Pràctiques de Llenguatges, Tecnologies i Paradigmes de programació. Curs 2016-17

Part III Programació Lògica

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Pràctica 6: Introducció a Prolog

Índex

1 Objectius de la Pràctica

Els objectius de la pràctica són dos:

1. Introduir a l'alumne en l'ús d'un llenguatge lògic per a representar una base de coneixement i realitzar consultes sobre ella. El llenguatge lògic usat és Prolog. Per açò, en la primera part de la pràctica es realitzen consultes a una base de coneixement implementada en un programa Prolog.

2. Estudiar dos conceptes bàsics del paradigma lògic que permeten resoldre les consultes: els conceptes d'unificació i *cerca amb reculada*. La segona part de la pràctica és un reforç de la matèria de teoria en la qual es practica sobre aquests dos conceptes. El segon d'aquests conceptes s'aborda fent ús de traces d'execució, i es recomana repassar-ho amb més temps després de la realització de la sessió de pràctiques.

2 SWI-Prolog

En aquesta secció s'introdueix l'ús del sistema SWI-Prolog així com conceptes bàsics de programació en el paradigma lògic. SWI-Prolog és una implementació en codi obert del llenguatge de programació Prolog. El nom SWI deriva de Sociaal-Wetenschappelijke Informatica, l'antic nom d'un grup de recerca en la Universitat d'Amsterdam on es va iniciar el seu desenvolupament.

2.1 Ús i càrrega de programes

La instal·lació de SWI-Prolog en entorn *Linux Ubuntu – Debian* és senzilla. Solament es requereix executar aquest comando:

```
sudo apt-get install swi-prolog
```

Descarrega el fitxer `family.pl` de PoliformaT, obri un terminal i estableix com a directori de treball el directori on has baixat el fitxer. Des de Linux, invoca a SWI-Prolog mitjançant el comando `swipl`.

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.0)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

El sistema executa un intèrpret i queda a l'espera de que introduïm consultes a la base de coneixement (programa Prolog) carregada en memòria. En SWI-Prolog aquestes consultes poden realitzar-se fent ús d'un shell interactiu que el seu prompt és `?-`. Des d'aquesta linea d'entrada de consultes, es poden carregar programes guardats en un fitxer. Per a carregar un programa podem usar l'ordre `consult`(nomfitxer), `compile`(nomfitxer) o [nomfitxer] seguits d'un punt. Podem eixir de l'interprete mitjançant el comando `halt.` o bé prement CTRL+D.

Exercici 1 *Carrega el programa guardat en el fitxer* `family.pl` *que està* en PoliformaT *fent ús d'alguna de les instruccions de càrrega anteriors. NO oblides que totes les consultes finalitzen amb un punt.*

El resultat ha de semblar-se al següent:

```
consult(family).
% family compiled 0.00 sec, 2,348 bytes
true.
```

Podem consultar els predicats que hem carregat mitjançant el comando `listing`. També podem utilitzar-ho per a conèixer algun predicat en particular, p. ej: `listing(fatherof)`.

Exercici 2 *Executa el predicat* `listing` *des de l'interprete* ***`swipl`*** *i llista la base de coneixement per a varis dels predicats carregats en memòria.*

Hauries d'haver obtingut alguna cosa semblat a açò:

```
?- listing.
fatherof(juan, maria).
fatherof(pablo, juan).
fatherof(pablo, marcela).
fatherof(carlos, debora).
fatherof(luisa, debora).
sonof(B, A) :-
    fatherof(A, B).
brotherof(A, C) :-
    fatherof(B, A),
    fatherof(B, C),
    A\==C.
grandfatherof(A, C) :-
    fatherof(A, B),
    fatherof(B, C).
relativeof(A, B) :-
    fatherof(A, B).
relativeof(A, B) :-
    sonof(A, B).
relativeof(A, B) :-
    brotherof(A, B).
relativeof(A,B) :-
    grandfatherof(A,B).
true.

?- listing(fatherof).
fatherof(juan, maria).
fatherof(pablo, juan).
fatherof(pablo, marcela).
fatherof(carlos, debora).
fatherof(luisa, debora).
true.
```

En primer lloc, és important destacar que l'ús de majúscules o minúscules és rellevant per al compilador. Les variables comencen per majúscules o per un símbol de subratllat. Per tant, `A` és una variable mentre que `juan` no ho és. A més, podem veure que un programa està format per fets i regles. Els fets són els predicats amb arguments entre parèntesis seguits d'un punt (per exemple, `fatherof(juan, maria).`). Les regles són els predicats amb arguments entre parèntesis seguits de `:-` i altres elements, en la seua majoría predicats (per exemple, `sonof(B, A) :- fatherof(A, B).`).

