كلية الهندسة المعلوماتية
السنة الثالثة

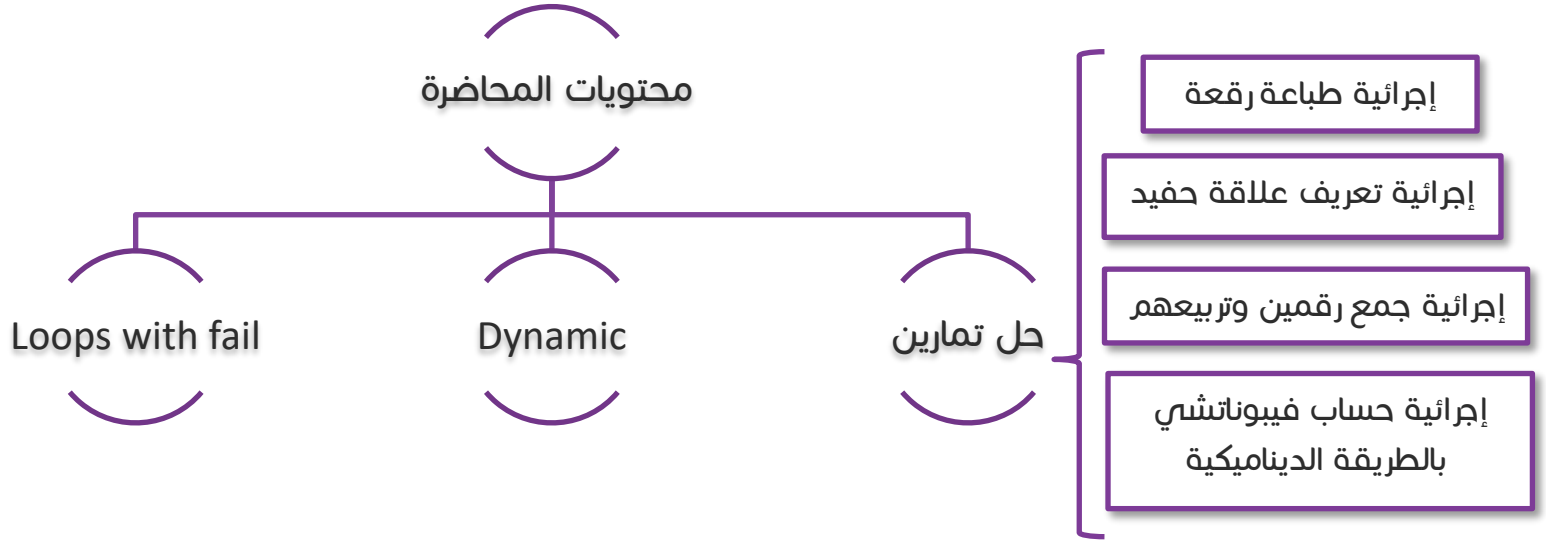
Fail and Dynamic

عملي مشترك

محتوى مجاني غير مخصص للبيع التجاري

مبادئ الذكاء الصنعي

RB Informatics ; 02/06/2022



تحدثنا في المحاضرة السابقة عن أحد استخدامات ال fail إذ تم الاستفادة منها لتحقيق مفهوم ال (not) وذلك عندما استخدمنا cut + fail

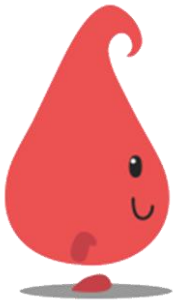
حيث يوجد العديد من الاستخدامات لل fail سوف نناقش في هذه المحاضرة مبدأ الحلقات looping باستخدام fail.

