UNIVERSITÉ DE NAMUR

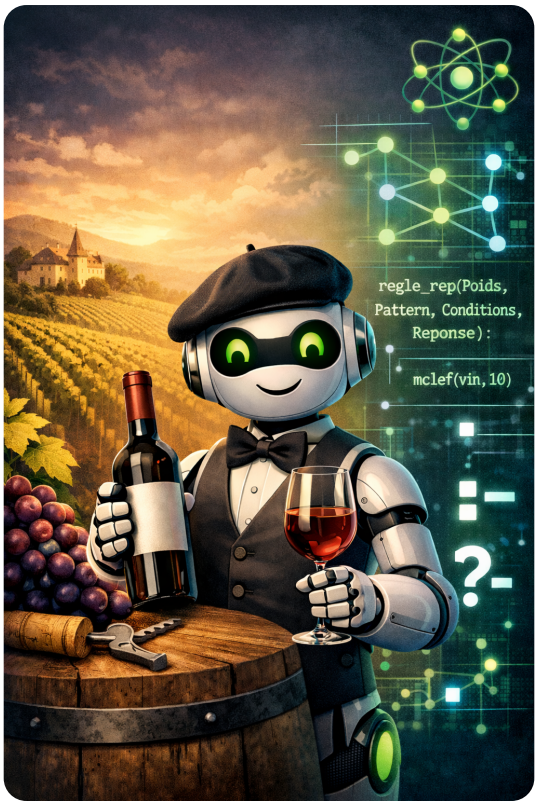
FACULTÉ D'INFORMATIQUE

Grandgousier

Bot conseiller en vins

Rapport de projet IA & programmation symbolique

IHDCM036



Homez Gilles

Crotteux Mathieu

Année académique 2025–2026

Professeur : Jean-Marie Jacquet

Table des matières

1. Introduction

Ce rapport présente le projet **Grandgousier**, un agent conversationnel écrit en PROLOG dont l'objectif est de conseiller des vins et de fournir des informations œnologiques à partir d'une base de connaissances structurée.
Le système repose sur une approche de *programmation symbolique* : les requêtes de l'utilisateur sont analysées sous forme de listes de mots, des mots-clés pondérés permettent d'identifier l'intention principale, et des règles Prolog déclenchent la réponse la plus appropriée. Cette approche s'inspire directement du programme *Eliza*, tout en étant adaptée au domaine spécifique du vin.

Objectifs du projet

- Construire une base de connaissances œnologique cohérente (vins, appellations, prix, descriptions, nez, bouche).
- Mettre en œuvre un moteur de dialogue basé sur des règles et des mots-clés pondérés.
- Reproduire fidèlement les dialogues exigés dans l'énoncé du projet.
- Illustrer les principes fondamentaux de la programmation symbolique appliquée à un agent conversationnel.

Organisation du rapport La section 2 décrit l'architecture générale du système. La section 3 détaille la base de connaissances. La section 4 présente le moteur de règles et le traitement de l'entrée utilisateur. La section 6 illustre le fonctionnement du système par des exemples concrets. Enfin, la section 9 conclut et propose des pistes d'amélioration.

2. Architecture générale du système

2.1. Vue d'ensemble

Le programme est structuré en deux parties principales :

- une **base de connaissances** contenant les faits œnologiques ;
- un **moteur de dialogue** chargé d'analyser les questions et de produire les réponses.

Cette séparation claire permet d'isoler les données du raisonnement, ce qui facilite l'extension de la base ou l'ajout de nouvelles règles sans modifier l'architecture globale.

2.2. Boucle principale et entrées/sorties

Le point d'entrée du programme repose sur une boucle de dialogue :

- lecture de la question utilisateur sous forme de liste de mots (via `read_atomics/1`) ;
- génération d'une réponse structurée (via `produire_reponse/2`) ;
- affichage de la réponse formatée (via `ecrire_reponse/1`).

La boucle s'interrompt lorsque l'utilisateur introduit le mot-clé `fin`.