El predicat `fatherof` d'aridad 2 expressa que el seu primer argument està relacionat amb el segon mitjançant la relació *és pare de*. La definició d'aquest predicat és extensional, és a dir, en

el programa apareixen explícitament els fets de la relació. El predicat `sonof(A,B)` es defineix intensionalmente mitjançant una regla i expressa que `A` és fill de `B` si `B` és pare de `A`. És a dir, podem saber qui és fill d'algú consultant la relació *és pare de* definida pel predicat `fatherof`. Fixa't que el *si* condicional de la frase anterior s'expressa en el programa amb el parell de símbols `:-`. Per exemple, la regla `relativeof(A, B) :- brotherof(A, B).` pot llegir-se com *A és familiar de B si A és germà de* B , o també com *si A és germà de B, llavors A és familiar de* B. La definició del predicat `brotherof` conté tres predicats separats per comes en el cos de la regla (darrere de `:-`). Les comes representen una conjunció de predicats. És a dir, en ser consultats els predicats en la base de coneixement, l'intèrpret ha de retornar una resposta d'èxit per a tots ells. La definició del predicat `grandfatherof` expressa la relació entre parells formats per pares i fills. En la definició del predicat `relativeof` d'aridad 2 pots observar que, a més de ser intensional, s'usen quatre clàusules en la seua definició. Aquesta és una forma simple d'expressar la disjunció en Prolog. En aquest cas, es defineix que el primer argument `A` és familiar del segon argument `B` *si* `A` és pare, fill, germà o avi de `B` .

2.2 Consultes a fets

A continuació anem a realitzar diverses consultes sobre la base de coneixement carregada en memòria. Podem llançar consultes des de l'intèrpret. Comencem realitzant consultes senzilles sobre els fets del programa. Per exemple, preguntar a l'intèrpret si Juan és pare de María, s'escriuria així:

```
?- fatherof(juan,maria).
true.
```

Si volguérem ser més precisos, el que hem fet és preguntar a l'intèrpret si en la base de coneixement es troba algun fet que prove que Juan és pare de María.

Exercici 3 *¿Què ocorre quan preguntes a la base de coneixement si coneix el fet que maria és pare de juan? Escriu la consulta en l'intèrpret i explica la resposta que retorna.*

Les dues consultes anteriors només requerien una cerca de fets usant els termes sense variables `maria` i `juan`. La informació que s'obtenia com a resposta era una afirmació (`true`) o una negació (`false`) de l'existència d'una demostració per a la consulta. En les consultes es poden usar variables per a extraure més informació de la base de coneixement. Per exemple, suposa que volem saber si amb el coneixement emmagatzemat en el nostre programa María té un pare, i en tal cas, saber de qui es tracta. Per a açò podem llançar a l'intèrpret la mateixa consulta que abans, però amb el primer argument com una variable. Com a resposta s'obté el valor de la variable amb el qual la nostra consulta és certa.

```
?- fatherof(X,maria).
X = juan.
```

És a dir, que l'intèrpret pot demostrar que `fatherof(X,maria)` és cert a partir dels fets del programa si `X=juan`.

En el cas que existisca més d'una resposta afirmativa a una consulta, el sistema retorna la primera que computa. Si desitgem rebre més respostes hem de prémer *r* (*redo*) per a conèixer altres respostes. Comprova-ho realitzant el següent exercici.

