Ecole d'Ingenieur Denis Diderot
Université Paris-Cité

Rapport

Intelligence Artificielle - 2SIE

Rapport du Mini-projet ELIZA en LISP

Auteurs

ELABJANI Nissrine
HIBAOUI Imane
ZIDANI Rababe

Encadrant

M. DENIS Christophe

Année universitaire

2024-2025

Table des matières

1 Introduction

Dans le cadre de notre formation en deuxième année du cycle ingénieur (2SIE), nous avons eu l'opportunité de travailler sur un mini-projet visant à implémenter un chatbot ELIZA en LISP. Ce projet s'inscrit dans le module d'intelligence artificielle, plus précisément dans la partie consacrée à l'IA symbolique.

Ce rapport décrit en détail les différentes étapes de développement, les fonctions réalisées, les problèmes rencontrés et les solutions mises en place. Il met également en lumière les apprentissages acquis ainsi que les perspectives d'amélioration possibles.

2 Fonctionnalités développées

Le mini-projet consistait à reproduire le comportement d'un chatbot thérapeute, inspiré d'ELIZA, en utilisant le langage de programmation symbolique LISP. Plusieurs fonctions ont été développées pour gérer la détection de motifs dans les phrases utilisateur, l'association de variables, la substitution de réponses, et l'interaction continue dans une boucle de dialogue.

La fonction **match-eq** permet de vérifier l'égalité stricte entre deux listes, utile pour des cas simples de comparaison. La fonction **match** quant à elle est essentielle : elle gère la reconnaissance de motifs avec des jokers, tels que `* x` ou `* y`, et met à jour dynamiquement les correspondances dans une structure de liaisons appelée `*bindings*`. Ces associations sont ensuite utilisées par **bind** pour créer ou compléter les paires (variable, valeur), et par **lookup** pour retrouver les valeurs associées.

Une fois les variables extraites, la fonction **subs** les remplace dans la structure de réponse brute, et la fonction **swap** ajuste les pronoms pour donner une impression de personnalisation : par exemple, `I` devient `YOU`, et `MY` devient `YOUR`. Pour éviter des réponses trop répétitives, la fonction **random-elt** permet de choisir une phrase aléatoire parmi une liste prédéfinie.

Enfin, la fonction **find-response** orchestre toutes ces opérations pour renvoyer une réponse complète, et la fonction **eliza** boucle tant que l'utilisateur n'a pas saisi une commande de sortie.

3 Contraintes et problèmes rencontrés

Au cours du développement, nous avons rencontré plusieurs obstacles techniques. Premièrement, la reconnaissance de motifs avec plusieurs jokers a été un vrai défi. Une version initiale de la fonction `match` ne pouvait gérer qu'un seul joker, ce qui bloquait la détection de phrases plus complexes comme `"I want a dog"` ou `"I love school because it's interesting"`.

Deuxièmement, nous avons utilisé initialement la fonction `split-sequence` pour découper les chaînes en mots. Cependant, cette fonction provient d'une bibliothèque externe non incluse par défaut dans l'environnement SBCL (Steel Bank Common Lisp), ce qui entraînait des erreurs de chargement et limitait la portabilité du code.

Troisièmement, nous avons constaté que SBCL ne tolère pas les virgules dans les chaînes saisies via `read-line` sans les entourer de backquotes. Cela a posé un problème lors de l'écriture naturelle d'entrées par l'utilisateur, comme `"Hi, how are you?"`, provoquant une erreur de parsing.

4 Solutions apportées

Pour résoudre ces problèmes, plusieurs solutions concrètes ont été mises en œuvre. D'abord, la fonction `match` a été complètement réécrite pour inclure une gestion récursive des jokers, permettant de capturer dynamiquement les sous-parties des phrases utilisateur, quel que soit leur nombre ou leur position dans la structure du motif.

Ensuite, pour remplacer la bibliothèque `split-sequence`, nous avons écrit une fonction maison appelée `split-string`, utilisant uniquement les primitives standard de LISP comme `make-string-input-stream` et `read`. Cela garantit une compatibilité totale avec SBCL sans dépendance externe.