Listing 1 – Extrait de la production de réponse

```
produire_reponse([fin],[L1]) :-
  L1 = [merci, de, m, '\'', avoir, consulte], !.
```

3. Base de connaissances œnologique

3.1. Représentation des vins

Chaque vin est identifié par un identifiant Prolog stable (ex. `chambolle_musigny_premier_cru_2012`). Les informations sont représentées par des faits indépendants :

- `nom(Id, NomLisible)`
- `prix(Id, Prix)`
- `provenance(Id, Region)`
- `appellation(Id, Appellation)`
- `nez(Id, Texte)`
- `bouche(Id, Texte)`
- `description(Id, Texte)`

Cette modélisation permet un accès flexible aux informations et favorise la réutilisation des données dans plusieurs règles.

Listing 2 – Exemples de faits de la base de connaissances

```
nom(chambolle_musigny_premier_cru_2012,
    'Chambolle Musigny 1er Cru 2012 - Les Noirots').

prix(chambolle_musigny_premier_cru_2012, 63.85).
appellation(chambolle_musigny_premier_cru_2012, chambolle_musigny).
provenance(chambolle_musigny_premier_cru_2012, bourgogne).
```

3.2. Accords mets-vins

Le système prend également en compte des plats typiques (canard, carbonnade, poisson, etc.). Ces plats sont normalisés vers des identifiants uniques, ce qui permet d'associer plusieurs formulations à un même concept.
Les recommandations sont alors produites à partir de règles plutôt que d'une analyse linguistique complète, ce qui correspond à l'esprit du projet.

4. Moteur de règles et compréhension d'entrée

4.1. Normalisation de l'entrée

L'entrée utilisateur est transformée en une liste de mots en minuscules. Avant l'appariement des règles, plusieurs normalisations sont appliquées :

- reconnaissance des noms de vins (formes longues, abrégées, collées) ;
- normalisation des appellations ;
- reconnaissance des plats ;
- décomposition de tokens composés (`estceque`, `puisje`, etc.).

Cette étape rend le système robuste face aux variations de formulation.

Listing 3 – Pipeline de normalisation

```
normaliser_question(Lmots,L_out) :-
  nom_vins_uniforme(Lmots,Ltmp),
  normaliser_appellations_tokens(Ltmp,Lapp),
  normaliser_plats_tokens(Lapp,Lplats),
  expand_compound_tokens(Lplats,Lsplit),
  maplist(normaliser_mot,Lsplit,L_out).
```

4.2. Mots-clés pondérés

Les mots-clés sont associés à des poids grâce au prédicat `mclef/2`. Ces poids permettent de prioriser les intentions les plus informatives (par exemple `nez` ou `bouche`) par rapport à des mots plus génériques.

Listing 4 – Exemples de mots-clés pondérés

```
mclef(nez,10).
mclef(bouche,10).
mclef(vin,5).
mclef(vins,5).
mclef(appellation,8).
mclef(canard,8).
```

4.3. Règles de réponse

Une règle de réponse est définie par le prédicat :

`regle_rep(MotCle, Priorite, Pattern, Reponse).`

Le `Pattern` est recherché comme sous-liste dans la question normalisée, et la `Reponse` est construite à partir de la base de connaissances.

Listing 5 – Règle pour une question sur le nez d'un vin

```
regle_rep(nez,1,
  [ quel, nez, presente, le, Vin ],
  Rep) :-
  nez(Vin,Rep),
  memoriser_vin(Vin).
```

4.4. Sélection de la règle

Le moteur :

1. identifie les mots-clés présents ;
2. sélectionne les règles associées ;
3. choisit la première règle compatible avec le pattern et les conditions.

Ce mécanisme simple est suffisant pour produire des dialogues cohérents et déterministes.

5. Mémoire et suivi de contexte

Afin de gérer des questions dépendantes du contexte (ex. « Puis-je le boire maintenant ? »), le système mémorise le dernier vin mentionné.

Listing 6 – Gestion du contexte

```
:- dynamic dernier_vin/1.

memoriser_vin(Vin) :-
  retractall(dernier_vin(_)),
  asserta(dernier_vin(Vin)).
```

Cette mémoire minimale améliore la fluidité du dialogue sans complexifier excessivement le moteur.

6. Exemples de dialogues

6.1. Recommandation par région

Utilisateur : Quels vins de bourgogne me conseillez vous
Système : Voici quelques vins de bourgogne que je peux vous proposer :

6.2. Demande de variantes

Utilisateur : auriez vous d autres vins de bourgogne
Système : J'ai aussi d'autres vins de bourgogne à vous proposer :

6.3. Questions œnologiques

Utilisateur : quel nez presente le chambolle musigny
Système : Nez complexe, intense de fruits rouges et d'épices.

