Data Structure

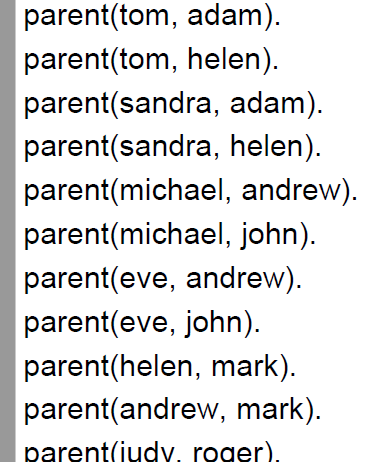
1. Values of data 数据的值
2. Operations that we can have for data 我们可以对数据进行的操作
3. How we represent data 我们如何表现数据

在PROLOG中 DATA STRUCTURE:TERM

有三个要素组成

1. Atom:任何identifier以小写字母开头
2. Number:整数
3. Variable变量：任何Identifier以大写字母开头

Compound term：有一个atom 叫做functor，以及一系列的语句每一个都是一个term



parent(tom, adam ). 记住要小写，加句号，意思是tom是adam的爹

arity:在一个parent(x,x)中参数的数量，parent/2 代表parent这个functor有两个argument factor 参数

单个的语句叫做clause,statement

一共有两种clause：

1. fact()既定事实，我们已经设置了tom是adaM的爹

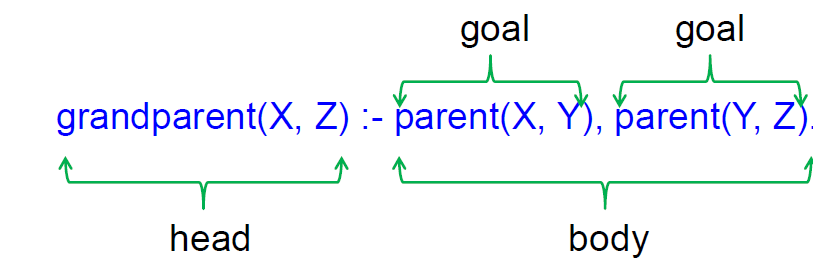
所有的fact()组成database

1. Rule:加于DATABASE上的规则，基于FACT

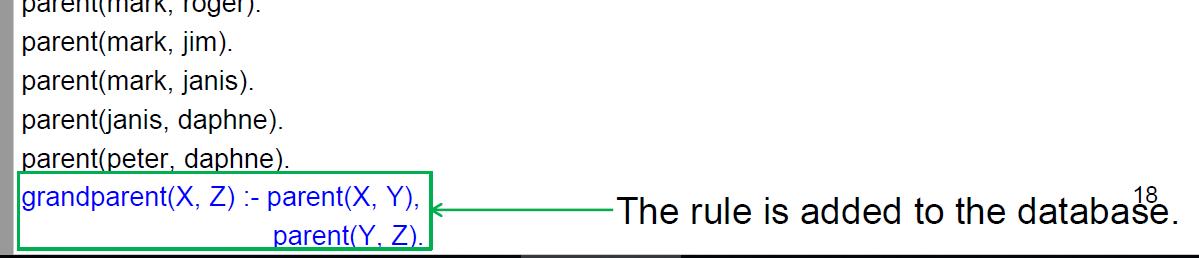
Head：-body.

Conclusion:-condition.

