7 SISTEMAS EXPERTOS

Un **sistema experto** es un programa que se comporta como un experto humano el algún dominio específico, usando conocimiento sobre ese dominio en particular. Es de esperar, que el programa sea capaz de explicar sus decisiones y su manera de razonar. También se espera que el programa sea capaz de contender con la incertidumbre y falta de información inherentes a esta tarea. En lo que sigue, nos basaremos en el texto de Bratko [19] para desarrollar los conceptos asociados a un sistema experto.

Sistema Experto

Algunas **aplicaciones** basada en sistemas expertos incluyen tareas como el diagnóstico médico, la localización de fallas en equipos y la interpretación de datos cuantitativos. Puesto que el éxito de estos sistemas depende en gran medida de su conocimiento sobre estos dominios, también se les conoce como **sistemas basados en el conocimiento** o *KBS*, por sus siglas en inglés. Aunque no todos los sistemas basados en el conocimiento ofrecen explicaciones sobre sus decisiones, ésta es una característica esencial de los sistemas expertos, necesaria principalmente en dominios inciertos como el diagnóstico médico, para garantizar la confianza del usuario en las recomendaciones del sistema; o para detectar un fallo flagrante en su razonamiento.

Aplicaciones

Sistemas basados en el conocimiento

La **incertidumbre** y la incompletez son inherentes a los sistemas expertos. La información sobre el problema a resolver puede ser incompleta o ni fiable. Las relaciones entre los objetos del dominio pueden ser aproximadas.

Incertidumbre

Ejemplo 7.1. *En el dominio médico, puede ser el caso de que **no estemos seguros** si un síntoma se ha presentado en un paciente, o que la medida de un dato es absolutamente correcta.*

Ejemplo 7.2. *El hecho de que ciertos fármacos **pueden** presentar reacciones adversas secundarias, pero éste no **suele** ser el caso. Este tipo de conocimiento requiere un manejo explícito de la incertidumbre.*

Para implementar un sistema experto, normalmente debemos considerar las siguientes funciones:

SOLUCIÓN DEL PROBLEMA. Una función capaz de usar conocimiento sobre el dominio específico del sistema, incluyendo el manejo de incertidumbre.

INTERFAZ DEL USUARIO. Una función que permita la interacción entre el usuario y el sistema experto, incluyendo la capacidad de explicar las decisiones y el razonamiento del sistema.

Cada una de estas funciones es extremadamente complicada, y puede depender del dominio de aplicación y de requerimientos prácticos específicos. Los problemas que pueden surgir en el diseño e implementación de un sistema experto tienen que ver con elegir una adecuada representación del conocimiento y los métodos de razonamiento asociados a tal representación. En lo que sigue, desarrollaremos un marco básico que podrá ser refinado posteriormente, conforme a nuestras necesidades. Es conveniente dividir el sistema experto en tres **módulos**, como se muestra en la figura 7.1:

Módulos de un sistema experto

BASE DE CONOCIMIENTOS. Incluye todo el conocimiento específico sobre un dominio de aplicación: Hechos, reglas y/o restricciones que describen las relaciones en el dominio. Puede incluir también métodos, heurísticas e ideas para resolver los problemas en ese dominio particular.

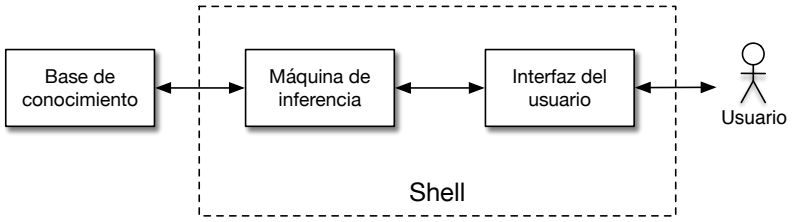

Figura 7.1: La estructura de un sistema experto.

MÁQUINA DE INFERENCIA. Incluye todos los procedimientos para usar activamente la base de conocimientos.

INTERFAZ DEL USUARIO. Se encarga de la comunicación entre el usuario y el sistema experto, debe proveer la información suficiente para que el usuario entienda el funcionamiento de la máquina de inferencia a lo largo del proceso.

También es conveniente ver la máquina de inferencia y la interfaz del usuario como un módulo independiente de cualquier base de conocimientos, lo que se conoce como un **shell** de sistema experto, o simplemente *shell*. Este esquema separa el conocimiento de los algoritmos que lo usan. Tal división es adecuada por los siguientes razones: La base de conocimientos depende claramente del cada dominio de aplicación específico, mientras que el *shell* es, al menos en principio, independiente de dicho dominio. De forma que, una forma racional de desarrollar sistemas expertos consiste en diseñar un solo *shell* que pueda usarse universalmente. Para ello las bases de conocimientos deberán apegarse al formalismo o representación que el *shell* puede manejar. En la práctica, este principio de diseño es difícil de mantener. Al menos que los dominios de aplicación sean muy similares entre si, lo normal es que el *shell* requiera de adecuaciones al cambiar de dominio. Sin embargo, aún en esos casos, la separación propuesta mantiene las ventajas de la modularidad.

Shell

En lo que sigue, aprenderemos a representar conocimiento usando reglas en formato si-entonces, conocidas como reglas de producción; así como los mecanismos de inferencia básicos sobre este formato: Los razonamientos por encadenamiento hacía adelante y hacía atrás. Estudiaremos las mejoras a estas reglas que se introducen al considerar la incertidumbre, las redes semánticas y la representación de conocimientos basada en marcos.

7.1 REGLAS DE PRODUCCIÓN

En principio, cualquier formalismo consistente con el que podamos expresar conocimiento acerca del dominio de un problema, puede ser considerado para su uso en un sistema experto. Sin embargo, el uso de reglas si-entonces, también llamadas **reglas de producción**, es tradicionalmente el formalismo más popular para representar conocimiento en un sistema experto. En principio, tales reglas son enunciados condicionales, pero pueden tener varias **interpretaciones**, por ejemplo:

Reglas de producción

Interpretaciones posibles

- Si condición P entonces conclusión C.
- Si situación S entonces acción A.
- Si las condiciones C_1 y C_2 son el caso, entonces la condición C no lo es.

De forma que, las reglas de producción son una forma natural de representar conocimiento, que además poseen las siguientes **características deseables**:

Características deseables

MODULARIDAD. Cada regla define una pequeña, relativamente independiente, pieza de información.

