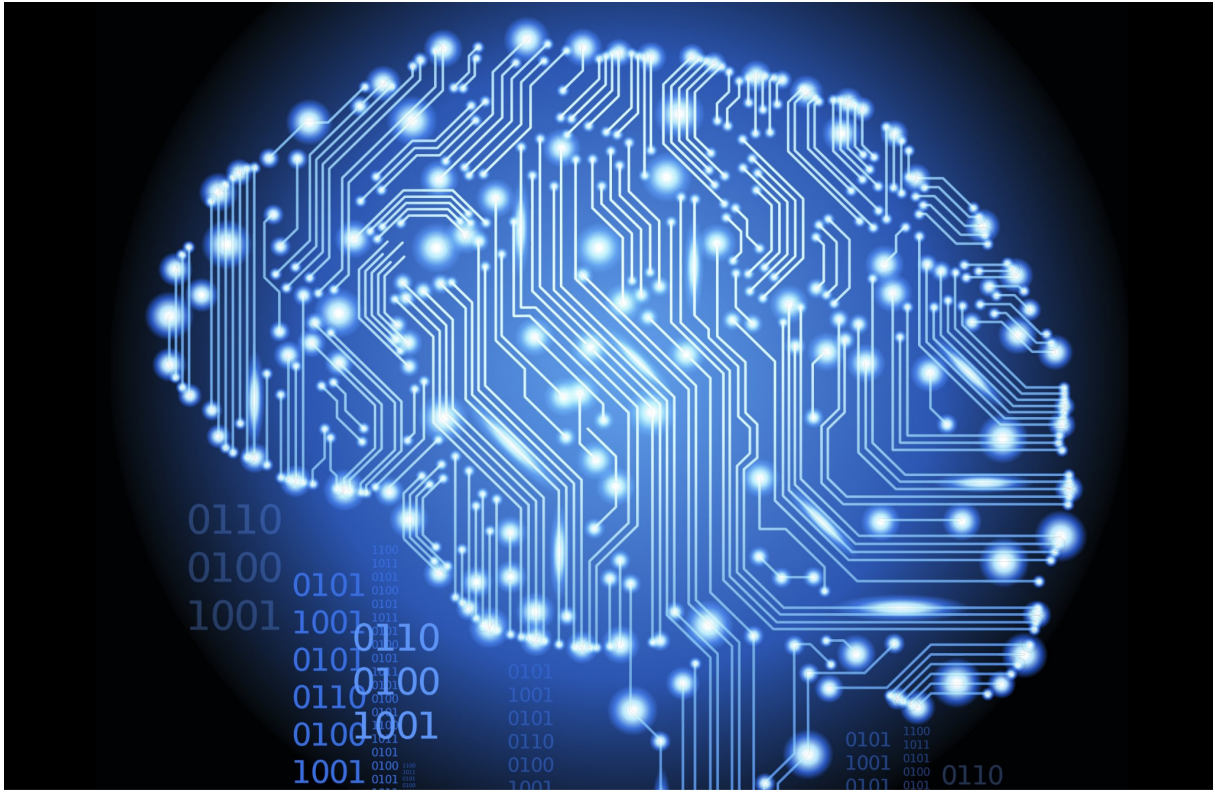


L021

RAPPORT PROJET A2023



Sommaire

I) Choix de conception

- 1) **Structure du projet**
- 2) **Structures des données**

II) Algorithmes des sous-programmes et explications

- 1) **Règles**
 - a) Création d'une règle vide
 - b) Ajout en queue d'une proposition à la prémisse d'une règle
 - c) Créer la conclusion d'une règle
 - d) Tester si une proposition appartient à la prémisse d'une règle de manière récursive
 - e) Supprimer une proposition de la prémisse d'une règle
 - f) Test si la prémisse d'une règle est vide
 - g) Accéder à la proposition se trouvant en tête d'une prémisse
 - h) Accéder à la conclusion d'une règle
- 2) **Base de connaissance**
 - a) Créer une base de connaissances vide
 - b) Ajouter une règle à une base (en queue)
 - c) Accéder à la règle se trouvant en tête de la base
- 3) **Moteur d'inférence**
 - a) Moteur d'inférence intermédiaire
 - b) Moteur d'inférence principal

