Prácticas de laboratorio de LTP
(Parte III : Programación Lógica)

Práctica 7: Introducción a Prolog

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Jose Luis Pérez
jlperez@dsic.upv.es

Introducción

Los objetivos de la práctica son dos:

1) Introducir el uso de un lenguaje lógico, concretamente **Prolog**, para representar una **base de conocimiento** y realizar consultas sobre ella.
2) Profundizar en dos conceptos básicos del paradigma lógico que permiten resolver las consultas:
   a) los conceptos de **unificación y búsqueda con retroceso** (o vuelta atrás),
   b) e introducir un mecanismo para realizar **cálculos aritméticos** simples.

Nota: En Poliformat se dispone los ficheros **cars.pl** y **flights.pl** que contienen las bases de conocimiento que utilizaremos en los ejercicios a realizar durante la sesión.

2. SWI-Prolog

SWI-Prolog es una implementación en código abierto del lenguaje de programación Prolog. El nombre SWI deriva de Sociaal-Wetenschappelijke Informatica, el antiguo nombre de un grupo de investigación en la Universidad de Amsterdam donde se inició su desarrollo.

SWI-Prolog está instalado en los laboratorios de prácticas y es accesible desde cualquier directorio escribiendo el comando **swipl** en la terminal Linux. Abre un terminal y crea un directorio de trabajo donde descargarás y editarás los programas (p. ej., **DiscoW/LTP/pr7**). Desde Linux, accede a dicho directorio e invoca a SWI-Prolog mediante el comando **swipl**:

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.1) Copyright (c) 1990-2013
University of Amsterdam, VU 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).
?-
```

2.1 Carga y ejecución de programas

El sistema ejecuta un intérprete que queda a la espera de que introduzcamos llamadas de ejecución al programa Prolog cargado en memoria (en la terminología Prolog, **consultas a la base de conocimiento**).

En SWI-Prolog estas consultas pueden realizarse haciendo uso de un shell interactivo cuyo prompt es **?-**.

```
?-
```

La **carga de programas** guardados en un fichero puede hacerse desde esta misma línea de entrada de consultas usando cualquiera de las órdenes

```
?- consult(nombrefichero).
....
?- compile(nombrefichero).
...
?- [nombrefichero].
```

Todas las entradas finalizan con un punto!!! y **nombrefichero** es el nombre del fichero sin incluir la extensión. Podemos salir del intérprete mediante el comando **halt,** o bien pulsando **CTRL+D**.

2.1 Carga y ejecución de programas

Ejercicio 1: Baja el fichero **cars.pl** que está en PoliformaT y cárgalo en Prolog haciendo uso de alguna de las instrucciones de carga anteriores.
¡No olvides que todas las consultas finalizan con un punto!
El resultado debe parecerse al siguiente:

```
?- consult(cars).
% cars compiled 0.00 sec, 49 clauses
True.
```

El intérprete devuelve **True** porque la operación se realizó correctamente. Podemos comprobar el código que hemos cargado mediante el comando **listing**. También podemos utilizarlo para conocer la definición de algún predicado en particular, p. ej: listing(model).

```
?- listing(model).
…
```

Ejercicio 2: Ejecuta el predicado **listing** desde el intérprete **swipl** y lista la base de conocimiento para varios de los predicados cargados en memoria.

2.1 Carga y ejecución de programas

```
brand(B, A) :- model(A, B).

isModelFrom(A, C) :-
    model(A, B), country(B, C).

since(ibiza, 1984).
since(cordoba, 1993).
since(altea, 2004).
since(golf, 1974).
since(touran, 2003).
since(clio, 1990).
since(twingo, 1993).
since(megane, 1995).
since(scenic, 1995).
since('2008', 2013).
since('3008', 2008).
since(corsa, 1982).

segment(ibiza, b).
segment(cordoba, b).
segment(altea, c).
segment(golf, c).
segment(touran, c).
segment(clio, b).
segment(twingo, a).
```

cars.pl

```
segment(megane, c).
segment(scenic, c).
segment('2008', b).
segment('3008', c).
segment(corsa, b).

isRelated(A, B) :- isSameBrand(A, B).
isRelated(A, B) :- isSameYear(A, B).
isRelated(A, B) :-
 segment(A, C), segment(B, C), A\==B.

model(ibiza, seat).
model(cordoba, seat).
model(altea, seat).
model(golf, volkswagen).
model(touran, volkswagen).
model(clio, renault).
model(twingo, renault).
model(megane, renault).
model(scenic, renault).
model('2008', peugeot).
model('3008', peugeot).
model(corsa, opel).
```