Enfin, pour éviter les erreurs dues à la ponctuation, nous avons introduit une fonction `clean-input` qui supprime automatiquement les caractères problématiques tels que les virgules, les points d'interrogation, ou les points-virgules, assurant ainsi que l'entrée utilisateur est correctement analysée.

5 Ce que nous avons appris

Ce mini-projet nous a permis de renforcer nos compétences dans plusieurs domaines clés. Sur le plan technique, nous avons acquis une maîtrise bien plus solide des listes, des fonctions récursives, de la manipulation symbolique, et de la structure de code modulaire en LISP.

Nous avons aussi compris en profondeur les fondements de l'intelligence artificielle symbolique : la manipulation de connaissances sous forme de règles, la recherche de correspondances dans des structures arborescentes, et la logique de transformation du langage naturel. Ce type d'IA, bien que dépassé dans certains domaines par les approches statistiques ou neuronales, reste d'une grande valeur pour des applications explicables, didactiques, et formelles.

Enfin, nous avons également gagné en autonomie pour la résolution de problèmes logiciels liés à l'environnement (SBCL), à la gestion des erreurs utilisateur, et à la maintenance de code lisible et réutilisable.

6 Pistes d'amélioration

Plusieurs pistes peuvent être envisagées pour enrichir ce chatbot symbolique :

— Intégrer une **mémoire contextuelle** pour que le bot se souvienne de sujets évoqués auparavant.

— Ajouter une **gestion de synonymes**, afin de reconnaître plusieurs façons de dire la même chose (ex : `angry`, `mad`, `furious`).

— Introduire un **score de pertinence** entre le motif et l'entrée, pour permettre une sélection plus intelligente des réponses.

— Développer une **interface graphique** simple ou une intégration web, pour rendre l'interaction plus conviviale.

— Étendre le bot à une **version multilingue**, en gérant plusieurs règles par langue.

7 Conclusion

Ce projet nous a permis de combiner des compétences en programmation fonctionnelle, en intelligence artificielle et en traitement du langage naturel symbolique. Nous avons non seulement reproduit un comportement historique de chatbot, mais aussi adapté ses mécanismes aux contraintes modernes. Le résultat est un agent conversationnel simple, fonctionnel, modulable et extensible, basé uniquement sur des mécanismes logiques et déterministes.

Nous sommes très satisfaits du chemin parcouru, et cette expérience a renforcé notre intérêt pour l'intelligence artificielle symbolique et les paradigmes alternatifs à l'apprentissage automatique classique.

Annexe : Résultats des tests du chatbot ELIZA sur wsl

Les tests suivants ont été réalisés pour vérifier le bon fonctionnement de chaque composant du système :

```
Test 1: (match-eq '(i feel happy) '(i feel happy))
Resultat: T

Test 2: (match-eq '(i feel sad) '(i feel happy))
Resultat: NIL

Test 3: (match '(i feel *) '(i feel happy))
Resultat: T
Bindings: NIL

Test 4: (wildcard '(* x i hate * y) '(sometimes i hate my job))
Resultat: NIL
Bindings: NIL

Test 5: (bind 'x 'cat '((x dog) (y sheep)))
Resultat: ((X CAT DOG) (Y SHEEP))

Test 6: lookup dans *bindings*
Bindings actuels: ((X DOGS) (Y CATS AND SHEEP))
lookup 'x     (X DOGS)
lookup 'y     (Y CATS AND SHEEP)
lookup 'z     NIL

Test 7: subs avec *bindings*
Phrase initiale: '(I think y are chased by x)
Resultat: (I THINK CATS AND SHEEP ARE CHASED BY DOGS)

Test 8: Fonction swap
swap 'i     YOU
swap 'my     YOUR
swap 'was     WERE
swap 'computer     COMPUTER

Test 9: Fonction random-elt
Liste 1: (yes no maybe)     NO
Liste 2: ("Tell me more." "Why do you think so?")     "Why do you
    think so?"

Demarrage de la session ELIZA... Tapez 'bisous' pour quitter.

ELIZA: Bonjour. Parlez-moi.
> bisous
Au revoir.
```

Listing 1 – Résultats des tests dans le terminal SBCL