AGREGACIÓN. Es posible agregar nuevas reglas al sistema, de forma relativamente independiente al resto de sus reglas.

FLEXIBILIDAD. Como una consecuencia de la modularidad, las reglas del sistema pueden modificarse con relativa independencia de las otras reglas.

TRANSPARENCIA. Facilitan la explicación de las decisiones tomadas por el sistema experto. En particular, es posible automatizar la respuesta a preguntad del tipo ¿Cómo se llegó a esta conclusión? y ¿Porqué estás interesado en tal información?

Conocimiento categórico vs probabilístico

Las reglas de producción a menudo definen relaciones lógicas entre conceptos del dominio de un problema. Las relaciones puramente lógicas pueden caracterizarse como **conocimiento categórico**, en el sentido de que son siempre absolutamente verdaderas. En algunos dominios, como el médico, el **conocimiento probabilístico** prevalece. En este caso, las relaciones no son siempre verdaderas y se establecen con base en su regularidad empírica, usando grados de certidumbre. Las reglas de producción deben ajustarse para contender con enunciados del tipo: Si condición A entonces conclusión C con un grado de certeza F.

Ejemplo 7.3. *Una regla con conocimiento no categórico en el dominio médico. Proviene del sistema experto desarrollado para el libro de Negrete-Martínez, González-Pérez y Guerra-Hernández [84] sobre la pericia artificial y su implementación incremental en Lisp. Aquí la regla se muestra en un pseudo-código más cercano a lo que haremos en Prolog:*

```
Regla 037:
IF
paciente(hipertermia,si), paciente(tos,si), paciente(insufResp,si)
;
paciente(estertoresAlveolares,si), paciente(condensacionPulmonar,si)
THEN
paciente(neumonia,[si,80]).
```

en este caso, la conclusión de la regla incluye un factor de certidumbre, no estrictamente probabilistico, que expresa que tan confiable es tal conclusión. Veremos más al respecto al abordar la incertidumbre.

Por lo general, si se quiere desarrollar un sistema experto, será necesario consultar a un experto humano en el dominio del problema y estudiar el dominio uno mismo. Al proceso de extraer conocimiento de los expertos y la literatura de un dominio dado, para acomodarlo a un formalismo de representación se le conoce como ingeniería del conocimiento [1].

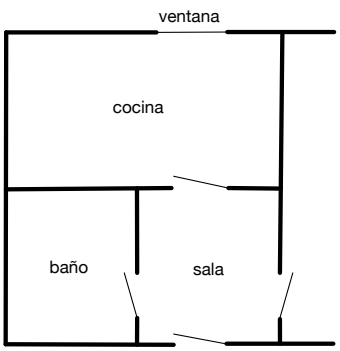

Figura 7.2: El plano de una casa donde debemos detectar fugas de agua.

1 En inglés suele usarse el intraducible término de *knowledge elicitation*.

El proceso de ingeniería del conocimiento, como tal, queda fuera de los alcances de este curso, pero jugaremos con un dominio sencillo en nuestros experimentos. La figura 7.2 muestra el plano de una casa donde deberemos localizar fugas de agua.

Luego de consultar con nuestro plomero favorito y consultar en internet los manuales de plomería para *dummies*, llegamos a la conclusión de que la fuga puede presentarse en el baño o en la cocina. En cualquier caso, la fuga causa que haya agua en el piso de la sala. Observen que asumimos que la fuga solo se da en un sitio, no en ambos al mismo tiempo. Esto se puede representar como una **red de inferencia**, tal y como se muestra en la figura 7.3.

Red de inferencia

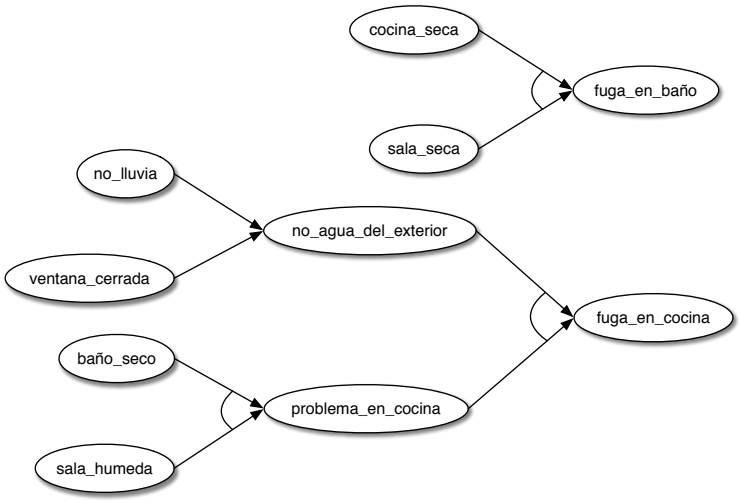

Figura 7.3: La red de inferencia de nuestro conocimiento sobre fugas.

Los nodos en la red de inferencia representan proposiciones y las ligas, reglas en la base de conocimientos. Los arcos que conectan ligas, indican la naturaleza conjuntiva entre las correspondiente proposiciones.

Ejemplo 7.4. *De acuerdo con lo anterior, una regla de nuestro sistema experto debería ser:*

```
IF baño_seco AND sala_humeda THEN problema_en_cocina.
```

A la red de inferencia representada de esta forma, también se le conoce como **grafo AND/OR**.

Grafo AND/OR

7.2 RAZONAMIENTO CON REGLAS DE PRODUCCIÓN

Una vez que el conocimiento se ha representado en algún formalismo, necesitamos de un procedimiento de razonamiento para extraer conclusiones de éste. Hay dos formas básicas de razonamiento con reglas de producción:

- Razonamiento hacía atrás.
- Razonamiento hacía adelante.

7.2.1 Razonamiento hacía atrás

Una vez que tenemos nuestra red de inferencias, el razonamiento hacía atrás funciona de la siguiente manera. Partimos de una hipótesis, por ejemplo, que hay una fuga en la cocina; y razonamos hacía atrás sobre la red de inferencias. Para resolver esta hipótesis necesitamos que no agua en el exterior y problema en la cocina sean verdaderas. La primera es el caso si no llueve o la ventana está cerrada, etc. El razonamiento se conoce como **hacía atrás**, porque forma una cadena de las hipótesis

Razonamiento hacía atrás

(problema en la cocina) hacía las evidencias (ventana cerrada).