```
country(seat, españa).
country(renault, francia).
country(peugeot, francia).
country(volkswagen, alemania).
country(opel, alemania).

isSameBrand(A, B) :-
        model(A, C), model(B, C), A\==B.
isSameYear(A, B) :-
        since(A, C), since(B, C), A\==B.
```

2.1 Carga y ejecución de programas

Podemos ver que un programa está formado por un conjunto de: **hechos** y **reglas**.

Se dice que las reglas tienen **cabeza** y **cuerpo** (separados por **:-**) mientras que los hechos pueden verse como cabezas donde el cuerpo es **True:**

Ej. **Reglas** (predicados **Brand** e **isModelFrom**)

cabeza cuerpo

```
brand(B, A) :- model(A, B).
isModelFrom(A, C) :-
    model(A, B), country(B, C).
...
```

Ej. **Hechos** (predicados **model**)

```
model(ibiza, seat).
model(cordoba, seat).
model(altea, seat).
model(golf, volkswagen).
...
```

En primer lugar, es importante destacar que el uso de mayúsculas o minúsculas es relevante para el compilador. Las variables empiezan por mayúsculas o por un símbolo de subrayado. Por tanto **A** es una variable mientras que **ibiza** es una constante.

2.1 Carga y ejecución de programas

```
model(ibiza, seat).
```

Este predicado (**model**) expresa el hecho de que **ibiza** es un modelo de **seat**.

La definición de este predicado es *extensional*, es decir, en el programa aparecen explícitamente los casos que definen la relación.

```
brand(B, A) :- model(A, B).
```

Este predicado (**brand**) se expresa de forma *intensional* mediante una regla.

Expresa que **B** es marca de **A** si **A** es modelo de **B**. Es decir, podemos resolver el problema de saber cuál es la marca de un modelo consultando el hecho **model**.

```
isRelated(A, B) :- isSameBrand(A, B).
isRelated(A, B) :- isSameYear(A, B).
isRelated(A, B) :-
 segment(A, C), segment(B, C), A\==B
```

Este predicado (**isRelated**) se expresa de forma *intensional* mediante tres reglas. que expresan que **A** está relacionado con **B** si **A** y **B** son de la misma marca, o del mismo año de lanzamiento, o del mismo segmento (tipo de coche).

El uso de varias reglas con la misma cabeza es una forma simple de expresar la *disyunción* en Prolog.

La última definición de **isRelated** contienen tres predicados separados por comas en el cuerpo de la regla. Las comas representan una *conjunción* de predicados.

2.2 Consultas a los hechos

Empezamos realizando consultas sencillas sobre los hechos del programa. Por ejemplo, preguntar si, de acuerdo con la información que consta en memoria, **clio** es un modelo de **Renault:**

```
?- model(clio,renault).
true.
```

Ejercicio 3: ¿Qué ocurre cuando preguntas si **renault** es un modelo de **clio**? Escribe la consulta y explica la respuesta que devuelve el intérprete.

```
?- model(clio,X).
X = renault.
```

Ahora que queremos saber si **clio** es un coche de alguna marca, y en tal caso, saber cuál es. Para ello podemos lanzar al intérprete la misma consulta que antes, pero usando una simple variable **X** como segundo argumento.

Es decir, a partir de los hechos del programa, el intérprete puede demostrar que la consulta es cierta si **X = renault**. En el caso de que exista más de una respuesta a la consulta planteada, el sistema retorna la primera que computa. Si deseamos recibir más respuestas debemos pulsar **r** (redo) para conocer otras respuestas.

2.2 Consultas a los hechos

Ejercicio 4: Pregunta cuáles modelos son de **renault** sustituyendo, en la consulta del ejemplo anterior, la variable **X** por la constante **renault** y la constante **clio** por una variable (por ejemplo **X**). ¿Qué ocurre cuando pulsas la tecla **r** después de la primera respuesta?

La siguiente ejecución ilustra el uso de dos variables para extraer de la base de conocimiento todos los pares (**X**,**Y**) donde **X** es una marca del país **Y** (los puntos y comas aparecen en el terminal tras pulsar la tecla ; o la tecla r):

```
?- country(X,Y).
X = seat,
Y = españa ;
X = renault,
Y = francia ;
X = peugeot,
Y = francia ;
X = volkswagen,
Y = alemania ;
X = opel,
Y = alemania.
```

Ejercicio 5: Realiza la consulta:

