Moteur d'inférence

Noé De Caestecker Tino Dolbeau

25 novembre 2025

Résumé

L'objectif de ce projet est de programmer un moteur d'inférence d'ordre Zéro+ au minimum adapté à un système expert de notre choix. Ce moteur d'inférence doit pouvoir être utilisé pour d'autres applications similaires à celle pour laquelle il a été créé.

Table des matières

1 Système expert

Afin de disposer rapidement d'une base pour commencer l'implémentation du moteur d'inférence, nous avons décidé de nous orienter vers une application de classement d'espèces animales. L'application prendra une description complète ou non des caractéristiques d'un animal (nombre de membres, squelette, poils, ...) et en déduira l'espèce ou la famille de l'animal (gastéropode, insecte, arachnide, ...).

2 Syntaxe

La syntaxe choisie se divise en 2 concepts : les faits et les règles.

Les faits seront composés d'une variable (sous forme d'une chaine de caractères ne contenant que des lettres et des underscores), d'un symbole arithmétique ou d'affectation et d'une valeur (valeur numérique ou chaine de caractères). Par exemple :

- nom_de_variable — pour une variable booléenne
- -nom_de_variable — pour la négation d'une variable booléenne
- nom_de_variable = 5.3 — pour une variable numérique
- nom_de_variable != 1.68 — pour une inégalité de variable numérique
- nom_de_variable > 0.3 — pour une comparaison de variable numérique
- nom_de_variable = bleu — pour une variable textuelle
- nom_de_variable != rouge — pour une inégalité de variable textuelle

Les règles seront quant à elles composés d'un nom de règle, d'une liste de faits représentant les prémisses et d'une autre liste représentant les conséquents. Par exemple :

- R1 : A => B
  une règle avec 1 prémisse booléenne et 1 conséquent booléen
- R2 : A, B = 3, H => B > 2, D < 2
  une règle avec 3 prémisses et 2 conséquents

Une base de faits et une base de règles d'exemple sont disponibles dans le dossier *bases* du répertoire du projet.

3 Utilisation

3.1 Compilation

Pour compiler le projet, exécuter les commandes :

```
# cd ./build
# cmake ..
# make
```

L'exécutable sera ensuite accessible depuis le dossier build avec la commande

```
# ./moteur_inference <options>
```

3.2 Exécution & options

Une fois le projet compilé, et pour pouvoir l'exécuter, une base de règle et une base de faits initiaux doivent être fournis. Les deux premiers arguments de l'exécutable représentent les chemins vers ces deux bases.

Par exemple, l'utilisation des bases fournies avec le projet donne la commande

```
# ./moteur_inference ../bases/rules.txt ../bases/facts.txt <options>
```

L'exécutable attend plusieurs autres options qui seront discutés dans la suite du document. Entre autres, la stratégie à utiliser doit être renseigné en utilisant les options -fc ou -bc respectivement pour le chainage avant et arrière (*forward chaining* et *backward chaining*)

Une liste des options est accessible via la commande

```
# ./moteur_inference -help
```

ou

```
# ./moteur_inference -h
```

4 Stratégies

4.1 Chaînage avant

Le chainage avant (utilisable avec l'option -f ou -fc) sature la base de faits en utilisant la base de règles. Un but peut être donné en utilisant l'option -b suivie d'un fait objectif (entre guillemets). Si un but est donné, les règles sont appliquées jusqu'a ce que le but soit atteint ou que la base de faits soit saturée.

Lors du chainage avant, l'ensemble des règles utilisés sont indiqués les une après les autres. La base de fait finale est ensuite affichée avec une indication si le fait but a été atteint ou non.

Exemple de sortie pour la commande

```
# ./moteur_inference ../bases/rules.txt ../bases/facts.txt -fc
```

```
- R1: tete => animal
- R4: animal, squelette_interne, crane => vertebre
- R5: vertebre, squelette_os => osteichtyen
- R6: osteichtyen, membre = 4 => tetrapode
- R7: tetrapode, gesier => oiseau_ou_croco
- R9: oiseau_ou_croco, trou_os_tempe => crocodilien
6 rules applied
Fact base saturated
Final fact base:
[
        tete
        bouche
        yeux
        squelette_interne
        crane
        squelette_os
        membre = 4
        gesier
        trou_os_tempe
        animal
        vertebre
        osteichtyen
        tetrapode
        oiseau_ou_croco
        crocodilien
]
```

4.2 Chaînage arrière

Le chainage arrière (utilisable avec l'option -b ou -bc) donne une démonstration d'un fait donné en paramètre. Contrairement au chainage arrière, le but doit être donné en utilisant l'option -b suivie du fait objectif (entre guillemets).