Exercici 4 *¿Què ocorre si preguntes de qui és pare Pablo? Per a açò reescriu la consulta de l'exemple anterior canviant la variable* `X` *pel terme* `pablo` *i el terme* `maria` *per una variable (per exemple* `X`*). ¿Què ocorre quan prems la tecla* `r` *després de la primera resposta?*

Açò mateix pot usar-se per a extraure informació sobre tots els parells (pare,fill) definits en el programa com s'il·lustra en la següent execució en la qual els punts i comes apareixen en el terminal després de prémer la tecla `;` o la tecla `r`.

```
?- fatherof(X,Y).
X = juan,
Y = maria ;
X = pablo,
Y = juan ;
X = pablo,
Y = marcela ;
X = carlos,
Y = debora.
X = luisa,
Y = debora.
```

Exercici 5 *Realitza la consulta* `fatherof(X,X).` *¿Què contesta l'intèrpret? Explica aquesta contestació.*

En aquesta secció hem realitzat consultes que només requerien *cercar* si un parell de termes (`juan`, `maria`, `pablo`...) estan relacionats entre si per un fet (`fatherof`). En el cas d'usar variables, aquestes s'instanciaben amb els termes que feien el fet cert.

2.3 Consultes a regles

Ara anem a realitzar consultes que requereixen deduir informació que no està expressada amb fets en el programa, però que pot obtenir-se a partir dels fets i les regles. Les regles són del tipus "Si és veritat l'antecedent (cos) llavors és veritat el conseqüent (cap)". Per exemple: fent ús de la regla `sonof(A,B) :- fatherof(B,A).` que expressa que si `B` és pare de `A` llavors `A` és fill de `B`, l'intèrpret pot arribar a la conclusió que María és filla de Juan després de la següent consulta:

```
?- sonof(maria,juan).
true.
```

també es pot preguntar qui és el pare de María:

```
?- sonof(maria,X).
X = juan.
```

o qui són els pares de Débora:

```
?- sonof(debora,X).
X = carlos ;
X = luisa.
```

Cap dels tres fets `sonof(maria,juan)`, `sonof(debora,carlos)` i `sonof(debora,luisa)` està explícitament en el programa. No obstant açò, l'intèrpret és capaç d'arribar a la conclusió que són deduïbles dels fets i regles del programa.

Exercici 6 *Fent ús de* `family.pl`*:*

- *Consultar qui és avi de María.*
- *Consultar tots els familiars de Pablo coneguts.*
- *Consultar tots les persones que són família de Débora.*
- *Usant un editor de text, afegir al fitxer* `family.pl` *una regla que definisca la relació* és nét de *similar a* és fill de *però usant el predicat* `grandfatherof` *en lloc de* `fatherof`*. Guarda els canvis i recarrega el fitxer amb el predicat* `reconsult(family).` *Esbrina de qui és néta María usant el predicat* `grandsonof`*.*
- *Llança la consulta* `relativeof(A,B).` *Afig al programa la regla* `relativeof(A,B) :- grandsonof(A,B)` *i reconsúltalo. ¿En què difereixen les dues respostes computades?*

3 Dos conceptes bàsics en Prolog

Cada paradigma fa ús de diferents conceptes en la seua semàntica operacional. Així, la semàntica operacional dels llenguatges imperatius es basa en el canvi d'estat. En els funcionals, l'encaixament de patrons i la reducció de termes són dos conceptes bàsics. En els llenguatges lògics, la unificació i cerca amb reculada o volta arrere[1] són dos conceptes fonamentals. En aquesta secció anem a practicar aquests dos conceptes.

3.1 Unificació de termes

Com hem vist en la secció anterior, els programes Prolog es componen de definicions de predicats de dos tipus: fets i regles. Igual que ocorre amb els llenguatges funcionals, en els lògics també existeixen termes. En els llenguatges lògics, els termes no són reductible usant un conjunt d'equacions. No obstant açò, en els termes dels llenguatges lògics poden aparèixer variables en les consultes[2]. Açò ofereix la possibilitat de recuperar informació a través dels paràmetres dels predicats. La unificació de termes permet que la informació dels paràmetres dels predicats siga reversible. És a dir, que en una consulta un paràmetre pot ser d'entrada i en una altra d'eixida. També podríem considerar que en una mateixa consulta un paràmetre pot ser tant d'entrada (part sense variables dels termes) com d'eixida (variables dels termes).