```
?- country(X,X).
```

¿Qué contesta el intérprete? Explica esta respuesta.

2.3 Consultas a las reglas

Ahora vamos a realizar consultas que requieren deducir información que no está expresada con hechos en el programa pero que puede obtenerse a partir de los hechos y las reglas. Recordemos que las reglas permiten expresar que:

"Si es verdad el antecedente (cuerpo) entonces es verdad el consecuente (cabeza)".

Por ejemplo, haciendo uso de la regla:

```
brand(B, A) :- model(A, B).
```

que expresa que **B** es marca de **A** si **A** es modelo de **B**. Sabiendo que **ibiza** es un modelo de **seat**, el intérprete puede llegar a la conclusión de que **seat** es la marca de **ibiza** tras la siguiente consulta:

```
?- brand(seat,ibiza).
true.
```

También se puede preguntar cuáles son modelos de **seat** o de **opel**:

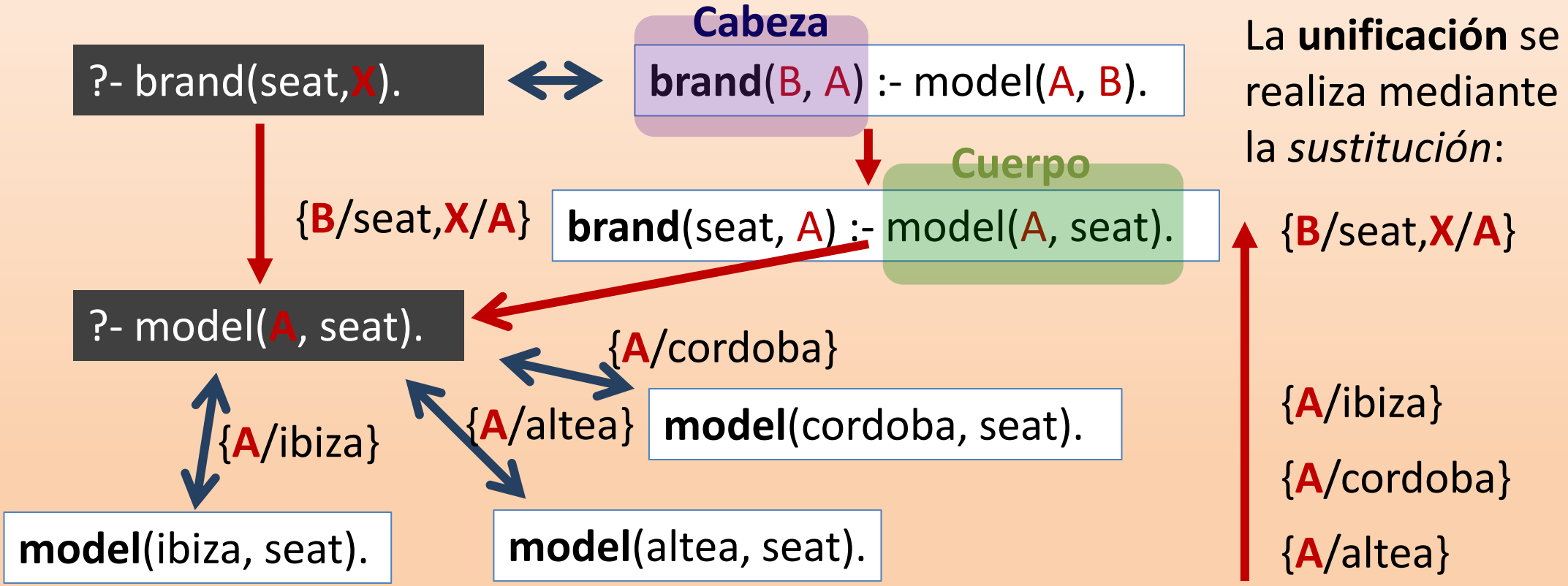
```
?- brand(seat,X).
X = ibiza ;
X = cordoba ;
X = altea.
?- brand(opel,X).
X = corsa.
```

2.3 Consultas a las reglas

Que está haciendo en realidad Prolog cuando hacemos esta consulta?:

?- brand(seat,**X**).

Lo que hace es buscar una **unificación** con algún *hecho* o *cabeza* de una relación de la base de conocimiento. En este caso con la *cabeza* de la regla:



La **unificación** se realiza con tres hechos de la base de conocimiento mediante tres *sustituciones.* Después deshacemos las sustituciones y obtenemos los valores de **X** para cada una de las ramas del árbol de deducción.

2.3 Consultas a las reglas

Ejercicio 6: Haciendo uso de **cars.pl**: Consultar cuáles modelos son de **alemania**. Consultar todos los modelos relacionados con el modelo **cordoba**.

Ejercicio 7: Usando un editor de texto, añadir al fichero **cars.pl** las siguientes reglas:

1. Una regla que defina la relación **isCountryOf** de forma similar a **brand** pero usando el predicado **isModelFrom** en lugar de **model**. Guarda los cambios y recarga el fichero con el predicado **reconsult(cars)**. Averigua de cuál país es el modelo **megane** usando el predicado **isCountryOf**.
2. Una regla que defina la relación **isClassic** que indique si un modelo fue comercializado antes del año 1995. Averigua cuáles modelos son clásicos usando el predicado **isClassic**.

Ejercicio 8: Prueba y modifica la regla **isRelated(A,B)**, que indica si dos modelos están relacionados porque sean de la misma marca, del mismo segmento, o del mismo año de lanzamiento: Lanza la consulta **isRelated(golf,X).** Amplía la relación **isRelated(A,B)**, añadiendo al programa una regla que indique que dos modelos están relacionados si ambos son clásicos (usando **isClassic**). Repite la consulta **isRelated(golf,X)**. En qué difieren las dos respuestas computadas?

3.1 Unificación de términos

En clases de teoría se estudia un algoritmo que determina si dos términos con variables unifican y, en el caso de hacerlo, devuelve un unificador (el más general).

Este algoritmo puede verse como un mecanismo de paso de parámetros bidireccional que generaliza el ajuste de patrones de los lenguajes funcionales permitiendo dar valores no solo a las variables del programa sino también a las de los términos de la consulta.

Informalmente, dos términos con variables unifican si su estructura es compatible y hay un valor de sus variables que los hace idénticos. Es decir, tras dar valor a las variables, los términos resultantes contienen exactamente los mismos símbolos, o simplemente se diferencian por un renombramiento de las variables. Veamos algunos ejemplos:

```
?- date(10,nov,2030) = date(X,nov,2030).
X = 10.
?- date(10,nov,2030) = date(X,nov,X).
False.
?- date(10,nov,2030) =  time(13,05).
False.
```

No unifican porque **X** no puede valer 10 y 2030 a la vez.

no unifican porque **date** y **time** son dos functores distintos.

3.1 Unificación de términos

```
?- X = time(13,05).
X = time(13,05).
?- X = date(10,nov,Y).
X = date(10,nov,Y).
?- date(10,oct,2030) = date(X,nov,2030).
False.
?- moment(date(10,nov,2030),Y) =moment(X,time(13,05)).
 X = date(10,nov,2030),
 Y = time(13,05).
?- moment(time(Time,Minutes)) = moment(time(13,05))
Time = 13,
Minutes = 05.
```

No unifican porque meses son distintos.

Ejercicio 9: Comprueba alguna de las afirmaciones que se hacen en los ejemplos anteriores ejecutando el algoritmo de unificación usado por el intérprete SWI-Prolog.

Ejercicio 10: Realiza lo mismo que en el ejercicio anterior pero usando ahora predicados. Por ejemplo: **model(X,volkswagen) = model(golf,Y).**

Unificación de términos compuestos (**flights.pl**)

Debemos cargar ahora el fichero **flights.pl** que contiene información sobre vuelos. Para cada vuelo directo existe un hecho representado por el predicado **flight** con ocho parámetros:

Origen (1), destino (2), fecha de salida (3), hora de salida (4), fecha de llegada (5), hora de llegada (6), duración (7) y precio (8).

Todas las fechas se representan con el functor **date** que agrupa tres términos como argumentos: el día, mes y año. Por ejemplo:

date(10,nov,2030)).

Todos los parámetros que representan una hora se escriben con el functor **time** y dos argumentos (hora y minutos). Por ejemplo:

time(13,05).

Un ejemplo de un hecho completo representando un vuelo es el siguiente:

flight(barcelona,madrid,
date(10,nov,2030),**time**(13,05),
date(10,nov,2030),**time**(15,05), 120, 80).

Unificación de términos compuestos (**flights.pl**)

El programa **flights.pl** contiene los cinco vuelos representados por los siguientes hechos:

flights.pl