6.4. Appellation

Utilisateur : que recouvre l appellation haut medoc
Système : Description d'un vin représentatif de l'appellation.

7. Validation, tests et limites

7.1. Stratégie de validation

La validation du système **Grandgousier** repose sur une combinaison de tests automatisés et de scénarios de dialogue inspirés de l'énoncé officiel. Contrairement à une validation purement manuelle, nous avons mis en place une véritable suite de tests unitaires en PROLOG, basée sur la bibliothèque `plunit`.
Ces tests couvrent à la fois :

- la cohérence de la base de connaissances œnologique ;
- la robustesse de la normalisation linguistique ;
- le bon déclenchement des règles de réponse ;
- la conformité des réponses produites avec les dialogues attendus.

L'ensemble des tests est regroupé dans une phase de test dédiée, chargée automatiquement avec le moteur principal.

7.2. Tests sur la base de connaissances

Une première série de tests vise à vérifier l'intégrité de la base de faits. Ces tests garantissent que chaque vin possède bien une provenance et une appellation, et que les informations essentielles sont présentes.

- vérification de l'existence des faits `provenance/2` et `appellation/2` ;
- cohérence taxonomique : chaque vin déclaré par `nom/2` possède une provenance et une appellation ;
- validation de faits spécifiques attendus par les dialogues de l'énoncé (ex. *La Fleur de Pomys*, *Hermitage rouge 2007*).

Ces tests permettent de détecter immédiatement une base incomplète ou incohérente lors de l'ajout de nouveaux vins.

7.3. Tests de normalisation linguistique

Une part importante de la robustesse du système repose sur la normalisation des entrées utilisateur. Plusieurs tests unitaires vérifient que des formulations variées mènent au même identifiant interne.
Les cas testés incluent notamment :

- noms de vins sans millésime explicite (*« nuits saint georges »*) ;
- formes compactées (*« nuitssaintgeorges »*, *« lafleurdepomys »*) ;
- variantes avec ou sans déterminant ;
- reconnaissance des premiers crus et millésimes.

Ces tests confirment que la normalisation produit systématiquement les identifiants Prolog attendus, indépendamment de la formulation exacte employée par l'utilisateur.

7.4. Tests des règles de réponse

Le cœur du moteur repose sur les règles `regle_rep/4`. Celles-ci sont validées par des tests qui comparent directement la réponse produite avec la réponse attendue.
Les catégories testées comprennent :

- questions sur le **nez** et la **bouche** d'un vin, y compris les formes longues et abrégées ;
- demandes de description détaillée (*« parlez-moi de »*, *« pourriez-vous m'en dire plus sur »*) ;
- recommandations par **appellation** (Bourgogne, Graves, Saint-Émilion) ;
- gestion des variantes (*« d'autres vins »*, absence de vins supplémentaires) ;
- recommandations par **plats** (canard, bœuf, poisson, boulets liégeois).

Les tests vérifient non seulement la présence des vins attendus, mais également le format exact des réponses, compatible avec le prédicat `ecrire_reponse/1`.

7.5. Tests sur les contraintes de prix

Des tests spécifiques sont consacrés aux recommandations fondées sur des contraintes numériques :

- intervalle de prix (*entre 20 et 35 euros*) ;
- prix minimum (*moins de 10 euros*) ;
- prix maximum (*plus de 60 euros*) ;
- absence de résultats dans une gamme donnée.

Ces tests confirment que les prédicats de sélection par prix fonctionnent correctement et retournent des réponses explicites lorsqu'aucun vin ne correspond aux critères.

7.6. Validation globale et non-régression

L'exécution complète de la suite de tests permet de valider le comportement global du système après chaque modification. Elle joue le rôle de test de non-régression : toute modification du moteur ou de la base de connaissances qui casserait un dialogue existant est immédiatement détectée.
Cette approche renforce considérablement la fiabilité du prototype et facilite son évolution.

