Compte Rendu TP03 – Système expert d'ordre 0+

Problématique

L'objet du TP03 est de réaliser le développement d'un SE d'ordre 0+ de sa phase d'expertise à sa phase d'utilisation.

Vous êtes nouveau à l'UTC et arrivez au Pic'Asso avec des amis. En tant que jeune étudiant, vous ne connaissez pas encore toutes les bières qui existent au Pic'Asso. Pour vous aider dans votre choix, un Système Expert vous est proposé.

Suite à l'étude des préférences de l'étudiant, le Système Expert lui proposera la bière la plus adaptée pour lui.

Précisions

Nous avons choisi d'intégrer à notre système expert toutes les bières du pic, mais aussi des bières qui ont été proposées lors d'évènements comme le WEI ou l'Estu Parking pour plus de diversité. En effet, le pic regorge de bières blondes par exemple, mais manque de diversité au niveau des autres bières.

Base de règles

Catalogue de bières

A l'aide de sources d'expertise fiables que sont le site www.saveur-biere.com, site fournisseur de bières répertoriant les caractéristiques de celles-ci, et les sites de brasserie comme www.brouwerijhuyghe.be, nous avons établi un tableau comprenant les 21 bières choisies pour ce système expert ainsi que toutes leurs caractéristiques. A savoir que l'amertume d'une bière est pour notre part une échelle entre 1 et 5, de peu amer à très amer.