```
flight(barcelona,madrid, date(10,nov,2030),time(13,05),
                         date(10,nov,2030),time(15,05), 120,80).
flight(barcelona,valencia, date(10,nov,2030),time(13,05),
                           date(10,nov,2030),time(15,05), 120,20).
flight(madrid,london, date(10,nov,2030),time(16,05),
                      date(10,nov,2030),time(17,35), 90,140).
flight(valencia,london, date(10,nov,2030),time(16,05),
                        date(10,nov,2030),time(17,35), 90,50).
flight(madrid,london, date(10,nov,2030),time(23,05),
                      date(11,nov,2030),time(00,25), 80,50).
```

Unificación de términos compuestos (**flights.pl**)

Ejercicio 11: Carga el fichero **flights.pl** y busca todos los vuelos desde **Valencia** a **Londres**. Usa los nombres de variables adecuados para que la salida sea:

```
?- flights ( ....  ).
DepartureDay = ArrivalDay, ArrivalDay = date(10, nov, 2030), DepartureTime
= time(16, 5), ArrivalTime = time(17, 35), Duration = 90, Price = 50.
```

Ejercicio 12: Realiza las siguientes consultas:

- Consulta todos los vuelos que salen desde Madrid el día 10 de noviembre de 2030. Para ello debes realizar una consulta en la que el tercer parámetro del predicado **flight** use el functor **date** con los parámetros de la fecha solicitada.
- Consulta los vuelos cuya hora de salida sea 13:05. Has de realizarlo de la misma forma que el ejercicio anterior pero con el functor **time**.
- Consulta los vuelos que salen a partir de las 16:00. Para ello debes realizar una consulta en la que el término del cuarto parámetro del predicado **flight** conste del functor **time** con dos variables (**H** y **M**) como parámetros. Después y separado por una coma, has de exigir que la hora sea mayor o igual que 16 (**H >=** 16). Recuerda que la coma entre predicados representa la conjunción lógica.

Unificación de términos compuestos (**flights.pl**)

- El programa **flights.pl** también contiene una regla que define el predicado de aridad 3 **connection_same_day** para representar vuelos con una única escala en el mismo día y con una hora de margen para realizar el enlace aéreo.

