RAPPORT TP3 IA01

OUEDRAOGO Taoufiq, BRENA–LABEL Yannis

IA01
Automne 2021

Table des matières

INTRODUCTION :

Le but de ce TP était de mettre en place un système expert d'ordre 0+ en autonomie et en utilisant nos connaissances acquises tout le long des séances de TD.

Contexte
Vous êtes un étudiant en fin de cursus visant le diplôme d'ingénieur et vous êtes en stage de fin d'étude. Vous payez votre logement par vos propres moyens et étant en stage de fin d'étude votre salaire n'est pas encore très élevé et vous pouvez avoir besoin d'acheter un ordinateur performant pour votre futur emploie (ex: pour utiliser des logiciels de 3D gourmand en performance), passer le permis, rénover votre futur logement, etc. Il y a plusieurs types de crédit disponible pour vous aider, c'est pourquoi nous avons mis en place ce système expert pour vous aider à choisir le crédit adéquat à votre besoin et si vous pouvez en bénéficier.

Le crédit octroyé dépendra surtout de l'**objet de la demande, de la situation professionnelle** du demandeur et du **montant désiré**.

Un **crédit** est une avance d'argent conditionné. C'est une pratique qui est de plus en plus prisée. On en distingue plusieurs types en fonction de la durée, du montant mais aussi de l'objet :

- **Crédit à la consommation :** financer les dépenses de la vie courante, d'équipement... sont plutôt des crédits à court terme.
- **Crédit immobilier :** financer l'acquisition d'un terrain, logement, des travaux de rénovation ou d'aménagement. Sont plutôt à long terme (10 15 ans)

Ce crédit pouvant aller **jusqu'à 35000 euros**, est **remboursable en 3 ans**.

Problématisation :

Quel serait **l'utilité** d'un tel **SE** dans le **domaine de la finance ?** et plus précisément dans les systèmes d'attribution de crédit ?

Nous avons développé un **système expert permettant de définir les crédits les mieux adaptés à chaque situation pendant le processus d'instruction pour chaque dossier**.

Dans notre cas, le système expert propose pour une personne **le crédit qui convient** au regard de sa situation. Et donc a un **rôle crucial** dans la **réduction du délai de traitement** de **milliers de dossiers** ce qui peut s'avérer être très utile. Donc :

- Automatiser le traitement
- Plus de dossiers traités en moins de temps
- Affecter les ressources humaines à d'autres tâches

Connaissances nécessaires au SE :

On a introduit la variable *question* pour distinguer le caractère demandable ou non d'une variable.
Ainsi lorsque le système rencontre une variable associée à la valeur *question*, il posera une question à l'utilisateur pour renseigner la vraie valeur de la variable en question.

diplome_vise = ingenieur
motif = ordinateur, permis, amenagement, decoration, renovation, question
type = consommation, travaux
montant = [0, 35k], question
credit = credit_renouvelable1, pret_personnel1, credit_renouvelable2, pret_personnel2
revenu >=208, question
possible? = oui, non
complement = garant

source : **COFIDIS** –-> https://www.cofidis.fr
ONEY –-> https://www.oney.fr
WIKIPEDIA –> https://fr.wikipedia.org/wiki/Cr%C3%A9dit

Base de règles :

Règle : (nom (premisse1=val1 premisse2=val2 …) conclusion)

règle = ["R2" , ("premisse1 = valeur1") , "variable = valeur"]

- **une règle avec 2 conclusions -> 2 règles avec 1 conclusion**
  SI motif=ordinateur ALORS type=consommation ET montant=question
  -> SI motif=ordinateur ALORS type=consommation
  -> SI motif=ordinateur ALORS montant=question

- **une règle avec 2 prémisses en OU -> 2 règles avec 1 prémisse**
  SI motif=amenagement OU motif=decoration ALORS type=travaux
  -> SI motif=amenagement ALORS type=travaux
  -> SI motif=decoration ALORS type=travaux