مقدمة

- في prolog عندما نستعلم عن طلب ما يعيد أول حل true ثم يقف ولا يعيد جميع الحلول الموجودة

أما في حال لم يجد أي حل صحيح يكون قد اختبر جميع الحقائق لديه ويتوقف عن التنفيذ ويعيد false

وبالنظر إلى عمل ال fail إذ بإضافتها سوف يعيد false أي سوف يكمل باقي الاستدعاءات ليتحقق من باقي الحقائق لديه فيمكن طباعة الحل قبل كتابة fail ، في هذه الحالة نكون قد حصلنا على الحل ولم يتوقف البرنامج فيكمل شجرة الاستدعاءات إلى أن يختبر جميع الشروط ونكون قد حصلنا على جميع الحلول الصحيحة الممكنة .

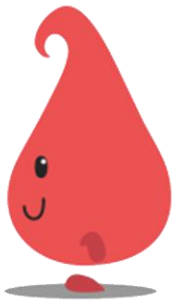


- مثال:

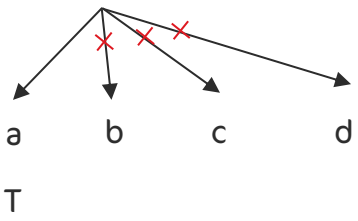
```
p(a).
p(b).
p(c).
p(d).
print_all :- p(X), write(X), nl  , fail .
```

Query

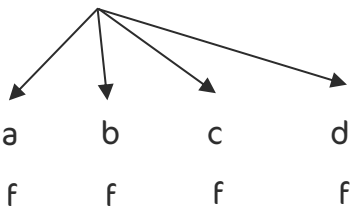
```
?- print_all .
= a ;
  b ;
  c ;
  d ;
false.
```



- في حال لم نستخدم fail سيطبع أول حل صحيح فقط وهو a



Output: a



Output: a
b
c
d
false

Dynamic

منذ أن بدأنا في ال prolog كنا نكتب جميع الحقائق والقواعد في ملف لاحقته pl وعند طلب الاستعلامات يقوم بقراءة ما لدينا من حقائق وقواعد ثابتة في هذا الملف وعند كل تعديل نذهب ونعدل على الملف لإضافة أو حذف حقيقة ما .

أما إذا أردنا إضافة حقيقة في أثناء تنفيذ البرنامج عندها يجب استخدام ال Dynamic .

حيث يعتمد مبدأ ال Dynamic على إضافة حقائق أو قواعد أثناء تنفيذ البرنامج بحيث نختصر الوقت في تنفيذ بعض البرامج العودية وذلك بتخزين القيمة التي حصلت عليها وعند الحاجة إليها لا أقوم بإعادة حسابها وإنما أعيد ما تم تخزينه مسبقاً .

ملاحظة : تبقى الحقائق والقواعد الديناميكية معرفة بعد إضافتها إلى أن يتم إغلاق البرنامج .

- ومن بعض التوابع التي سوف نستخدمها :

assert(factname(value)).	يخزن الحقيقة في نهاية قاعدة المعرفة
asserta(factname(value)).	يخزن الحقيقة في بداية قاعدة المعرفة
retract(factname(value)).	يحذف الحقيقة
retractall(factname(_)).	يحذف جميع الحقائق

مثال :

- بفرض أن قاعدة المعرفة فارغة، وعند طلب الاستعلام :

```
?- p(1) .
```

- سوف يعيد لنا خطأ أنه لم يجد حقيقة بهذا الاسم في قاعدة المعرفة لديه

وعند طلب :

```
?- assert(p(1)) .
= true
```

- يعيد true ؛ لأنه قام بإضافة الحقيقة التالية وعند إعادة الطلب :

```
?- p(1) .
= true
```

- وعند إضافة حقيقة ثانية باستخدام assert(p(2)) يقوم بإضافتها في آخر قاعدة المعرفة فعند طلب:

```
?- p(X) .
X = 1 ;
X = 2.
```

- أما في حال أردنا أن نضيف في بداية قاعدة المعرفة وذلك باستخدام asserta(p(3)).

```
?- p(X) .
X = 3 ;
X = 1 ;
X = 2.
```

حيث جميع الأمثلة السابقة بدأنا من قاعدة معرفة فارغة.