En classes de teoria s'estudia un algorisme que determina si dos termes amb variables unifiquen, i en el cas de fer-ho, retorna un unificador (el més general). Aquest algorisme pot veure's com un mecanisme per a realitzar l'encaix de patrons vist per als llenguatges funcionals però que permet donar valors a les variables dels termes de la consulta. Dos termes amb variables unifiquen si les estructures dels termes són idèntiques donant valors a les seues variables. És a dir, després de donar valor a les variables, tenen els mateixos noms en els mateixos llocs. Açò és cert sempre, excepte quan es tracta de dues variables que estan en el mateix lloc dins de les estructures dels dos termes. En aquest cas les variables poden conservar el seu nom i considerar-se idèntiques (són equivalents). Vegem alguns exemples amb parells de termes:

- `date(10,nov,2016)` i `date(10,nov,2016)` unifiquen ja que són idèntics.
- `date(10,nov,2016)` i `date(X,nov,2016)` unifiquen si `X` = 10.

[1] També conegut com *backtracking*

[2] En l'apèndix A pots trobar una classificació d'aquests termes

- `date(10,nov,2016)` i `date(X,nov,X)` no unifiquen perquè `X` no pot valdre 10 i 2016 alhora.
- `date(10,nov,2016)` i `time(13,05)` no unifiquen perquè `date` i `time` són dos functors diferents.
- `X` i `time(13,05)` unifiquen si `X` = `time(13,05)`.
- `X` i `date(10,nov,Y)` unifiquen si `X` = `date(10,nov,Y)`.
- `date(10,oct,2016)` i `date(X,nov,2016)` no unifiquen perquè els mesos són diferents.
- els termes `moment(date(10,nov,2016),Y)` i `moment(X,time(13,05))` unifiquen si `X` = `date(10,nov,2016)` i `Y` = `time(13,05)`.
- els termes `moment(time(Time,Minutes))` i `moment(time(13,05))` unifiquen si `Time` = `13` i `Minutes` = `05`.

Perquè dos predicats unifiquen han de tenir el mateix nom, la mateixa aridad i han d'unificar cadascun dels termes dels seus paràmetres. Per exemple:

- els fets `fatherof(X,debora)` i `fatherof(carlos,debora)` unifiquen si `X` = `carlos`.
- els fets `fatherof(X,debora)` i `fatherof(luisa,debora)` unifiquen si `X` = `luisa`.
- els fets `fatherof(X,debora)` i `sonof(X,debora)` no unifiquen perquè els noms dels predicats són diferents.
- els fets `fatherof(X,debora)` i `fatherof(luisa,Y)` unifiquen si `X` = `luisa` i `Y` = `debora`.

Exercici 7 *Comprova alguna de les afirmacions que es fan en els exemples anteriors executant l'algorisme d'unificació usat per l'intèrpret SWI-Prolog. Per a açò tria parells de termes i col·loca'ls a l'esquerra i dreta del símbol igual a manera de consulta. Per exemple:*

```
date(10,nov,2016) = date(X,nov,2016).
grandsonof(time(Time,moment)) = moment(time(13,05)).
```

Exercici 8 *Realitza el mateix que en l'exercici anterior però amb predicats. Per exemple:* `fatherof(X,debora) = fatherof(luisa,Y).`

A continuació anem a practicar la unificació amb predicats que contenen termes compostos en els seus arguments. Per a açò, usarem el fitxer `flights.pl` que està *en PoliformaT* i que conté informació sobre vols. Per a cada vol directe existeix un fet representat pel predicat `flight` amb vuit paràmetres: origen, destinació, data d'eixida, hora d'eixida, data d'arribada, hora d'arribada, durada i preu. En els paràmetres de dates, aquestes es representen amb termes amb el functor `date` i tres termes per al dia, mes i any (per exemple: `date(10,nov,2012)`). Els paràmetres d'hora es representen amb el functor `time` amb dos paràmetres hora i minuts. Un exemple d'un fet complet sobre un vol és el següent:

```
flight(barcelona,madrid,date(10,nov,2016),time(13,05),
                        date(10,nov,2016),time(15,05),120,80).
```

en el qual es representa que existeix un vol directe des de Barcelona a Madrid amb eixida el 10 de Novembre de 2016 a les 13:05 i arribada el mateix dia a les 15:05 amb una durada de 120 minuts a un preu de 80 euros. Els cinc fets del programa són els següents:

```
flight(barcelona,madrid,date(10,nov,2016),time(13,05),
                        date(10,nov,2016),time(15,05),120,80).
flight(barcelona,valencia,date(10,nov,2016),time(13,05),
                        date(10,nov,2016),time(15,05),120,20).
flight(madrid,london,date(10,nov,2016),time(16,05),
                       date(10,nov,2016),time(17,35),90,140).
flight(valencia,london,date(10,nov,2016),time(16,05),
                        date(10,nov,2016),time(17,35),90,50).
flight(madrid,london,date(10,nov,2016),time(23,05),
                       date(11,nov,2016),time(00,25),80,50).
```

