بسمه تعالی

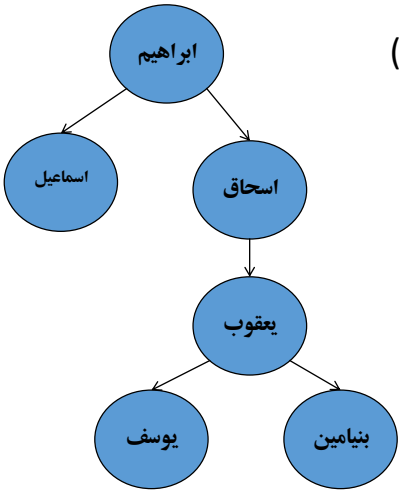
هوش مصنوعی

پرولوگ ۱،۲،۳

نیمسال اوّل ۱۴۰۴-۱۴۰۳

دکتر مازیار پالهنگ
آزمایشگاه هوش مصنوعی
دانشکدهٔ مهندسی برق و کامپیوتر
دانشگاه صنعتی اصفهان

پرولوگ

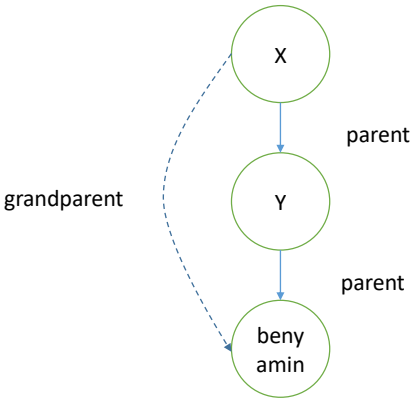


- یک زبان برنامه نویسی منطقی (logic programming)
- رابطهٔ خانوادگی

```
parent(ebrahim,esmail).
parent(ebrahim,eshagh).
parent(eshagh,yaghoub).
parent(yaghoub,yousof).
parent(yaghoub,benyamin).
```

ولی بزرگ

- چه کسی ولی بزرگ بنیامین است؟
- چه کسی ولی بنیامین است؟ فرض Y
- چه کسی ولی Y است؟ فرض X

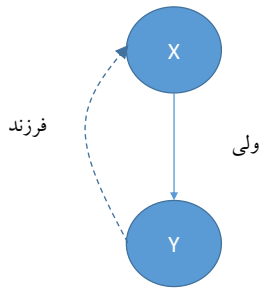


parent(Y,benyamin), parent(X,Y).

grandparent(X,Z) :- parent(Y,Z), parent(X,Y).

فرزند بودن

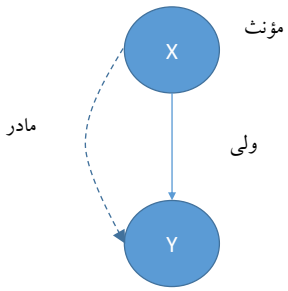
- برای هر X و Y، Y یک فرزند X است اگر X یک ولی Y باشد.




```
offspring(Y,X) :- parent(X,Y).
```

مادر بودن

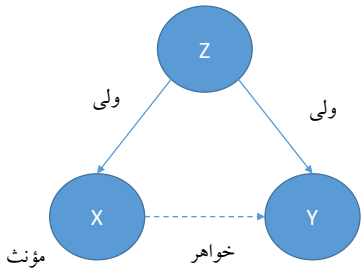
- برای هر X و Y، X یک مادر Y است اگر:
- X ولی Y باشند و
- X مؤنث باشد.




```
mother(X,Y) :- parent(X,Y), female(X).
```

خواهر بودن

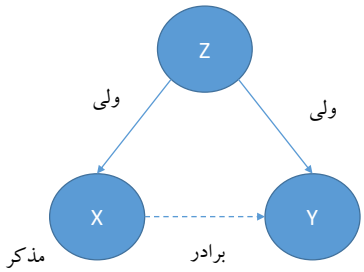
- برای هر X و Y، X یک خواهر Y است اگر:
- هر دوی X و Y ولی مشترکی داشته باشند و
- X مؤنث باشد.




```
sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X).
```

