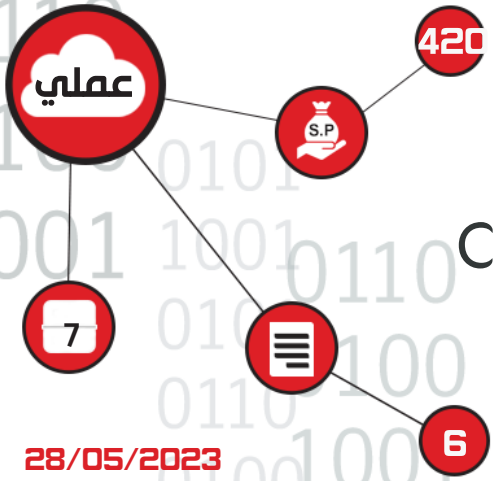

Cut & Fail in prolog

عملي مشترك
محتوى مجاني غير مخصص للبيع التجاري

28/05/2023

RB Informatics;

مبادئ الذكاء الصنعي

- تكلمنا سابقاً عن مبدأ عمل ال prolog وذكرنا أنّه قائم على مبدأ العودية أو ال Back track.
- بالحالة العامة عند كتابة Rules معينة و query متعلقة بها وقمنا بإيجاد الحل عند الضغط على (;) فإنّه يقوم بإرجاع (false) لأنه في السنتاكس الخاص بال prolog يقوم بإيجاد الحل ولكن يعود مرة أخرى لاختبار حلول أخرى لذلك يرد (false).
- ولتجنب هذه الحالة نستخدم cut.

مبدأ عمل ال cut:

- أولاً رمزها هو " ! " وتوضع داخل ال body التابع لل Rules أي في الجانب الأيمن بعد اختبار الشروط.
- مبدأ عملها هو عندما يكون الشرط الذي يسبقها صحيح يقوم بقطع جميع استدعاءات الأفرع الأخرى (القواعد الأخرى) أي يكتفي بالحل الصحيح الحالي ولا يقوم بالاستدعاء من أجل الحلول الباقية الصحيحة.

ملاحظة:

غالباً يكون ترتيب القواعد عند وجود ال cut هو:

- القاعدة التي تحتوي على الشرط العام وتحتوي ال cut.
- القاعدة التي تخالف الشرط العام.
- القاعدة التي تحتوي شرط التوقف.

مثال بسيط لاستخدام ال cut:

```
likes (john , pizza).
likes (john , sushi).
likes (sara , sushi).
eats (john , X) :- likes (john , X).
```

لو قمنا بال query التالي:

```
?- eats (john , X).
   X = pizza ;
   X = sushi.
```

- ولكن لو أردت أن يظهر لي أكلة واحدة فقط نعدل في قاعدة eats.

```
eats (john , X) :- likes (john , X) , ! .
```

والآن لو قمنا بال query:

```
?- eats (john , X).
   X = pizza.
```

وهنا أفادنا استخدام cut لمنع البحث عن المزيد من الأكلات التي يحبها الشخص.

وبالتالي يتم توفير الوقت والجهد في البحث وتحسين أداء البرنامج.

مثال لفهم فكرة ال cut:

أراد شخص شراء ساعة ذكية بحيث تكون الساعة ذات تصميم جميل وتكون ذكية.

لدينا :

```
design (apple).
design (samsung).
design (huawei).
smart (apple).
smart (samsung).
smart (huawei).

watch (X) :- design (X) , smart (X) , ! .
```

ويكون الخرج هنا X = apple **فقط**.

لن يعيد النتائج الأخرى لأننا وضعنا " ! " أي نريد الاكتفاء بحل وحيد صحيح وليس جميع الحلول الصحيحة.

