Grandgousier

Travail réalisé par Banach Piotr et Castiaux Julien
Dans le cadre du cours de Technique d'intelligence artificielle
Dispensé à la faculté d'Informatique de l'UNamur
Durant l'année académique 2019-2020

Code source accessible en ligne : https://git.drlazor.be/UNamur/grandgousier

Description

Le programme Grandgousier propose un prototype de bot capable de conseiller les utilisateurs sur le choix de vins adaptés à leurs repas et les renseigne sur leurs caractéristiques. Il prend en charge une série de dialogues en manipulant entre autres le prix des vins, leur région et localité ainsi que leur nez et effet en bouche.

Base de données

La base de données des vins se présente sous la forme d'un fichier CSV. Chaque ligne contient l'ensemble des informations relatives à un vin :

- Identifiant
- Nom complet
- Catégorie (vin de Loire, de Bordeaux, ...)
- Localité (zone géographique plus précise)
- Litrage
- Prix
- Type (blanc, rouge, rosé)
- Année de consommation conseillée
- Cote des internautes (étoiles sur 5)
- Description complète reprenant :
  - Nez
  - Bouche
  - Commentaire

Les fichiers CSV ont l'avantage de pouvoir être ouverts avec la plupart des tableurs ou bien de simple éditeur de texte, un autre avantage est que le fichier est très facilement exploitable dans du code.

Génération de la base de connaissances

La base de connaissance contient trois types d'informations:

- Les propriétés des vins (nom, prix, description, ...)

- Des fonctions de conversion entre nom canonique et identificateur interne
- Des mots clef associés aux catégories de vin ou à leur localité

Cette base de connaissance Prolog est générée depuis la base de données CSV via un programme Python. Python est un langage de programmation qui se prête bien à la manipulation de texte, qui est disponible sur les systèmes d'exploitation classiques (Windows, Mac OS, Linux) et qui propose nativement une bibliothèque pour traiter des fichiers CSV.

Tests unitaires

Afin d'assurer la non-régression des différentes modifications apportées à Grandgousier, le projet est fourni avec une suite de tests unitaires. Un nouvel interpréteur SWI-PL est démarré pour chaque test et est contrôlé par du code, une simple API est proposée pour interroger Grandgousier et récupérer sa réponse. Les erreurs sur la sortie d'erreur standard (par exemple les warnings prolog) sont interprétées comme des erreurs.

Chaque requête comprise par Grandgousier est reprise dans un test unitaire, les réponses attendues sont récupérées depuis la base de donnée en CSV, ceci évite de devoir changer la suite de test en cas de modification de la DB. On s'assure également que la génération automatique de la base de connaissance ainsi que le bot fonctionnent correctement.

Code prolog

Le code prolog est structuré en plusieurs grandes parties produisant le programme décrit ci-dessus:

- une partie transformant les questions entrées par l'utilisateur en éléments (lexèmes) exploitables par le programme et formatant les réponses en texte lisible
- une boucle principale cherchant les réponses aux questions en faisant appel à différentes règles
- les détails des différentes règles décrivant la structure autorisée pour les questions et la logique pour y apporter des réponses
- une base de connaissances à laquelle les règles pourront faire appel et comprenant la liste des vins et de leurs caractéristiques, répartie en deux fichiers :
  - `vinsdb.pl`, une partie générée dynamiquement comme expliqué plus haut (voir Génération de la base de connaissances)
  - `vinsdb_statique.pl`, une partie statique comprenant quelques informations encodées manuellement

Le code est basé sur le squelette de programme avec éléments de réponse proposé avec l'énoncé du projet et les efforts de développements se sont concentrés sur

la partie permettant de produire une réponse (les parties de conversion d'une question en liste de mots et d'écriture de réponse ont été gardées telles quelles). La suite du présent rapport se concentrera donc sur l'extension de cette première, les choix et la mise en œuvre de son implémentation.

Ajout de règles 'simples'

De manière très similaire à la règle n°1 (liée au mot clé 'bouche') proposée dans le squelette de programme de l'énoncé, deux autres règles simples (sans conditions autres que la détection du mot clé et du pattern de mots dans la question) ont été ajoutées : la règle n°6 (`nez`) et n°8 (`lappellation`), allant chercher une réponse toute prête dans les prédicats `nez(Vin,Reponse)` et `description_appellation(Appellation,Reponse)` de la base de connaissances.

