Compte rendu du TD n°5
UV IA01

I – Ordres

Un moteur d'ordre 0 ne traite que des faits pouvant avoir des valeurs booléennes :

Si beautemps = vrai alors …

Un moteur d'inférence d'ordre 0+ fait intervenir des faits pouvant avoir des valeurs :

Si temps = 'beau' alors …

Un moteur d'ordre 1 peut analyser des variables :

Si pere x = père y alors …

II – Chaînage avant et chaînage arrière

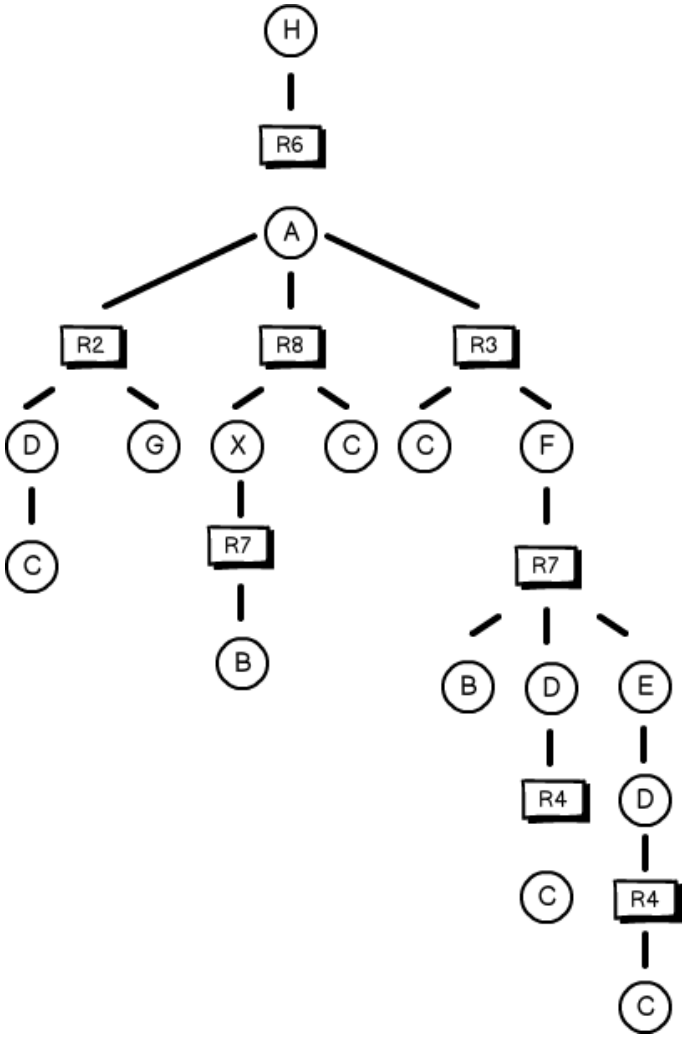
A – Chaînage avant

On part des règles, et on les explore toutes pour tenter d'arriver à un but.

Règles candidates	**Choix de la règle**	**BF**
R4, R7	On prend toujours la première, ici R4	{ B, C }
R5, R7	-	{B , C , D}
R1, R7	-	{B , C , D, E}
R3, R7	-	{B , C , D, E, F}
R6, R7	-	{ B , C , D, E, F , A}
-	-	{ B , C , D, E, F , A, **H**}

B – Chaînage arrière

On part du but, et on regarde quelles règles on a pu appliquer pour arriver à ce but. En descendant de cette manière, si on arrive aux faits présents dans notre base, alors on sait qu'on a pu arriver à notre but.



III – Représentation

Représentation de la base de faits : (B C)

Représentation des règles :

(

(R1 (B D E) (F))

(R2 (D G) (A))

(R3 (C F) (A))

(R4 (C) (D))

(R5 (D) (E))

(R6 (A) (H))

(R7 (B) (X))

(R8 (X C) (A))

)

IV – Algorithmes

Nous allons tenter de créer un système qui à partir d'une base de faits, peut dire si il est possible d'en déduire un fait H.