Tal procedimiento es muy fácil de implementar en Prolog, de hecho, ¡es la forma en que este lenguaje razona! La manera directa de implementar nuestra red de inferencias es usando las reglas provistas por Prolog:

```
% Base de conocimientos para fuga en casa

% Conocimiento previo

fuga_en_bano :-
    sala_seca,
    cocina_seca.

problema_en_cocina :-
    sala_humeda,
    bano_seco.

no_agua_del_exterior :-
    ventana_cerrada
    ;
    no_lluvia.

fuga_en_cocina :-
    problema_en_cocina,
    no_agua_del_exterior.

% Evidencia

sala_humeda.
bano_seco.
ventana_cerrada.
```

Observen que la evidencia se representa como hechos, mientras que el conocimiento previo hace uso de reglas Prolog.

Nuestra hipótesis puede ahora verificarse mediante la siguiente consulta:

```
?- fuga_en_cocina.
true
```

Sin embargo, usar las reglas de Prolog directamente tiene algunas desventajas:

1. La sintaxis de estas reglas no es la más adecuada para alguien que no conoce Prolog. Este es el caso de nuestros expertos humanos, que deben especificar nuevas reglas, leer las existentes y posiblemente modificarlas, sin saber Prolog.

2. La base de conocimientos es sintácticamente indistinguible del resto del sistema experto. Esto, como se mencionó, no es deseable.

Como vimos en el tutorial de Prolog, es muy sencillo definir una sintaxis diferenciada para las reglas de producción de nuestro sistema experto, haciendo uso de la definición de operadores. De forma que un interprete para razonamiento hacía atrás basado en reglas de producción luciría así:

```
% Un interprete de encadenamiento hacía atrás para reglas if-then.

:-  op( 800, fx, if).
:-  op( 700, xfx, then).
:-  op( 300, xfy, or).
:-  op( 200, xfy, and).

is_true( P) :-
   fact( P).

is_true( P) :-
   if Cond then P,         % Una regla relevante,
   is_true( Cond).         % cuya condición Cond es verdadera.

is_true( P1 and P2) :-
```

```
16    is_true( P1),
17    is_true( P2).
18
19 is_true( P1 or P2)  :-
20    is_true( P1)
21    ;
22    is_true( P2).
```

Hemos introducido los operadores infijos `then`, `or` y `and`, junto con un operador prefijo `if` para representar nuestras reglas de producción de una manera diferenciada de las reglas de Prolog. Por ello es necesario establecer la semántica de nuestras reglas, a través del preciado `is_true`. Observen que los operadores definidos también se usan en las reglas semánticas.

Adecuando el formato de las reglas anteriores, nuestra base de conocimientos quedaría implementado de la siguiente forma:

```
1  %%%base de conocimiento para fugas, en fomato de reglas if-then
2
3  %%%Conocimiento previo
4
5  if sala_humeda and cocina_seca
6  then fuga_en_bano.
7
8  if sala_humeda and bano_seco
9  then problema_en_cocina.
10
11 if ventana_cerrada or no_lluvia
12 then no_agua_del_exterior.
13
14 if problema_en_cocina and no_agua_del_exterior
15 then fuga_en_cocina.
16
17 %%%Evidencias
18
19 fact(sala_humeda).        %Cambiar a sala_seca para que falle la meta
20 fact(bano_seco).
21 fact(ventana_cerrada).    %Comentar para probar explicaciones porque
```

Observen el uso del predicado `fact` para representar los hechos del sistema experto. La hipótesis de fuga en la cocina puede verificarse de la siguiente forma:

```
1 ?- is_true(fuga_en_cocina).
2 true
```

Desventajas

Una **desventaja** del procedimiento de inferencia definido es que el usuario debe incluir en la base de conocimientos, toda la evidencia con la que cuenta en forma de hechos, antes de poder iniciar el proceso de razonamiento. Lo ideal sería que las evidencias fueran provistas por el usuario conforme se van necesitando, de manera interactiva. Esto lo resolveremos más adelante.

7.3 RAZONAMIENTO HACÍA ADELANTE

En el razonamiento hacía atrás partíamos de las hipótesis para ir hacía la evidencia. Algunas veces resulta más natural razonar en la dirección opuesta: A partir de nuestro conocimiento previo y la evidencia disponible, explorar que conclusiones podemos obtener. Por ejemplo, una vez que hemos confirmado que la sala está mojada y que el baño está seco, podríamos concluir que hay un problema en la cocina.

Programar un procedimiento de razonamiento hacía adelante en Prolog, sigue siendo sencillo, si bien no trivial como es el caso del razonamiento hacía atrás. Nuestro interprete es el siguiente:

```
1 %Un intérprete simple para el razonamiento hacia adelante
2
```

```
forward :-
    new_derived_fact(P),    % Se deriva un nuevo hecho.
    !,
    write('Nuevo hecho derivado: '), write(P), nl,
    assert(fact(P)),
    forward   % Buscar más hechos nuevos.
    ;
    write('No se derivaron más hechos.').   % Terminar, no más hechos derivados.

new_derived_fact(Concl) :-
    if Cond then Concl,   % Una regla
    \+ fact(Concl),       % cuya conclusión no es un hecho
    composed_fact(Cond).  % Su condición es verdadera?

composed_fact(Cond) :-
    fact(Cond).           % Un hecho simple

composed_fact(Cond1 and Cond2) :-
    composed_fact(Cond1),
    composed_fact(Cond2). % Ambos operandos verdaderos.

composed_fact(Cond1 or Cond2) :-
    composed_fact(Cond1)
    ;
    composed_fact(Cond2). % Al menos un operando verdadero.
```

Nuestra base de conocimiento seguirá siendo la usada en el caso del razonamiento hacía atrás. Algo de modularidad hemos conseguido. Por simplicidad, asumimos que las reglas en esa base de conocimiento no tienen variables. El interprete parte de la evidencia conocida, especificada mediante el predicado `fact` en la base de conocimientos, deriva los hechos que se pueden inferir mediante las reglas de producción y los agrega a la base de conocimiento, mediante el predicado `assert` (lo cual nos lleva a definir `fact` como dinámico). La ejecución de este procedimiento de razonamiento es como sigue:

```
?- forward.
Nuevo hecho derivado: problema_en_cocina
Nuevo hecho derivado: no_agua_del_exterior
Nuevo hecho derivado: fuga_en_cocina
No se derivaron más hechos.
true.
```

