effacer_Voisine : Indique si le corps céleste a dégagé la zone orbitale environnante

(Valeur posible : t / nil)

Horizon_evenement : Indique si le corps céleste a un horizon d'événements, qui est un terme astronomique utilisé pour décrire les trous noirs.

(Valeur posible : t / nil)

Position : Indique que les positions relatives des corps célestes sont fixes et peuvent être décrites en général

(Valeur posible : centre_galaxie / incertain)

Signal_Impulsion : Indique si le corps céleste émettra des signaux d'impulsion

(Valeur posible : periodique / normal / nil)

!ATTENTION:

Tous les types d'un corps céleste possible :

etoile / etoile_sequence_principale / etoile_compacte / nain_blanc / etoile_neutron / pulsar / geante_rouge/ trou_noir / planete / planete_naine_gazeuse / planete_jovienne / planete_tellurique / planete_naine / asteroide / satellite /

B - REGLES

R11 SI Etat_Matiere=plasma ET Forme=sphere ET Mode_Lumineux=radiation ALORS Type=etoile

R12 SI Etat_Matiere=degeneree ET Forme=sphere ET Reaction_nucleaire=nil ALORS Type=etoile_compacte

R13 SI Vitesse_Rot ET Forme=sphere ET Masse <2.1 ET Masse > 1.35 ALORS Type=etoile_neutron

R14 SI Masse > 3.3 ET Forme=sphere ET Horizon_evenement=vrai ALORS Type=trou_noir