مثال آخر :

حل مسألة ال max بطريقة ال cut:

الطريقة السابقة:

```
max (X , Y , X) :- X>Y.
max (X , Y , Y) :- X=<Y.

?- max (2 , 3 , L).
   L = 3 ;
   False.
```

باستخدام ال cut :

```
max(X , Y , X) :- X>Y , ! .
max(X , Y , Y) :- X =< Y.

?- max(3 , 1 , L).
    L = 3.
```

تحقق من ال query من السطر الأول وتم عمل ال cut لعدم تجربة حلول أخرى، بينما لو كان:

```
?- max(1 , 3 , L).
    L = 3.
```

لم يصل لل cut من الأصل لأنه عند تطبيق السطر الأول تكون النتيجة خاطئة لذا ينتقل للسطر الثاني.

تمرين (1): تابع يقوم بقسم السلسلة إلى سلسلتين الأولى موجبة والثانية سالبة:

- سنقوم بتمرير ثلاث سلاسل: السلسلة التي أريدها والسلسلة التي ستحمل العناصر الموجبة والسلسلة التي ستحمل العناصر السالبة.
- شرط التوقف: في حال كانت السلسلة فارغة فالسلسلتين الباقيتين حتماً سيكونوا فارغتين.
- في حال كان رأس السلسلة موجب فيتم إضافته إلى القائمة الثانية ثم يتم استدعاء الدالة مرة أخرى وباستخدام ال cut لن يتم الدخول إلى القاعدة الثانية.
- وفي حال كان رأس السلسلة سالب فيتم إضافته إلى القائمة الثالثة ثم استدعاء الدالة مرة أخرى.

```
split( [ ] , [ ] , [ ] ).
split( [ H | T ] , [ H | Tp ] , N) :- H>=0 , split( T , Tp , N) , ! .
split( [ H | T ] , P , [ H | TN ] ) :- H < 0 , split( T , P , TN).
```

نقوم بال query التالية:

```
?- split( [ 1 , 2 , 3 , -1 , -2 , -3 ] , P , N ).
    P = [ 1 , 2 , 3 ] ,
    N = [ -1 , -2 , -3 ].

?- split([  -1 , -2 , -3 ] , P , N ).
    P = [ ] ,
    N = [ -1 , -2 , -3 ].
```

تمرين (2): تابع يقوم بقص السلسلة من العنصر الأول إلى العنصر N:

- نمرر السلسلة التي أريدها مع المتغير N والسلسلة الناتجة.
- شرط التوقف: في حال كانت السلسلة فارغة فسيعيد سلسلة فارغة.
- في حال كان المتغير N يساوي الصفر فالسلسلة الناتجة تكون هي ذاتها السلسلة المدخلة وهنا نستخدم ال cut لكي لا يدخل إلى القاعدة الثانية.
- وفي حال كان أكبر من الصفر نعرف متغير N1 وهو منقوص المتغير N بمقدار واحد ثم نستعدي التابع عودياً من أجل ذيل السلسلة T و N1.

```
cut( [ ] , _ , [ ] ).
cut( L , 0 , L ) :- !.
cut( [ _ | T ] , N , L ) :- N>0 , N1 is N-1 , cut(T , N1 , L) , ! .
```

نقوم بال query التالي :

```
?- cut( [ a , b , c , d , e ] , 3 , L ).
    L = [ d , e ].

?- cut( [ 1 , 2 , 3 , 4 , 5 ] , 0 , L ).
    L = [ 1 , 2 , 3 , 4 , 5 ].
```

تمرين (3): تابع يقوم بإيجاد السلسلة الأولى فرق الثانية.

(أي العناصر التي تنتمي للسلسلة الأولى ولا تنتمي للسلسلة الثانية).