III) Jeux d'essais

- 1) **Option 1 : Création d'une nouvelle base de connaissance**
- 2) **Option 2 : création d'une nouvelle liste de faits**
- 3) **Option 3 : Accéder au moteur d'inférence**
 - a) Afficher la base de faits
 - b) Afficher la base de connaissance
 - c) Exécuter le moteur d'inférence
 - d) Quitter

IV) Commentaires

I) Choix de conception

Pour notre projet, nous avons pris des dispositions afin d'organiser notre code de manière claire et facile à comprendre. Voici ce que nous avons décidé :

1. Structure du Projet :

Nous avons divisé notre projet en cinq parties, chacune avec son propre fichier. Chaque fichier a également un "header" qui lui est associé. Les fichiers "header" ont une double fonction. D'abord, ils contiennent les descriptions des fonctions présentes dans chaque fichier. De plus, ils spécifient ce que chaque fonction renvoie.

Ces parties sont :

base_de_fait : Contient des faits (proposition) qui sont certains

base_de_connaissances : Contient toute la logique du moteur d'inférence, c'est-à-dire toutes les règles.

moteur_inference : Effectue des déductions et des raisonnements et met donc à jour la base de faits.

menu : S'occupe de l'interaction avec l'utilisateur.

regle : Définit la logique du système avec des implications logiques.

En plus de cela, il y a un fichier **main** qui sert à tester les différentes fonctionnalités.

2. Structures des Données :

Nous avons créé trois façons de stocker les informations importantes pour que le système fonctionne bien :

- **Structure de Règle**

```
typedef struct Regle
{
    char* premisses; //sachan
    char* conclusion;
    struct Regle* next; // Po
}Regle;
```

- **Structure de Base de Connaissance**

```
typedef struct BC
{
    Regle* regle;
    struct BC* next;
}Elem_BC;
```

- **Structure de Base de Faits**

```
typedef struct Base_faits
{
    char* proposition; //sachan
    struct Base_faits *next;
}Elem_BF;
```

Pour plus de clarté, nous avons décidé de représenter la base de faits, la base de connaissances et les règles de manière emboîtée.

Base de connaissance :

Règle 1	Règle 2	Règle 3
<div>Conclusion</div> <div>Prémisse</div> <div>Proposition 1</div> <div>Proposition 2</div> <div>Proposition 3</div>	<div>Conclusion</div> <div>Prémisse</div> <div>Proposition 1</div> <div>Proposition 2</div> <div>Proposition 3</div>	<div>Conclusion</div> <div>Prémisse</div> <div>Proposition 1</div> <div>Proposition 2</div> <div>Proposition 3</div>

Base de fait :

Proposition 1
Proposition 2
Proposition 3
Proposition 4

II) Algorithmes des sous-programmes et explications

1) REGLES

a) Création d'une règle vide

Algorithme : creer_regle_vide

Données : \emptyset

Variables :

- newel : règle avec prémisse, conclusion et élément qui le suit

Résultat : newel : Règle

Algorithme :

Début

```
|  
|   premisses(newel) ← NULL  
|   conclusion(newel) ← NULL  
|   next(newel) ← NULL  
|   creer_Regle_Vide ← newel  
|
```

Fin

b) Ajout en queue d'une proposition à la prémisse d'une règle

Algorithme : ajout_proposition_regle

Données :

- Règle r

- Proposition X

Variables :

- newel : règle avec prémisse, conclusion et élément qui le suit

- temp : règle permettant le parcours de la liste de proposition

Résultat : Règle r avec la proposition ajouté en queue

Algorithme :

Début

```
|  
|   Si appartenance_premisse(r,X) = FAUX alors  
|       |  
|       |   newel ← creer_regle_vide()           // création d'une premisses d'une règle  
|       |   premisses(newel) ← X  
|       |   conclusion(newel) ← conclusion(r)  
|       |  
|       |   Si longueur(premisse(regle)) = 0 alors //longueur est une fonction qui renvoie 0  
|       |       |  
|       |       |   ajout_proposition_regle ← newel // s'il n y a pas de premisses dans une règle  
|       |   Sinon  
|       |       |  
|       |       |   temp ← r  
|       |
```

```

Tant que next(temp) ≠ indéfini faire
    temp ← next(temp)           // on cherche le dernier élément de la liste
    Fait
        next(temp) ← newel
        ajout_proposition_regle(r,X) ← r //on renvoie la règle avec l'élément ajouté en queue
    finSi
Sinon
    ajout_proposition_regle(r,X) ← r
finSi
Fin