R15 SI Masse > 1000000 ET Forme=sphere ET Position=centre_galaxie

ALORS Type=trou_noir

R16 SI Reaction_nucleaire=nil ET Forme=sphere ET Orbite_Etoile=vrai

ET Effacer_Voisine=vrai
ALORS Type=planete

R17 SI Reaction_nucleaire=nil ET Forme=sphere ET Orbite_Etoile=vrai

ET Effacer_Voisine=faux
ALORS Type=planete_naine

R18 SI Reaction_nucleaire=nil ET Forme= irregulier ET Orbite_Etoile=vrai

ET Effacer_Voisine=faux
ALORS Type=asteroide

R19 SI Reaction_nucleaire=nil ET Orbite_Etoile=faux ET Orbite_Planete=vrai

ET Effacer_Voisine=faux
ALORS Type=satellite

R21 SI Type=etoile ET Reaction_nucleaire=normal

ALORS Type=etoile_sequence_principale

R22 SI Type=etoile_compacte ET Masse <1.33 ET Masse >0.17

ALORS Type=nain_blanc

R23 SI Type=etoile_compacte ET Masse <2.1 ET Masse >1.35

ALORS Type=étoile_neutron

R24 SI Type=etoile ETReaction_nucleaire=pres_de_fin

ET Masse <8 ET Masse >0.3
ALORS Type=geante_rouge

R25 SI Type=planete ET Etat_Matiere=gaz

ALORS Type=planete_jovienne

R26 SI Type=planete ET Etat_Matiere=solide

ALORS Type=planete_tellurique

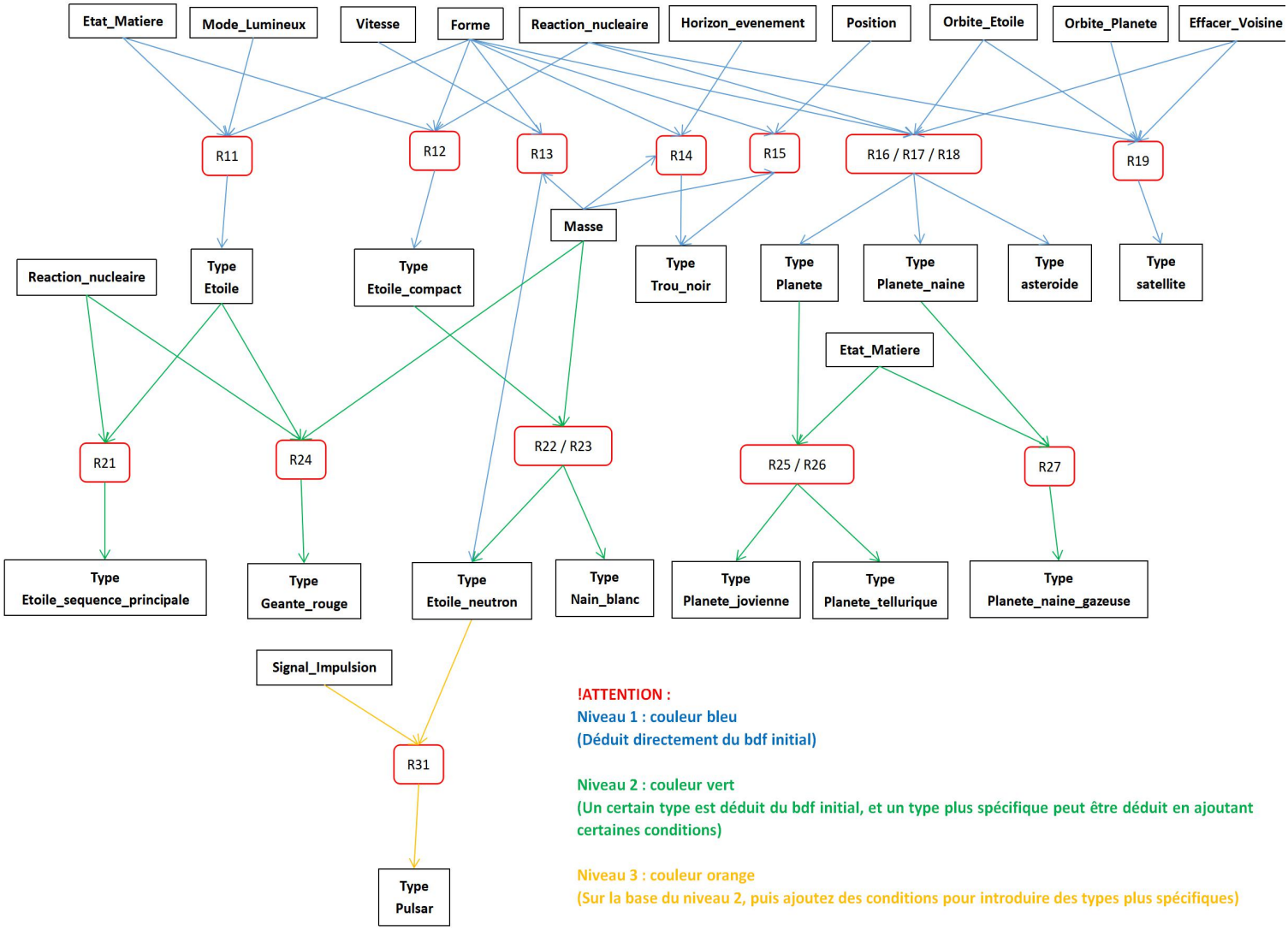
R27 SI Type=planete_naine ET Etat_Matiere=gaz

ALORS Type=planete_naine_gazeuse

R31 SI Type=etoile_neutron ET Signal_Impulsion=periodique

ALORS Type=pulsar

Voici un arbre d'inférence de notre base de régles :



III - Implémentation en Lisp

1 . Base de règles :

```
(setq *BDR* '(
    (R11 ((Etat_Matiere plasma) (Forme sphere) (Mode_Lumineux radiation)) (Type etoile))
    (R12 ((Etat_Matiere degeneree) (Forme sphere) (Reaction_nucleaire nil)) (Type etoile_compacte))
    (R13 ((Vitesse_Rot rapide) (Forme sphere) (Masse (1.35 2.1))) (Type etoile_neutron))
    (R14 ((Masse (> 3.3)) (Forme sphere) (Horizon_evenement t)) (Type trou_noir))
    (R15 ((Masse (> 1000000)) (Forme sphere) (Position centre_galaxie)) (Type trou_noir))
    (R16 ((Forme sphere) (Reaction_nucleaire nil) (Orbite_Etoile t) (Effacer_Voisine t)) (Type planete))
    (R17 ((Forme sphere) (Reaction_nucleaire nil) (Orbite_Etoile t) (Effacer_Voisine nil))
        (Type planete_naine))
    (R18 ((Forme irregulier) (Reaction_nucleaire nil) (Orbite_Etoile t) (Effacer_Voisine nil))
        (Type asteroide))
    (R19 ((Reaction_nucleaire nil) (Orbite_Planete t) (Orbite_Etoile nil) (Effacer_Voisine nil))
        (Type satellite))

     (R21 ((Type etoile) (Reaction_nucleaire normal)) (Type etoile_sequence_principale))
     (R22 ((Type etoile_compacte) (Masse (0.17 1.33))) (Type nain_blanc))
     (R23 ((Type etoile_compacte) (Masse (1.35 2.1))) (Type etoile_neutron))
     (R24 ((Type etoile) (Reaction_nucleaire pres_de_fin) (Masse (0.3 8))) (Type geante_rouge))
     (R25 ((Etat_Matiere gaz) (Type planete)) (Type planete_jovienne))
     (R26 ((Etat_Matiere solide) (Type planete)) (Type planete_tellurique))
     (R27 ((Type planete_naine) (Etat_Matiere gaz)) (Type planete_naine_gazeuse))

     (R31 ((Type etoile_neutron) (Signal_Impulsion periodique)) (Type pulsar))
))
```