- نمرر السلسلتين التي أريدهما ومتغير من أجل النتيجة.
- شرط التوقف: في حال كانت السلسلة الأولى فارغة فحتماً النتيجة ستكون سلسلة فارغة.
- نختبر وجود رأس السلسلة الأولى في السلسلة الثانية من خلال التابع member في حال وجوده نتجاهله (نستخدم ال cut لمنع الدخول لبقية القواعد وإضافة العنصر للسلسلة) ثم نستدعي التابع عودياً من أجل باقي السلسلة (T).
- وفي حال عدم وجوده نقوم بإضافته إلى سلسلة النتيجة ونستدعي عودياً من أجل البقية.

```
list_diff( [ ] , _ , [ ] ).
list_diff( [ H | T ] , L2 , L ) :- member( H , L2 ) , ! , list_diff( T , L2 , L).
list_diff( [ H | T ] , L2 , [ H | L ] ) :- list_diff( T , L2 , L).
```

نقوم بال query التالي:

```
?- list_diff( [ 1 , 2 , 3 , 4 ] , [ 2 , 4 ] , L).
    L = [ 1 , 3 ].
```

تمرين (4): تابع يقوم بدمج سلسلتين:

- نمرر السلسلة الأولى والسلسلة الثانية ومتغير الناتج.
- شرط التوقف: في حال كانت السلسلة الأولى فارغة فالناتج هو السلسلة الثانية.
- إذا كان رأس السلسلة الأولى موجود في السلسلة الثانية فيتم تجاهله (أي نستخدم cut) ويتم استدعاء عودي للدالة باستخدام القائمة المتبقية T والقائمة الثانية L.
- وإذا لم يكن العنصر الأول H في السلسلة الأولى موجود في السلسلة الثانية يتم إضافته إلى رأس القائمة الثانية ويتم استدعاء الدالة بشكل متكرر.
- ويستمر الاستدعاء العودي حتى تصبح القائمة الأولى فارغة.

```
collecting( [ ] , L , L ).
collecting( [ H | T ] , L , M ) :- member( H , L ) , ! , collecting( T , L , M ).
collecting( [ H | T ] , L , M ) :- collecting( T , [ H | L ] , M).
```

نقوم بال query التالي:

```
?- collecting( [ 1 , 2 , 3 ] , [ 2 , 3 , 4 ] , M).
   M = [ 1 , 2 , 3 , 4 ].
```

Fail

- تشبه ال Not ولكن لا تقوم بنفس عملها.
- يتم استخدام كلمة fail للإشارة إلى فشل الاستعلام أو عدم توفر الحلول المطلوبة ، ويتم استخدامها عادةً في حالات الفشل في تنفيذ الأهداف المحددة في القواعد أو الاستعلامات .
- وعند حدوث "fail" يقوم ال prolog بالتحقق من وجود حلول أخرى ممكنة ، وإذا لم يتم العثور على أي حلول ممكنة يتم إرجاع fail كنتيجة للاستعلام .

مثال 1 :

```
check (X) :- X>0 , X < 10.
check ( _ ) :- fail.
```

لدينا تابع check يقوم بالتحقق من قيمة العنصر إذا كان بين 0 و 10 فسيتم إرجاع العنصر أو سيتم إرجاع fail.

مثال 2 :

ليكن لدينا الحقائق التالية:

```
married( aya , fares).
married( maya , mohamad ).
married(anahalla , karam).
single(kayla).
```

1. نكتب قاعدة لنثبت أن شخص ما متزوج:

يكون الشخص متزوج إذا كان متزوج من أحد أو أحد متزوج منه ويكفي إثبات طرف واحد لتصح العلاقة.

```
is_married( X ) :- married( X , _ ) , ! ; married( _ , X ).
```

2. نكتب قاعدة لنثبت أن شخص ما single:

جميع الأشخاص single **حتى يتم إثبات أنهم** married.

```
single(X) :- is_married(X) , ! , fail.
single(X).
```

أي إذا استطعت أن تثبت أن الشخص married **قف وافشل "رد** false**" ، أمّا خلاف ذلك سينتقل للسطر الثاني ويرد الناتج.**

```
Query :  ?- single(aya).
           false.

         ?- single(kayla).
           true.
```

When something is important enough, you do it even if the odds "
".are not in your favor

-Elon Musk

-نهاية المحاضرة-