7.7. Limites et pistes d'amélioration

Malgré une couverture fonctionnelle étendue, certaines limites subsistent :

- **Couverture linguistique bornée** : les questions hors patrons explicitement codés peuvent ne pas être comprises.
- **Désambiguïsation limitée** : plusieurs intentions peuvent partager des mots-clés similaires, ce qui dépend fortement de l'ordre et du poids des règles.
- **Heuristiques dépendantes du lexique** : certaines extensions (ex. maturité de dégustation) reposent sur la présence de mots-clés textuels.

Parmi les pistes d'amélioration possibles :

- enrichissement des synonymes et des patrons linguistiques ;
- renforcement du contexte de dialogue (mémorisation des filtres et des vins déjà proposés) ;
- ajout de nouvelles phases de tests unitaires pour couvrir des dialogues plus complexes.

8. Déroulement séquentiel du raisonnement dans `produire_reponse/2`

Une fois la question lue et transformée en liste de mots par les prédicats fournis (`lire_question/1` et `read_atomics/1`), le raisonnement proprement dit commence dans le prédicat `produire_reponse/2`. Ce prédicat traite la question de manière strictement séquentielle, selon les étapes décrites ci-dessous.

8.1. Entrée dans `produire_reponse/2`

À l'entrée de `produire_reponse/2`, la question n'est plus une chaîne de caractères mais une liste de tokens. Par exemple, pour la question utilisateur :

> *« Est-ce que je peux boire le Chablis 2019 maintenant ? »*

la représentation initiale est de la forme :

```
[est, ce, que, je, peux, boire, le, chablis, 2019, maintenant]
```

8.2. Normalisation de la question

La première étape effectuée dans `produire_reponse/2` est l'appel au prédicat `normaliser_question/2`. Cette étape transforme la liste brute de tokens en une forme standardisée, appelée *question normalisée*.
Dans l'exemple précédent, la normalisation conduit typiquement à :

```
Qn = [est, ce, que, je, peux, boire, Vin, maintenant]
```

où le nom du vin reconnu est remplacé par la variable logique `Vin`. À partir de ce point, `produire_reponse/2` ne travaille plus jamais sur la question brute, mais uniquement sur la question normalisée `Qn`.

8.3. Identification des mots-clés

Une fois la question normalisée obtenue, `produire_reponse/2` parcourt la liste `Qn` afin d'identifier les mots-clés qu'elle contient. Un mot est considéré comme mot-clé s'il apparaît dans les faits `mclef/2`.
Dans l'exemple :

```
Qn = [est, ce, que, je, peux, boire, Vin, maintenant]
```

les mots-clés détectés sont notamment :

```
boire -> poids 8
vin   -> poids 5
```

8.4. Sélection du mot-clé dominant

Parmi les mots-clés présents dans la question normalisée, `produire_reponse/2` sélectionne celui dont le poids est le plus élevé. Ce mot-clé est considéré comme le plus discriminant pour déterminer le type de question posée.
Dans l'exemple, le mot-clé dominant est :

```
boire
```

Le rôle de ce mot-clé n'est pas de produire la réponse, mais d'orienter la recherche vers une famille pertinente de règles.

8.5. Recherche des règles correspondantes

Une fois le mot-clé dominant déterminé, `produire_reponse/2` recherche les règles de dialogue associées à ce mot-clé à l'aide du prédicat `regle_rep/4`. Seules les règles liées au mot-clé sélectionné sont considérées.
Chaque règle contient un patron décrivant une forme possible de question.

8.6. Vérification du patron

Pour chaque règle candidate, `produire_reponse/2` vérifie si le patron de la règle est présent dans la question normalisée `Qn`. Cette vérification est réalisée par le prédicat `match_pattern/2`, qui teste si le patron est une sous-liste de `Qn`.
La première règle dont le patron correspond est retenue, conformément au mécanisme de résolution standard de Prolog basé sur l'ordre des clauses.

8.7. Exécution de l'action associée

Lorsque la règle est validée, l'action associée est exécutée. Cette action déclenche un prédicat spécialisé chargé de construire la réponse (par exemple `reponse_accessibilite/2` dans le cas présent).
Ces prédicats interrogent la base de connaissances `base_vins.pl`, produisent les lignes de réponse et mettent éventuellement à jour le contexte du dialogue, notamment via le prédicat `dernier_vin/1`.