2 . Base de faits

Notre base de faits est implémentée par les fonctions d'initialisation init_BDF et init_Champs :

```
(defun init_BDF ()
  (setq *BDF* nil)
  (format t "Veuillez saisir les donnees du corps celeste : ~%")
    (dolist (champ list_champs)
       (init_Champs champ)
      )
  (setq *BDF* (append *BDF* (list (list 'Type 'inconnu))))
  (format t "~%------initialisation finish !------~%")
)
(setq list_champs '(Etat_matiere Forme Mode_Lumineux Orbite_Etoile Orbite_Planete Reaction_nucleaire Masse
                    Vitesse_Rot Effacer_Voisine Horizon_evenement Position Signal_Impulsion))
```

!ATTENTION:

Ici on rajoute un champ Type avec la Valeur inconnu pour appliquer les chainage facilement.

```
(defun init_Champs (nom_Champ)
   (let ((valeur nil))
      (format t "Quelle est le ~S du corps celeste ?~%" nom_Champ)
      (cond
        ((equal nom_Champ 'Etat_matiere)
          (format t "(roche / glace / metal / hydrogene / helium / plasma / degeneree / inconnu)~%")
        )
        ((equal nom_Champ 'Forme)
          (format t "(sphere / irregulier / inconnu)~%")
        )
        ((equal nom_Champ 'Mode_Lumineux)
          (format t "(radiation / nil / inconnu)~%")
        )
        ((equal nom_Champ 'Orbite_Etoile)
          (format t "(t / nil / inconnu)~%")
        )
        ((equal nom_Champ 'Orbite_Planete)
          (format t "(t / nil / inconnu)~%")
        )
        ((equal nom_Champ 'Reaction_nucleaire)
          (format t "(normal / nil / pres_de_fin / inconnu)~%")
        )
        ((equal nom_Champ 'Masse)
          (format t "(Entrez un nombre ou reel)~%")
        )
        ((equal nom_Champ 'Vitesse_Rot)
          (format t "(Entrez un nombre ou reel)~%")
        )
        ((equal nom_Champ 'Effacer_Voisine)
          (format t "(t / nil / inconnu)~%")
        )
        ((equal nom_Champ 'Horizon_evenement)
          (format t "(t / nil / inconnu)~%")
        )
        ((equal nom_Champ 'Position)
          (format t "(centre_galaxie / incertain / inconnu)~%")
        )
        ((equal nom_Champ 'Signal_Impulsion)
          (format t "(periodique / normal / nil / inconnu)~%")
        )
      )
      (setq valeur (read))
      (setq *BDF* (append *BDF* (list (list nom_Champ valeur))))
   )
)
```

Notion :

Ici, nous invitons l'utilisateur à entrer la valeur pour éviter les erreurs de saisie utilisateur (la fonction pour éviter les erreurs de données sera implémentée plus tard)

Exemple : Base de faits

Prenons les données de la terre comme exemple :

(
  (Etat_matiere roche) (Forme sphere)
  (Mode_Lumineux nil) (Orbite_Etoile t)
  (Orbite_Planete nil) (Reaction_nucleaire nil)
  (Masse 3.0e-6) (Vitesse_Rot 86400)
  (Effacer_Voisine t) (Horizon_evenement nil)
  (Position incertain) (Signal_Impulsion nil)
  (Type inconnu)
)