برادر بودن

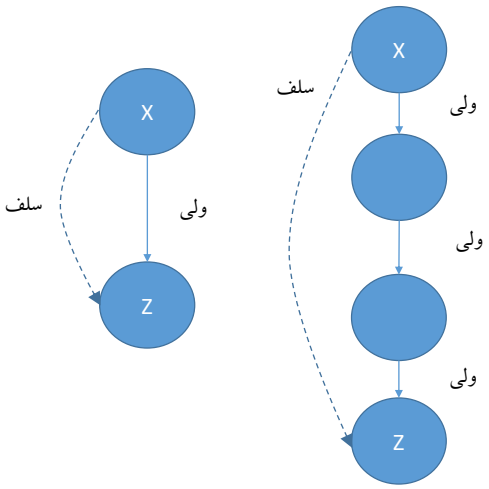
- برای هر X و Y، X یک برادر Y است اگر:
- هر دوی X و Y ولی مشترکی داشته باشند و
- X مذکر باشد.




```
brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X).
```

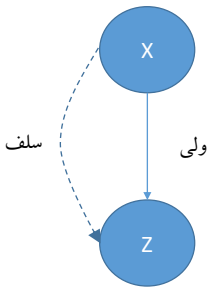
قوانین بازگشتی

- سلف (جد) بودن.
- با دو قانون.
- سلف مستقیم،
- سلف غیر مستقیم

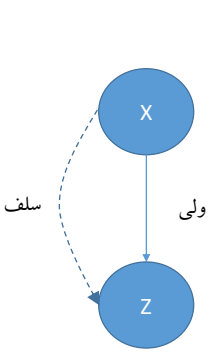


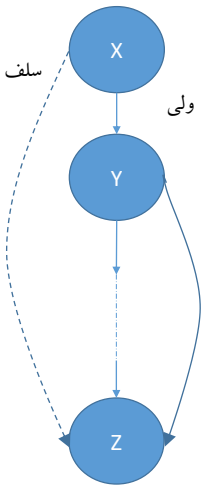
قوانین بازگشتی

- سلف (جد) بودن.
- با دو قانون.
- **سلف مستقیم:**
- برای هر X و Z، X یک سلف Z است اگر
- X یک ولی Z باشد.



قوانین بازگشتی





- سلف (جد) بودن.
- با دو قانون.
- **سلف مستقیم**:
- برای هر X و Z، X یک سلف Z است اگر
- X یک ولی Z باشد.
- **سلف غیر مستقیم**:
- برای هر X و Z، X یک سلف Z است اگر یک Y وجود داشته باشد بطوری که:
- X یک ولی Y باشد و
- Y یک سلف Z باشد.

- قانون نهائی:

```
predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :- parent(X,Y), predecessor(Y,Z).
```

- حال اگر سؤال کنیم:

```
predecessor(ebrahim,yaghoub)?
```

متغیرها

- شروع با حرف بزرگ انگلیسی یا _ (underscore)
- سایر کاراکترها حروف الفبای انگلیسی یا ارقام

X
Result
Book_list
_X23
_23

متغیر گمنام

- می توان از نام گذاری متغیرها خودداری کرد و متغیر گمنام داشت.

```
haschild(X) :- parent(X,Y).
```

- در مثال فوق نام فرزند مهم نیست:

```
haschild(X) :- parent(X,_).
```

- هر بار که یک کاراکتر underscore تنها در یک جمله ظاهر می شودد آن نشان دهندهٔ یک متغیر گمنام جدید است.

```
Somebody_has_child :- parent(_,_).
```

- اگر متغیر گمنام در سؤال وجود داشته باشد مقدار آن هنگامی که پرولوگ به سؤال جواب می دهد نشان داده نخواهد شد.

parent(X,_)?

ساختارها

- اشیاء ساخت یافته (ساختارها) اشیائی هستند که دارای اجزاء متعددی هستند.
- خود اجزاء نیز می توانند به نوبهٔ خود ساختار باشند.
- مانند تاریخ که شامل روز، ماه و سال است.
- نیاز به یک functor
- مثلاً برای تاریخ می توان از فانکتور date استفاده کرد.

```
date(20,khordad,1399)
```

- اجزاء می توانند متغیر یا ساختار دیگری باشند:

```
date(D,khordad,1399)
```

فصل

- همانگونه که دیده شد , در جملات به معنای عطف است.
- برای فصل ;

P :- Q;R.

- P درست است اگر Q درست باشد یا R درست باشد.
- اولویت , از ; بیشتر است

P :- Q,R;S,T,U.

P :- (Q,R);(S,T,U).

P :- Q,R.
P :- S,T,U.