SI diplome_vise=ingenieur **ALORS** motif=question

SI motif=ordinateur **OU** motif=permis **ALORS** type=consommation **ET** montant=question

SI motif=amenagement **OU** motif=decoration **OU** motif=renovation **ALORS** type=travaux **ET** montant=question

SI type=consomation **ET** montant<3000 **ALORS** credit=credit_renouvelable1 **ET** revenu=question

SI type=travaux **ET** montant<3000 **ALORS** credit=credit_renouvelable2 **ET** revenu=question

SI type=consomation **ET** montant>3000 **ALORS** credit=pret_personnel1 **ET** revenu=question

SI type=travaux **ET** montant>3000 **ALORS** credit=pret_personnel2 **ET** revenu=question

SI credit=pret_personnel2 **OU** credit=credit_renouvelable2 **ET** revenu>2430 **ALORS** possible?=oui

SI credit=credit_renouvelable1 **ET** revenu>208 **ALORS** possible?=non

SI credit=credit_renouvelable2 **ET** revenu>500 **ALORS** possible?=non

SI possible?=non **ALORS** complement=garant

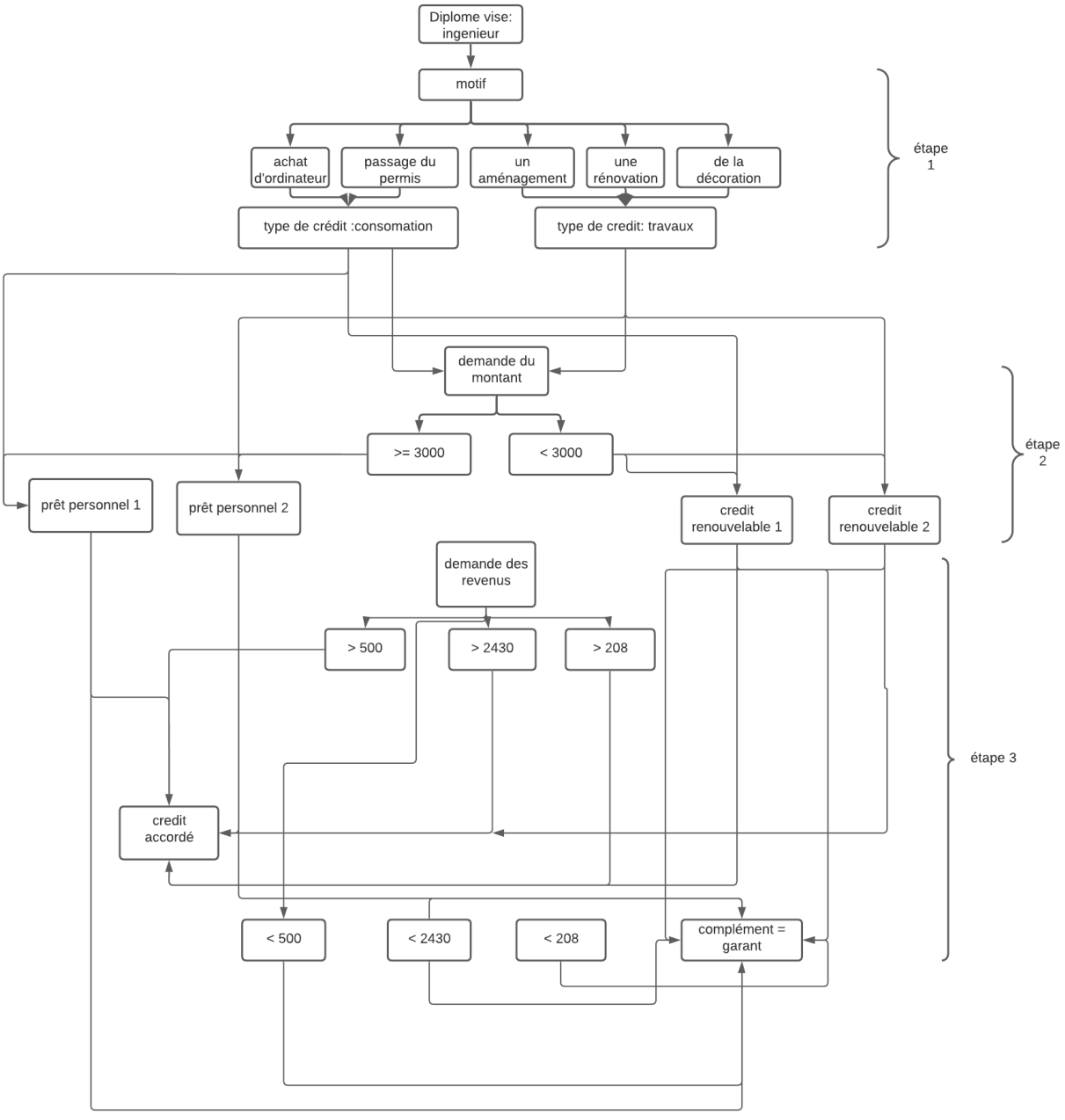
Lisp :

```
(setq BR '(
(R1     ((equal diplome_vise ingenieur))      (motif question))
(R2     ((equal motif ordinateur))            (type consommation))
(R3     ((equal motif permis))                (type consommation))
(R4     ((equal motif amenagement))           (type travaux))
(R5     ((equal motif renovation))            (type travaux))
(R6     ((equal motif decoration))            (type travaux))
(R7     ((equal type travaux))                (montant question))
(R8     ((equal type consommation))                   (montant question))
(R9     ((eq type consommation) (< montant 3000 ))          (credit credit_renouvelable1))
(R10    ((equal type travaux) (< montant 3000))             (credit credit_renouvelable2))
(R11    ((equal type consommation) (>= montant 3000))       (credit pret_personnel1))
(R12    ((equal type travaux) (>= montant 3000))            (credit pret_personnel2))
(R13    ((equal credit credit_renouvelable1))         (revenu question))
(R14    ((equal credit credit_renouvelable2))         (revenu question))
(R15    ((equal credit pret_personnel1))              (revenu question))
(R16    ((equal credit pret_personnel2))              (revenu question))
(R17    ((equal credit pret_personnel1) (>  revenu 500))              (possible? oui))
(R18    ((equal credit pret_personnel2) (>  revenu 2430))             (possible? oui))
(R19    ((equal credit credit_renouvelable1) (> revenu 208))          (possible? oui))
(R20    ((equal credit credit_renouvelable2) (> revenu 2430))                 (possible? oui))
(R21    ((equal credit credit_personnel1) (<  revenu 500))   (possible? non))
(R22    ((equal credit credit_personnel2) (<  revenu 2430))  (possible? non))
(R23    ((equal credit credit_renouvelable1) (< revenu 208))          (possible? non))
(R24    ((equal credit credit_renouvelable2) (< revenu 2430))         (possible? non))
(R25    ((equal possible? non))         (complement garant))
))
```