3 . Fonctions utiles

Prenons l'exemple de base de faits ci-dessus, le contenu de matiere, masse et vitesse ne peut pas correspondre au contenu de la base de régle, nous avons donc ajouté quelques fonctions qui peuvent convertir des données.

```
(defun Transformation_donnee (bdf)
  (Transform_Matiere bdf)
  (Transform_Masse bdf)
  (Transform_Vitesse_Rot bdf)
  (format t "~%------Transformation finish !------~%~%")
)
```

```
(defun Transform_Matiere (bdf)
  (let ((matiere (cadr (assoc 'Etat_Matiere bdf))))
     (cond
       ((or
          (equal matiere 'roche)
          (equal matiere 'glace)
          (equal matiere 'metal)
        )
        (setf (nth 1 (nth 0 bdf)) 'solide))
       ((or
          (equal matiere 'hydrogene)
          (equal matiere 'helium)
          )
        (setf (nth 1 (nth 0 bdf)) 'gaz))
       ((and
         (not (equal matiere 'plasma))
         (not (equal matiere 'degeneree))
         )
        (setf (nth 1 (nth 0 bdf)) 'inconnu))
     )
  )
)
```

```
(defun Transform_Masse (bdf)
  (let ((masse (cadr (assoc 'Masse bdf))) (list_interval_masse nil))
       (if (not (or (floatp masse) (numberp masse)))
          (progn
            (setf (nth 1 (nth 6 bdf)) nil)
            (return-from Transform_Masse nil)))
       (if (and (> masse 0.17) (< masse 1.33))
         (setq list_interval_masse (append list_interval_masse (list (list 0.17 1.33))))
       )
       (if (and (> masse 1.35) (< masse 2.1))
         (setq list_interval_masse (append list_interval_masse (list (list 1.35 2.1))))
       )
       (if (and (> masse 0.3) (< masse 8))
         (setq list_interval_masse (append list_interval_masse (list (list 0.3 8))))
       )
       (if (> masse 3.3)
         (setq list_interval_masse (append list_interval_masse (list (list '> 3.3))))
       )
       (if (> masse 1000000)
         (setq list_interval_masse (append list_interval_masse (list (list '> 1000000))))
         )
       (setf (nth 1 (nth 6 bdf)) list_interval_masse)
  )
)
```

```
(defun Transform_Vitesse_Rot (bdf)
  (let ((vitesse (cadr (assoc 'Vitesse_Rot bdf))))
    (if (or (floatp vitesse) (numberp vitesse))
       (if (<= vitesse 30)
           (setf (nth 1 (nth 7 bdf)) 'rapide)
         (setf (nth 1 (nth 7 bdf)) 'normal)
       )
      (setf (nth 1 (nth 7 bdf)) 'inconnu)
    )
  )
)
```

Notion :

Pour l'état de la substance, la fonction Transform_Matiere permet d'agréger le type de substance correspondant en solide ou en gaz (le plasma et le dégénérescence sont les valeurs initiales entrées par l'utilisateur et n'ont pas besoin d'être modifiées)

Pour la masse, car dans la base de régle correspondante, la masse du corps céleste peut rencontrer plusieurs plages pour l'inférence, donc cette fonction ajoute toutes les plages éligibles à la valeur de masse

ex: (Masse ((1.35 2.1) (0.3 8))) avec masse = 1.5

Pour la vitesse de rotation, nous la divisons en rapide et normale selon une certaine norme (vitesse de rotation moyenne des étoiles à neutrons)

Parmi les trois items ci-dessus, si matiere ou vitesse est en dehors des valeurs possibles, cet item sera assigné comme inconnu, si masse est en dehors des valeurs possibles, alors il sera assigné comme nil

4 . Moteur d'inférence

4.1 Introduction

Dans ce système, nous avons choisi deux moteurs d'inférence, le chainage avant en profondeur d'abord et le chainage arrière en largeur d'abord.

```
(defun moteur_inference (choix_chainage bdf bdr)
  (cond
   ((equal choix_chainage 'avant)
       (avant_Profondeur bdf bdr)
   )
   ((equal choix_chainage 'arriere)
       (format t "~%~% Veuillez saisir le type de corps céleste que vous souhaitez confirmer :~%~%")
       (setq type_celeste_BUT (read))
       (arriere_Largeur type_celeste_BUT bdf bdr)
   )
  )
)
```

4.2 Fonctions utiles dans les moteurs

regleSuffisant :

Cette fonction permet d'obtenir une liste de tous les règles éligibles en fonction de la base de faits actuelle

```
(defun regleSuffisant (bdf bdr)
  (let ((regleSuff nil))
    (dolist (regle bdr)
      (let ((verifier t))
        (dolist (champ_regle (cadr regle))
          (cond
            ((equal (car champ_regle) 'Masse)
              (if (member (cadr champ_regle) (cadr (assoc (car champ_regle) bdf)) : test 'equal)
                nil (setq verifier nil)))
            (t
              (if (member champ_regle bdf : test 'equal)
                nil (setq verifier nil)))
          )
        )
        (if (and
              (equal verifier t)
              (or
                (member (nth 12 bdf) (cadr regle) : test 'equal)
                (equal (nth 1 (nth 12 bdf)) 'inconnu)
```

```
                    ))
                    (push (car regle) regleSuff))
            )
        )
        (reverse regleSuff)
    )
)
```

verifier_compatible :

Cette fonction est utilisée pour confirmer si la base de faits saisie par l'utilisateur provoquera des conflits dans les résultats d'inférence avant le moteur d'inférence appliquer. (Par exemple, si l'utilisateur saisit des données erronées, la base de faits inférera deux types contradictoires (ex: étoile à neutrons, planète))

```
(setq type_non_compatible '(
  (etoile etoile_neutron) (etoile trou_noir) (etoile planete) (etoile planete_naine)
  (etoile satellite)(etoile_neutron planete) (etoile_neutron planete_naine)
  (etoile_neutron satellite) (trou_noir planete_naine) (trou_noir planete) (trou_noir satellite)
                                    )

(defun verifier_compatible (bdf bdr)
      (let ((reglesuff_1er (regleSuffisant bdf bdr)) (list_type nil) (res t))
          (dolist (regle reglesuff_1er)
              (setq list_type (append list_type (list (cadr (caddr (assoc regle bdr))))))
          )
          (dolist (champ type_non_compatible)
              (if (and
                      (member (car champ) list_type)
                      (member (cadr champ) list_type))
                  (setq res nil)
              )
          )
          res
      )
)
```

appliquer_regle :