- أما في حال كانت الحقيقة ρ(1) موجودة قبل بدء البرنامج وأردنا أن نضيف ρ(2) فرضاً ، سيعطينا خطأ بأننا لم نحدد أن الحقيقة ρ(1) هي حقيقة ديناميكية ونقوم بذلك من خلال

```
:- dynamic ρ/1
ρ(0) .
```

(1) (2)

حيث :

1. اسم الحقيقة التي نريد أن نجعلها ديناميكية.
2. عدد بارمترات هذه الحقيقة .

نقوم بإضافة هذا السطر قبل الحقائق التي نريد تحويلها إلى حقائق ديناميكية؛ يُفضل وضعها أول الملف.

مثال على الإدخالات الديناميكية:

```
?- assert(happy(mary)) .
    = true
```
Query

```
?- assert(happy(harry)) .
    = true
```
Query

```
?- assert(happy(sara)) .
    = true
```
Query

```
?- assert(happy(harry)) .
    = true
```
Query

```
?- listing .
    = happy(mary)
    = happy(harry)
    = happy(sara)
    = happy(harry)
```
Query

```
?- happy(X) .
    X = mary ;
    X = harry ;
    X = sara ;
    X = harry ;
```
Query

بهذه الطريقة قمنا بإضافة متغيرات dynamic أثناء runtime البرنامج .

نقوم باستعمال تعليمة أخرى لتقوم بحذف المدخلات :

```
?- retract(happy(marry)) .
    = true
```

```
?- listing
```

نلاحظ من نتائج الكويري أنه قام بحذف marry وإرجاع فقط الثلاث الباقية.

```
?- retractall(happy(marry)) .
    X = harry ;
    X = sara ;
    X = harry ;
```

```
?- listing .
```

نلاحظ حذف جميع المتحولات .

1. كتابة إجراء يأخذ رقمين يجمعهم ثم يربعهم .

```
:- dynamic lookup/3 .
add_and_squer(X,Y,Z ): Z is (X+Y) * (X+Y) , assert(lookup(X,Y,Z)) .
```

Query-1-

```
?- add_and_squer(2,3,Z ) .
Z = 25.
```

Query-2-

```
?- add_and_squer(3,2,Z ) .
Z = 25.
```

الطريقة صحيحة؛ لكن لا تحقق مفهوم ال dynamic لأننا نريد تحقيق الشرط التالي :

I. إذا الرقمين تم جمعهما سابقاً اجلب النتيجة.

II. وإلا اجمع وخزن النتيجة .

❖ نعدل على ال Rule

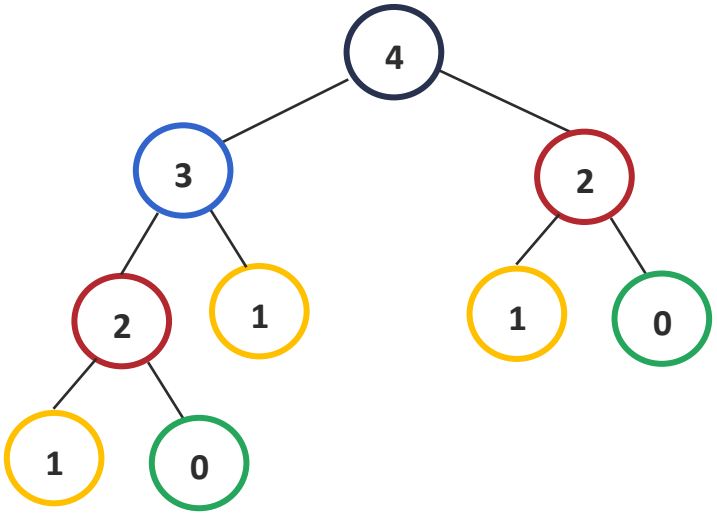
```
add_and_squer(X,Y,Z ):- lookup(X,Y,Z), ! .
add_and_squer(X,Y,Z ) :- Z is (X+Y) * (X+Y), assert(lookup(X,Y,Z)) .
```

