Démonstration Py4Graal -- visite Hcéres LIRMM 2025

Jean-François Baget, Carole Beaugeois (2025), Tom Salembien (2021)

Py4Graal est une interface Python permettant de manipuler la plateforme de raisonnement Integraal. La démonstration suivante, présentée sous la forme d'un notebook Jupyter, est très inspirée du tutoriel écrit par Carole Beaugeois au cours de son stage de M3.

Quelques utilitaires

Quelques fonctions d'affichage pour alléger le code qui va suivre...

In [1]:
```
from py4graal import *
from pathlib import Path

def display_kb(factbase: FactBase, rulebase: RuleBase, query: Query):
    print("Factbase:", factbase, "Rulebase:", rulebase, "Query:", query, sep="\n

def display_ans(answers:Answers):
    print("Answers:")
    for answer in answers:
        print(answer)

def display_analysis(analysis: Analyser):
    print("Saturation finie? " + str(analysis.isfes))
    print("Réécriture finie? " + str(analysis.isfus))
    print("Terminaison d'un algorithme? " + str(analysis.isdec))
```

Modus Ponens

Nous allons illustrer les différentes formes de raisonnement présentes dans Integraal à l'aide du classique (et simplissime) exemple du *modus ponens*:

- Socrate est un homme
- *or* tous les hommes sont mortels
- *donc* Socrate est mortel

Ce qui peut s'écrire, en DLGP comme en DLGPE, de la façon suivante:

```
% Modus Ponens

man(socrates).       % Socrate est un homme
mortal(X) :- man(X). % Tous les hommes sont mortels

[Q] ?(X) :- mortal(X).   % Qui est mortel ?
```

> **Note:** les appels suivants nécessitent un fichier `mp.dlgpe` contenant une requête nommée `Q`.

Evaluation de requête

L'évaluation d'une requête dans une base de faits ne tient pas compte de la base de règles...

In [2]:
```
dlgpe = Path("mp.dlgpe").read_text(encoding="utf-8")
graal = Graal()
factbase, rulebase, qc =  graal.load_dlgp(dlgpe)
query = Query(graal.environment, "Q")
display_kb(factbase, rulebase, query)
answers = query.evaluate(factbase)
display_ans(answers)
```

```
Factbase:
man(socrates).
Rulebase:
[R:1761914900082_6] mortal(X) :- man(X).
Query:
[Q] ?(X) :- mortal(X).
Answers:
```

Saturation

On peut rajouter à la base de faits tout ce qui pourrait être déductible par les règles...

In [3]:
```
fbsave = factbase.copy()
factbase.saturate(rulebase)
display_kb(factbase, rulebase, query)
answers = query.evaluate(factbase)
display_ans(answers)
```

```
Factbase:
mortal(socrates), man(socrates).
Rulebase:
[R:1761914900082_6] mortal(X) :- man(X).
Query:
[Q] ?(X) :- mortal(X).
Answers:
{'X': 'socrates'}
```

Réécriture

On peut réécrire une requête par une base de règles de façon à ce que l'évaluation de la requête réécrite dans la base de faits initiale donne le même résultat que l'évaluation de la requête initiale dans la base de faits saturée.

In [4]:
```
factbase = fbsave
query.rewrite(rulebase)
display_kb(factbase, rulebase, query)
```

```
answers = query.evaluate(factbase)
display_ans(answers)
```

```
Factbase:
man(socrates).
Rulebase:
[R:1761914900082_6] mortal(X) :- man(X).
Query:
[Q] ?(X) :- mortal(X).
?(X) :- man(X).
Answers:
{'X': 'socrates'}
```

Le problème de l'arrêt

Le problème de déduction dans le formalisme des règles existentielles étant indécidable, il s'ensuit que ni la saturation ni la réécriture ne sont assurées de s'arrêter.

Saturation

Contrairement à ce qu'on peut observer en Datalog, la présence de variables existentielles dans la tête des règles peut mener à une génération infinie de nouvelles variables.

In [5]:
```
factbase =  graal.create_factbase("human(socrates).")
rulebase = graal.create_rulebase("parent(Y, X), human(Y) :- human(X).")

for i in range(3) :
    print(str(factbase.saturate(rulebase, checker=Checker.RESTRICTED, rank=1)))
```

```
parent(Graal_EE18, socrates), human(Graal_EE18), human(socrates).
parent(Graal_EE35, Graal_EE18), parent(Graal_EE18, socrates), human(Graal_EE18),
human(socrates), human(Graal_EE35).
parent(Graal_EE35, Graal_EE18), parent(Graal_EE52, Graal_EE35), parent(Graal_EE1
8, socrates), human(Graal_EE18), human(socrates), human(Graal_EE35), human(Graal_
EE52).
```

Réécriture

Mais, même en Datalog, la réécriture peut ne pas s'arrêter...