Una interfaz para cargar los modulos definidos hasta ahora, se implementa como sigue:

```
% Cargar el sistema basado en conocimiento.

:- dynamic(fact/1).

:- [encadenamientoAtras].
:- [encadenamientoAdelante].
:- [kbFugasIfThen].
```

7.4 COMPARANDO LOS RAZONAMIENTOS

Las reglas de producción, representadas en nuestra red de inferencias (Fig. 7.3), forman cadenas que van de la izquierda a la derecha. Los elementos en el lazo izquierdo de estas cadenas son información de entrada, mientras que los del lado derecho son información derivada:

$$\text{Información de Entrada} \quad \rightarrow \cdots \rightarrow \quad \text{Información Derivada}$$

Estos dos tipos de información reciben varios nombres, dependiendo del contexto en el cual son usados. La información de entrada puede llamarse **datos**, por ejem- *Datos*

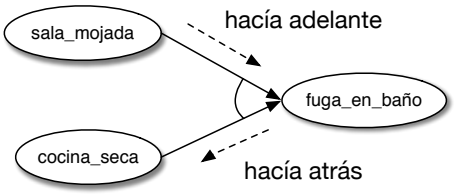

Figura 7.4: Un razonamiento en ambos sentidos disparado por la evidencia de que la sala está mojada.

plo, en el caso de información de entrada que requiere mediciones. La información derivada puede llamarse **metas**, o hipótesis, o diagnóstico o explicación. De forma que las cadenas de inferencia conectan varios tipos de información como:

Metas

Datos	$\rightarrow \cdots \rightarrow$	Metas
Evidencia	$\rightarrow \cdots \rightarrow$	Hipótesis
Observaciones	$\rightarrow \cdots \rightarrow$	Diagnósticos
Descubrimientos	$\rightarrow \cdots \rightarrow$	Explicaciones
Manifestaciones	$\rightarrow \cdots \rightarrow$	Causas

Ambos razonamientos involucran búsqueda, pero difieren en su dirección. El razonamiento hacía atrás busca de las metas a los datos, mientras que el razonamiento hacía adelante lo hace en el sentido inverso. Como el primero inicia con una meta, se dice que es un proceso **dirigido por metas**. El razonamiento hacía adelante, puede verse como un procesamiento **dirigido por datos**.

Razonamiento dirigido por metas y por datos

Una pregunta evidente es ¿Qué tipo de razonamiento es mejor? La respuesta depende del problema. Si lo que queremos es verificar si una hipótesis determinada es verdadera, entonces el razonamiento hacía atrás resulta más natural. Si es el caso que tenemos numerosas hipótesis y ninguna razón para comenzar con una de ellas en particular, entonces el encadenamiento hacía adelante es más adecuado. En general, el razonamiento hacía adelante es más adecuado en situaciones de monitoreo, donde los datos se adquieren continuamente.

La forma de la red de inferencias puede ayudar a decidir que tipo de razonamiento usar. Si solo hay unos pocos nodos de datos (el flanco izquierdo de la gráfica) y muchos nodos meta (el flanco derecho), entonces el encadenamiento hacía adelante sería el más apropiado; y viceversa, el encadenamiento hacía atrás es más apropiado en el caso contrario.

Las tareas de un experto son normalmente más intrincadas y requieren de una combinación de encadenamientos en ambas direcciones. En medicina, por ejemplo, normalmente algunas observaciones iniciales disparan razonamientos del médico hacía adelante, para generar una hipótesis de diagnóstico inicial. Ésta debe ser confirmada o rechazada con base en evidencia adicional, la cual se obtiene mediante procesos de razonamiento hacía atrás. En nuestro ejemplo, observen que la evidencia de que la sala está mojada podría disparar un razonamiento como el mostrado en la figura 7.4.

7.5 GENERANDO EXPLICACIONES

Existen formas estándar de generar explicaciones en los sistemas basados en reglas de producción. Los tipos de explicación computables son respuestas a las preguntas de tipo ¿Cómo? y ¿Porqué?

7.5.1 Explicaciones tipo ¿Cómo?

Cuando nuestro sistema experto genera una respuesta, el usuario puede preguntar –¿Cómo obtuviste esa respuesta? La explicación consiste en la traza de como se derivo la respuesta en cuestión.

Ejemplo 7.5. *Supongamos que la última respuesta computada por nuestro sistema es que hay una fuga en la cocina. La explicación sería como sigue, por que:*

1. *Hay un problema en la cocina, lo cual fue concluído a partir de que la sala está mojada y el baño seco.*
2. *El agua no provino del exterior, lo cual fue concluído a partir de que la ventana estaba cerrada.*

Árbol de derivación

Tal explicación es de hecho el **árbol de derivación** de la meta, es decir, el árbol-SLD (Def. 5.30) que Prolog construye cada vez que le planteamos una meta. Definamos `<=` como un operador infijo, para poder representar el árbol de prueba de una proposición P de la siguiente manera:

1. Si P es un hecho, entonces el árbol de prueba es P mismo.
2. Si P fue derivado usando una regla: if *Cond* then P, el árbol de prueba es P `<=` *PruebaCond*; donde *PruebaCond* es a su vez, el árbol de prueba de *Cond*.
3. Sean P_1 y P_2 proposiciones cuyos árboles de prueba son A_1 y A_2, entonces el árbol de prueba de P_1 `and` P_2 es A_1 `and` A_2. El caso de `or`, se resuelve análogamente.

Esta definición se puede implementar en Prolog, refinando el predicado `is_true/1` con un segundo argumento que unifique con el árbol de prueba:

```
% is_true(P,A) A es el árbol de prueba de la meta proposicional P

:- op(800, xfx, <=).

is_true(P,P) :-
    fact(P).

is_true(P, P <= CondA) :-
    if Cond then P,
    is_true(Cond,CondA).

is_true(P1 and P2, A1 and A2) :-
    is_true(P1,A1),
    is_true(P2,A2).

is_true(P1 or P2, A1 or A2) :-
    is_true(P1,A1)
    ;
    is_true(P2,A2).
```

Si incluimos este módulo en nuestra versión 2 de la interfaz, tendremos:

```
% Cargar el sistema basado en conocimiento.

:- dynamic(fact/1).

:- [encadenamientoAtras].
:- [encadenamientoAdelante].
:- [explicacionesComo].
:- [kbFugasIfThen].
```