Ajout de règles sur base de mots clés flexibles

Afin de pouvoir reconnaître des questions comme "Auriez-vous un Graves ?" ou "'Auriez-vous des vins de Bourgogne ?" où seuls des noms de localité/catégorie indiquent clairement la nature de la question, nous avons décidé que les mots clés de ces règles devraient être des variables.

Cela est d'une part rendu possible en ajoutant un prédicat de mot clé `mclef/2` pour chacun de ces noms à la génération de la base de connaissances, et d'autre part en vérifiant dans les règles que le mot clé est bien du bon type (par exemple dans la règle n°4 liée à une variable de nom de localité où on vérifie que le mot clé appartient à un prédicat `localites/2`).

Une stratégie similaire a été mise en place dans plusieurs règles grâce au prédicat `determinant/1`, rendant les patterns de questions plus flexibles en autorisant plusieurs déterminants par une variable (ex: "Auriez-vous des vins de Bordeaux ?" ou "Auriez-vous des vins du Rhône ?").

Pré-traitement de la question et autocorrection

Plusieurs opérations sont appliquées à la question de l'utilisateur après sa conversion en liste de mots afin de faciliter la reconnaissance des patterns liés aux règles. En plus de la fonction d'uniformisation des noms de vins proposée dans le squelette de programme de l'énoncé (permettant de reconnaître des noms à plusieurs mots en les substituant par un identifiant), une fonction similaire a été mise en place pour uniformiser les noms de localités. Les deux fonctions ainsi que leurs règles d'uniformisation sont automatiquement générées en même temps que la base de connaissances.

Additionnellement, une fonction d'autocorrection `correction_question/2` a été ajoutée, parcourant récursivement la liste de mots formant la question et corrigeant 0, un ou plusieurs mots. Ceci est possible grâce à :

- l'ajout du prédicat `vocabulaire/1` dans la base de connaissances statique contenant une liste de mots qui pourront être corrigés (composée manuellement sur base de noms de vins, localités, mots clés,...)
- l'utilisation de la fonction Prolog `isub`, indiquant la similarité entre deux mots sur une échelle de 0 à 1. Dans notre cas, les mots seront corrigés s'ils présentent une similarité d'au moins 0.9 avec un des mots de la liste de vocabulaire. Par exemple, à partir de la question "Auriez-vou des vins de Bourgoge ?", deux fautes de frappes auront été détectées et la question sera corrigée en "Auriez-vous des vins de Bourgogne ?".

Tous ces traitements sont appliqués dans la fonction `produire_reponse/2` avant de lancer la reconnaissance des règles et patterns.

Classement des vins par étoiles

Étant donné que plusieurs règles produisent comme réponse une liste de vins et que cette liste peut être un peu trop longue pour l'utilisateur, nous avons décidé de proposer les vins par série de trois comme suggéré dans l'énoncé (sauf pour la proposition des vins en fonction de leur prix). Afin de rendre la sélection plus pertinente, nous avons ajouté un système de cotation des vins (prédicat `etoile/2` de la base de connaissances généré aléatoirement pour chaque vin dans ce prototype mais qui pourrait tout aussi bien correspondre à une note donnée aux vins par des internautes) pour proposer les *meilleurs* vins en priorité.

Ceci est réalisé par :

- La fonction `tri_etoiles/2` qui forme des paires 'Étoile-Vin' en allant chercher la cote du vin, trie les paires sur base de leur clé (Étoile), enlève les clés (pour juste garder les vins) puis inverse la liste pour avoir un tri décroissant (meilleur cote d'abord).
- La fonction `top_trois/3` qui sélectionne trois vins (si disponibles) dans une liste à partir d'un indice. L'indice se base sur le contexte courant, répéter la question aura pour effet de mettre à jour l'indice dans le contexte et donc de récupérer les trois suivants.

Établissement d'un contexte dynamique

Pour permettre au bot de réagir à des questions faisant référence à la réponse précédente (ex: "Auriez-vous d'autres vins de Bordeaux ?", "Pouvez-vous m'en dire plus sur ... (nom de vin) ?"), nous avons également mis en place un contexte dynamique pouvant *mémoriser* des élément de réponse apportés.