Cette fonction est utilisée pour exécuter une règle

```
(defun appliquer_regle (regle bdr bdf)
   (if (equal (nth 1 (nth 12 bdf)) 'inconnu)
     (format t "~%----------------------------------------------------------------------------------~%"))
   (if (member regle (regleSuffisant bdf bdr))
```

```
      (let ((regleComplet (assoc regle bdr)))
        (format t "~%Type ancienne : ~S ~% les conditions ~S appliquent le regle ~S ~%Type maintenant : ~S" (nth 1 (nth 12 bdf)) (cadr regleComplet) regle (cadr (caddr regleComplet)))
        (setf (nth 1 (nth 12 bdf)) (cadr (caddr regleComplet)))
        )
   )
   (format t "~%")
)
```

inference_inverse :

Cette fonction permet de trouver toutes les règles qui peuvent dériver type_but et d'écrire les conditions correspondantes (les conditions correspondantes de chaque regle sont placées entre parenthèses)

```
(defun inference_inverse (type_but bdr)
        (let ((list_conditions nil))
           (dolist (x bdr)
               (if (equal (cadr (caddr x)) type_but)
                     (setq list_conditions (append list_conditions (list (cadr x))))
               )
           )
           list_conditions
      )
)
```

diff :

Cette fonction permet de trouver les différentes parties des deux listes (la différence entre la première et la seconde)

```
(defun diff (L M) (cond ((null L) nil) ((member (car L) M :test 'equal) (diff (cdr L) M)) (t (cons (car L) (diff (cdr L) M)))))
```

flatten :

Cette fonction est utilisée pour diviser le nid, et finalement convertir la liste en (((...) (...)) ((...) (...)) ((...) (...)))

```
(defun flatten (L) (if L (append (car L)(flatten (cdr L)))))
```

successeur :

Cette fonction est utilisée pour déduire une liste de toutes les conditions qui peuvent être déduites pour obtenir list_conds

Par exemple, si list_conds vaut ((Type etoile_neutron)), si les conditions (Cond1 Cond2) et (Cond3 Cond4) peuvent chacune déduire les list_conds, le résultat renvoyé

par la fonction est ((cond1 cond2) (cond3 cond4))
De même, si list_conds est ((Type etoile_neutron) cond5), alors le résultat est ((cond5 cond1 cond2) (cond5 cond3 cond4))

```
(defun successeur (list_conds bdr)
    (let ((result nil) (conds_sans_type nil)
            (champ_type (assoc 'Type list_conds))
            )
      (setq conds_sans_type (diff list_conds (list champ_type)))
      (cond
        ((equal conds_sans_type nil)
            (dolist (x (inference_inverse (cadr champ_type) bdr))
                (setq result (append result (list x)))
            )
        )
        ((not (null champ_type))
            (dolist (x (inference_inverse (cadr champ_type) bdr))
                (setq result (append result (list (append conds_sans_type x))))
            )
        )
      )
    result
    )
)
```

verifier_conditions :

Cette fonction permet de confirmer si la base de faits satisfait à une liste de conditions donnée

```
(defun verifier_conditions (list_conditions bdf)
    (let ((res t) (mass (cadr (assoc 'Masse bdf))))
    (dolist (x list_conditions)
        (cond
          ((equal (car x) 'Masse)
              (if (not (member (cadr x) mass : test 'equal))
                    (setq res nil)
              ))
          ((not (member x bdf : test 'equal)) (setq res nil))
        )
    )
    res)
)
```

4.3 Chainage avant en profondeur d'abord

4.3.1 Principe de fonctionement

La fonction de ce moteur d'inférence est que pour toutes les bases de faits qui peuvent être appliquer règle, chaque récursion appliquera l'un des régles, obtiendra un nouveau type pour remplacer le type d'origine, puis répétez les étapes ci-dessus pour cette nouvelle base de faits jusqu'à ce que vous obteniez le type final.
Puisqu'il y a un cas où le même type est obtenu à travers des régles différentes, la base de faits initiale sera retournée après la fin de l'inférence, et elle ne se terminera pas tant que tous les règles applicables à cette base de faits n'auront pas été parcourus.