Python :

```
bdr = [["R1",("diplome_vise = ingenieur"),"motif = question"],
    ["R2",("motif = ordinateur"),"type = consommation"],
    ["R3",("motif = permis"),"type = consommation"],
    ["R4", ("motif = amenagement"), "type = travaux"],
    ["R5", ("motif = renovation"), "type = travaux"],
    ["R6", ("motif = decoration"), "type = travaux"],
    ["R7", ("type = consommation"), "montant = question"],
    ["R8", ("type = travaux"), "montant = question"],

    ["R9",  ("type = consommation","montant < 3000"),"credit = credit_renouvelable1"],

    ["R10", ("type = travaux","montant < 3000"),"credit = credit_renouvelable2"],

    ["R11", ("type = consommation","montant >= 3000"), "credit = pret_personnel1"],

    ["R12", ("type = travaux","montant >= 3000"), "credit = pret_personnel2"],

    ["R13", ("credit = credit_renouvelable1"), "revenu = question"],

    ["R14", ("credit = credit_renouvelable2"), "revenu = question"],

    ["R15", ("credit = pret_personnel1"), "revenu = question"],

    ["R16", ("credit = pret_personnel2"), "revenu = question"],

    ["R17", ("credit = pret_personnel1","revenu > 500"), "possible = oui"],

    ["R18", ("credit = pret_personnel2","revenu > 2430"), "possible = oui"],

    ["R19", ("credit = credit_renouvelable1","revenu > 208"), "possible = oui"],

    ["R20", ("credit = credit_renouvelable2","revenu > 2430"),"possible = oui"],

    ["R21",("credit = pret_personnel1","revenu < 500"),
        "possible = non"],


    ["R22",("credit = pret_personnel2","revenu < 2430"),
    "possible = non"],

    ["R23", ("credit = credit_renouvelable1","revenu < 208"),
    "possible = non"],

    ["R24", ("credit = credit_renouvelable2","revenu < 2430"),"possible = non"],

    ["R25", ("possible = non"),"complement=garant"]]
```