La sortie du chainage arrière est un arbre des règles à utiliser pour démontrer le fait voulu. Si un fait demandable nécessaire n'est pas présent dans la base de faits et que sa négation n'y est pas non plus, le programme peut alors demander à l'utilisateur si ce fait est vrai, faux ou indéterminé.

Exemple de sortie pour la commande

```
# ./moteur_inference ../bases/rules.txt ../bases/facts.txt -bc -g crocodilien
```

```
R9: oiseau_ou_croco, trou_os_tempe => crocodilien
 +- R7: tetrapode, gesier => oiseau_ou_croco
 |  +- R6: osteichtyen, membre = 4 => tetrapode
 |  |  +- R5: vertebre, squelette_os => osteichtyen
 |  |  |  +- R4: animal, squelette_interne, crane => vertebre
 |  |  |  |  +- R1: tete => animal
 |  |  |  |  |  +- tete
 |  |  |  |  +- squelette_interne
 |  |  |  |  \- crane
 |  |  |  \- squelette_os
 |  |  \- membre = 4
 |  \- gesier
 \- trou_os_tempe
```

Exemple de demande de fait pour "demandable > 2"

```
Is this true ? (y/n/?) demandable > 2
-
```

5 Critères de choix

En plus de choisir la stratégie à utiliser, il est possible de préciser selon quel critère les règles sont choisies. Par défaut, les règles sont testées dans l'ordre de leur présence dans la base de règles, ce qui n'est en général pas le meilleur choix. Deux critères sont possibles : le nombre de prémisses ou les prémisses les plus récemment ajoutées dans la base de faits.

5.1 Nombre de prémisses

Le critère du nombre de prémisses peut être activé avec l'option -max_premisses. Si c'est le cas, les règles ayant le plus grand nombre de prémisses seront testés en premier. Si plusieurs règles peuvent être appliquées et ont le même nombre de prémisses, la première dans l'ordre de la base de règles sera sélectionnée.

5.2 Prémisses récentes

Le critère des prémisses les plus récentes peut être activé avec l'option -recent_premisses. Les règles utilisant les faits les plus récemment ajoutés dans la base de faits seront testés en premier. Tout comme le critère du nombre de prémisses, en cas d'égalité, la première règle concernée dans l'ordre de la base de règles sera sélectionnée.

Le calcul utilisé pour le choix de la règle est la somme des indices de chaque fait dans la base de faits (les faits les plus récents ayant les indices les plus grands).

6 Contraintes de l'implémentation sur le langage

L'utilisation des faits arithmétiques dans notre moteur nous a forcé à réaliser des choix d'implémentation ne permettant pas l'utilisation de variables multi-valuées. Il est aussi impossible d'affecter à une variable une autre variable (ou de comparer des variables pour la même raison).

7 Cohérence

7.1 Cohérence des faits

La cohérence de la base de faits est réalisée à l'importation de celle-ci. Toute contradiction est renseignée dans la console sous forme de warning.

Exemple de contradiction de règles :

```
Consistency Warning:
        fact    "membre > 6"
        is inconsistent with fact       "membre = 6"
```

7.2 Cohérence des règles

La cohérence de la base de règles est aussi réalisée à l'importation. Les cohérences entre des règles sont testées uniquement 2 à 2. Les règles ayant un conséquent en contradiction avec une prémisse sont aussi relevées.

Exemple de contradiction entre 2 règles :

```
Consistency Warning:
        rule    "T1: T = eleve, P = haha => V = rapide"
        leads to        "T1: T = eleve => V = lent"
```

Exemple de contradiction dans une règle :

```
Consistency Warning:
        "A > 3"  contradicts    "A < 1"
        in rule "R3: A > 3 => "A < 1"
```