- لو قمنا بالاستعلام عن ال Query السابق -2- لن يقوم بحسابها مرة ثانية وإنما يرجع نتيجة الـ Query 1 نفسها لأنه قام بحسابها سابقاً .
- ولمسح جميع البيانات في الذاكرة المخزنة ضمن lookup :

```
?- retractall(lookup(_,_,_)) .
```

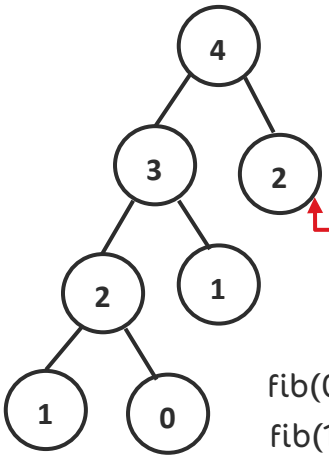
2. تمرين كتابة برنامج لحساب سلسلة فيبوناتشي .

حيث تكون شجرة الاستدعاءات العودية لـ fib(4) على الشكل التالي:



- نلاحظ عدد كبير من التكرارت وذلك من أجل حساب رقم صغير لفيبوناتشي وفي حال أردنا حساب عدد كبير سوف يكون حسابه مستحيل وذلك لكثرة الاستدعاءات العودية المكلفة للوقت (رتبة التعقيد الزمني لذا الحل تكون 2^n)

- أما في حال استخدام البرمجة الديناميكية في حل مسألة فيبوناتشي ، نقوم بتخزين كل نتيجة قمنا بحسابها من أجل العودة إليها في حال احتجنا لها مرة أخرى فتصبح الشجرة السابقة من الشكل:



لا داعي لحساب fib(2) مرة أخرى إذ قمنا بحسابها في أول فرع عودي، ونقوم بإرجاع القيمة المحسوبة فقط.

- فتكون القاعدة قبل تحويلها إلى ديناميكية من الشكل:

```
fib(0,0).
fib(1,1).
fib(X,Z) :- X>1, X1 is X-1, X2 is X-2, fib(X1,Y1), fib(X2,Y2), Z is Y1+Y2.
```

- ولتحويلها إلى قاعدة ديناميكية نقوم بإضافة:

1. سطر تعريف أن الحقيقة ديناميكية في بداية التابع.
2. تعليمة لإضافة القيم الناتجة من الحساب.

```
:- dynamic fib/2.
fib(0,0).
fib(1,1).
fib(X,Z) :- X>1, X1 is X-1, X2 is X-2, fib(X1,Y1), fib(X2,Y2), Z is Y1+Y2, asserta(fib(X,Z)).
```

ملاحظة:
لماذا استخدمنا في المثال السابق asserta ولم نستخدم assert ؟
لأنه لما تحدثنا سابقاً أن البرولوغ يقوم بتنفيذ سطر سطر بالترتيب ، ولو قمنا بإضافة الناتج في نهاية قاعدة المعرفة وطلب حساب فيبوناتشي لعدد قد تم حسابه سابقاً سوف يبدأ من السطر الأول ويعود لحساب الناتج مرة أخرى ونكون لم نستفد من عملية التخزين هذه بشيء فيجب إضافة الناتج في البداية للاختبار وجوده قبل البدء بعملية حسابه.