Nous allons voir les deux fonctions principales, puis passerons rapidement les fonctions mineures.

A – Fonctions principales

Fonction verifier (but)

Début

Ok <- faux

Si vrai (but) alors ok <- vrai

Sinon

EC <- regles_candidates(but)

Pour chaque règle R de EC et tant que ok = faux

Ok <- verifier_et (R)

Fin pour

Si ok = faux alors ok <- question (but) Fin si

Retourner ok

Fin

Fonction verifier_et (R)

Début

Ok <- vrai

Prem -> premisses (R)

Pour chaque P de prem et tant que ok = vrai

Ok <- verifier (P)

Fin pour

Retourner ok

Fin

B – Fonctions auxiliaires

Fonction question (But) : Demande à l'utilisateur si But fait partie de la base de faits. Retourne vrai ou faux.

Fonction vrai (But) : Retourner vrai si But appartient à la base de faits, faux sinon.

Fonction premisses(R) : Retourner les prémisses de la règle R

V – Pour aller plus loin

Nous allons implémenter l'algorithme ci-dessus.

```
(defun verifier (But)
	(let ((ok nil) (regles-candidates ()))
		(if (vrai But)
			(setf ok T)
			(progn
				(setf regles-candidates (regles_candidates But))
				(while (and regles-candidates (not ok))
					(setf ok (verifier-et (car regles-candidates)))
					(pop regles-candidates)
				)
				(if (equal ok nil) (setf ok (question But)))
			)
		)
		ok
	)
)
```

```
(defun verifier-et (R)
	(let ((ok T) (prem (premisses R)))
		(while (and prem ok)
			(setf ok (verifier (car prem)))
			(pop prem)
		)
		ok
	)
)
```

```
(defun vrai (But)
	(if (member But BF)
		T
		nil
	)
)
```

```
(defun question (But)
	(princ (concatenate 'string "Est ce que " (symbol-name But) " est un
fait qui appartient a la base de faits ? (T ou nil)"))
	(setq resultat (read))
)
```

```
(defun premisses (R)
	(cadr R)
)
```

```
(defun regles_candidates (But)
      (let ((regles-candidates ()))
            (dolist (r regles (reverse regles-candidates))
                  (if (equal (caddr r) (list But))
                        (push r regles-candidates )
                  )
            )
      )
)
```

Règles :

```
(defvar regles '(
      (R1 (B D E) (F))
      (R2 (D G) (A))
      (R3 (C F) (A))
      (R4 (C) (D))
      (R5 (D) (E))
      (R6 (A) (H))
      (R7 (B) (X))
      (R8 (X C) (A))
))
```

Base de faits :

```
(defvar BF '(B C))
```

Testons tout cela en commençant par demander si H est déductible.

En nous référant à l'arbre ci-dessus (branche du milieu), nous pouvons en déduire qu'en répondant 'Non' à toutes les questions du système expert, ce dernier doit tout de même pouvoir en déduire que H est vrai car il peut trouver B et C qui sont bien dans notre base de faits.

```
CG-USER(3): (verifier 'H)
Est ce que G est un fait qui appartient a la base de faits ? (T ou nil)nil
T
```

Faisons un autre test. En modifiant la base de faits comme ci-dessous :

```
(defvar regles '(
      (R1 (B Z) (H))
))
```

Le système devrait demander si Z est dans la base de faits. Si l'utilisateur répond oui, alors il devrait retourner vrai, sinon si la réponse est non, alors en derniers recourt il devrait demander si H (qui est la déduction recherchée) est bien dans la base de faits. C'est ce qu'on appelle un système expert, car il s'adapte en fonction de la base de faits et demande des informations à l'utilisateur.

Vérifions :

```
CG-USER(1): (verifier 'H)
Est ce que Z est un fait qui appartient a la base de faits ? (T ou nil)nil
Est ce que H est un fait qui appartient a la base de faits ? (T ou nil)nil
NIL
CG-USER(2): (verifier 'H)
Est ce que Z est un fait qui appartient a la base de faits ? (T ou nil)T
T
```