意思是，如果BODY部分成立，那么HEAD的逻辑关系就成立



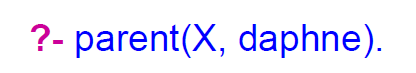
逗号表示并列关系，分好表示或。



如果X是Y的爹，Y是Z，的爹，那么X就是Z的爷

Query质疑

一共有两种query

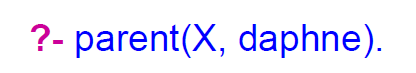
怎么query?-

Ground query



会弹出true of false（PETER是不是DAPHNE的爹）

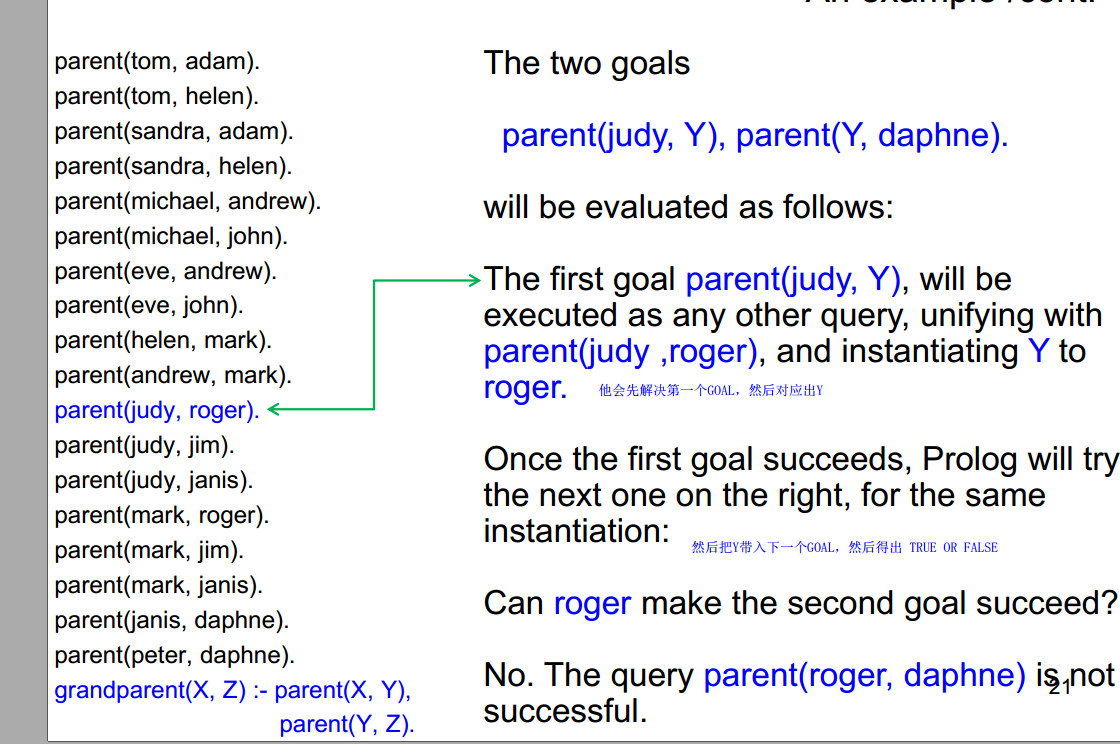
Non-Ground query

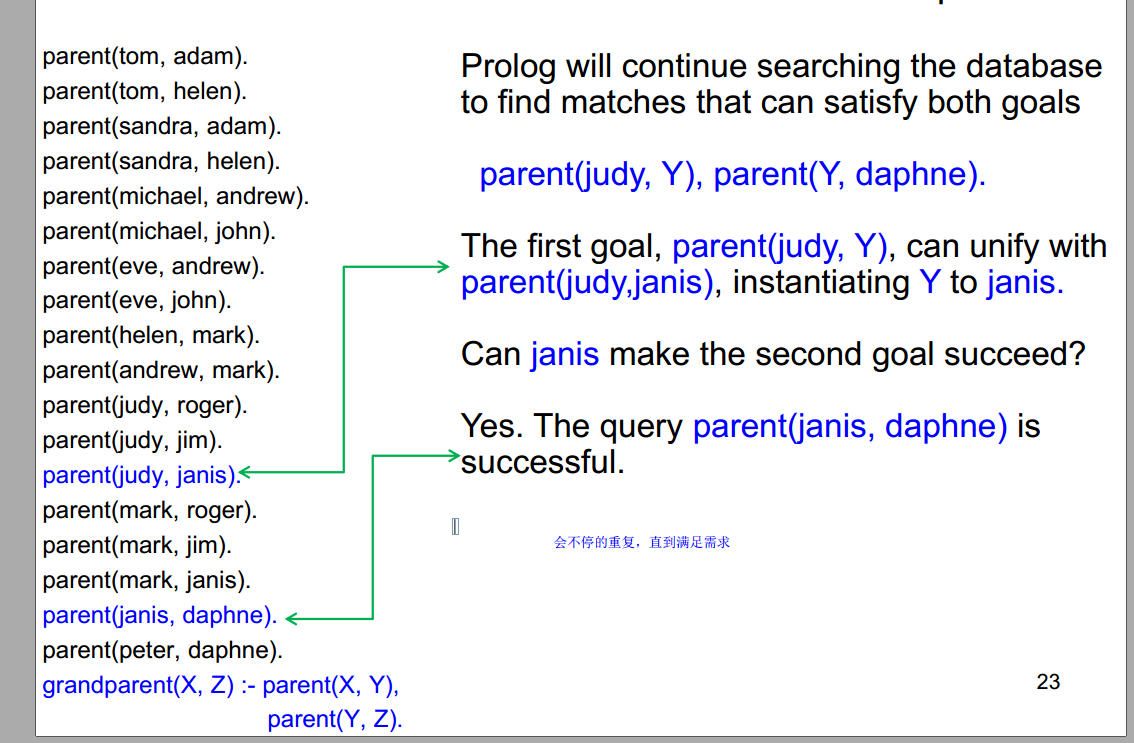


有一个X,输入这个会弹出X的值（daphne的父母）

对于RULE怎么具体验证：

1. ground query
2. 找到对应规则，通过第一个GOAL给出Y，然后把Y带入第二个goal得到YES OR TRUE。
3. 不停的重复，直到得到YES。OR遍历完以后得到NO