Exercici 9 *Carrega el fitxer* `flights.pl` *i cerca tots els vols des de València a London. Usa els noms de variables adequats perquè l'eixida siga:*

```
DepartureDay = date(10, nov, 2016),
DepartureTime = time(16, 5),
ArrivalDay = date(10, nov, 2016),
ArrivalTime = time(17, 35),
Duration = 90,
Price = 50.
```

Exercici 10 *Realitza les següents consultes.*

- *Consulta tots els vols que ixen des de Madrid el dia 10 de Novembre de 2016. Per a açò has de realitzar una consulta en la qual el tercer paràmetre del predicat* `flight` *use el functor* `date` *amb els paràmetres de la data sol·licitada.*
- *Consulta els vols l'hora d'eixida dels quals siga 13:05. Has de realitzar-ho de la mateixa forma que l'exercici anterior però amb el functor* `time`*.*
- *Consulta els vols que ixen a partir de les 16:00. Per a açò has de realitzar una consulta en la qual el terme del quart paràmetre del predicat* `flight` *conste del functor* `time` *amb dues variables (*`H` *i* `M`*) com a paràmetres. Després i separat per una coma, has d'exigir que l'hora siga major o igual que 16 (* `H>=16`*). Recorda que la coma entre predicats representa la conjunció lògica.*

El programa també conté una regla que defineix el predicat `connection_same_day` d'aridad 3 per a vols amb una única escala en el mateix dia. Aquest predicat té tres paràmetres per a l'origen, destinació i la data. Fixa't com en el cos de la clàusula apareix una nova variable `Connection` que representa la ciutat de l'enllaç aeri. Els paràmetres amb símbol subratllat no són rellevants per a la resolució del problema. El cos d'aquest predicat usa el predicat `is` per a avaluar expressions aritmètiques assignant a la variable de l'esquerra el resultat d'avaluar l'expressió de la dreta. En aquesta expressió es sumen els minuts transcorreguts des de l'inici del dia, deixant 60 minuts per a poder realitzar l'enllaç. L'última condició en el cos de la clàusula és que el viatger tinga suficient temps per a agafar el vol.

```
connection_same_day(Origin,Destination,Date):-
                        flight(Origin,Connection,Date,_,Date,time(Hl1,Ms1),_,_),
                        flight(Connection,Destination,Date,time(Hs2,Ms2),Date,_,_,_),
                        Hl1_in_minutes is Hl1 * 60 + Ms1 + 60,
                        Hs2_in_minutes is Hs2 * 60 + Ms2,
                        Hl1_in_minutes <= Hs2_in_minutes.
```

Exercici 11 *Consulta tots els vols amb una única escala que poden realitzar-se el dia 10 de Novembre de 2016.*

Exercici 12 *¿Què ocorreria si en la quarta i última aparició de la variable* `Date` *del cos de la clàusula* `connection_same_day` *es canviara per una altra variable* `Another_date`*?*

3.2 Cerca amb reculada

Hem vist que l'intèrpret proporciona totes les solucions que són deduïbles dels fets i regles que defineixen predicats en un programa Prolog. Aquestes solucions van apareixent (es van deduint) una després d'una altra a mesura que l'usuari les sol·licita. Per a açò s'exploren totes les possibilitats fent ús tant de fets com de regles. Aquesta part de la pràctica es dedica a observar mitjançant traces l'estratègia que utilitza l'intèrpret per a localitzar totes les solucions.