Arbre de déduction



Jeux d’essais

```
Bonjour quel est le motif de votre demande ?
Vous avez le choix entre
1: achat d'un ordinateur
2: passage du permis
3: un aménagement
4: une rénovation
5: pour de la décoration

Entrer le numéro correspondant à votre motif: 3

Quel est le montant que vous désirez obtenir: 5000

Entrer vos revenus mensuels : 1000
Dû à vos revenu vous aurez besoin d'un garant
Chemin parcouru:  ['R1', 'R4', 'R8', 'R12', 'R16', 'R22']
```

```
Bonjour quel est le motif de votre demande ?
Vous avez le choix entre
1: achat d'un ordinateur
2: passage du permis
3: un aménagement
4: une rénovation
5: pour de la décoration

Entrer le numéro correspondant à votre motif: 1

Quel est le montant que vous désirez obtenir: 1000

Entrer vos revenus mensuels : 300
Vous pouvez bénificier du  credit_renouvelable1
Chemin parcouru:  ['R1', 'R2', 'R7', 'R9', 'R13', 'R19']
```

```
Bonjour quel est le motif de votre demande ?
Vous avez le choix entre
1: achat d'un ordinateur
2: passage du permis
3: un aménagement
4: une rénovation
5: pour de la décoration

Entrer le numéro correspondant à votre motif: 4

Quel est le montant que vous désirez obtenir: 1000

Entrer vos revenus mensuels : 950
Dû à vos revenus vous aurez besoin d'un garant
Chemin parcouru:  ['R1', 'R5', 'R8', 'R10', 'R14', 'R24']
```

```
Bonjour quel est le motif de votre demande ?
Vous avez le choix entre
1: achat d'un ordinateur
2: passage du permis
3: un aménagement
4: une rénovation
5: pour de la décoration

Entrer le numéro correspondant à votre motif: 4

Quel est le montant que vous désirez obtenir: 2000

Entrer vos revenus mensuels : 2500
Vous pouvez bénificier du  credit_renouvelable2
Chemin parcouru:  ['R1', 'R5', 'R8', 'R10', 'R14', 'R20']
```

Moteur d'inférence

- **Chaînage avant en profondeur d'abord**

=> permet de trouver tous les crédits auxquels une personne serait éligible

(on applique les règles jusqu'à trouver le but)

Modèle intéressant car permet de visualiser la progression du raisonnement jusqu'au but.

Cependant, un grand nombre d'opérations (toutes les possibilités sont testées.

Pour réduire la complexité, on s'arrête lorsqu'une solution est trouvée.