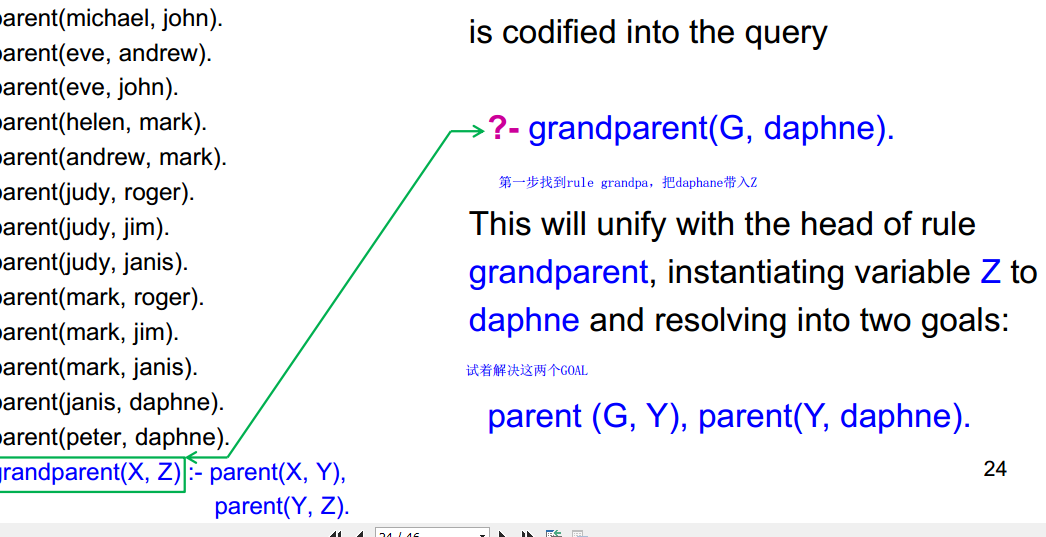


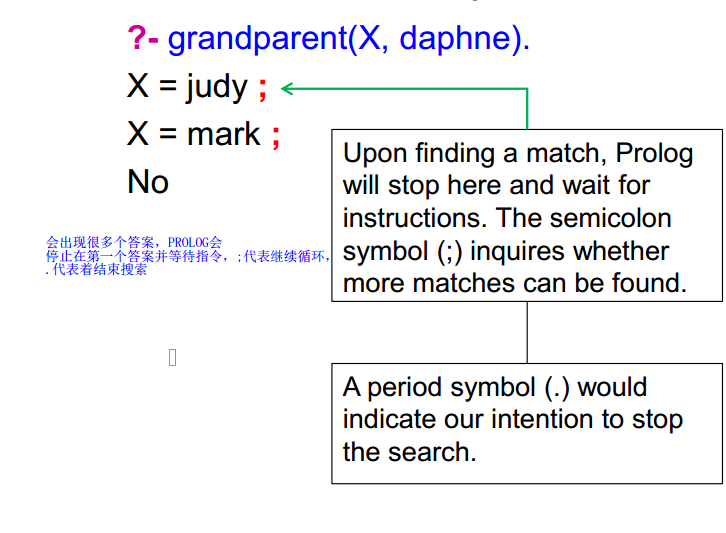


1. Non-ground

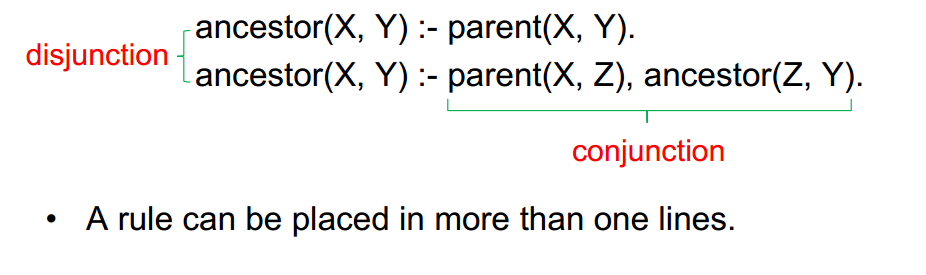
第一步找到对应法则，带入空缺的那位，然后找到两个GOAL（）两个都是NON-GROUND，