لیستها

- دنباله ای از هر تعداد از اشیاء

[ball,book,pen]

- لیست تهی []
- لیستی که تهی نیست می تواند بصورت دو چیز در نظر گرفته شود:
- اولین شئ که سر (head) لیست نامیده می شود.
- باقیماندهٔ لیست که دم (tail) لیست نامیده می شود.
- در مثال فوق سر لیست ball و دم آن لیست [book,pen]

- علامت | سر و دم لیست را از هم جدا می کند و می تواند مقداری عمومی تر باشد:

[a,b,c] = [a|[b,c]] = [a,b | [c]] = [a,b,c | []]

- از لیستها می توان برای نمایش مجموعه ها استفاده کرد.
- ترتیب اعضاء در مجموعه مهم نیست ولی در یک لیست مهم است.
- یک شئ می تواند چند بار در لیست ظاهر شود.

عضویت

- X یک عضو L است اگر یا:
- X سر L باشد، یا
- X عضوی از دم L باشد.

member(X,L)

```
member(X, [X | Tail].
member(X, [Head | Tail] :- member(X,Tail).
```

member(b,[a,b,c]).

member(b,[a,[b,c]]).

الحاق

- الحاق دو لیست L1 و L2 و ایجاد L3

conc(L1,L2,L3)

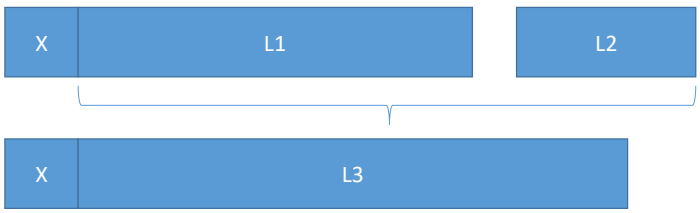
- مثال:

conc([a,b],[c,d],[a,b,c,d])

• اگر آرگومان اول لیست تهی باشد، آنگاه آرگومان دوم و سوم مشابه هستند.

conc([],L,L).

• اگر آرگومان اول تهی نباشد، آنگاه یک سر و یک دم دارد.



conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).

اضافه کردن یک عضو

- اضافه کردن عضو جدید در ابتدای لیست:

```
add(X,L,[X|L]).
```

حذف یک عضو

del(X,L,L1).

- اگر X سر لیست باشد آنگاه نتیجه پس از حذف دم لیست می باشد.
- اگر X در دم لیست باشد، آنگاه آن از دم حذف می شود.

```
del(X,[X | Tail],Tail).
del(X,[Y | Tail],[Y | Tail1]) :- del(X,Tail,Tail1).
```

ریاضی

- محاسبات رقمی در پرولوگ زیاد استفاده نمی شود.
- امکانات محاسباتی آن محدود و ساده است.
- جمع +، تفریق –، ضرب *، تقسیم اعشاری /، تقسیم صحیح div، باقیماندهٔ تقسیم mod
- برای عملگر انتساب از is استفاده می شود.

```
X is 3/2, Y is 3 div 2.
```

```
X = 1.5
Y = 1
```

ریاضی

- عملگرهای مقایسه ای >، >=، <، =<
- مقادیر X و Y (بعد از ارزیابی) برابرند.
- مقادیر X و Y برابر نیستند.
- دقت: = و =:= با هم یکسان نیستند.

X =:= Y

X =\= Y

```
4 = 4.
true
2+2 = 4.
false
2+2 =:=4
true
```

ریاضی

- یافتن طول یک لیست: length(List, N)
- اگر لیست تهی است، طول آن صفر است.
- اگر لیست تهی نیست، آنگاه لیست یک سر و یک دم دارد و طول آن برابر است با یک به علاوهٔ طول دم لیست.

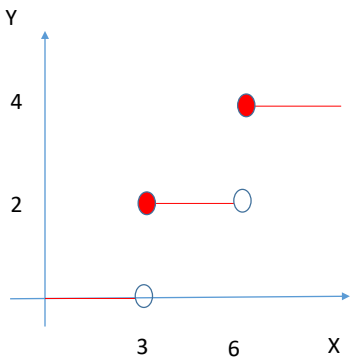
```
length([],0)

length([_|Tail],N) :- length(Tail,N1), N is 1+N1.
```

- دو مؤلفهٔ بدنهٔ قانون را نمی توان جابجا کرد.