```

c) Créer la conclusion d'une règle

Algorithme : créer_conclusion_regle

Données :

- Règle r,
- Proposition X

Variables : ∅

Résultat : Règle r

Algorithme :

Début

```

Si longueur(prémisse(r)) = 0 alors           //Erreur, on ne peut pas mettre de conclusion si il n y a pas
    créer_conclusion_regle(r,X) ← r           // de prémisse
Sinon
    conclusion(r) ← X
    créer_conclusion_regle(r,X) ← r
finSi

```

Fin

d) Tester si une proposition appartient à la prémisse d'une règle de manière récursive

Algorithme : appartenance_premisse

Données :

- Règle r,
- Proposition X

Variables :

- temp : règle permettant le parcours de la liste de proposition

Résultat : Booléen (Vrai ou Faux)

Algorithme :**Début**

Si r = indéfini **alors**

appartenance_premisse(r,X) \leftarrow FAUX

Sinon Si next(r) = indéfini **alors**

Si premisses(r) = X **alors**

appartenance_premisse(r,X) \leftarrow VRAI

Sinon

appartenance_premisse(r,X) \leftarrow FAUX

finSi

Sinon

Si appartenance_premisse(reste(r), X) = VRAI **alors**

appartenance_premisse(r,X) \leftarrow VRAI

Sinon

temp \leftarrow r

Tant que next(temp) \neq indéfini **faire**

temp \leftarrow next(temp)

fait

Si premisses(temp) = X **alors**

appartenance_premisse(r,X) \leftarrow VRAI

Sinon

appartenance_premisse(r,X) \leftarrow FAUX

finSi

finSi

finSi

Fin

e) Supprimer une proposition de la prémisse d'une règle

Algorithme : supprimer_proposition

Données :

- Règle r
- Proposition X

Variables :

- newel : règle avec prémisse, conclusion et élément qui le suit
- temp : règle permettant le parcours de la liste de proposition

Résultat : Règle r

Algorithme :

Début

Si appartenance_premisse(r,X) **alors**

 newel ← creer_regle_vide()

Si next(r) = indéfini **alors**

 liberer(r)

 appartenance_premisse(r,X) ← indéfini

Sinon

 temp ← r

Tant que temp ≠ indéfini **faire**

Si premisses(temp) ≠ X **alors**

 newel = ajout_proposition_regle(newel, premisses(temp))

finSi

 temp ← next(temp)

fait

 supprimer_proposition(r,X) ← newel

finSi

Sinon

 supprimer_proposition(r,X) ← r

finSi

Fin

f) Test si la prémisse d'une règle est vide

Algorithme : vide_premisse

Données :

- Premisse p

Variables : \emptyset

Résultat : Booléen (Vrai ou Faux)

Algorithme :

Début

Si longueur(p) = 0 **alors**

 vide_premisse \leftarrow VRAI

Sinon

 vide_premisse \leftarrow FAUX

finSi

Fin

g) Accéder à la proposition se trouvant en tête d'une prémisses

Algorithme : tete_premisse

Données :

- Règle r

Variables : \emptyset

Résultat : Prémisses

Algorithme :

Début

 tete_premisse(r) \leftarrow premisses(r)

Fin

h) Accéder à la conclusion d'une règle

Algorithme : conclusion_regle

Données :

- Règle r

Variables : \emptyset

Résultat : conclusion

Algorithme :

Début

|

2) BASE DE CONNAISSANCE

a) Créer une base de connaissances vide

Algorithme : creer_base_connaissances

Données : \emptyset

Variables :

- newel : liste de type base de connaissances

Résultat : newel

Algorithme :

Début

| regle(newel) \leftarrow creer_regle_vide()

| next(newel) \leftarrow NULL

| creer_base_connaissances() \leftarrow newel

Fin

b) Ajouter une règle à une base (en queue)

Algorithme : ajout_en_queue_bc

Données :

- liste : liste de type base de connaissances

- regle : Regle

Variables :

- newel : liste de type base de connaissances

- temp : liste de type de base de connaissances utilisée pour parcourir la liste des bases de connaissances