每一次循环只会出来一个答案并等待指令，按;才会继续 按.就结束



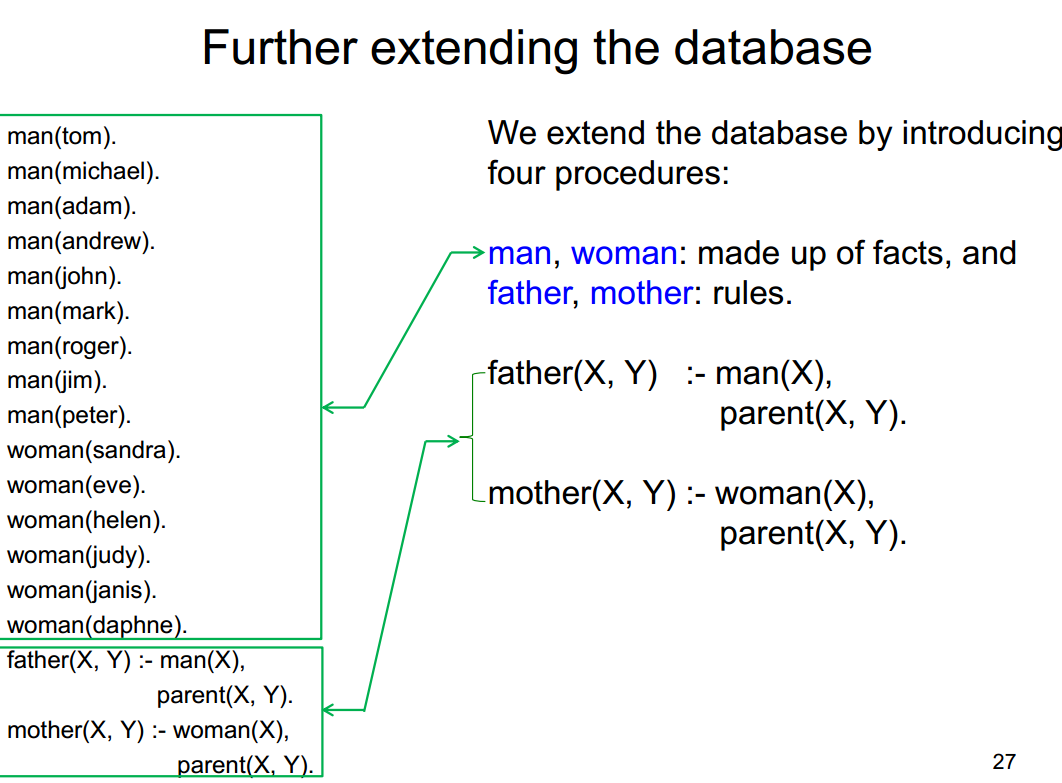


Disjunction

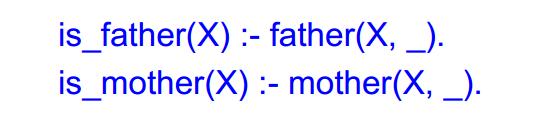


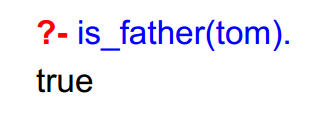
一条rule可以被多种方式描述，叫做disjuntion

一条rule有用逗号隔开的不同GOAL，两个GOAL之间有CONJUNCTION

更进一步拓展DATABASE：为个人添加属性  


如果一个关系里面有一个参数不重要，我们可以用 \_ 下划线来代替 anonymous variable匿名的变量





怎么样写ANCESTOR这种一连串的关系：  
错误ancestor(X,Y):-parent(x,y);parent(x,z),parent(z,y)

第一个是父母关系，第二个是爷孙关系

正确ancestor(x,y):-parent(x,y);parent(x,z),ancestor(z,y)

比如说X,Z1,Z2,Y

X是Y的的爹吗，不是。X是Z1的爹，并且Z1是Y的爷爷吗？针对第二句，无限循环

最后得到第一个GOAL，Z2是Y的爹

路径图· path(x,y):-edge(x,y);edge(X,Z),PATH(Z,Y)

总而言之，第一个是基本条件，第二个GOAL里面前半句是引出她的下一个，后半句是recursion

Path(x,y):-path(x,z),edge(z,y);edge(x,y)

不行，是从左往右的，第一句话会无限循环

Qualifier限定词:forall

Qualify(x):-forall man(x),parent(x,\_)

所有男的都是parent

?-forall（man（x）,parent(x,\_)）

?-forall(son(x,\_),man(x)) Yes,只要是儿子，就是男的

只要是男的，就是Parent是错的

Forall的关系：只要就

Arithmetic operation 四则运算符号

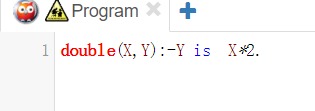
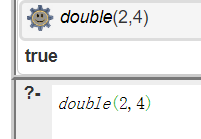
/\*-+没有=，只有is 7is 1+6

Is:右边是一个四则运算，左边是一个变量

比如X is 6+1

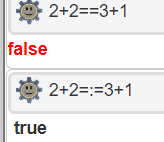
//mod，整除

X is 6//4答案是1，不是求余，而是求整数部分

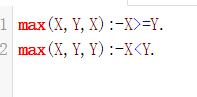
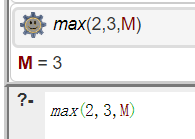
 

== 与\==等于与不等于，只是在形式上的，A==A,2+2==3+1就是false,因为两边不一样

=:=与=\=内容上的2+2=:=3+1



<,>,>=,=<

注意这里XY一定要用大写，不然不算作变量

另外一种写法

Max(X,Y,Z):-X>=Y,Z is X;Z is Y.

最好的写法max（X,Y,Z）:-X>=Y ->Z is X;Z is Y.