In [6]:
```
rulebase = graal.create_rulebase("ancestor(X, Z) :- ancestor(X, Y), ancestor(Y,
query = graal.create_query("?(X) :- ancestor(socrates, X).")
# query.rewrite(rulebase)
```

Py4Graal n'a pas encore accés à la réécriture *par étapes*, mais elle donnerait quelque chose comme ça:

```
?(X, Z) :- ancestor(X, Z).
?(X, Z) :- ancestor(X, Y), ancestor(Y, Z).
?(X, Z) :- ancestor(X, Y1), ancestor(Y1, Y2), ancestor(Y2, Z).
?(X, Z) :- ancestor(X, Y1), ancestor(Y1, Y2), ancestor(Y2, Y3),
ancestor(Y3, Z).
```

Analyse d'une base de règle

L'analyse d'une base de règles (à l'aide de l'outil Kiabora) permet (parfois) de prouver que la saturation (ou la réécriture) par une base de règles s'arrêtera, quelle que soit la base de faits (ou la requête).

In [7]:
```
rulebase = graal.create_rulebase("hasParent(X, Y), human(Z) :- human(X).")
analyser = rulebase.analyse()
display_analysis(analyser)
```

```
Saturation finie? False
Réécriture finie? True
Terminaison d'un algorithme? True
```

In [8]:
```
rulebase = graal.create_rulebase("ancestor(X, Z) :- ancestor(X, Y), ancestor(Y,
analyser = rulebase.analyse()
display_analysis(analyser)
```

```
Saturation finie? True
Réécriture finie? False
Terminaison d'un algorithme? True
```

In [9]:
```
rulebase = graal.create_rulebase("hasParent(X, Y), human(Y) :- human(X). ancesto
analyser = rulebase.analyse()
display_analysis(analyser)
```

```
Saturation finie? False
Réécriture finie? False
Terminaison d'un algorithme? True
```

> **Note:** nécessite un fichier `kb.dlgpe`.

In [10]:
```
dlgpe = Path("kb.dlgpe").read_text(encoding="utf-8")
fb, rulebase, qc = graal.load_dlgp(dlgpe)
analyser = rulebase.analyse()
display_analysis(analyser)
```

```
Saturation finie? False
Réécriture finie? False
Terminaison d'un algorithme? True
```

Analyse plus fine possible avec l'outil Kiabora.

Compilation d'une base de règles

La compilation est une technique d'optimisation de la réécriture développée au cours de la thèse de Mélanie König.

Sans compilation

L'exemple suivant montre une explosion exponentielle de la taille de la requête réécrite.

In [11]:
```
dlgpe = """
a(X, Y) :- a1(Y, X).
```

```
a(X, Y) :- a2(X, Y, X).
a(X, Y) :- a3(Y, X).
b(X, Y) :- b1(X, Y).
b(X, Y) :- b2(X, Y).
b(X, Y) :- b3(X, Y).
r(X, Y) :- a(X, Z), b(Z, Y).
"""
rulebase = graal.create_rulebase(dlgpe)
query = graal.create_query("?(X) :- r(a, X).")
query.rewrite(rulebase)
print(query)
```

```
?(X) :- b3(Graal_EE54, X), a(a, Graal_EE54).
?(X) :- b(Graal_EE54, X), a1(Graal_EE54, a).
?(X) :- b(Graal_EE54, X), a3(Graal_EE54, a).
?(X) :- b2(Graal_EE54, X), a(a, Graal_EE54).
?(X) :- b1(Graal_EE54, X), a3(Graal_EE54, a).
?(X) :- b(Graal_EE54, X), a(a, Graal_EE54).
?(X) :- b1(Graal_EE54, X), a1(Graal_EE54, a).
?(X) :- b3(Graal_EE54, X), a1(Graal_EE54, a).
?(X) :- b1(Graal_EE54, X), a(a, Graal_EE54).
?(X) :- b3(Graal_EE54, X), a3(Graal_EE54, a).
?(X) :- r(a, X).
?(X) :- b3(Graal_EE54, X), a2(a, Graal_EE54, a).
?(X) :- a2(a, Graal_EE54, a), b2(Graal_EE54, X).
?(X) :- b(Graal_EE54, X), a2(a, Graal_EE54, a).
?(X) :- a1(Graal_EE54, a), b2(Graal_EE54, X).
?(X) :- b1(Graal_EE54, X), a2(a, Graal_EE54, a).
?(X) :- a3(Graal_EE54, a), b2(Graal_EE54, X).
```

Avec compilation

La réécriture avec compilation réécrit la requête uniquement avec les règles non compilables, mais en tenant compte des règles compilées.