```
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.
```

- Este predicado tiene tres parámetros: origen, destino y fecha. Fíjate como en el cuerpo de la cláusula aparece una nueva variable Connection que representa la ciudad del enlace aéreo.
- El cuerpo de este predicado usa el predicado **is** para evaluar expresiones aritméticas asignando a la variable de la izquierda el resultado de evaluar la expresión de la derecha. En esta expresión se suman los minutos transcurridos desde el inicio del día, dejando 60 minutos para poder realizar el enlace.

Unificación de términos compuestos (**flights.pl**)

Ejercicio 13: Consulta todos los vuelos con una única escala que pueden realizarse el día 10 de Noviembre de 2030.

Ejercicio 14: ¿Qué ocurriría si la cuarta y última aparición de la variable Date en el cuerpo de la cláusula **connection_same_day** se reemplazara por otra variable Another_date?

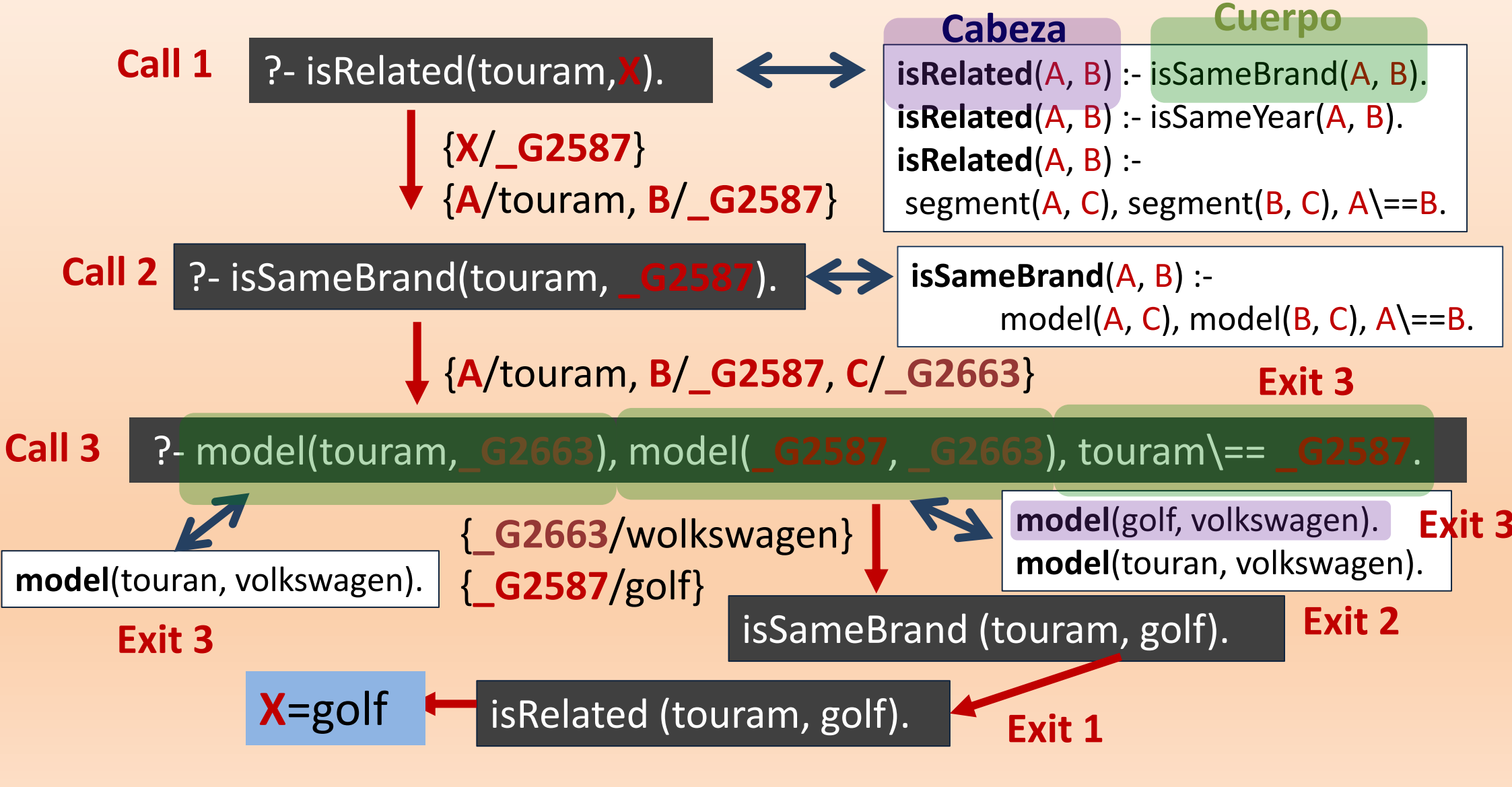
3.2 Búsqueda con retroceso

En Prolog, el proceso de búsqueda se concreta de la siguiente manera:

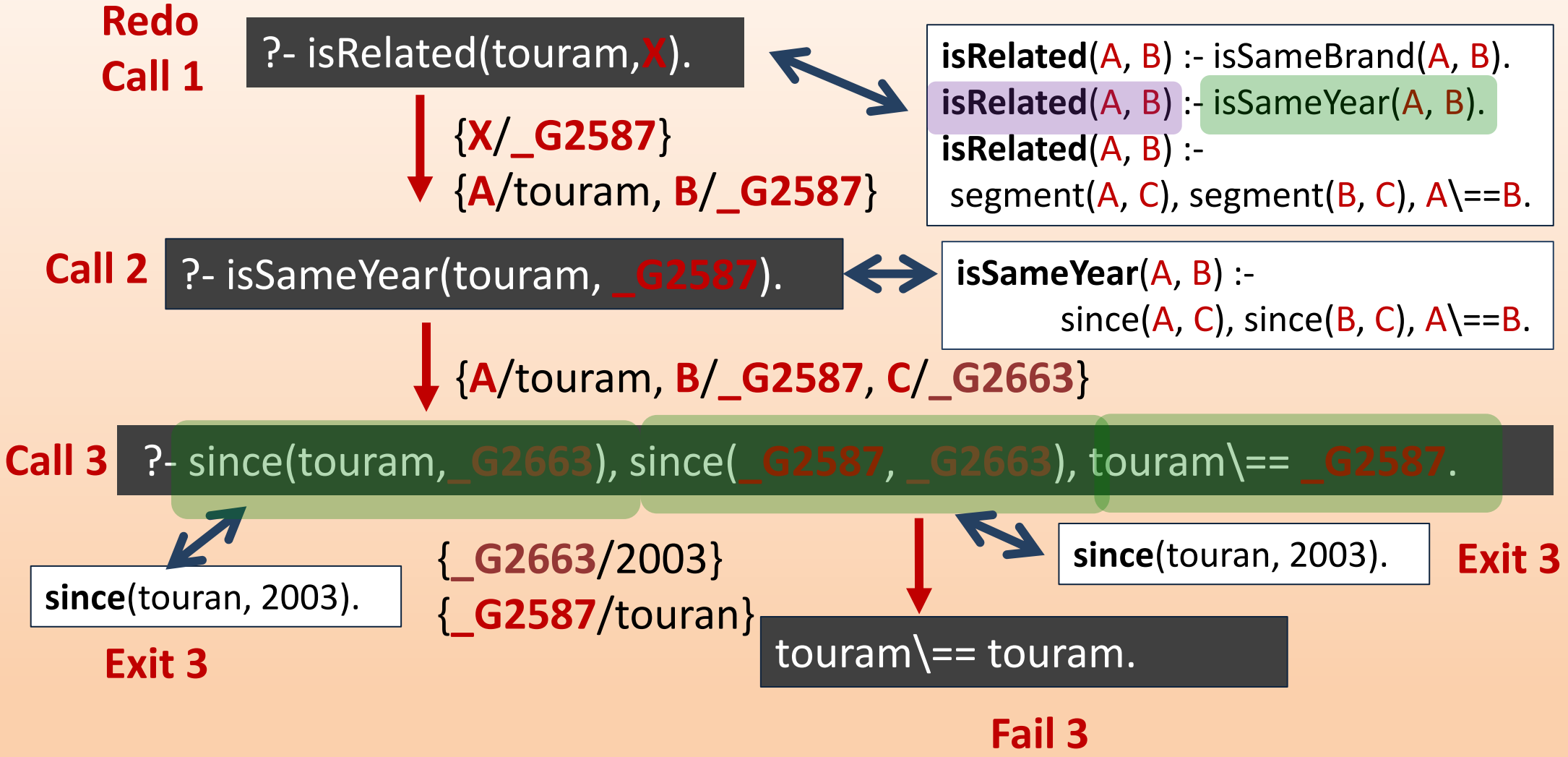