Vegem primer una breu introducció a aquesta estratègia. Una *clàusula* Prolog és tant un fet com una regla. D'aquesta forma diem que un programa Prolog està format per clàusules. Unificar amb una clàusula consisteix a unificar amb un fet o amb el cap d'una regla. En aquesta unificació es poden instanciar tant les variables de la consulta com les de la clàusula, donant lloc a un pas de paràmetres tant d'eixida com d'entrada respectivament. Donada una consulta, l'intèrpret ha de localitzar una clàusula que unifique amb ella. Cada vegada que s'unifica amb una clàusula, l'intèrpret proporciona una nova còpia d'aquesta (variant). Totes les variables d'una variant són noves (estan renomenades) i mantenen la mateixa relació que en la clàusula original. Depenent del tipus de clàusula, l'intèrpret es comporta de forma diferent. Si una consulta unifica amb un fet, s'obté una solució basada en el resultat de la unificació. Si unifica amb el cap d'una regla, el resultat de la unificació s'aplica al cos de la regla. D'aquesta forma, el resultat d'unificar amb el cap (variables instanciadas amb termes) es propaga als predicats del cos. A continuació, es llancen cadascun dels predicats del cos com si foren consultes. Si tots els predicats del cos tenen èxit, els seus resultats es retornen a través de l'unificador del cap de la clàusula amb la consulta.

El fet que siga necessària una exploració es deu al fet que els predicats de les consultes poden unificar amb diverses clàusules del programa. Cada vegada que té èxit una d'aquestes unificacions, s'està fent una elecció entre diferents possibilitats que poden portar a una deducció correcta. La reculada o volta arrere es produeix si es realitza una elecció que porta a l'intèrpret a decidir que no pot deduir la veracitat d'una determinada consulta. Aquest intenta altres alternatives desfent l'última elecció i intentant unificar amb una altra clàusula. En Prolog, el procés de cerca es concreta de la següent manera: es van triant clàusules d'a dalt a a baix en l'ordre en el qual apareixen en el programa. Quan s'unifica amb el cap d'una regla, els predicats del cos d'aquesta es van resolent (consultant) d'esquerra a dreta. Cada predicat resolt en aquest ordre té aplicat l'unificador resultant tant de la unificació amb el cap com les realitzades per a resoldre els predicats a la seua esquerra. Cada vegada que l'intèrpret falla a resoldre un d'aquests predicats del cos (després de cercar totes les possibilitats d'unificació d'a dalt a a baix), fa un pas arrere tornant al predicat més pròxim a l'esquerra del que ha fallat. Desfà l'última unificació que va fer per a aquest predicat i intenta unificar amb una altra clàusula del programa que estiga més a baix.

L'intèrpret SWI-Prolog proporciona predicats que permeten seguir aquestes cerques i reculades: `debug`, `nodebug`, `trace` i `notrace`. Aquests predicats col·loquen al *shell* en manera de depuració i permeten observar com l'intèrpret realitza l'exploració en la base de coneixement. El