کنترل عقبگرد

- مثال زیر را در نظر بگیرید:
- اگر x<3 انگاه y=0
- اگر 3 =>x و x<6 آنگاه y=2
- اگر 6=>x آنگاه y=4




```
f(X,0) :- X < 3.          %Rule 1
f(X,2) :- 3=<X, X<6.      %Rule 2
f(X,4) :- 6=<X.
```

- آزمایش ۱:

f(1,Y), 2<Y?

```
f(X,0) :- X < 3.          %Rule 1
f(X,2) :- 3=<X, X<6.      %Rule 2
f(X,4) :- 6=<X.
```

- باعث عقبگرد بیهوده
- جلوگیری از عقبگرد با عملگر قطع یا cut (!):

```
f(X,0) :- X < 3, !.       %Rule 1
f(X,2) :- 3=<X, X<6, !.   %Rule 2
f(X,4) :- 6=<X.
```

• آزمایش ۲:

```
f(X,0) :- X < 3, !.          %Rule 1
f(X,2) :- 3=<X, X<6, !.      %Rule 2
f(X,4) :- 6=<X.
```

f(7,Y)?

Y=4

```
f(X,0) :- X < 3, !.          %Rule 1
f(X,2) :- X<6, !.            %Rule 2
f(X,4) .
```

- حال اگر قطع را برداریم رفتار برنامه متفاوت خواهد بود.

```
f(X,0) :- X < 3, !.          %Rule 1
f(X,2) :- X<6, !.            %Rule 2
f(X,4) .
```

```
f(1,Y)?
```

```
f(X,0) :- X < 3.             Y=0;
f(X,2) :- X<6                Y=2;
f(X,4) .                     Y=4;
                             false
```

- اگر داشته باشیم:

H :- B1, B2, …., Bm, !, …., Bn.

- تا قبل از رسیدن به قطع عقبگرد انجام می شود.
- هنگام رسیدن به قطع عقبگرد متوقف می شود.
- ضمنا هر تلاش دیگر برای قانون دیگری که سر آن با قانون فعلی یکسان است نیز متوقف می شود.
- مثال:

C :- P,Q,R,!,S,T,U.
C :- V.
A :- B,C,D.

A?

- قطع فقط در اجرای هدف C تأثیر گذار است و از دید A پنهان است.

- بازدید دوباره:

```
member(X, [X | Tail]).
member(X, [Head | Tail] :- member(X,Tail).
```

- اگر یک عضو چند بار در لیست باشد همهٔ آنها را جک می کند.

```
member(X, [X | _ ]) :- !.
member(X, [ _ | Tail] :- member(X,Tail).
```

نقیض به عنوان شکست

- امین همهٔ حیوانات را دوست دارد به جز مارها را.
- امین همهٔ حیوانات را دوست دارد:

```
likes(amin,X) :- animal(X).
```

- استفاده از fail هدفی که همیشه شکست می خورد.

```
likes(amin,X) :- snake(X), !, fail.
likes(amin,X) :- animal(X).

likes(amin,X) :- sanke(X), !, fail; animal(X).
```

• X با Y متفاوت است:

```
different(X,X) :- !, fail.
different(X,Y).

different(X,Y) :- X=Y, !; fail; true.
```

نقیض

- درست باشد هر گاه Goal نادرست است.

```
not(Goal)
```

- بصورت:

```
not(P) :- P, !, fail; true.
```

- بجز مار:

```
not(snake(X)).
```

- برخی پرولوگها اجازه نوشتن بصورت عملگر پیشوندی

```
not snake(X).
```

.•

likes(amin,X) :- animal(X), not(snake(X)).

different(X,Y) :- not(X=Y).

• دقت:

```
P :- a,b.
P :- c.
```
$P \Leftrightarrow (a \& b) \vee c.$

```
P :- a, !, b.
P :- c.
```
$P \Leftrightarrow (a \& b) \vee (\sim a \& c).$

```
P :- c.
P :- a, !, b.
```
$P \Leftrightarrow c \vee (a \& b).$

- قطعی که معنای توصیفی برنامه را تغییر ندهد قطع سبز (green cut) گویند.
- همانند:

P :- c.
P :- a, !, b. $P \Leftrightarrow c \vee (a \& b)$.

- قطعی که معنای توصیی برنامه را تغییر دهد قطع قرمز (red cut) نامیده می شود.
- احتیاط در استفاده از نقیض:

not(human(amin))?

true

- فرض جهان بسته (closed world assumption)

دانشگاه صنعتی اصفهان

مازیار پالهنگ - پرولوگ