8.8. Cas d'échec

Si aucun mot-clé pertinent n'est détecté ou si aucune règle associée ne correspond à la question normalisée, `produire_reponse/2` génère une réponse par défaut indiquant que la question n'a pas été comprise.

9. Conclusion

Le projet **Grandgousier** illustre efficacement les principes de la programmation symbolique appliqués à un agent conversationnel.
Il met en œuvre :

- une base de connaissances riche et structurée ;
- un moteur de règles fondé sur des mots-clés pondérés ;
- un mécanisme simple mais efficace de suivi de contexte.

Le système répond aux exigences de l'énoncé et va même au-delà, notamment grâce à l'heuristique d'évaluation de la maturité des vins. Des améliorations futures pourraient inclure une gestion plus fine des ambiguïtés linguistiques ou une extension du contexte de dialogue.
La liste exhaustive des patrons reconnus par les règles `regle_rep/4`, ainsi qu'un schéma détaillé du déroulement de `produire_reponse/2`, sont fournis en annexe (Annexe A et Annexe B).

A. Patrons de formulation (`regle_rep/4`)

Cette section recense de manière exhaustive les patrons (listes de tokens) reconnus par le bot au niveau des règles `regle_rep/4`, regroupés par type de question. Les variables suivantes désignent :

- `Vin` : un vin (nom d'une entrée de la base),
- `App` : une appellation,
- `Plat` : un plat (ou ingrédient),
- `X, Y, Min, Max` : des nombres (prix),
- `Det` : un déterminant/segment optionnel (ex. `le`, `ce`, etc.).

A.1. Bouche d'un vin

```
[que, donne, le, Vin, en, bouche]
[que, donne, Vin, en, bouche]
[comment, est, Vin, en, bouche]
[que, donne, Vin, en, bouche, '?']
[bouche, de, Vin]
```

A.2. Nez d'un vin

```
[quel, nez, presente, le, Vin]
[quel, nez, presente, Vin]
[quel, nez, pour, Vin]
[quel, nez, pour, Vin, '?']
[nez, de, Vin]
```

A.3. Description d'un vin / appellation

```
[pourriezvous, men, dire, plus, sur, le, Vin]
[pourriezvous, men, dire, plus, sur, Vin]
[pourriez, men, dire, plus, sur, le, Vin]
[pourriez, men, dire, plus, sur, Vin]
[que, pouvezvous, me, dire, sur, le, Vin]
[parlez, moi, du, Vin]
[parlez, moi, de, Vin]
[parle, moi, du, Vin]
[parlezvous, du, Vin]
```

A.4. Recommandations par appellation

```
[quels, vins, de, App, me, conseillezvous]
[quels, vins, de, App, me, conseillez, vous]
[auriezvous, des, vins, de, App]
[avezvous, des, vins, de, App]
[quel, vin, de, App, me, conseillezvous]
[quel, vin, de, App, me, conseillez, vous]
[vous, auriez, un, App]
[vous, auriez, des, App]
[auriez, vous, un, App]
[auriez, vous, des, App]
```

A.5. Autres vins d'une appellation

```
[auriezvous, dautres, vins, de, App]
[auriezvous, d, autres, vins, de, App]
[auriez, vous, d, autres, vins, de, App]
[auriezvous, autres, vins, de, App]
[auriez, vous, autres, vins, de, App]
```

A.6. Accessibilité / "prêt à boire"

```
[est, ce, un, vin, que, je, peux, boire, tout, de,
suite, Vin]
[est, ce, un, vin, que, je, peux, boire, tout, de, suite,
Det, Vin]
[est, ce, que, je, peux, boire, Vin, tout, de, suite]
[est, ce, que, je, peux, boire, Det, Vin, tout, de, suite]
[est, ce, que, je, peux, boire, Vin, maintenant]
[est, ce, que, je, peux, boire, Det, Vin, maintenant]
[puis, je, boire, Vin, maintenant]
[puis, je, boire, Det, Vin, maintenant]
[puis, je, deja, boire, Vin]
[puis, je, deja, boire, Det, Vin]
[je, peux, deja, boire, Vin]
[je, peux, deja, boire, Det, Vin]
[peut, on, boire, Vin, tout, de, suite]
[peut, on, boire, Det, Vin, tout, de, suite]
[peut, on, boire, Vin, maintenant]
[peut, on, boire, Det, Vin, maintenant]
[Vin, est, il, pret, a, boire]
[Det, Vin, est, il, pret, a, boire]
[je, peux, boire, Vin, tout, de, suite]
[je, peux, boire, Det, Vin, tout, de, suite]
[je, peux, boire, Vin, maintenant]
[je, peux, boire, Det, Vin, maintenant]
[est, ce, un, vin, que, je, peux, boire, tout, de, suite]
[est, ce, un, vin, que, je, peux, boire, maintenant]
[est, ce, que, je, peux, le, boire, tout, de, suite]
[est, ce, que, je, peux, le, boire, maintenant]
```