In [12]:
```
query = graal.create_query("?(X) :- r(a, X).")
compilation = rulebase.compile(CompilationType.ID)
print(compilation.compiled)
print("-------------")
query.compiled_rewrite(compilation)
print(query)
```

```
[R:1761914900430_34] a(X, Y) :- a1(Y, X).
[R:1761914900430_35] b(X, Y) :- b2(X, Y).
[R:1761914900430_36] b(X, Y) :- b3(X, Y).
[R:1761914900430_37] b(X, Y) :- b1(X, Y).
[R:1761914900430_38] a(X, Y) :- a3(Y, X).
[R:1761914900430_39] a(X, Y) :- a2(X, Y, X).
-------------
?(X) :- a(a, Graal_EE105), b(Graal_EE105, X).
?(X) :- r(a, X).
```

La requête réécrite avec compilation n'est pas suffisante

In [13]:
```
factbase = graal.create_factbase("a3(c, a), b2(c, b).")
answers = query.evaluate(factbase)
display_ans(answers)
```

```
Answers:
```

Solution 1: réécrire avec les règles compilées

In [14]:
```
qsave = query.copy()
if (compilation.compiled is not None):
    query.rewrite(compilation.compiled)
print(query)
answers = query.evaluate(factbase)
display_ans(answers)
```

```
?(X) :- a(a, Graal_EE105), b(Graal_EE105, X).
?(X) :- b2(Graal_EE105, X), a3(Graal_EE105, a).
?(X) :- a2(a, Graal_EE105, a), b(Graal_EE105, X).
?(X) :- a(a, Graal_EE105), b1(Graal_EE105, X).
?(X) :- b3(Graal_EE105, X), a2(a, Graal_EE105, a).
?(X) :- a(a, Graal_EE105), b3(Graal_EE105, X).
?(X) :- b2(Graal_EE105, X), a2(a, Graal_EE105, a).
?(X) :- a3(Graal_EE105, a), b(Graal_EE105, X).
?(X) :- a1(Graal_EE105, a), b3(Graal_EE105, X).
?(X) :- a3(Graal_EE105, a), b3(Graal_EE105, X).
?(X) :- r(a, X).
?(X) :- b1(Graal_EE105, X), a1(Graal_EE105, a).
?(X) :- b1(Graal_EE105, X), a2(a, Graal_EE105, a).
?(X) :- a(a, Graal_EE105), b2(Graal_EE105, X).
?(X) :- a1(Graal_EE105, a), b(Graal_EE105, X).
?(X) :- b1(Graal_EE105, X), a3(Graal_EE105, a).
?(X) :- b2(Graal_EE105, X), a1(Graal_EE105, a).
Answers:
{'X': 'b'}
```

Cette réécriture de la réécriture semble inutile, mais est pourtant plus rapide que la réécriture directe sans compilation ! Voir expérimentations Mélanie König.

Solution 2: saturer avec les règles compilées

In [15]:
```
query = qsave
fbsave = factbase.copy()
if (compilation.compiled is not None):
    factbase.saturate(compilation.compiled)
print(factbase)
answers = query.evaluate(factbase)
display_ans(answers)
```

```
a3(c, a), b2(c, b), a(a, c), b(c, b).
Answers:
{'X': 'b'}
```

Solution 3: évaluer avec les règles compilées

In [16]:
```
factbase = fbsave
# answers = query.compiled_evaluate(factbase, compilation)
```

Certainement une bonne solution, mais pas encore implémentée dans Integraal. C'est pourtant ce qu'on avait, il y a quelques années, avec Cogitant (2000)!

Mappings

Les mappings permettent de construire une base de faits à partir de données extraites d'une source quelconque, comme par exemple une base de données relationnelle, un store RDF, un fichier Excel ou un document Json... Malheureusement, Py4Graal ne prend pas (encore) en compte ces mappings, et, pour la suite de cette démonstration, nous allons utiliser un autre outil, Integraal CLI.

> **Memento pour la démo du CLI:**
>
> - Utilisation des mappings en saturation
>   - `load mappings.dlgpe`
>   - `peek`
>   - `saturate`
>   - `peek` or `inspect` or `save export.dlgpe`
>   - `assert '?(X) :- hasColor(X, "blue").'`
>   - `evaluate`
> - Utilisation des mappings en réécriture
>   - `load mappings.dlgpe`
>   - `assert '?(X) :- hasColor(X, "blue").'`
>   - `peek`
>   - `rewrite`
>   - `peek`
>   - `evaluate`