4.3.2 Implémentation

```
(defun avant_Profondeur1 (bdf bdr regleOld)
  (let ((regleS (regleSuffisant bdf bdr)) (retourner nil)) ;;liste des règles suffisantes (chaque récursion)
    (cond
      ((not (null regleS))
          (while (and (not retourner) regleS)
            (let ((type_copy (nth 1 (nth 12 bdf))))
              (push (car regleS) regleOld)
              (appliquer_regle (car regleS) bdr bdf)
              (setq retourner (avant_Profondeur1 bdf bdr regleOld))
              (if retourner (format t "~%Retourner avant d'appliquer ~S ~%" (pop regleS)))
              (setf (nth 1 (nth 12 bdf)) type_copy)
              (if (not regleS)
                    (return-from avant_Profondeur1 t)
                (progn
                  (pop regleOld)
                  (setq retourner nil)
                  )))))
      ((and (null regleOld) (null regleS))
          (format t "~%Erreur ! Donnees insuffisantes ou erreur !~%")
      )
      (t      ;;quand on a trouvé le type final
        (format t "~%Le raisonnement est termine, le resultat du raisonnement est TYPE == ~S" (nth
1 (nth 12 bdf)))
        (format t "~%Les regles utilise sont : ")
        (print (reverse regleOld))
        t
      )
    )
```

Chaque application d'une règle modifiera le contenu de la base de faits, de sorte que la valeur de type est copiée avant chaque récursivité pour garantir que la base de faits reste inchangée après la récursivité correspondante.

Cette condition correspond à la situation où les données initiales de base de faits sont insuffisantes ou erronées, ce qui rend impossible l'application d'une règle

```
   )
)

(defun avant_Profondeur (bdf bdr)
    (if (equal (verifier_compatible bdf bdr) t)    ;;vérifier qu'il n'y a pas des types non-compatible
        (avant_Profondeur1 bdf bdr nil)
        (format t "~%Erreur ! Donnee invalide !~%")))
```

4.3.3 Exemple d'implémentation

Exemple 1 : La terre (Il n'y a qu'un seul chemin d'inférence)

Ici le BDF est :

```
(
(Etat_matiere roche) (Forme sphere) (Mode_Lumineux nil) (Orbite_Etoile t) (Orbite_Planete nil) (Reaction_nucleaire nil)
(Masse 3.0e-6) (Vitesse_Rot 86400) (Effacer_Voisine t) (Horizon_evenement nil)(Position incertain) (Signal_Impulsion nil)
(Type inconnu)
)
```

```
Entrez votre choix de la chainage (avant ou arriere) :
avant

--------------------------------------------------------------------------------

Type ancienne : INCONNU
 les conditions ((FORME SPHERE) (REACTION_NUCLEAIRE NIL) (ORBITE_ETOILE T)
                 (EFFACER_VOISINE T)) appliquent le regle R16
Type maintenant : PLANETE

Type ancienne : PLANETE
 les conditions ((ETAT_MATIERE SOLIDE) (TYPE PLANETE)) appliquent le regle R26
Type maintenant : PLANETE_TELLURIQUE

Le raisonnement est termine, le resultat du raisonnement est TYPE == PLANETE_TELLURIQUE
Les regles utilise sont :
(R16 R26)
Retourner avant d'appliquer R26

Retourner avant d'appliquer R16

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

Exemple 2 : Pulsar (il y a deux chemins d'inférence possibles)

Ici le BDF est :

```
((Etat_matiere degeneree) (Forme sphere) (Mode_Lumineux radiation) (Orbite_Etoile nil) (Orbite_Planete nil)
(Reaction_nucleaire nil) (Masse 1.5) (Vitesse_Rot 20) (Effacer_Voisine t) (Horizon_evenement nil)(Position inconnu)
(Signal_Impulsion periodique) (Type inconnu)
)
```

```
Entrez votre choix de la chainage (avant ou arriere) :
avant

--------------------------------------------------------------------------------

Type ancienne : INCONNU
 les conditions ((ETAT_MATIERE DEGENEREE) (FORME SPHERE) (REACTION_NUCLEAIRE NIL)) appliquent le regle R12
Type maintenant : ETOILE_COMPACTE

Type ancienne : ETOILE_COMPACTE
 les conditions ((TYPE ETOILE_COMPACTE) (MASSE (1.35 2.1))) appliquent le regle R23
Type maintenant : ETOILE_NEUTRON

Type ancienne : ETOILE_NEUTRON
 les conditions ((TYPE ETOILE_NEUTRON) (SIGNAL_IMPULSION PERIODIQUE)) appliquent le regle R31
Type maintenant : PULSAR

Le raisonnement est termine, le resultat du raisonnement est TYPE == PULSAR
Les regles utilise sont :
(R12 R23 R31)
Retourner avant d'appliquer R31