A->B;C IF A ,THEN B ELSE C

Factorial(N,F)阶乘

ifN<=0 F=1

else ,F=N\*factorial(n-1,f)



背后逻辑

Factiral(4,f)->factorial(3,f1),F=F1\*4

Factorial(3,f1)->factorial(2,f2),f1=f2\*3

Factorial(2,f2)->factorial(1,f3),f2=f3\*2

Factorial(1,f3)->factorial(0,f4),f3=f4\*1

Factorial(0,f4)->f4 is 1

然后一路倒推

List

List[a,b,c]，a是head，[b,c]是tail

List[]空集，没有head,or tail.

[a,[b,c],d]有三个element

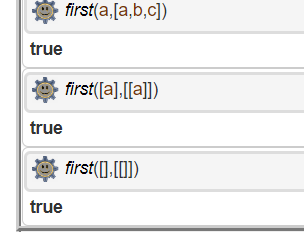
[[a,b],c,d]的head是[a,b],tail是[c,d]

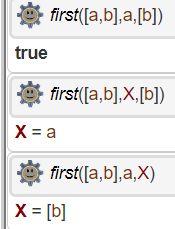
[[a]]≠[a]

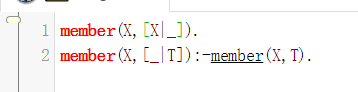
[a]的head是a,tail是[]

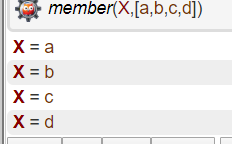
[X+Y,X+Y]的head是X+Y,tail是[X+Y]

[H|T] 指一个有着head H与tail T的人

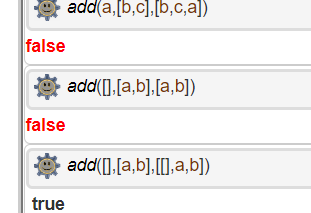






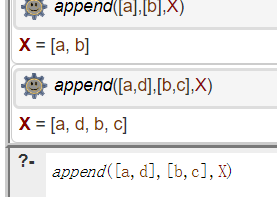


背后逻辑：?-member(d,[a,b,c,d])会对应到第二个式子，然后:-成member(d,[b,c,d])然后:-成member(d,[c,d])最后成为member(d,[d])，然后对应到式子1,true

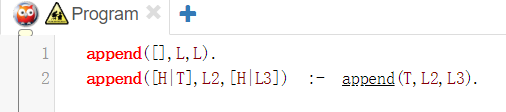


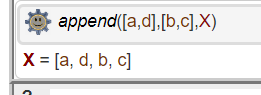
二三区别，空集也是一个元素

Append，一个内置的function,和add的区别在于add只能加一个element，而append可以把两个List合二为一，只是去除左右括号



背后逻辑：





分析：append([a,d],[b,c],X)第二条,:-append([d],[b,c],L3)，X是[a,L3]

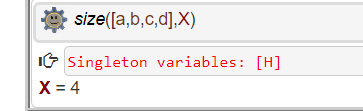
Append([d],[b,c],L3]):-append([],[b,c],L4) L3是[d,L4]

Append([],[b,c],L4)第一条，L4=[b,c]L3=[d,b,c],X=[a,d,b,c].

简单的来说就是不停地把第一个和第三个每次去掉一个H，直到去除到第一个为空，然后第二个复制到LIST3作为BASE，然后把之前去除的H轮流加上

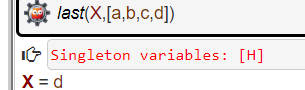
当做stack理解





背后逻辑：

LIST不停减去H作为STACK，而N同时每次减1，直到LIST剪完，N剩下的数字为0作为BASE，然后把STACK一次次加1。

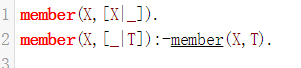


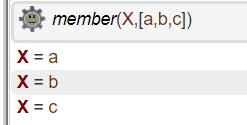
背后逻辑，第二个不停地抽去Head作为STACK，直到只剩一个元素。

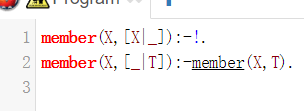
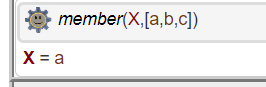
！：cut

感叹号代表直接终止

10.23

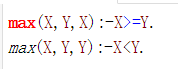
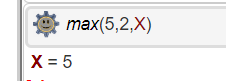
\_>member(X,[a,b,c])的答案是X=a,X=b,X=c

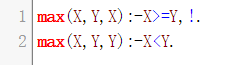
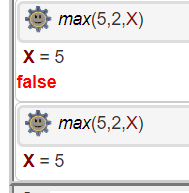
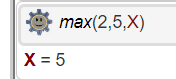