A.7. Filtrage par prix

```
[auriezvous, des, vins, entre, X, et, Y, eur]
[auriez, vous, des, vins, entre, X, et, Y, eur]
[avezvous, des, vins, entre, X, et, Y, euros]
[avez, vous, des, vins, entre, X, et, Y, euros]
[auriezvous, des, vins, entre, X, et, Y]
[auriez, vous, des, vins, entre, X, et, Y, euros]
[auriezvous, des, vins, entre, X, et, Y, euros]
[avezvous, des, vins, a, moins, de, Max, euros]
[avezvous, des, vins, a, moins, de, Max]
[avezvous, des, vins, a, plus, de, Min, euros]
[avezvous, des, vins, a, plus, de, Min]
[des, vins, a, moins, de, Max, euros]
[des, vins, a, moins, de, Max]
[des, vins, a, plus, de, Min, euros]
[des, vins, a, plus, de, Min, eur]
[des, vins, a, plus, de, Min]
```

A.8. Accords mets–vins

```
[je, cuisine, du, Plat, quel, vin, me, conseillezvous]
[je, cuisine, des, Plat, quel, vin, me, conseillezvous]
[Plat, quel, vin, me, conseillezvous]
[je, fais, un, Plat, quel, vin, servir]
[quel, vin, avec, du, Plat]
[quel, vin, avec, des, Plat]
[quel, vin, pour, Plat]
```

A.9. Définition d'une appellation

```
[que, recouvre, appellation, App]
[que, recouvre, appellation, App, '?']
[que, recouvre, l, ''', appellation, App]
[que, recouvre, l, ''', appellation, App, '?']
```

A.10. Fin de session

```
[fin]
```

B. Flux d'exécution

```
grandgousier/0
  ↓
lire_question/1
  ↓
normaliser_question/2
  ↓
produire_reponse/2
      (A) Préparation / repérage
           normaliser_question/2
           mclef/2 + member/2
               → repère un ‘‘mot-clé’’ présent dans la question normalisée

      (B) Sélection d'une règle de dialogue
           clause/2 sur regle_rep/4
               → récupère une règle candidate associée au mot-clé détecté
           match_pattern/2
                → vérifie que le patron de mots attendu est bien présent

      (C) Exécution de l'action (le vrai contenu de la réponse)
           call(Body)
                → déclenche UN des ‘‘constructeurs’’ de réponse suivants (selon la règle)

             1) Réponses de dégustation (vin explicite)
                 bouche_reponse/2
                 nez_reponse/2
                    s'appuient sur base_vins : bouche/2, nez/2 (+ mise en forme)
                    appellent souvent memoriser_vin/1 pour le contexte

             2) Réponse « parle / parlez-moi de ... »
                 description_ou_appellation/2
                    si on reconnaît un vin : description/2
                    sinon : bascule vers definition_appellation/2

             3) Recommandations / listes de vins
                 reponse_conseil/3
                     récupère des contraintes (appellation, etc.)
                     filtre/sélectionne des vins via base_vins
                     limite le nombre de résultats
                 repondre_vins_prix/...
                      cas typiques « moins / plus ... euros »

             4) Accords mets-vins (plats)
                 reponse_plat/2
                    construit un profil de plat
                    transforme en recommandations
                    formate la sortie

             5) « Puis-je le boire maintenant ? »
                 reponse_accessibilite/2
                    utilise dernier_vin/1 si nécessaire
                    applique une heuristique de scoring

      (D) Cas d'échec (fallback)
           → aucune règle ne correspond
  ↓
ecrire_reponse/1
```