Retourner avant d'appliquer R23

Retourner avant d'appliquer R12


--------------------------------------------------------------------------------

Type ancienne : INCONNU
 les conditions ((VITESSE_ROT RAPIDE) (FORME SPHERE) (MASSE (1.35 2.1))) appliquent le regle R13
Type maintenant : ETOILE_NEUTRON

Type ancienne : ETOILE_NEUTRON
 les conditions ((TYPE ETOILE_NEUTRON) (SIGNAL_IMPULSION PERIODIQUE)) appliquent le regle R31
Type maintenant : PULSAR

Le raisonnement est termine, le resultat du raisonnement est TYPE == PULSAR
Les regles utilise sont :
(R13 R31)
Retourner avant d'appliquer R31

Retourner avant d'appliquer R13

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

Exemple 3 : cas non compatible

((Etat_matiere plasma) (Forme sphere) (Mode_Lumineux radiation) (Orbite_Etoile nil) (Orbite_Planete nil)

(Reaction_nucleaire normal) (Masse 1.5) *(Vitesse_Rot 20)* (Effacer_Voisine t) (Horizon_evenement nil)(Position inconnu)

(Signal_Impulsion nil) (Type inconnu)

)

Ici on entrez un champ mauvais *(Vitesse_Rot 20)* ,donc ce BDF peut inférer deux types différentes et non compatibles (etoile et etoile_neutron),mais en pratique c'est impossible , donc ça c'est le cas non compatible.

```
Entrez votre choix de la chainage (avant ou arriere) :
avant

Erreur ! Donnee invalide !

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

Exemple 4 : cas compatible mais pas de résultat (c-a-d Il ne peut rien raisonner)

((Etat_matiere plasma) (Forme sphere) (Mode_Lumineux nil) (Orbite_Etoile t) (Orbite_Planete nil) (Reaction_nucleaire normal) (Masse 1.5) (Vitesse_Rot 20001) (Effacer_Voisine t) (Horizon_evenement nil)(Position inconnu) (Signal_Impulsion nil) (Type inconnu)

)

Ici un BDF qui n'a pas de sens (Il n'y a pas de corps céleste avec ce genre de données dans le monde réel) , donc il n'y a pas de résultats qui peuvent inféré par le moteur.

```
Entrez votre choix de la chainage (avant ou arriere) :
avant

Erreur ! Donnees insuffisantes ou erreur !

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

4.4 Chainage arrière en largeur d'abord

4.4.1 Principe de fonctionement

Ce moteur d'inférence demande d'abord à l'utilisateur le type_BUT qui doit être interrogé, puis déduit les conditions requises en fonction de ce type. Puisque le moteur d'inférence est chainage arrière en largeur d'abord, chaque récursion obtiendra tout les groupes de conditions qui peut déduire type_BUT . Avant chaque récursion, le moteur d'inférence utilisera la fonction verifier_conditions pour confirmer s'il existe un groupe de conditions qui correspond à la base de faits initiale dans le groupe de conditions inversement déduit. S'il existe, il terminera directement la boucle et notifiera à l'utilisateur que le type_BUT a été confirmé avec succès.

4.4.2 Implémentation

```
(defun arriere_Largeur1 (list_cadidates bdf bdr)
    (if (null list_cadidates)
        (format t "~%On ne peut pas verifier le type BUT~%")
      (format t "~%On a les conditions maintenant :~%"))
    (dolist (x list_cadidates)     ;;ici on peut diviser les différentes groupes des conditions et les
                                   ;;vérifier successivement
      (format t "~S ~%----------------------------------------------------------~%" x)
      (if (equal (verifier_conditions x bdf) t)
          (progn
            (format t "Conditions verifiees avec succes !~%~%")
            (return-from arriere_Largeur1 nil)
          )
```

```
                (format t "et on n'a pas encore verifier les conditions dans le base de faits~%~%")
            )
        )
        (format t "~%************************************************************~%")
    (cond
        ((not list_cadidates) nil)
        (t (arriere_Largeur1 (flatten (mapcar #'(lambda(xx) (successeur xx bdr)) list_cadidates)) bdf bdr)
        )
    )
)

(defun arriere_Largeur (type_but bdf bdr)
   (if (equal (verifier_compatible bdf bdr) t)
         (arriere_Largeur1 (list (list (list 'Type type_but))) bdf bdr)
      (format t "~%Erreur ! Donnee invalide !~%")))
```