Construit le raisonnement à partir de la BF

exemple :

(diplome_vise ingenieur) -> (eq motif question) poser question pour le motif -> question est remplacer par la réponse rentrée et devient (eq motif réponse) -> ajouter (eq motif réponse) à bdf -> prendre resultat (eq motif reponse) dans BR -> ajouter le resultat (eq type xxx) dans bdf -> (eq montant question) poser question pour montant -> remplacer question par reponse (eq montant reponse) -> ajouter à bdf -> chercher (eq type xxx) et le montant correspondant dans BR -> ajouter resultat (eq credit xxxx) à bdf -> poser question sur revenu (eq revenu question) -> remplacer question par reponse (eq revenu reponse)

- **Chaînage arrière (plusieurs problèmes rencontrés) , essais en lisp**

=> permet de trouver toutes les situations favorables à un crédit particulier

(on part du but et on utilise les règles pour remonter aux faits)

Des boucles infinies sur certaines règles (problèmes dans l'initialisation des faits dans la BF)

Utile car permet de retrouver les faits nécessaires pour atteindre un but

Développe que les règles ayant un lien avec le but (plus optimal?)

En python

Fonctions de service

```
10  def question_motif ():
11      motif = int(input("Bonjour quel est le motif de votre demande ?\n"
12                        "Vous avez le choix entre\n"
13                        "1: achat d'un ordinateur\n"
14                        "2: passage du permis\n"
15                        "3: un aménagement\n"
16                        "4: une rénovation\n"
17                        "5: pour de la décoration\n\n"
18                        "Entrer le numéro correspondant à votre motif: "))
19      if motif == 1:
20          bdf.append("motif = ordinateur")
21      if motif == 2:
22          bdf.append("motif = permis")
23      if motif == 3:
24          bdf.append("motif = amenagement")
25      if motif == 4:
26          bdf.append("motif = renovation")
27      if motif == 5:
28          bdf.append("motif = decoration")
29
```

- Demande à l'utilisateur quel est le motif de sa demande de crédit
- Puis l'ajoute à la base de fait

```
30  def question_montant ():
31      montant = float(input("Quel est le montant que vous désirez obtenir: "))
32      if  montant < 3000 :
33          bdf.append("montant < 3000")
34      elif   montant >= 3000:
35          bdf.append("montant >= 3000")
36
```

- Demande à l'utilisateur le montant de sa demande
- Le compare puis ajoute à la base de fait si il est plus grand ou plus petit que 3000 euros

```
def question_revenu ():
    revenu = float(input("Entrer vos revenus mensuels : "))
    if revenu > 500:
        bdf.append("revenu > 500")
    if revenu <500:
        bdf.append("revenu < 500")
    if revenu > 2430:
        bdf.append("revenu > 2430")
    if revenu < 2430:
        bdf.append("revenu < 2430")
    if revenu > 208:
        bdf.append("revenu > 208")
    if revenu < 208:
        bdf.append("revenu < 208")

```

- Demande à l'utilisateur ses revenus
- Et suite à la réponse son revenu est comparé aux différentes prémisses puis celle qui sont respecté sont ajoutées à la base de fait

```
def premisse_respecter1 (bdf,bdr):
    for i in bdr:
        for j in bdf:
            if j == premisse(i):
                bdf.append(ccl_regle(i))
                chemin.append(numRegle(i))
                break
    return bdf

```

- Permet de vérifier les prémisses qui contiennent une seule condition
- En parcourant la base de règle et en comparant chaque élément de la base de fait aux prémisses des différentes règles
- Et dès qu'une prémisse est respectée son résultat est ajouté à la base de fait ainsi que son numéro de règle pour le chemin parcouru

```
def premisse_respecter2 (bdf,bdr):
    fait = (bdf[2],bdf[3])          #fait est la premisse permettent de savoir crédit qui nous sera proposé
    for i in bdr:
        if fait == premisse(i):
            bdf.append(ccl_regle(i))
            chemin.append(numRegle(i))
    return bdf

```

- Permet de vérifier la première prémisse qui contient deux conditions dans la base de règle
- En prenant les éléments directement concerné dans la base de fait pour ne pas avoir à la parcourir

```
def premisse_respecter3 (bdf,bdr):
    c1 = (bdf[4],bdf[5])         #c1, c2 et c3 sont les 3 premisses qui vérifie si le credit est possible
    c2 = (bdf[4],bdf[6])
    c3 = (bdf[4],bdf[7])
    for i in bdr:                #on parcours la base de règle pour vérifier quel règle qu'une des prémisses présente dans la base de fait vérifie
        if c1 == premisse(i) or c2 == premisse(i) or c3 == premisse(i):
            bdf.append(ccl_regle(i))
            chemin.append(numRegle(i))
            break
    return bdf
```