✓ أما في الحل السابق نكون قد حصلنا على أجوبة صحيحة بطريقة ديناميكية وتخفيض رتبة التعقيد الزمني من 2^n إلى n ، بقي لدينا مشكلة واحدة وهي أن البرنامج يعطي قيم متكررة ولا يقف عند إعطاء الحل:

Query

```
?- fib(6, X).
X=8 ;
X=8....
```

✓ ولتفادي هذه المشكلة نقوم بتخزين الناتج في حقيقة جديدة وليس ضمن نفس الحقيقة على الشكل التالي:

```
:- dynamic solution /2.
fib(0,0).
fib(1,1).
fib(X,Z) :- solution(X,Z), !.
fib(X,Z) :- X>1, X1 is X-1, X2 is X-2, fib(X1,Y1), fib(X2,Y2), Z is Y1+Y2, asserta(solution(X,Z)).
```

حيث: solution هي الحقيقة التي خزنا فيها النواتج واستعلمنا عن وجود الناتج مسبقاً فإن لم يكن موجود حسبناه وخزناه أما إن كان موجود أعدناه وتوقف البرنامج لاستخدام cut .

Query

```
? fib(10,X).
X= 55.
```

Find all

بفرض لدينا الحقائق التالية:

```
like( a, apple ).
like( a, banana ).
like( b, apple ).
like( c, apple ).
```

- عند الاستعلام عن:

Query

```
? like( X, apple).
X=a ;
X=b ;
X=c.
```

إذ يقوم بإعطاء حل في كل مرة، أما في حال أردنا الحلول التي تحقق العلاقة جميعاً ضمن سلسلة نستخدم find all على الشكل التالي:

```
?- find all( X, like(X, apple), L).
L= [ a, b, c ].
```

إذ يقوم بوضع المتحول الذي نريد جميع القيم له من ثم القاعدة المرادة ومتحول الذي نخزن فيه سلسلة الحل

بعض الأمثلة الأخرى عن find all :

Query

```
?- find all( X, like( a, X ), L ).
L= [ apple, banana ].
```

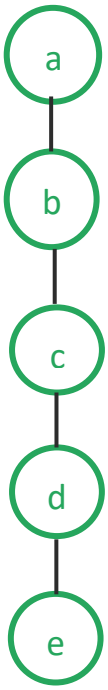
Query

```
?- find all( X, like( X, Y ), L ).
L= [ a, a, b, c ].
```

Query

```
?- find all( Y, like( X, Y ), L ).
L= [ apple, banana, apple, apple ].
```

3. تمرين: لتكن لدينا الشجرة التالية ونريد تعريف علاقة " حفيد / خلف " .




```
child( a, b ).
child( b, c ).
child( c, d ).
child( d, e ).
```

ابن a هو b

```
descendents( X, Y) :- child( X, Y ).
descendents( X, Y) :- child( X, Z ), descendents( Z, Y ).
```

- نريد إيجاد جميع أحفاد a:

Query

```
?- descendents( a, X ).
```

أعادت حلول منفصلة

```
X= b ;
X= c ; X = d ; X= e .
```

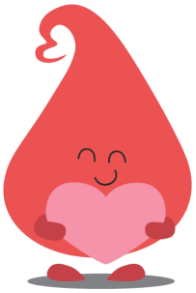
Query

```
?- findall( X, descendents( a, X ), L ).
L= [ b, c, d, e ].
```

4. اكتب إجرائية تطبع عناصر المصفوفة " رقعة" :

```
getNumber(To, To, To) :- !.
getNumber(From, To, N) :- From<To, N=From.
getNumber(From, To, N) :- F is From+1, getNumber(F, To, N).
get_row(R)   :- size(Row, _), getNumber(1, Row, R).
get_col(c)   :- size(_, Column), getNumber(1, Column, C).
print_col(R) :- get_column(C), write(‘ * ‘), write(‘ ‘), write(‘_’), fail.
print_grid   :- get_row(R), \+ print_col(R), nl, fail.
```

تم إضافة لعبة X/O في ملف تطبيق أكواد المحاضرة QR.



ختاماً .. نسأل الله التوفيق لكم جميعاً .

كان معكم فريق مبادئ الذكاء الصنعي القسم العملي

لا تنسونا من صالح دعائكم ..