第一步加了感叹号就成了a

解释：当有两个Line时，第一部成功了，我们就CUT不循环其他LINE

与上面相比减少时间，

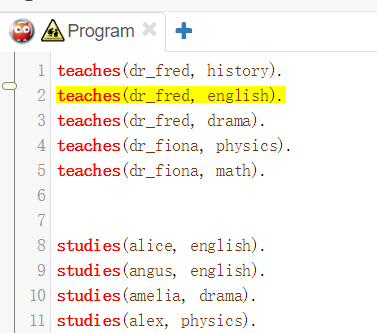
为什么：因为第一步如果成立，那么就不会进行第二步

写得更简单点

，要么1，要么直接2

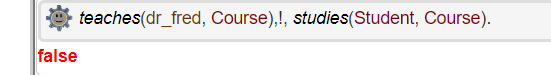
并且query 也只会进行一次

Query中加入感叹号



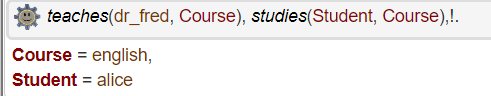


基本的，会告诉你COURSE,STUDENT

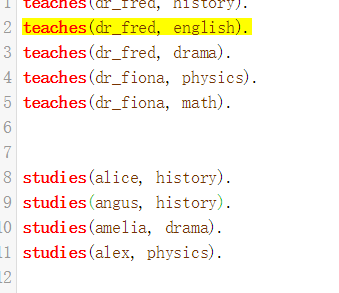


加了感叹号，直接false,为什么，因为他进行完第一个query，就不会再BACKTRACK，COURSE=HISTORY，而我们没有这样的STUDENT。

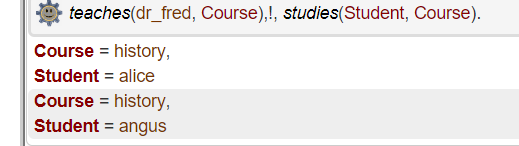
结尾家感叹号



在第二句成立前，能无限BACKTRACK，第二句第一次成立，即history过后我们找到english(有对应student)，然后进行一次就中断



Studie里面把科目改成history



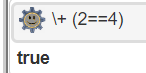
总结的来说，在rule里，第一句加了cut，第一句成功就不看第二句，第一句失败直接第二句

在query里加，在没成功前可以无限backtrack，成功一次就无法backtrack。

写法上在rule里一定要：-的后面

在query里要加逗号

\+ 意思是not

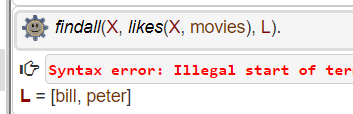
答案是true, not false

Iswoman(X)::-\+isman(X). 不是男的就是女的

内置function们

Findall

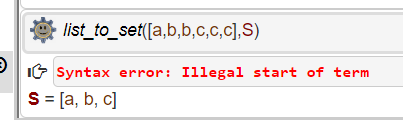




X是temp变量，第二个是GOAL，第三个是我们所求的List

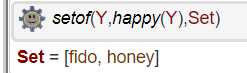
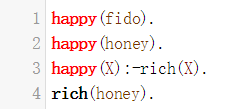
我们用X代入goal中得到所需List

Llisttoset



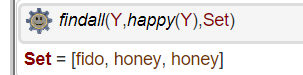
可以把list转换成SET，而SET是不能重复的，换句话说，剔除了重复元素

Setof



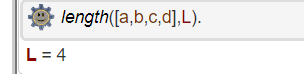
setof(X,Y,Set)

X是暂时变量，Y是含有X的goal,Set是我们要的Set

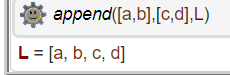


Findall就有两个，因为findall是List可重复，SET不可重复

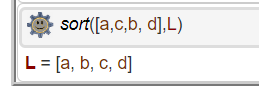
Length

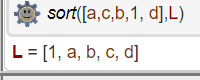


Append

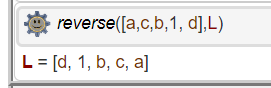


Sort

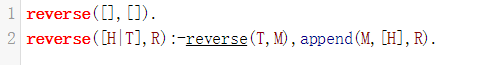




Reverse



背后逻辑



M加上H是右边的list，换句话说 M是倒过来以后的list除了H以外的所有

相当于把TAIL倒过来，

每一次都会减少一个H，直到空List.