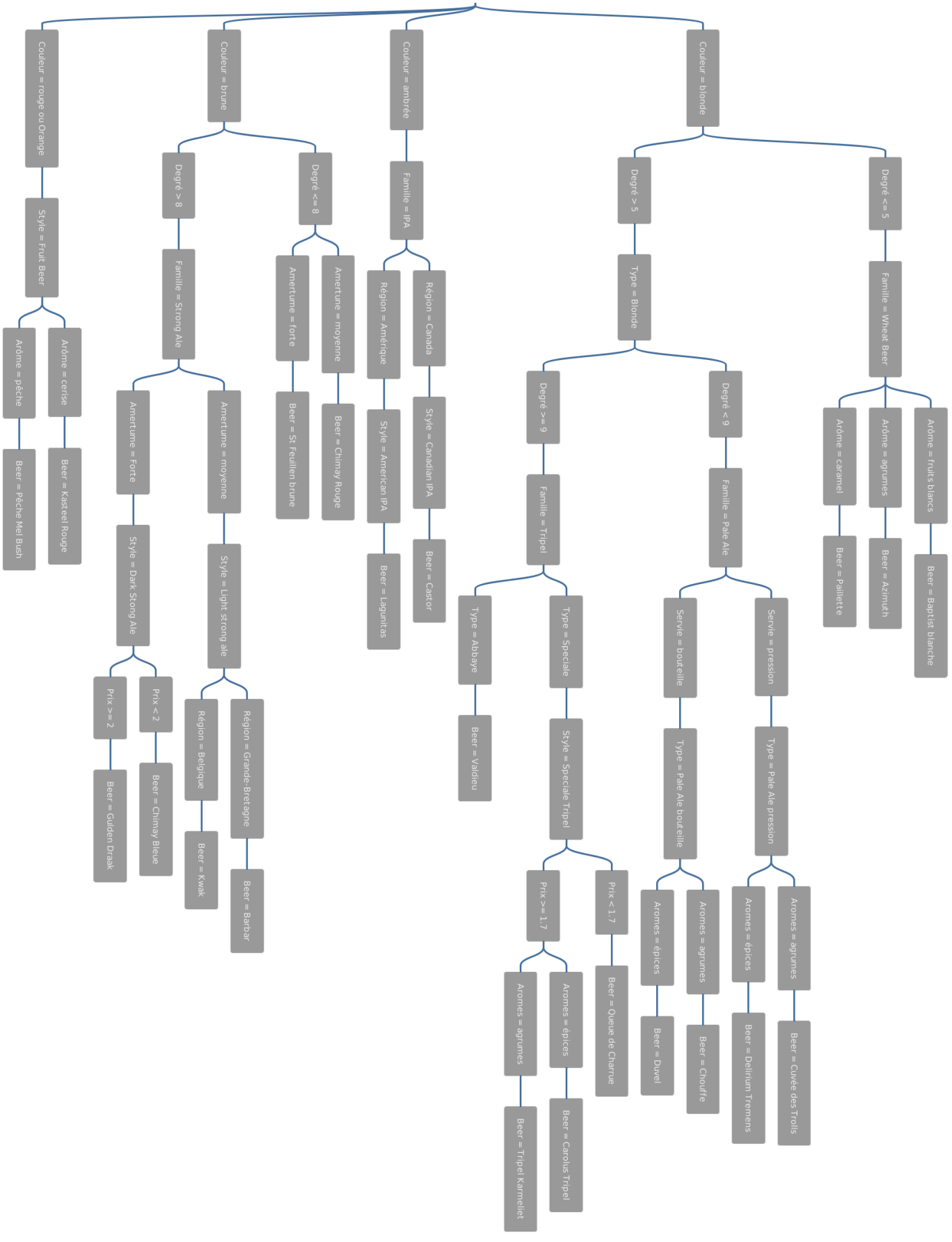
Nom bière	Degré	Prix	Région	Couleur	Amertume	Aromes	Style bière	Type	Famille de style	Servie
Lagunitas	6.2	1.65	Americaine	Ambrée	4	Agrumes	American ipa		IPA	Bouteille
Castor	6	1.55	Canada	Ambrée	3	Agrumes	Canadian IPA		IPA	Pression
Baptist blanche	5	1.5	Belgique	Blanche	1	Fruit blanc	wheat beer		Wheat Beer	Bouteille
Azimuth	4.5	1.4	Francaise	Blanche	1	Agrumes	wheat beer		Wheat Beer	Bouteille
Paillette	4.7	1.3	Francaise	Blanche	1	Caramel	Wheat beer		Wheat Beer	Bouteille
Chouffe	8	1.70	Belgique	Blonde	3	Agrumes	Belgian pale ale	Garde	Pale ale	Bouteille
Duvel	8.5	1.75	Belgique	Blonde	3	Epices	Belgian pale ale	Speciale	Pale ale	Bouteille
Delirium Tremen	8.5	1.8	Belgique	Blonde	2	Epices	Belgian pale ale	Speciale	Pale ale	Pression
Cuvée des Trolls	7	1.75	Belgique	Blonde	2	Agrumes	Belgian pale ale		Pale ale	Pression
Triple Karmeliet	9	1.8	Belgique	Blonde	2	Agrumes	Tripel	Speciale	Tripel	Pression
Carolus tripel	9	1.8	Belgique	Blonde	2	Epices	Tripel	Speciale	Tripel	Bouteille
Queue de charru	9	1.6	Belgique	Blonde	2	Caramel	Tripel	Speciale	Tripel	Pression
Valdieu	9	1.5	Belgique	Blonde	2	Epices	Tripel	Abbaye	Tripel	pression
Chimay rouge	7	1.7	Belgique	Brune	2	Caramel	Dubbel	Trappiste	Dubbel	pression
St Feuillen brune	8	1.85	Belgique	Brune	4	Caramel	Dubbel		Dubbel	Bouteille
Kwak	8.4	1.85	Belgique	Brune	2	Caramel	Light strong ale	Speciale	Strong ale	pression
Gulden Draak	10.7	2	Belgique	Brune	4	Cacao	Dark strong ale	Speciale	Strong ale	Bouteille
Chimay bleue	9	1.85	Belgique	Brune	4	Epices	Dark strong ale	Trappiste	Strong ale	Bouteille
Barbar	8.5	1.6	Belgique	Brune	2	Epices	Light strong ale	Bock	Strong ale	pression
Peche mel	8.5	1.85	Belgique	Fruitée	1	Peche	Fruit beer	Speciale	Fruit beer	Bouteille
Kastel Rouge	8	1.8	Belgique	Rubis	1	Cerise	Fruit beer	Speciale	Fruit beer	pression

L'objectif de ce tableau est de mettre en valeur des caractéristiques précises pour pouvoir différencier chaque bière. Ainsi, chaque requête aboutira toujours à une unique proposition.

Arbre

Nous avons ainsi créé un arbre avec un chemin menant à chaque bière. L'arbre est exhaustif et non redondant de par la représentation de toutes les bières, le fait qu'un chemin mène à une unique bière et que tous les chemins mènent à une bière.

L'arbre de toutes les bières répertoriées est présenté ci-dessous.



Représentations des objets

Pour implémenter notre système expert, nous avons dû définir une syntaxe précise pour la base de règles, la base de fait et la base de questions qui seront posées à l'utilisateur.

Règles :

Syntaxe :

(((prémisse 1)(prémisse 2))(conclusion))

Exemple :

(((eq COULEUR blonde)(<= DEGRE 5))(eq FAMILLE wheat_bear))

Notes : Les règles sont écrites au format préfixé avec les primitives LISP pour pouvoir être évaluées facilement. Lorsqu'une règle contient plusieurs prémisses, elles doivent toutes être vérifiées pour aboutir à la conclusion (opérateur ET logique). Pour l'opérateur OU logique, il suffit de décomposer en plusieurs règles distinctes.