Résultat : newel, liste

Algorithme :

Début

| regle(newel) \leftarrow regle

| **Si** regle_vide(regle(liste)) **alors**

| | ajout_en_queue_bc(liste, regle) \leftarrow newel

| **Sinon**

| | temp \leftarrow liste

```

| | Tant que next(temp) indéfinie faire
| |   temp ← next(temp)
| | fait
| |   next(temp) ← newel
| |   ajout_en_queue_bc(liste,regle) ← liste
| finSi
Fin

```

c) Accéder à la règle se trouvant en tête de la base

Algorithme : tete_bc

Données :

- liste : liste de type base de connaissances

Variables : ∅

Résultat : regle(liste)

Algorithme :

Début

```

|   tete_bc ← regle(liste)
|

```

Fin

3) MOTEUR D'INFÉRENCE

a) Moteur d'inférence intermédiaire

Algorithme : Moteur inférence I

Données :

- liste_bf : liste de faits

- liste_bc : liste des bases de connaissances

Variables :

- actuel_bc : liste des bases de connaissances

- actuel_bf : liste de faits

- compteur : entier qui sera utilisé pour vérifier si chaque proposition de la base de faits appartient à la prémisse d'une règle

- longueur_regle : entier qui contient la longueur de la prémisse d'une règle

Résultat : liste_bf: Règle

Algorithme :**Début**

actuel_BC ← cree_base_connaissances()

Tant que actuel_BC ≠ indéfini **faire**

longueur_regle ← longueur_premisse(actuel_BC)

compteur ← 0

actuel_BF ← creer_bf_vide()

Tant que actuel_BF ≠ indéfini **faire**

Si appartenance_premisse(regle(actuel_bc), proposition(actuel_bf)) **alors**

compteur ← compteur + 1

finSi

actuel_bf ← next(actuel_bf)

Fait

Si compteur = longueur_regle **alors**

liste_bf ← ajout_en_queue_bf(liste_bf, conclusion(regle(actuel_bc)))

finSi

compteur ← 0

actuel_bc ← next(actuel_bc)

Fait

moteur_inference_l(liste_bf,liste_bc) ← liste_bf

Fin

b) Moteur d'inférence principal

Algorithme : moteur_inference_p**Données :**

- liste_bf : liste de faits

- liste_bc : liste des bases de connaissances

Variables :

- comparaison : Booléen (Vrai ou Faux)

- temp : liste_bf

Résultat : liste_bf: Règle

Algorithme :**Début**

comparaison ← FAUX

Tant que comparaison = FAUX **faire**

temp ← copieLlisteBF(liste_bf)
liste_bf ← moteur_inference_l(liste_bf, liste_bc)
comparaison ← Comparaison_BF (temp, liste_bf)

Fait

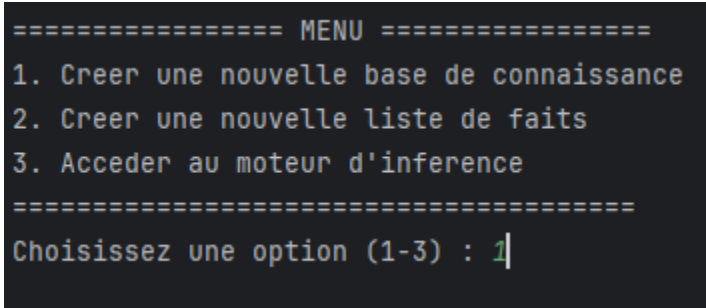
moteur_inference_p(liste_bf, liste_bf) ← liste_bf

Fin

III) Jeux d'essais

Au lancement du programme, un menu se présente à l'utilisateur, lui demandant ce qu'il veut faire, il a le choix entre créer une nouvelle base de connaissance (1), créer une nouvelle liste de faits (2) ou d'accéder au moteur d'inférence (3). L'utilisateur doit répondre une option entre 1 et 3 en fonction de son besoin

```
===== MENU =====  
1. Creer une nouvelle base de connaissance  
2. Creer une nouvelle liste de faits  
3. Acceder au moteur d'inferece  
=====
```



```
Choisissez une option (1-3) : 1|
```