Y su uso, se ilustra con la siguiente consulta:

```
?- [loadSystemV2].
true.
```

```
?- is_true(fuga_en_cocina,A).
A =  (fuga_en_cocina<=(problema_en_cocina<=sala_humeda and
bano_seco)and(no_agua_del_exterior<=ventana_cerrada or _66))
```

Por supuesto, podemos mejorar la manera en que el árbol de derivación se despliega para el usuario. Para ello, definiremos un predicado `show_tree/1`:

```
% show_tree(A): Imprime el árbol de derivación A.

show_tree(A) :-
    show_tree(A,0).

% show_tree(A,N): Imprime el árbol de derivación A,
% al nivel de indentación N.

show_tree(A,N) :-
    var(A), tab(N), writeln('cualquier_cosa').

show_tree(A1 <= A2,N) :- !,
    tab(N), write(A1), writeln(' <='),
    show_tree(A2,N).

show_tree(A1 and A2,N) :- !,
    N1 is N + 2,
    show_tree(A1,N1), tab(N1), writeln('and'),
    show_tree(A2,N1).

show_tree(A1 or A2,N) :- !,
    N1 is N + 2,
    show_tree(A1,N1), tab(N1), writeln('or'),
    show_tree(A2,N1).

show_tree(A,N) :-
    tab(N),writeln(A).
```

Ahora podemos hacer la siguiente consulta:

```
?- [loadSystemV2].
true.

?- is_true(fuga_en_cocina,A), show_tree(A).
fuga_en_cocina <=
  problema_en_cocina <=
    sala_humeda
    and
    bano_seco
  and
  no_agua_del_exterior <=
    ventana_cerrada
    or
    cualquier_cosa
A =  (fuga_en_cocina<=(problema_en_cocina<=sala_humeda and
bano_seco)and(no_agua_del_exterior<=ventana_cerrada or _12250))
```

7.5.2 Explicaciones tipo ¿Porqué?

Interacción

La explicación sobre porqué un dato es relevante, a diferencia de las del tipo ¿Cómo?, se necesitan en tiempo de ejecución, no al final de una consulta. Esto require que el usuario pueda **interactuar** con la máquina de inferencia del sistema experto, através de la interfaz.

Una forma de introducir interacción en nuestro sistema, es que éste pregunte al usuario por las evidencias que va necesitando en su proceso de razonamiento. En ese momento, el usuario podrá preguntar, porqué tal información es necesaria. Será necesario especificar que proposiciones pueden ser preguntadas al usuario, mediante el predicado `askable/1`.

Ejemplo 7.6. *Se puede agregar a nuestra base de conocimientos, las siguientes líneas:*

```
%%%Hechos que se pueden preguntar la usuario

askable(no_lluvia).
askable(ventana_cerrada).
```

Si hemos agregado $askable(P)$ a nuestra base de conocimientos, cada vez que la meta P se presente, el sistema experto preguntará si P es el caso, con tres respuesta posibles: `si`, `no`, y `porque`. La última de ellas, indica que el usuario está interesado en saber porqué tal información es necesaria. Esto es particularmente necesario en situaciones donde el usuario no sabe la respuesta, y establecer si P es o no el caso, requiere de cierto esfuerzo.

En caso de que el usuario responda `si` al sistema, la proposición P es agregada a la memoria de trabajo, usando `assert(fact(P))`. Debemos agregar también una cláusula al estilo de `already_asked(P)` a la memoria de trabajo, para evitar preguntar de nuevo por P cuando esto ya no es necesario. La respuesta a `porque` debería ser una cadena de reglas que conectan la evidencia P con la meta original que se le ha planteado al sistema. Veamos la implementación de este mecanismo en Prolog:

```
:- dynamic already_asked/1.

% iis_true(P,A): P es el caso con la explicación A.
% Versión interactiva con el usuario.

iis_true(P,A) :-
    explore(P,A,[]).

% explore(P,A,T): A es una explicación de porque P es verdadera, H
% es una cadena de reglas que liga P con las metas anteriores.

explore(P, P, _) :-
    fact(P).

explore(P1 and P2, A1 and A2, T) :-
    explore(P1, A1, T),
    explore(P2, A2, T).

explore(P1 or P2, A1 or A2, T) :-
    explore(P1, A1, T)
    ;
    explore(P2, A2, T).

explore(P, P <= ACond, T) :-
    if Cond then P,
    explore(Cond, ACond, [if Cond then P | T]).

explore(P,A,T) :-
    askable(P),
    \+ fact(P),
    \+ already_asked(P),
    ask_user(P, A, T).

% ask_user(P,A,T): Pregunta al usuario si P es el caso,
% generando las explicaciones A y T.

ask_user(P, A, T) :-
    nl, write('Es cierto que: '), write(P),
    writeln(' Conteste si/no/porque:'),
    read(Resp),
    process_answer(Resp,P,A,T).

process_answer(si,P, P <= preguntado,_) :-
    asserta(fact(P)),
    asserta(already_asked(P)).

process_answer(no,P,_,_) :-
    asserta(already_asked(P)),
    fail.

```

```
process_answer(porque,P,A,T) :-
    display_rule_chain(T,0), nl,
    ask_user(P,A,T).

% display_rule_chain(R,N): Despliega las reglas R, identando a
% nivel N.

display_rule_chain([],_).

display_rule_chain([if Cond then P | Reglas], N) :-
    nl, tab(N), write('Para explorar si '),
    write(P), write(' es el caso, usando la regla '),
    nl, tab(N), write(if Cond then P),
    N1 is N + 2,
    display_rule_chain(Reglas,N1).
```

que agregaremos a nuestra tercera versión de la interfaz. Observen que el predicado principal se llama ahora `iis_true/2`, para indicar que es la versión interactiva del anterior `is_true/2` [2] De esta forma, nuestra interfaz `loadSystemV3.pl` quedaría como:

```
% Cargar el sistema basado en conocimiento.

:- dynamic(fact/1).

:- [encadenamientoAtras].
:- [encadenamientoAdelante].
:- [explicacionesComo].
:- [explicacionesPorque].
:- [kbFugasIfThen].
```