Les prédicats sont d'abord indiqués comme dynamiques (modifiables lors de l'exécution) par la déclaration `:- dynamic ().`, indiquant le nom et l'arité du prédicat dynamique. Ils peuvent ensuite être retirés et réinsérés dans le programme grâce aux fonctions Prolog `retract` et `assert`.

Nous avons développé deux utilisations du contexte dynamique dans le cadre de ce prototype :

- pour pouvoir proposer plus de vins d'une certaine catégorie (règle n°3, question du type "Auriez-vous d'autres vins de Bordeaux ?"), nous utilisons un prédicat dynamique `contexte_vins(Cat,N)` où `Cat` est le dernier mot clé de catégorie utilisé dans une réponse et `N` l'indice à partir duquel il faudra commencer dans la liste des vins si on redemande cette catégorie. Inspiré de la gestion du contexte dynamique dans le programme Eliza, ce prédicat sera mis à jour dans la fonction `lvins_categorie/3` à chaque fois qu'on propose une série de trois vins supplémentaires (la mot clé de catégorie sera écrasé et l'indice incrémenté de 3).
- pour pouvoir donner plus d'informations sur un vin cité précédemment (règle n°5, question du type "Pouvez-vous m'en dire plus sur..."), un prédicat dynamique `contexte_vins` est écrasé par la liste de vins proposée à l'utilisateur à chaque fois que cette dernière est élaborée. Lorsque l'utilisateur demande plus d'informations sur un vin, il suffit de vérifier qu'il appartient à cette liste (ce qu'on fait dans la fonction `member_vin/2`).

Formatage des réponses

Afin de rendre l'interaction avec l'utilisateur plus naturelle et éviter par exemple de répondre "Oui" à une question ouverte ("Que me conseillez-vous... ?"), des phrases d'introduction ont été ajoutées pour certaines règles et sont passées comme variable à la fonction formatant une réponse. Cela permet par exemple pour la règle n°2 (proposition de vins sur base du prix) d'avoir la dialogue suivant:

- Auriez-vous des vins entre 10 et 12 EUR ?
- Oui, voici les vins dans cette gamme de prix : ...

De plus, plusieurs formatages de réponse en lignes ont été prévus, permettant de traiter aussi bien des tuples de type '(Vin,Prix)' que des listes de localités et catégories de vins comme c'est le cas pour le conseil de vins liés aux repas (règle n°7).

Perspectives

Étant donné que ce programme n'est qu'un prototype et que le temps consacré à son implémentation était limité, de nombreuses pistes d'amélioration auraient pu être explorées, notamment:

- l'ajout du litrage des vins, information disponible dans la base de données CSV mais non exploitée. Un prédicat `litrage/2` liant le litrage à l'identifiant des vins aurait pu être généré puis utilisé pour passer des tuples `(Vin, Prix, Litrage)` à la fonction de formatage des réponses (au lieu de `(Vin, Prix)`).
- l'extension de la base de connaissance liée au repas et aux appellations. Ces informations devant être recherchées et encodées manuellement à partir d'Internet, nous n'avons inclus qu'un type de repas (canard) et trois appellations.
- l'autocorrection aurait également pu être améliorée, par exemple:
  - en affichant à l'écran la question de l'utilisateur corrigée. Ceci aurait demandé du temps car la correction est appliquée sur une liste de mots préformatée (sans majuscules, apostrophes,...) et on aurait dû soit trouver un moyen d'inverser ce formatage soit stocker la question à l'état brut et y appliquer les corrections.
  - en étendant la liste de vocabulaire pris en considération pour la correction ou considérant des groupes de mots plutôt que des mots uniques (les mots trop courts ne seront pour le moment jamais corrigés)
  - en trouvant des alternatives à la fonction Prolog `isub` qui parfois indique une faible similarité entre deux mots très proches
- l'introduction de concepts d'apprentissage automatique, par exemple en surveillant les caractéristiques des vins auxquels l'utilisateur s'intéresse (type, prix, ...) pour mieux le conseilleur au fur et à mesure de l'exécution
- l'utilisation du poids des mots clés pour privilégier les plus importants. Dans l'état actuel du programme il est peu probable qu'une question puisse correspondre à plusieurs patterns, mais cela aurait été intéressant si le nombre de scénarios de conversations devait être étendu