[a,b,c]

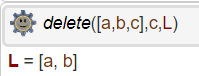
执行第二句，reverse([b,c],M),append(M,[a],R)

执行第二句（reverse[c],M1）,append(M1,[b],M)

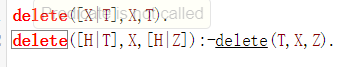
执行第二句（reverse[],M2），append(M2,[c],M1)

M2是空集，M1是[C],M是[C,,B],R是[C,B,A]

Delete



背后逻辑



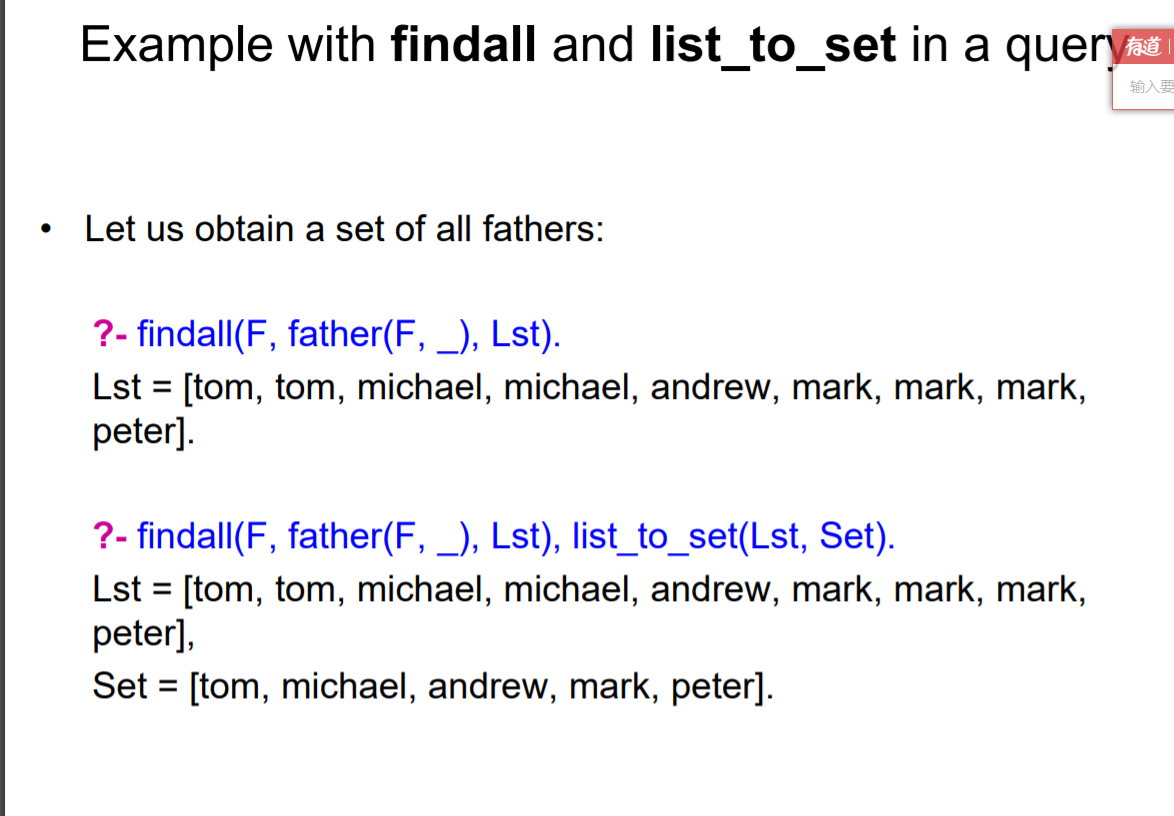
Delete([a,b,c],c,L)

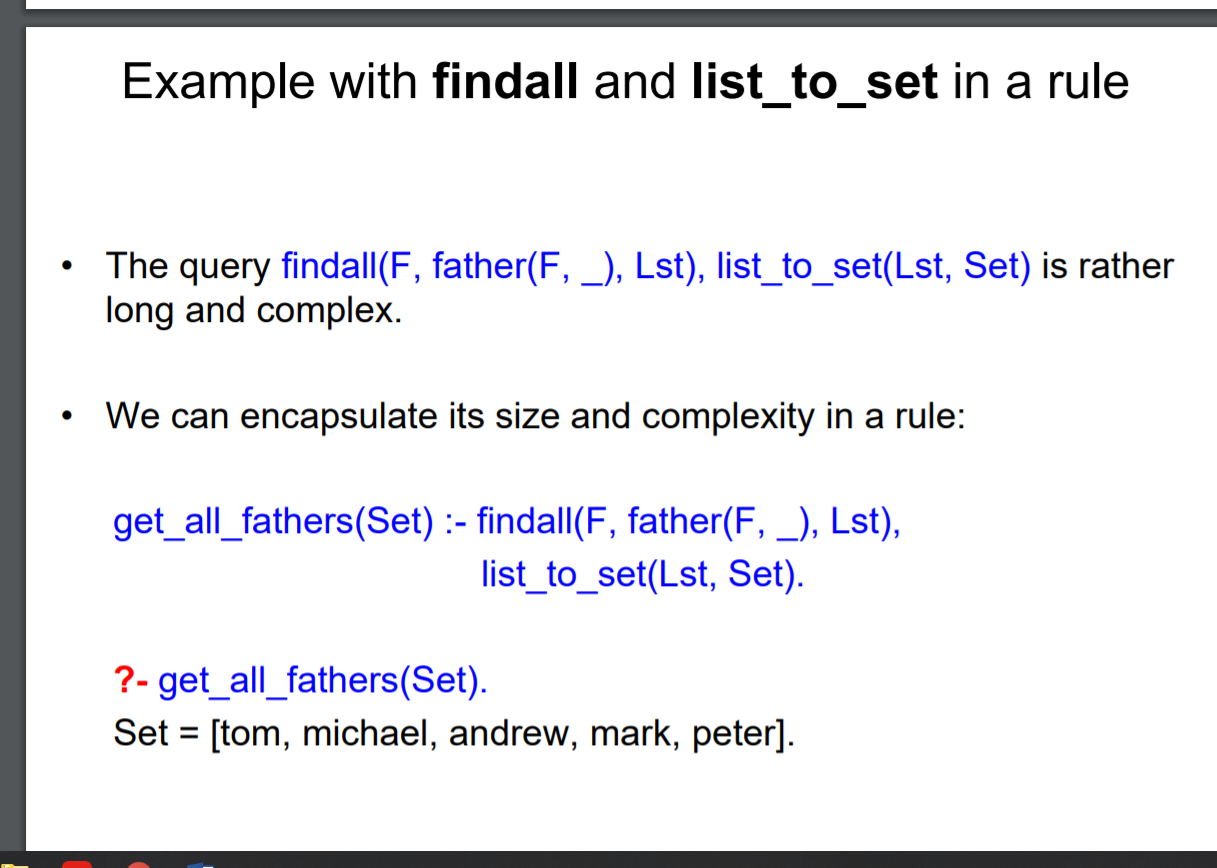
Delete([b,c],c,L1)