Una consulta al sistema, sería como sigue. Antes de ejecutar la consulta, verifique que ha comentado la línea correspondiente a `fact(ventana_cerrada` en la base de conocimientos original, para que el sistema experto deba preguntar por este hecho o por `no_lluvia`. El usuario puede pedir explicaciones al respecto en ambos casos.

```
?- [loadSystemV3].
true.

?- iis_true(fuga_en_cocina,A), show_tree(A).

Es cierto que: ventana_cerrada Conteste si/no/porque:
|: porque.

Para explorar si no_agua_del_exterior es el caso, usando la regla
if ventana_cerrada or no_lluvia then no_agua_del_exterior
  Para explorar si fuga_en_cocina es el caso, usando la regla
  if problema_en_cocina and no_agua_del_exterior then fuga_en_cocina

Es cierto que: ventana_cerrada Conteste si/no/porque:
|: si.

fuga_en_cocina <=
  problema_en_cocina <=
    sala_humeda
    and
    bano_seco
  and
  no_agua_del_exterior <=
    ventana_cerrada <=
    preguntado
    or
    cualquier_cosa

A =  (fuga_en_cocina<=(problema_en_cocina<=sala_humeda and bano_seco)
and(no_agua_del_exterior<=(ventana_cerrada<=preguntado)or _6410))
```

2 De otra forma hay un conflicto de nombres, que Prolog resuelve definiendo nuevamente el predicado en cuestión a partir del último archivo cargado.

Obviamente el módulo de explicaciones del tipo ¿Porqué? solo funciona con reglas proposicionales. Si usásemos variables, habría que preguntar también por los valores de estas. De cualquier forma, nuestro sistema ilustra los principios que queríamos presentar sobre las explicaciones en los sistemas expertos.

7.6 INCERTIDUMBRE

Hasta ahora, hemos asumida que la representación de conocimiento de nuestro problema es categórica, en decir, las respuestas a todas nuestras preguntas son verdaderas o falsas. Como datos, las reglas también han sido concebidas de manera categórica. Sin embargo, mucha de la experticia humana no es categórica. Cuando un experto busca la solución a un problema dado, en muchas ocaciones se asumen muchos hechos, que si bien generalmente son ciertos, suelen presentar excepciones. Tanto los datos como las reglas asociadas a un dominio de problema, pueden ser parcialmente certeros. Tal incertidumbre se puede modelar si asignamos alguna calificación, otra que no sea verdadero o falso, a los hechos reconocidos. Estas calificaciones pueden expresarse como descriptores, como verdadero, altamente probable, probable, poco probable, imposible. Alternativamente, el grado de certeza pueda expresarse como un número real en algún intervalo, por ejemplo, entre 0 y 1, o entre -5 y +5. Tales valores reciben diversos nombres como **grados de certeza**, o grados de creencia, o probabilidad subjetiva. La mejor manera de expresar incertidumbre es usando probabilidades, debido a sus fundamentos matemáticos. Sin embargo, razonar correctamente usando la probabilidad es más demandante que los esquemas *ad hoc* de incertidumbre, que aquí revisaremos.

Grados de certeza

En esta sección extenderemos nuestra representación basada en reglas de producción con un **esquema de incertidumbre** simple, que aproxima las probabilidades de manera muy rudimentaria. Cada proposición P será asociada a un número FC entre 0 y 1, que representará su grado de certeza. Usaremos un par $P : FC$ para esta representación. La notación adoptada para las nuevas reglas será: `if` *Cond* `then` $P : FC$.

Proposiciones y reglas con certeza

En toda representación que maneje incertidumbre, es necesario especificar la forma de combinar la certeza de los hechos y las reglas en el sistema.

Ejemplo 7.7. *Sean dos proposiciones, P_1 y P_2, con valores asociados de certeza $c(P_1)$ y $c(P_2)$ respectivamente ¿Cual es la certeza $c(P_1$ `and` $P_2)$?*

La misma pregunta aplica para el resto de los operadores en nuestro lenguaje de reglas de producción. En nuestro caso, el esquema de combinación será el siguiente

$$c(P_1 \textit{ and } P_2) = \text{mín}(c(P_1), c(P_2)) \tag{7.1}$$

$$c(P_1 \textit{ or } P_2) = \text{máx}(c(P_1), c(P_2)) \tag{7.2}$$

En el caso de una regla `if` P_1 `then` $P_2 : FC$, entonces:

$$c(P_2) = c(P_1) \times FC \tag{7.3}$$

Por simplicidad asumiremos que las reglas tienen implicaciones únicas. Si ese no fuese el caso, las reglas en cuestión pueden escribirse con ayuda del operador `or`, para satisfacer esta restricción de formato. Nuestra nueva base de conocimientos, llamada `kbFugasIncert.pl` incluye:

```
1 %%%base de conocimiento para fugas, en fomato de reglas if-then
2
3 %%%Conocimiento previo con certidumbre
4
5 if sala_humeda and cocina_seca
6 then fuga_en_bano.
7
8 if sala_humeda and bano_seco
```

```
then problema_en_cocina:0.9.

if ventana_cerrada or no_lluvia
then no_agua_del_exterior.

if problema_en_cocina and no_agua_del_exterior
then fuga_en_cocina.

% Evidencia con certidumbre

given(sala_humeda, 1).      % verdadero
given(bano_seco, 1).
given(cocina_seca, 0).      % falso
given(no_lluvia, 0.8).      % muy probable
given(ventana_cerrada, 0).
```

Observen que la regla de la línea 9, o mejor dicho, su conclusión, ha sido relativizada con un grado de certeza de 0.9. La máquina de inferencia basada en las ecuaciones 7.1-7.3, es como sigue:

```
% Interprete basado en reglas con incertidumbre

% cert(P,C): C es el grado de certeza de la proposición P

cert(P,C) :-
    given(P,C).

cert(P1 and P2, C) :-
    cert(P1,C1),
    cert(P2,C2),
    min(C1,C2,C).

cert(P1 or P2, C) :-
    cert(P1,C1),
    cert(P2,C2),
    max(C1,C2,C).

cert(P, C) :-
    if Cond then P:C1,
    cert(Cond,C2),
    C is C1 * C2.

cert(P, C) :-
    if Cond then P,
    cert(Cond,C1),
    C is C1.

% max y min

max(X,Y,X) :-  X>=Y,!.
max(_,Y,Y).

min(X,Y,X) :-  X=<Y,!.
min(_,Y,Y).
```

Una nueva interfaz versión 4 carga los archivos correspondientes:

```
% Cargar el sistema basado en conocimiento.

:-  dynamic(fact/1).

:-  [encadenamientoAtras].
:-  [encadenamientoAdelante].
:-  [incertidumbre].
:-  [kbFugasIncert].
```

La consulta al sistema es como sigue:

```
?- cert(fuga_en_cocina,FC).
FC = 0.8
```

Dificultades con la incertidumbre

Esquemas como el aquí propuesto son fácilmente criticables por no seguir los axiomas de la probabilidad formal.