predicat `trace` és interactiu i permet interaccionar en el procés (pots veure les opcions prement *h* durant la traça). Aquest és el predicat que anem a usar per a observar i seguir l'execució de l'intèrpret. Per a entrar en manera traça, n'hi ha prou amb realitzar la consulta `trace.` Vegem aquest procés amb l'exemple que apareix a continuació en què es llança la consulta `relativeof(pablo,X).` sobre el programa `family.pl` sense el predicat `grandfatherof` ni la clàusula `relativeof` que ho usa. Cada vegada que es va a realitzar una consulta, aquesta va precedida de la paraula `Call [1]`. Fixa't que apareix la variable `_G555` procedent de la variant de la clàusula. Si l'intèrpret aconsegueix demostrar que la consulta és deduïble a partir de les clàusules del programa, la mostra precedida de la paraula `Exit` i amb els seus paràmetres instanciats als termes que la fan certa. En `[2]` la variable `_G555` apareix instanciada a `juan`. Si l'usuari demana que es continue amb la cerca (prement `r`), l'intèrpret fa un pas arrere, desfà l'última unificació i reprèn la cerca d'una nova clàusula a partir de l'última amb la qual va unificar. La continuació de la cerca apareix precedida de la paraula `Redo`. En `[3]` torna a aparèixer la variable `_G555` i se cerca la següent clàusula que unifique amb la consulta però a partir de la següent de l'última que va unificar. En `[4]` s'unifica amb la següent clàusula. Si s'insisteix a demanar més solucions a la consulta inicial, l'intèrpret no troba un altre fet `fatherof` amb el qual unificar i cerca una altra regla `relativeof [5]`. Totes les altres clàusules que defineixen el predicat `relativeof` fallen ja que les consultes dels seus cossos no unifiquen amb cap clàusula. Les fallades en la cerca apareixen precedits de la paraula `Fail`. Per a eixir de la manera traça usa la consulta `notrace.` i `nodebug.` per a eixir de manera depuració.

```
[trace]  ?- relativeof(pablo,X).
   Call: (6) relativeof(pablo, _G555) ? creep            <--- [1]
   Call: (7) fatherof(pablo, _G555) ? creep              <--- [1]
   Exit: (7) fatherof(pablo, juan) ? creep               <--- [2]
   Exit: (6) relativeof(pablo, juan) ? creep             <--- [2]
X = juan ;
   Redo: (7) fatherof(pablo, _G555) ? creep              <--- [3]
   Exit: (7) fatherof(pablo, marcela) ? creep            <--- [4]
   Exit: (6) relativeof(pablo, marcela) ? creep
X = marcela ;
   Redo: (6) relativeof(pablo, _G555) ? creep            <--- [5]
   Call: (7) sonof(pablo, _G555) ? creep
   Call: (8) fatherof(_G555, pablo) ? creep
   Fail: (8) fatherof(_G555, pablo) ? creep
   Fail: (7) sonof(pablo, _G555) ? creep
   Redo: (6) relativeof(pablo, _G555) ? creep
   Call: (7) brotherof(pablo, _G555) ? creep
   Call: (8) fatherof(_G606, pablo) ? creep
   Fail: (8) fatherof(_G606, pablo) ? creep
   Fail: (7) brotherof(pablo, _G555) ? creep
   Fail: (6) relativeof(pablo, _G555) ? creep
false.
```

Exercici 13 *Comprova que estiga carregada la base de coneixement* `family.pl` *(pots usar* `listing.`*) i en el cas de no estar-ho, usa* `consult(family).` *Executa la traça de la consulta* `relativeof(pablo,X).` *considerant les clàusules que no estaven en l'exemple anterior (*`grandfatherof` *i* `grandsonof`*).*

Exercici 14 *Cerciora't que estiga carregada la base de coneixement* `flights.pl`*. Executa la traça de la consulta* `connection_same_day(Origin, Destination, Connection).`

4 Avaluació

L'assistència a les sessions de pràctiques és obligatòria per a aprovar l'assignatura. L'avaluació d'aquesta tercera part de pràctiques es realitzarà mitjançant un examen individual en el laboratori.

Apèndixs

A Classificació dels termes de Prolog

- Simples
  - Variables: es representen mitjançant cadenes formades per lletres, dígits i el símbol de subratllat, però han de començar necessàriament per una lletra majúscula o per un símbol de subratllat (per exemple: `X`, `Result_1`, `_total3`). Les que comencen amb un símbol de subratllat són variables anònimes i s'usen quan es necessita treballar amb variables els possibles valors de les quals no interessen.
  - Constants
    - Constants simbòliques: cadenes formades per lletres, dígits i el símbol de subratllat, que han de començar necessàriament per una lletra minúscula. Per exemple: `f`, `barcelona`, `book33a`, `white_book`. **Nota**: A aquestes constants també li les coneix com a àtoms, però s'usa aquesta terminologia per a no confondre'ls amb les fórmules atòmiques que són un predicat (per exemple: `fatherof(...)`, `flight(...)`.
    - Nombres: s'utilitzen per a representar tant nombres enters com a reals.
      - Enters: es representen amb la notació habitual (0, 1, -320, 1520, etc).
      - Reals: es poden representar tant en notació decimal com en notació científica. En tots dos casos haurà d'haver-hi sempre almenys un dígit a cada costat del punt.
- Compostos: es construeixen mitjançant un símbol de funció denominat functor (denotat per una constant simbòlica) i seguit, entre parèntesi, per una sèrie de termes separats per comes (arguments). Per exemple: `date(10,nov,2016)`, `time(13,05)`). **Nota**: en escriure un terme compost no pot haver-hi cap espai entre el functor i el parèntesi obert previ als arguments.