Faits :

Syntaxe :

((CLE1 valeur1)(CLE2 valeur2) (CLE3 valeur3)

Exemple :

((COULEUR rouge)(STYLE fruit_beer) (AROME cerise)(BEER "Kasteel Rouge"))

Notes : Chaque fait est construit sous la forme d'un tuple "Clé-Valeur" qui est pertinent dans un système d'ordre 0+.

Question :

Syntaxe :

(NUMERO "question à afficher" CLE (Correspondances réponses possibles))

Exemple :

(Q9 "Préférez-vous une bière anglaise (0) ou Belge (1) ?" REGION ((0 gb) (1 belgique)))

Notes : La plupart des questions correspond à cette forme type "question à choix multiples" : l'utilisateur a simplement à saisir un nombre correspondant à une des réponses. Cette construction a pour avantage de limiter les erreurs de saisie et de pouvoir créer plusieurs questions identiques en faisant varier les réponses en fonction des faits déjà saisies par l'utilisateur. Pour les variables DEGRE et PRIX, l'utilisateur rentre simplement une valeur de son choix.

Gestion des questions

Nous avons défini dans une liste *questions* toutes les questions possibles qui pourront être posées à l'utilisateur. La fonction getQuestion, permet de renvoyer la question voulue selon la prémisse connue et la variable inconnue que nous recherchons. En effet, il existe deux types de questions pour la région et l'arôme selon l'endroit de l'arbre où on se situe. Pour les autres variables inconnues, on

cherche à faire correspondre la variable de la question (récupérée grâce à la fonction getVariable) et la variable inconnue.

La fonction askQuestion récupère en paramètre la prémisse connue et la prémisse inconnue, nous permettant de récupérer la bonne question avec la fonction getQuestion, celle-ci est ensuite posée à l'utilisateur, on push alors dans la base de faits la correspondance de la réponse entrée par l'utilisateur et la vraie valeur. Cependant, si la question concerne le degré ou le prix, on push directement la réponse de l'utilisateur dans la base de faits.

Fonctions de service

Pour accéder plus facilement à des éléments des bases et pour faciliter la lecture du code, nous avons implémenté quelques fonctions de services simples :

- getVariable : retourne la variable concernée par une question, exemple : getVariable('(Q9 "Préférez-vous une bière anglaise (0) ou Belge (1) ?" REGION ((0 gb) (1 belgique)))) retourne REGION.
- responsePossible : retourne l'ensemble des correspondances entre le nombre entré par l'utilisateur et la réponse souhaitée à rentrer dans la base de fait, exemple : responsePossible('(Q9 "Préférez-vous une bière anglaise (0) ou Belge (1) ?" REGION ((0 gb) (1 belgique)))) retourne ((0 gb) (1 belgique)).
- getPremisses : retourne l'ensemble des prémisses nécessaires pour évaluer une règle, exemple : getPremisses('(((eq COULEUR blonde)(<= DEGRE 5))(eq FAMILLE wheat_bear))) retourne ((eq COULEUR blonde)(<= DEGRE 5)).
- getConclusion : retourne la conclusion d'une règle, exemple getConclusion('(((eq COULEUR blonde)(<= DEGRE 5))(eq FAMILLE wheat_bear))) retourne (eq FAMILLE wheat_bear).
- trueRule : s'assure que l'ensemble des prémisses d'une règle sont vérifiées dans la base de fait (grâce à un funcall) et renvoie True si c'est le cas, False sinon.

Moteur d'inférence

Le but étant de déterminer quelle est la bière la plus adaptée aux goûts saisis par l'utilisateur, nous avons choisi de réaliser un moteur d'inférence à chaînage avant. Le principe est de réaliser un moteur capable de saturer une base de fait de manière autonome, et de poser une question à l'utilisateur en cas de blocage. On répète ces étapes jusqu'à l'obtention d'une bière dans la base de faits.

En pratique, le moteur est une fonction récursive contenant deux boucles principales.