Ejemplo 7.8. *Asumamos que el factor de certeza de a es 0.5 y el de b es 0. Entonces, el factor de certeza de a* ***or*** *b es 0.5. Ahora, si en factor de certeza de b se incrementa a 0.5, esto no tiene efecto en el factor de certeza de la disyunción, que sigue siendo 0.5; lo cual resulta, al menos, contraintuitivo.*

Por otra parte, se ha argumentado que la teoría de probabilidad, aunque matemáticamente bien fundada, no es ni práctica, ni apropiada en estos casos, por las siguientes razones:

- Los expertos humanos parecen tener problemas para razonar basados realmente en la teoría de probabilidad. La verosimilitud que usan, no se corresponde con la noción de probabilidad definida matemáticamente.
- El procesamiento de información con base en la teoría de probabilidad, parece requerir de información que no siempre está disponible; o asumir simplificaciones que no siempre se justifican.

Estas objections aplican a esquemas simplificados como el aquí propuesto. En realidad, la opinión unánime es a favor del uso de enfoques híbridos que aprovechen la teoría de probabilidad, pero para ello hacen falta máquinas de inferencia más elaboradas, por ejemplo, basadas en redes bayesianas.

7.7 RAZONAMIENTO BAYESIANO

Redes de creencias

Las redes bayesianas, también llamadas **redes de creencias**, proveen una manera de usar el cálculo de probabilidades para el manejo de la incertidumbre en nuestras representaciones de conocimiento. Estas redes proveen mecanismos eficientes para manejar las dependencias probabilísticas, mientras explotan las independencias; resultando en un a representación natural de la **causalidad**. En lo que sigue, revisaremos como se usan las dependencias e independencias entre variables en una red bayesiana e implementaremos un método para computar las **probabilidades condicionales** en los modelos de redes bayesianas.

Causalidad

Probabilidades condicionales

7.7.1 Probabilidades, creencias y redes Bayesianas

¿Cómo podemos manejar la incertidumbre correctamente, con principios bien fundamentados y pragmatismo? Como comentamos en el capítulo precedente sobre sistemas expertos (Sección 7.6), conseguir pragmatismo y rigor matemático es una meta difícil de alcanzar, pero las redes Bayesianas ofrecen una buena solución en esa dirección.

En el resto del capítulo asumiremos que el medio ambiente puede representarse por medio de un vector de **variables** que pueden tomar valores aleatoriamente de un dominio, es decir, un conjunto de valores posibles. En los ejemplos que introduciremos, asumiremos que nuestras variables tienen un **dominio booleano**, es decir, podrán tener como valor falso o verdadero.

Variables

Dominio booleano

Ejemplo 7.9. *`ladron` y `alarma` son dos variables de este tipo. La variable `alarma` es verdadera cuando la alarma suena, y la variable `ladron` es verdadera cuando un ladrón ha entrado en casa. En cualquier otro caso, ambas variables son falsas.*

De esta forma, el estado del medio ambiente en un momento dado, se puede especificar completamente computando el valor de las variables de su representación

en ese momento. Observen que nuestro supuesto de variables booleanas no constituye una limitación significativa. De hecho, al terminar el capítulo será evidente como contender dominios multi-valor.

Cuando las variables son booleanas, es normal pensar en ellas como si representasen **eventos**.

Eventos

Ejemplo 7.10. *Cuando la variable* `alarma` *se vuelve verdadera, el evento alarma sonando sucedió.*

Un agente, natural o artificial, normalmente no está completamente seguro de cuando estos eventos son verdaderos o falsos. Normalmente, el agente razona acerca de la **probabilidad** de que la variable en cuestión sea verdadera. Las probabilidades en este caso, son usadas como una medida del grado de creencia. Este grado depende de que tanto el agente conoce su medio ambiente. De forma que tales creencias se conocen tambien como **probabilidades subjetivas**, donde subjetivo no quiere decir arbitrario, al menos no en el sentido de que no estén gobernadas por la teoría de la probabilidad.

Probabilidad

Probabilidades subjetivas

Necesitaremos un poco de notación. Sean X e Y dos proposiciones, entonces, como de costumbre:

- $X \wedge Y$ es la conjunción de X e Y.
- $X \vee Y$ es la disyunción de X e Y.
- $\neg X$ es la negación de X.

La expresión $p(X)$ denota la probabilidad de que la proposición X sea verdadera. La expresión $p(X|Y)$ denota la probabilidad condicional de que la proposición X sea verdadera, dado que la proposición Y lo es.

Un **meta** típica es este contexto toma esta forma: Dado que los valores de ciertas variables han sido observados ¿Cuales son las probabilidades de que otras variables de interés tomen cierto valor específico? O, dado que ciertos eventos han sido observados ¿Cual es la probabilidad de que sucedan otros eventos de interés?

Metas

Ejemplo 7.11. *Si observamos que la alarma suena ¿Cual es la probabilidad de que un ladrón haya entrado en la casa?*

La mayor dificultad para resolver estas metas está en manejar la dependencia entre las variables del problema. Sea el caso que nuestro problema contempla n variables booleanas; entonces, necesitamos $2^n - 1$ números para definir la distribución de probabilidad completa entre los 2^n estados posibles del medio ambiente! Esto no es sólo caro computacionalmente, sino imposible, debido a que toda esa información no suele estar a la disposición del agente.

Afortunadamente, suele ser el caso que no es necesario contar con todas esas probabilidades. La distribución de probabilidades completa, no asume nada acerca de la independencia entre variables. Afortunadamente, algunas cosas son independientes de otras. Las redes bayesianas proveen una forma elegante para declarar la dependencia e independencia entre variables. La Figura 7.5, muestra una red bayesiana acerca de ladrones y sistemas de alarma. La idea es que el sensor se dispare si un ladrón ha entrado en casa, activando la alarma y llamando a la policía. Pero, un relámpago fuerte puede también puede activar el sensor, con las complicaciones correspondientes. Esta red bayesiana puede responder a preguntas del tipo: Supongamos que hace buen tiempo y escuchamos la alarma. Dados estos dos hechos ¿Cual es la probabilidad de que un ladrón haya entrado en casa?