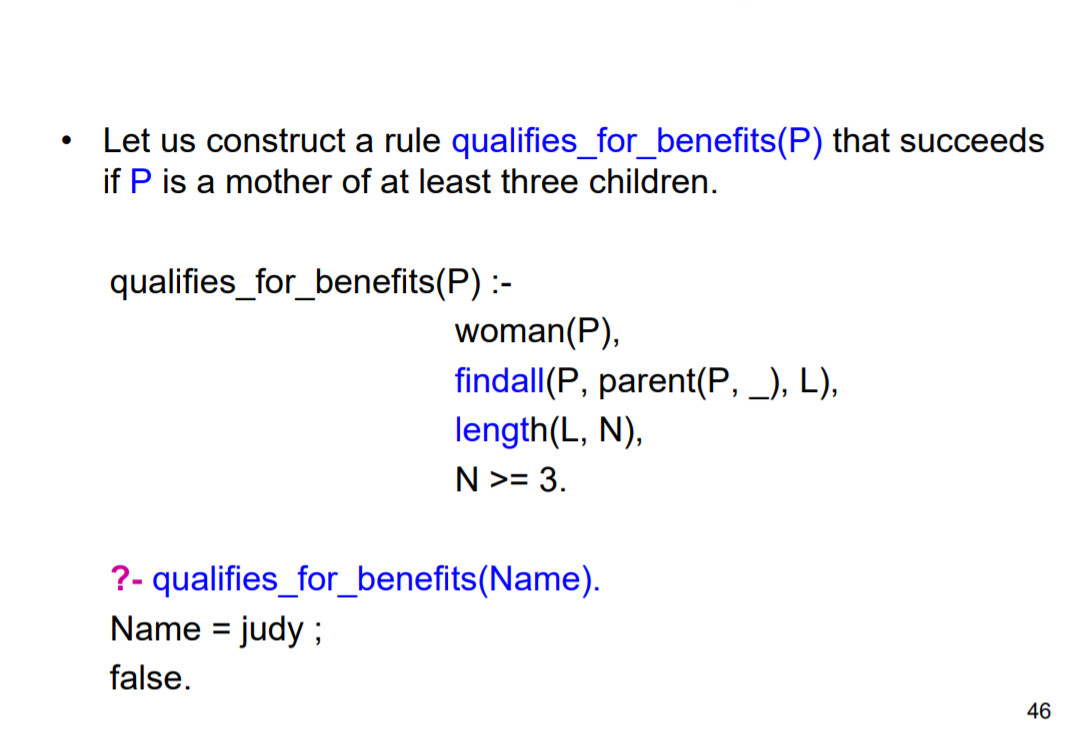
Delete([c],c,L2)

L2 IS [],L1 is [b], Lis [a,b]

Example







电路图

NOT:

Rev(0,1)

REV(1,0)

OR

or(0,1,1)

Or(1,1,1)

Or(0,0,0)

Or(1,0,1)

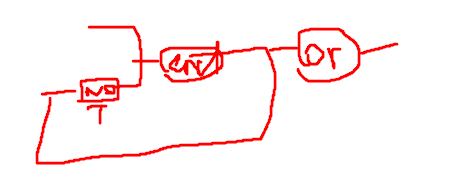
And

And(0,0,0)

and(0,1,0)

And(1,0,0)

And(1,1,1)



circuit(X,Y,Output):-rev(Y,T1),and(T1,X,T2),or(T2，Y1，Output)