- Permet de vérifier les éléments de la base de fait qui respecte les prémisses de la règle R17 à R24
- En assignant directement à c1, c2 et c3 les éléments concernés par les prémisses des règles 17 à 24

```
def bilan (bdf):
    for i in bdf:
        if i == "possible = non":
            bdf.append(ccl_regle(bdr[24]))
            print("Dû à vos revenus vous aurez besoin d'un garant")
        elif i == "possible = oui":
            for i in bdf:
                if i == "credit = credit_renouvelable1" or i == "credit = credit_renouvelable2" or i == "credit = pret_personnel1" or i == "credit = pret_perso
                    print("Vous pouvez bénificier du",i[8:])
```

- Cette fonction réalise le bilan en parcourant la base de fait et cherche si possible égale oui ou non pour conclure

Programme principal

```
def main():
    if premisse_respecter1(bdf, bdr)[1] == "motif = question" :
        del(bdf[1])                 #supprime "motif = question" pour le remplacer
        question_motif()
    premisse_respecter1(bdf, bdr)
    del(chemin[1])              #supprime la règle 1 qui est de nouveau ajouté au chemin
    del(bdf[2])                 #supprime "motif = question" qui est de nouveau ajouté
    for i in bdf:
        if i == "montant = question":
            del(bdf[-1])            #supprime "montant = question" pour le remplacer
            question_montant()
    premisse_respecter2(bdf, bdr)
    if premisse_respecter1(bdf, bdr)[-1] == "revenu = question":
        del(bdf[5:9])               #supprime "revenu = question" pour le remplacer
        question_revenu()           #et les résultats des règles déjà évaluées qui le sont de nouveaux dans la base de fait
    del(chemin[4:7])                #supprime les noms des règles déjà évaluées qui sont rajoutées de nouveaux dans le chemin parcouru
    premisse_respecter3(bdf, bdr)
    bilan(bdf)
    print("Chemin parcouru: ",chemin)
```

- La fonction principale utilise donc les différentes fonctions précédemment présenté dans l'ordre de la base de règle
- Et nous renvoie le bilan de notre demande et le chemin parcouru

Avantages/Inconvénients Python :

-meilleure visibilité et aspect graphique (interface) -structure plus modulable (plus facile d'ajouter des éléments) -meilleure maîtrise -des possibilités variés (liste[index]...)	- première expérience sur les SE -risques de déviation -redéfinition de toutes les fonctions de services -certaines fonctions lisp n'existant pas en python (exemple cdr...)

Limites :

- Prendre en compte plus de variables (mensualités, situations, frais, crédit en cours …)
- Domaine très vaste mais notre base de règle est limitée
- Amélioration possibles de l'interface
- Couplage possible avec un raisonnement à partir de cas

Plusieurs difficultés sont rencontrées lors de l'implémentation du SE notamment avec les parcours en chaînage arrière et avant en profondeur. C'est pour cela que nous nous sommes tournés vers Python. (fichier code lisp fournit)

Conclusion :

Dans ce TP, nous avons réalisé un SE sur un problème complexe, celui de l'attribution des crédits bancaires.

Nous avons restreint le domaine d'application de ce SE et avons omis certains aspects importants à prendre en compte lors des demandes de prêts (par exemple les frais de remboursement, les mensualités de remboursement…) mais tout de même les possibilités restent nombreuses.

Ce TP nous a permis de comprendre les réelles difficultés des développements de systèmes experts. En effet, nous avons désiré avoir un SE robuste, répondant aux besoins tout en étant adaptable et simple d'utilisation.
Le fait de l'avoir réalisé dans un langage autre que celui utilisé en td a été à la fois compliquée et enrichissant, ce qui nous servira d'expérience pour la suite de notre cursus.