1. Se **van eligiendo cláusulas de arriba a abajo** en el orden en el que aparecen en el programa.
2. Cuando se unifica con la cabeza de una regla, **los predicados del cuerpo de ésta se van resolviendo (consultando) de izquierda a derecha**.
3. Cada predicado resuelto en este orden tiene aplicado el unificador resultante tanto de la unificación con la cabeza como las realizadas para resolver los predicados a su izquierda.
4. Si el intérprete **falla en resolver** uno de estos predicados del cuerpo (tras buscar todas las posibilidades de unificación de arriba a abajo), **da un paso atrás volviendo al predicado más próximo a la izquierda** del que ha fallado.
5. Se **deshace la última unificación** que hizo para este predicado **e intenta unificar con otra cláusula** del programa que esté **más abajo**.

3.2 Búsqueda con retroceso

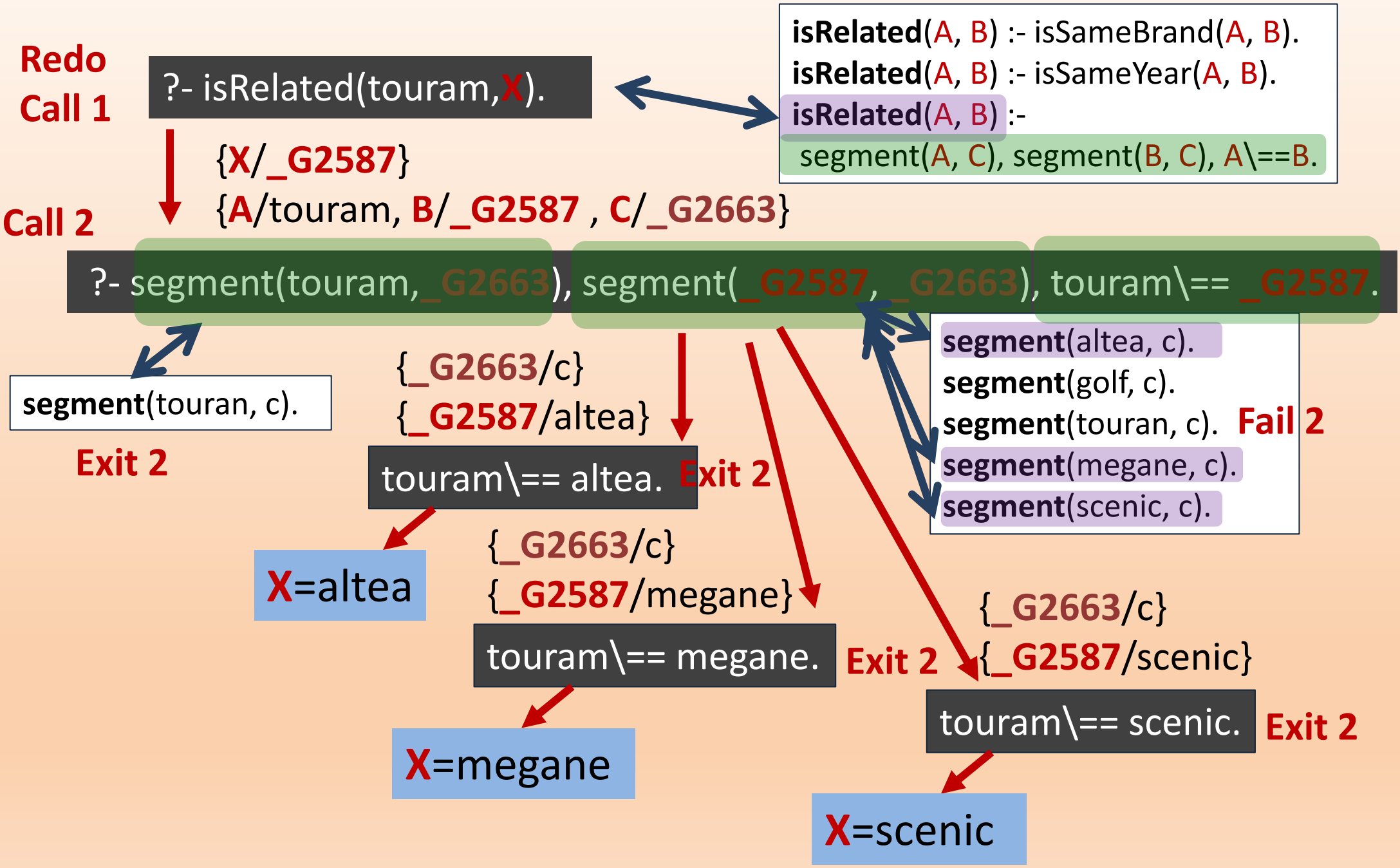
Para ilustrar este algoritmo de resolución consideraremos el ejemplo en que se lanza la consulta ?- **isRelated(touran,X).** sobre el programa del fichero **cars.pl** sin el predicado **isClassic** ni la cláusula **isRelated** que lo usa:



3.2 Búsqueda con retroceso

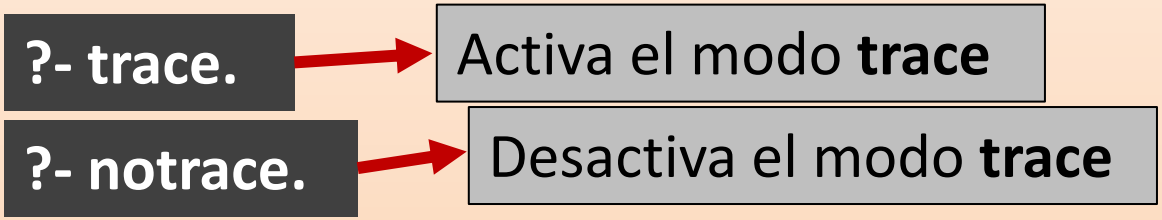


3.2 Búsqueda con retroceso



3.2 Búsqueda con retroceso

El intérprete SWI-Prolog proporciona predicados que permiten seguir las búsquedas y retrocesos: **debug**, **nodebug**, **trace** y **notrace**. Estos predicados inician el modo de depuración y permiten observar cómo el intérprete realiza la exploración en la base de conocimiento. El predicado **trace** es interactivo y permite interaccionar en el proceso (puedes ver las opciones pulsando h durante la traza):



Ejercicio 15: Comprueba que esté cargada la base de conocimiento **cars.pl** (se puede usar **listing**.) y en el caso de no estarlo, usa **consult(cars)**. Ejecuta la traza de la consulta **isRelated(touran,X).** excluyendo, y luego incluyendo, las cláusulas que se eliminaron en el ejemplo anterior (**isClassic**, y su **isRelated** correspondiente).

Ejercicio 16: Cerciórate de que esté cargada la base de conocimiento **flights.pl**. Ejecuta la traza de la consulta **connection_same_day(Origin, Destination, Connection)**.