1) Option 1 : Création d'une nouvelle base de connaissance

Tout d'abord, on doit entrer le nombre de règles qu'on veut créer. Ensuite, la création d'une règle se fait toujours de la même manière, on indique le nombre d'élément en comptant les différentes prémisses et la conclusion, c'est-à-dire que si on a une règle $A \text{ et } B \Rightarrow C$, alors on a 3 éléments, deux prémisses A et B et une conclusion C. Ensuite, on rentre les propositions, celles-ci doivent être un caractère compris entre A et Z et enfin on rentre la conclusion en dernier. La création de règles se termine quand toutes les règles ont été données.

```
Entrer le nombre de regles que vous voulez creer: 2

=====
  Creation d'une nouvelle regle
=====
Nombre d'elements dans la premisses et la conclusion : 3

=====
Saisie des propositions
Et de la conclusion (3 nombres elements au total)
=====
Proposition 1 : A
Proposition 2 : B
Conclusion : C

=====
  Nouvelle regle creee avec succes!
=====
```

2) Option 2 : création d'une nouvelle liste de faits

Au lancement de l'option 2, il sera demandé à l'utilisateur de rentrer le nombre d'élément qu'il y aura dans la liste de faits, ainsi si A est vrai et B est vrai, on aura 2 éléments dans cette liste. Ensuite, ces propositions devront être rentré par l'utilisateur une à une, celles-ci doivent être un caractère compris entre A et Z.


```

=====
Choisissez une option (1-3) : 2
=====
    Creation d'une nouvelle liste de faits
=====
Nombre d'elements dans la liste de faits : 2
=====
Saisie des propositions (Base de faits)
(2 nombres elements au total)
=====
Proposition 1 : A
Proposition 2 : B
=====
    Nouvelle liste de faits creee avec succes!
=====

```

3) Option 3 : Accéder au moteur d'inférence

Une fois que la base de connaissance et la liste de faits ont été correctement remplis, le moteur d'inférence nous propose 4 options.

```

=====
                Moteur inference
=====
1. Afficher la base de faits
2. Afficher la base de connaissance
3. Executer le moteur d'inference
0. Quitter
=====
Choix :1

```

a) Afficher la base de faits

Cette option affiche la base de faits préalablement rempli par l'utilisateur.

```
Base de faits :  
  
=====
Affichage de la liste de faits
=====
A -> B -> NULL
=====
Fin de la liste de faits
=====
```

b) Afficher la base de connaissance

Cette option affiche la base de connaissance préalablement rempli par l'utilisateur.

```
Choix :2  
  
Base de connaissance :  
  
=====
Affichage de la Base de Connaissances
=====
```

--- Regle 1 ---
A -> B -> || C ||

c) Exécuter le moteur d'inférence

Cette option permet de déduire tout les nouveaux fait certains en fonction de la base de fait initiale, ainsi que de la base de connaissance. Elle affichera d'abord la liste de faits avant que le moteur d'inférence agisse, puis après qu'il ait agit en incluant dans la liste finale tous les nouveaux faits certains.

```
Choix :3

Base de faits avant :

=====
Affichage de la liste de faits
=====
A -> B -> NULL
=====
Fin de la liste de faits
=====

Base de faits apres :

=====
Affichage de la liste de faits
=====
A -> B -> C -> NULL
=====
Fin de la liste de faits
=====
```

d) Quitter

Cette option arrête le programme.

IV) Commentaires

Nous pensons avoir répondu aux demandes de ce projet, et sommes très contents du résultat. Nous pouvons créer une liste de faits, une base de connaissance, et le moteur d'inférence fonctionne comme demandé : il ressort tous les faits en fonction de la liste de faits et de la base de connaissance. Néanmoins, il y a plusieurs détails qui aurait pu être améliorés comme une interface plus intuitive et plus esthétique et également le fait de pouvoir choisir autre chose qu'un caractère pour les éléments des différentes listes.

Ce projet a été une occasion précieuse pour approfondir nos connaissances acquises au cours de l'UV LO21 et les mettre en pratique dans un contexte réel, intégrant à la fois la programmation et l'algorithmie. Travailler en binôme a été une expérience enrichissante, favorisant la collaboration et l'entraide, renforçant ainsi notre efficacité collective. Nous avons pris plaisir à relever ce défi en équipe et sommes impatients de continuer à développer nos compétences dans des projets futurs.