4.4.3 Exemple d'implémentation

<u>Les même BDFs que dans les exemples ci-dessus</u>

Exemple 1 : La terre

```
Entrez votre choix de la chainage (avant ou arriere) :
arriere


 Veuillez saisir le type de corps celeste que vous souhaitez confirmer :

planete_tellurique

On a les conditions maintenant :
((TYPE PLANETE_TELLURIQUE))
------------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On a les conditions maintenant :
((ETAT_MATIERE SOLIDE) (TYPE PLANETE))
------------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On a les conditions maintenant :
((ETAT_MATIERE SOLIDE) (FORME SPHERE) (REACTION_NUCLEAIRE NIL) (ORBITE_ETOILE T) (EFFACER_VOISINE T))
------------------------------------------------------
Conditions verifiees avec succes !
```

Exemple 2 : Pulsar

```
Entrez votre choix de la chainage (avant ou arriere) :
arriere


 Veuillez saisir le type de corps celeste que vous souhaitez confirmer :

pulsar

On a les conditions maintenant :
((TYPE PULSAR))
------------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On a les conditions maintenant :
((TYPE ETOILE_NEUTRON) (SIGNAL_IMPULSION PERIODIQUE))
------------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On a les conditions maintenant :
((SIGNAL_IMPULSION PERIODIQUE) (VITESSE_ROT RAPIDE) (FORME SPHERE) (MASSE (1.35 2.1)))
------------------------------------------------------
Conditions verifiees avec succes !


Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

Exemple 3 : cas non compatible

```
Entrez votre choix de la chainage (avant ou arriere) :
arriere


 Veuillez saisir le type de corps celeste que vous souhaitez confirmer :

etoile

Erreur ! Donnee invalide !

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

Exemple 4 : cas compatible mais pas de résultat

Ici ce n'est qu'une des tentatives , sur le BDF non compatible , il va toujours déduire "On ne peut pas vérifier le type BUT"

```
Entrez votre choix de la chainage (avant ou arriere) :
arriere


 Veuillez saisir le type de corps celeste que vous souhaitez confirmer :

etoile

On a les conditions maintenant :
((TYPE ETOILE))
-----------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On a les conditions maintenant :
((ETAT_MATIERE PLASMA) (FORME SPHERE) (MODE_LUMINEUX RADIATION))
-----------------------------------------------------
et on n'a pas encore verifier les conditions dans le base de faits


************************************************************

On ne peut pas verifier le type BUT

************************************************************

Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?
```

5 . Programme principal

Enfin, nous avons implémenté la partie principale, la fonction corps_celeste qui interagit avec l'utilisateur. Cette fonction initialisera automatiquement la base de faits et permettra d'exécuter plusieurs inférences lorsque l'utilisateur sélectionne le chaînage. En même temps, elle demandera à l'utilisateur s'il doit effectuer un classification d'un autre corps céleste à la fin.(c.-à-d. Réinitialiser la base de faits et répéter les étapes ci-dessus).

```
(defun corps_celeste ()
   (format t "~%*****************************************programme commencer*****************************************~%"
   (let ((programme 0))
    (while (equal programme 0)
     (init_BDF)
     (Transformation_donnee *BDF*)
     (let ((autrechoix 0) (votreChoix nil))
      (while (equal autrechoix 0)
       (format t "Entrez votre choix de la chainage (avant ou arriere) :~%")
       (setq votreChoix (read))
       (moteur_inference votreChoix *BDF* *BDR*)
       (format t "~%Voulez-vous appliquer l'autre chainage sur ce BDF (oui : 0 / non : 1)?~%")
       (setq autrechoix (read))
      )
     )
     (format t "~%Voulez-vous quitter le programme ? (Entrez 1 pour quitter , entrez 0 pour continuer)~%")
     (setq programme (read))
    )
   )
     (format t "~%*****************************************programme terminer*****************************************~%
)
```

IV - Conclusion

Dans ce tp, nous avons établi un système expert sur la classification des corps célestes Les deux moteurs d'inférence que nous utilisons ont leurs propres avantages et inconvénients.

Pour le moteur d'inférence de chainage avant en profondeur d'abord, il peut montrer clairement tous les chemins de résultats possibles et le processus de raisonnement spécifique de chaque étape.L'inconvénient est qu'il traversera toutes les règles possibles, donc la complexité en temps sera relativement élevée.

Pour le moteur d'inférence de chainage arrière en largeur d'abord, il peut vérifier si le type saisi par l'utilisateur peut correspondre au BDF à la vitesse la plus rapide, car tant que le groupe de conditions correspondant au BDF apparaît, le raisonnement se terminera directement, donc la complexité temporelle est faible.. Mais il ne peut être utilisé que pour tester si un certain type répond aux exigences, et il ne peut pas déduire directement de quel type est le corps céleste.

Sources :

https://fr.wikipedia.org/wiki/Objet_c%C3%A9leste
https://fr.wikipedia.org/wiki/%C3%89toile
https://fr.wikipedia.org/wiki/Trou_noir
https://fr.wikipedia.org/wiki/Types_de_plan%C3%A8tes