

Projet LO21: Rapport final

Adrien Burgun

Automne 2020

Résumé

Le projet de ce semestre pour le cours de **LO21** (Algorithmique et Programmation II) porte sur un « *système expert* ». Un système expert est constitué de 3 éléments :

Une base de connaissance, qui prend la forme suivante :

$$A \wedge B \wedge \dots \wedge Z \Rightarrow \Omega$$

Où A, B, \dots sont les symboles (d'arité zéro, aussi appelés « propositions ») constituant la *prémisse* et Ω est la *conclusion*.

Une base de faits, qui est la liste des symboles ayant la valeur « *Vrai* » (qui correspond à l'état « *Certain* »).

Un symbole ne faisant pas partie de cette liste a par défaut la valeur « *Faux* » (qui correspond à l'état « *Incertain* »).

Un moteur d'inférence, qui, à partir de la base de connaissance et la base de faits, déduit quels autres symboles sont aussi vrais et les ajoute à la base de faits.

Nous définirons d'abords le type « *Règle* », constituant la base de connaissance. Nous définirons ensuite le type « *BC* » (Base de Connaissance).

Nous décrirons enfin le moteur d'inférence comme implémenté dans ce projet, avec différents exemples.

1 Règles

Soit *regle_t* le type représentant une règle sous la forme d'une liste chaînée de symboles :

L'implémentation faite dans ce projet utilise des noms anglais pour les variables, fonctions et types ; le type *Règle* a pour équivalent C le type *rule_t*.

Le dernier élément d'une telle liste chaînée correspond à la conclusion de la règle, tandis que tous les autres éléments appartiennent à la prémisse (contrainte du projet).

1.1 Créer une règle vide

Nous représenterons une règle vide par un pointeur nul. Voici l'algorithme permettant de créer une règle vide :

Algorithme 1 : NouvelleRègle	
Variables : R : La règle vide à retourner	
Résultat : R : Règle	
1	Début NouvelleRègle()
2	$R \leftarrow \text{NULL}$
3	Fin

1.2 Créer un noeud d'une règle

Voici la fonction permettant de créer un noeud (ou symbole) dans une règle :

Algorithme 2 : CréerNoeud	
Variables : R : Le noeud de la règle à retourner $symbole$: La valeur de $symbole$ à mettre dans le noeud $suivant$: Pointeur sur le prochain noeud de la liste	
Données : $symbole$: Chaîne de caractères; $suivant$: Règle	
Résultat : R : Règle	
1	Début CréerNoeud($symbole$, $suivant$)
	// Nous supposons que l'espace de mémoire requis pour R est déjà alloué
2	$(R \rightarrow symbole) \leftarrow symbole$
3	$(R \rightarrow suivant) \leftarrow suivant$
4	Fin

1.3 Ajouter une proposition à la prémisse d'une règle

L'ajout des propositions (symboles) à la prémisse d'une règle se fait par l'algorithme *AjoutPrémisse* défini ci-dessous. Cet ajout se fait en queue de la liste chaînée (contrainte

du projet). La liste chaînée donnée en entrée est modifiée par l'algorithme et est retournée.

Algorithme 3 : AjoutPrémisse

Variables : R : La règle à modifier
 R' : Une variable temporaire pour traverser la liste chaînée
 $symbole$: Le nom de la proposition (symbole) à insérer
Données : R : Règle, $symbole$: Chaîne de caractères
Résultat : R : Règle

```

1 Début AjoutPrémisse( $R$ ,  $symbole$ )
2   Si  $R = NULL$  alors
3      $R \leftarrow$  CréerNoeud( $symbole$ ,  $NULL$ )
4   Sinon
5      $R' \leftarrow R$ 
6     // Répéter jusqu'à ce qu'on atteigne le dernier élément
7     Tant que ( $R' \rightarrow next$ )  $\neq NULL$  faire
8        $R' \leftarrow (R' \rightarrow next)$ 
9     Fin
10    //  $R'$  contient désormais le dernier élément de la liste
11    ( $R' \rightarrow suivant$ )  $\leftarrow$  CréerNoeud( $symbole$ ,  $NULL$ )
12  FinSi
13 Fin
```

1.4 Créer la conclusion d'une règle

Créer la conclusion d'une règle revient à ajouter une proposition (symbole) à la fin de la règle. Pour ce faire, nous ré-utilisons l'algorithme [AjoutPrémisse](#) défini plus tôt.

Algorithme 4 : AjoutConclusion

Variables : R : La règle à modifier
 $symbole$: Le nom de la proposition (symbole) à insérer comme conclusion
Données : R : Règle, $symbole$: Chaîne de caractères
Résultat : R : Règle

```

1 Début AjoutConclusion( $R$ ,  $symbole$ )
2    $R \leftarrow$  AjoutPrémisse( $R$ ,  $symbole$ )
3 Fin
```

1.5 Tester si une proposition appartient à la prémisse d'une règle

Nous testons si une proposition appartient à la prémisse d'une règle en traversant celle-ci de manière récursive (contrainte du projet).

Les 3 cas minimaux sont :

$R = []$ (règle vide) : retourner « Faux »

$R = \{\text{symbole} : "...", \text{suivant} : \text{NULL}\}$ (conclusion) : retourner « Faux »
 $R = \{\text{symbole} : \text{symbole_recherché}, \text{suivant} : ...\}$ (symbole trouvé) : retourner « Vrai »

Dans les autres cas, nous retournons de manière récursive le résultat de la même fonction, appelée sur $(R \rightarrow \text{suivant})$.