- La première boucle permet la saturation de la base de faits. Cette dernière parcourt la base de règles, pour chaque règle, elle vérifie si l'ensemble des prémisses sont vérifiées grâce à la fonction trueRule. Si c'est le cas, elle ajoute la conclusion à la base de faits. Tant qu'un nouveau fait a pu être déduit, on recommence l'opération jusqu'à ne plus rien pouvoir déduire. Avant de passer à l'étape suivante, on vérifie qu'aucune bière n'est déja dans la base de fait, sinon on l'affiche et on met fin au programme.
- Dans l'autre cas, on rentre alors dans la deuxième boucle. On reparcourt alors la base de règles. Quand on tombe sur une règle dont seule la première prémisse est vérifiée, on pose alors la question concernée à l'utilisateur. On utilise pour cela la fonction askQuestion avec en paramètre la clé de la prémisse manquante (DEGRE ou PRIX par exemple). Dès qu'un nouveau fait est ajouté aux faits connus, on doit tout de suite repasser dans la première boucle pour restaurer au maximum la base de faits avec celui qu'on vient d'ajouter. A la fin de cette boucle, on vérifie à nouveau s'il existe un fait BEER dans la base (grâce à la fonction assoc), si ce n'est pas le cas, on relance la fonction depuis le début.

Notes : pour garder une base de faits consistante durant tout le processus, la base est définie comme une variable globale initialisée vide dans une fonction externe qui lance également le moteur. Cette fonction initialise également deux autres variables globales qui seront utilisées par le moteur pour savoir par exemple si un nouveau fait a été ajouté à la base de faits.

Scénarios d'utilisation

Pour lancer le programme complet, il faut importer l'ensemble des fonctions du fichier SE.cl puis lancer dans la console la fonction parcours en tapant "(parcours)". Deux exemples d'utilisation vous est présenté ci-dessous :

```
CL-USER> (parcours )
(parcours )

"Quelle apparence de biere preferez-vous ? (0 : blonde, 1 : brune, 2 : ambree, 3 : rouge, 4 : orange)"
>
0

"Quel degre d'alcool voulez-vous ? (entrez un nombre entre 0 et 12)"
>
10

"Preferez-vous une biere speciale (0) ou une biere d'abbaye (1) ?"
>
0

"Quel est le prix ideal de cette biere pour vous ? (entrez un nombre entre 0.5 et 3)"
>
2

"Preferez-vous un arome d'agrumes (0) ou d'epices (1) dans votre biere?"
>
0

La bière la plus adaptée pour vous au pic est la "Tripel Karmeliet"
NIL
```

```
CL-USER> (parcours)
(parcours)

"Quelle apparence de biere preferez-vous ? (0 : blonde, 1 : brune, 2 : ambree, 3 : rouge, 4 : orange)"
>
1

"Quel degre d'alcool voulez-vous ? (entrez un nombre entre 0 et 12)"
>
11

"Quelle amertume voulez-vous dans votre biere ? (0 : moyenne, 1 : forte)"
>
1

"Quel est le prix ideal de cette biere pour vous ? (entrez un nombre entre 0.5 et 3)"
>
2

La bière la plus adaptée pour vous au pic est la "Gulden Draak"
NIL
```

Commentaires

Ce système expert semble répondre à nos attentes : on compte 37 règles, 21 conclusions utiles et chaque combinaisons de préférences saisis par un utilisateur aboutit à une unique bière. De plus, le catalogue de bières sélectionnées semble couvrir un assez large éventail de goûts pour pouvoir proposer une bière adaptée à tous les utilisateurs.

Un autre avantage de notre système tient dans la gestion des questions : seulement les questions nécessaires sont posées, et ces questions dépendent entièrement des choix déjà faits par l'utilisateur. Il n'a donc pas à remplir complètement une base de faits en répondant à des questions qui ne seraient pas pertinentes dans son cas.

Nous avons tout de même rencontré plusieurs difficultés lors de la réalisation du projet. Dans un premier temps, il a fallu trouver des caractéristiques précises pour différencier chaque bière tout en restant assez pertinentes pour pouvoir être posées aux utilisateurs. Le catalogue du pic contenant beaucoup de bières blondes assez similaires, nous avons donc introduit les notions d'arômes, ou même de prix dans le cas de bières aux caractéristiques trop proches.

Une des difficultés principales résidait dans la nécessité d'adapter précisément les réponses possibles aux questions. En effet, à plusieurs endroits de l'arbre, on doit demander la valeur pour la clé arôme mais en fonction de la branche dans laquelle on se trouve, il faut pouvoir adapter le jeu de réponse présenté à l'utilisateur.

Conclusion

Ce projet nous a permis d'appliquer les concepts abordés dans le cours en réalisant l'élaboration complète d'un système expert. La conduite du devoir passe d'abord par une phase de réflexion et de recherche pour déterminer la structure globale de notre projet. S'ensuit une phase d'implémentation en LISP, le tout pour aboutir à un outil fonctionnel et unique. Cela rend l'approfondissement du concept de système expert ludique et enrichissant.