Algorithme 5 : TestAppartenance	
	Variables : R : La règle à étudier <i>symbole</i> : La nom de la proposition à rechercher <i>résultat</i> : Si oui ou non la proposition à été trouvée Données : R : Règle, <i>symbole</i> : Chaîne de caractères Résultat : <i>résultat</i> : Booléen
1	Début TestAppartenance($R, \text{symbole}$)
2	Si $R = \text{NULL}$ alors
	// Règle vide
3	<i>résultat</i> \leftarrow Faux
4	Sinon si $(R \rightarrow \text{suivant}) = \text{NULL}$ alors
	// Conclusion
5	<i>résultat</i> \leftarrow Faux
6	Sinon si $(R \rightarrow \text{symbole}) = \text{symbole}$ alors
	// Symbole trouvé
7	<i>résultat</i> \leftarrow Vrai
8	Sinon
9	<i>résultat</i> \leftarrow TestAppartenance($(R \rightarrow \text{suivant}), \text{symbole}$)
10	FinSi
11	Fin

1.6 Supprimer une proposition de la prémisse d'une règle

Nous supprimons une proposition de la prémisse d'une règle de manière récursive. Les cas minimaux sont les suivants :

$R = []$ (règle vide) : retourner NULL
 $R = \{\text{symbole} : "...", \text{suivant} : \text{NULL}\}$ (conclusion) : retourner R

Dans le cas général, nous attribuons à $(R \rightarrow \text{suivant})$ la valeur retournée par cette fonction, appelée sur $(R \rightarrow \text{suivant})$, et nous retournons $(R \rightarrow \text{suivant})$ si le noeud

correspond au symbole et R sinon.

Algorithme 6 : SupprimerSymbole

Variables : R : La règle à modifier

$symbole$: La nom de la proposition à rechercher

R' : La règle privée de $symbole$ dans sa prémisses

Données : R : Règle, $symbole$: Chaîne de caractères

Résultat : R' : Règle

```
1 Début SupprimerSymbole( $R$ ,  $symbole$ )
2   Si  $R = NULL$  alors
3     // Règle vide
4      $R' \leftarrow NULL$ 
5   Sinon si  $(R \rightarrow suivant) = NULL$  alors
6     // Conclusion
7      $R' \leftarrow R$ 
8   Sinon
9     Si  $(R \rightarrow symbole) = symbole$  alors
10      // Retourner le reste de la liste, sans ce noeud
11       $R' \leftarrow SupprimerSymbole((R \rightarrow suivant), symbole)$ 
12    Sinon
13       $(R \rightarrow suivant) \leftarrow SupprimerSymbole((R \rightarrow suivant), symbole)$ 
14       $R' \leftarrow R$ 
15    FinSi
16  FinSi
17 Fin
```

1.7 Tester si la prémisse d'une règle est vide

Voici la fonction retournant « Vrai » si la prémisse d'une règle est vide et « Faux » si la prémisse d'une règle contient au moins 1 symbole :

Algorithme 7 : PrémisseVide

```
Variables :  $R$ : La règle à étudier  
 $résultat$ : Si oui ou non la prémisse d'une règle est vide  
Données :  $R$ : Règle  
Résultat :  $résultat$ : Booléen  
1 Début PrémisseVide( $R$ )  
2   Si  $R = NULL$  alors  
3     // La règle est vide, donc sa prémisse est vide  
4      $résultat \leftarrow$  Vrai  
5   Sinon si  $(R \rightarrow suivant) = NULL$  alors  
6     // La règle n'a qu'une conclusion, donc sa prémisse est vide  
7      $résultat \leftarrow$  Vrai  
8   Sinon  
9      $résultat \leftarrow$  Faux  
10  FinSi  
11 Fin
```

1.8 Accéder à la proposition se trouvant en tête d'une prémisse

Voici la fonction retournant la valeur de la proposition se trouvant en tête d'une prémisse. Si la prémisse est vide, alors la fonction retourne NULL.

Algorithme 8 : PremierSymbole

```
Variables :  $R$ : La règle à étudier  
 $résultat$ : La valeur du premier symbole de la prémisse, si existant  
Données :  $R$ : Règle  
Résultat :  $résultat$ : Chaîne de caractères  
1 Début PremierSymbole( $R$ )  
2   Si PrémisseVide( $R$ ) alors  
3     // La prémisse est vide: nous retournons NULL  
4      $résultat \leftarrow$  NULL  
5   Sinon  
6      $résultat \leftarrow (R \rightarrow symbole)$   
7   FinSi  
8 Fin
```

1.9 Accéder à la conclusion d'une règle

La conclusion se trouvant à la fin d'une règle, nous traversons simplement la liste chaînée jusqu'au dernier élément de celle-ci. Si la liste est vide, alors la fonction retourne NULL.

Algorithme 9 : ConclusionRegle

```
Variables :  $R$ : La règle à étudier  

 $R'$ : Variable temporaire pour traverser la liste chaînée  

résultat: La valeur du premier symbole de la prémisse, si existant  

Données :  $R$ : Règle  

Résultat : résultat: Chaîne de caractères  

1 Début ConclusionRegle( $R$ )  

2   Si ( $R \rightarrow \text{suivant}$ ) = NULL alors  

3     | résultat  $\leftarrow$  NULL  

4   Sinon  

5     |  $R' \leftarrow R$   

6     | // Avancer jusqu'à la fin de la liste  

7     | Tant que ( $R' \rightarrow \text{suivant}$ )  $\neq$  NULL faire  

8     | |  $R' \leftarrow (R' \rightarrow \text{suivant})$   

9     | Fin  

10    | résultat  $\leftarrow (R' \rightarrow \text{symbole})$   

11  FinSi  

12 Fin
```