La estructura de esta red bayesiana indica algunas dependencias probabilísticas, y también algunas independencias. Nos dice, por ejemplo, que `ladron` no es dependiente de `relampago`. Sin embargo, si se llega a saber que `alarma` es verdadera, entonces bajo las condiciones dadas, `ladron` y `relampago` ya no son independientes.

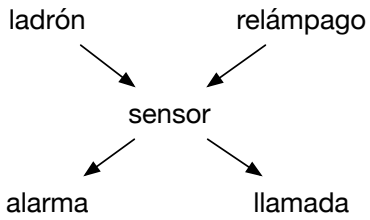

Figura 7.5: Una red bayesiana para detectar ladrones en casa.

Ejemplo 7.12. *En nuestro pequeño ejemplo, es evidente que las ligas en la red bayesiana indican causalidad. El* `ladron` *es causa de que el* `sensor` *se dispare. El* `sensor` *por su parte, es la causa de que la* `alarma` *se encienda. De forma que la estructura de la red permite razonamientos como el siguiente: Si* `alarma` *es verdadera, entonces un* `ladron` *pudo entrar en casa, puesto que esa es una de las causas de que la alarma suene. Si entonces nos enteramos de que hay una tormenta, la presencia del ladrón debería volverse menos probable. La alarma se explica por otra causa, un posible relámpago.*

En el ejemplo anterior, el razonamiento es de diagnóstico y predictivo al mismo tiempo. Al sonar la alarma, podemos diagnosticar que la causa posible es la presencia de un ladrón. Al enterarnos de la tormenta, podemos predecir que la causa real fue un relámpago.

Inferencias probabilísticas
Descendiente
Independencia

Formalicemos ahora el significado de las ligas en una red bayesiana ¿Qué clase de **inferencias probabilísticas** podemos hacer dada una red? Primero definiremos que un nodo Z es `descendiente` de X si hay un camino siguiendo las ligas desde X hasta Z. Ahora, supongamos que $Y_1, Y_2, \ldots$ son padres de X en una red bayesiana. Por definición, la red implica la siguiente relación de **independencia** probabilística: X es independiente de los nodos que no son sus descendientes, dados sus padres. De forma que, para computar la probabilidad de X, basta con tomar en cuenta los descendientes de X y sus padres $(Y_1, Y_2, \ldots)$. El efecto de las demás variables en X es acumulado através de sus padres.

Ejemplo 7.13. *Supongamos que sabemos que* `sensor` *es verdadera, y que estamos interesados en la probabilidad de* `alarma`*. Todos los demás nodos en la red no son descendientes de* `alarma`*, y dado que el valor del único padre de* `alarma` *es conocido, su probabilidad no depende de ninguna otra variable. Formalmente:*

$$p(alarma \mid ladron \wedge relampago \wedge sensor \wedge llamada) = p(alarma \mid sensor)$$

Aunque las ligas en una red bayesiana pueden interpretarse naturalmente como relaciones causales, debe observarse que éste no es necesariamente el caso.

Causa raíz

Para completar la representación de un modelo probabilístico con una red bayesiana debemos, además de especificar la estructura de la red, definir algunas probabilidades de la siguiente manera: Para los nodos que no tienen padres, conocidos como **causas raíz**, es necesario especificar sus probabilidades *a priori*. Para los demás nodos debemos especificar su probabilidad condicional, tomando en cuenta el estado de sus padres, es decir $p(X \mid$ padres de $X)$. La red bayesiana de nuestro ejemplo se puede especificar en Prolog de la siguiente manera:

```
1  %%%Red bayesiana alarma-ladron
2
3  % Estructura de la Red
4
5  padre( ladron, sensor).    % el ladrón tiende a disparar el sensor
6  padre( relampago, sensor). % un relámpago fuerte también
7  padre( sensor, alarma).
8  padre( sensor, llamada).
9
10 % Probabilidades
11
```

```
p( ladron, 0.001). % Probabilidad a priori
p( relampago, 0.02).
p( sensor, [ ladron, relampago], 0.9). % Probabilidad condicionada
p( sensor, [ ladron, not relampago], 0.9).
p( sensor, [ not ladron, relampago], 0.1).
p( sensor, [ not ladron, not relampago], 0.001).
p( alarma, [ sensor], 0.95).
p( alarma, [ not sensor], 0.001).
p( llamada, [ sensor], 0.9).
p( llamada, [ not sensor], 0.0).
```

Observen que hay dos causas raíz en la red: `ladron` y `relampago`, por lo que es necesario especificar sus probabilidades *a priori*. El nodo `sensor` tiene dos padres, `ladron` y `relampago`, por lo que sus probabilidades condicionadas, dado el estado de sus padres, forman una tabla de cuatro entradas (2^n, donde n es el número de padres). Las variables `alarma` y `llamada` sólo tienen un padre, por lo que sus tablas de probabilidad condicional tienen dos entradas. Con ello, hemos tenido necesidad de representar 10 probabilidades; pero sin información sobre la independencia entre variables, hubiésemos tenido que representar $2^5 - 1 = 31$ probabilidades. La estructura de la red y las probabilidades especificadas nos han ahorrado 21 probabilidades.

7.7.2 Cálculo de probabilidades

Recordemos algunas fórmulas del cálculo de probabilidades, que nos serán de utilidad para razonar con las redes bayesianas. Sean X e Y dos proposiciones, entonces:

- $p(\neg X) = 1 - p(X)$
- $p(X \wedge Y) = p(X)p(Y)$
- $p(X \vee Y) = p(X) + p(Y) - p(X \wedge Y)$
- $p(X) = p(X \wedge Y) + (X \wedge \neg Y) = p(Y)p(X \mid Y) + p(\neg Y)p(X \mid \neg Y)$

Las proposiciones X es Y se dice **independientes** si $p(X \mid Y) = p(X)$ y $p(Y \mid X) = p(Y)$. Esto es, si Y no afecta mi creencia sobre X y viceversa. Si X e Y son independientes, entonces:

- $p(X \wedge Y) = p(X)p(Y)$

Variables disjuntas

Se dice que las proposiciones X e Y son **disjuntas** si no pueden ser verdaderas al mismo tiempo, en cuyo caso: $p(X \wedge Y) = 0$ y $p(X \mid Y) = 0$ y $p(Y \mid X) = 0$.

Sean $X_1, \ldots, X_n$ proposiciones, entonces:

$$p(X_1 \wedge \cdots \wedge X_n) = p(X_1)p(X_2 \mid X_1)p(X_3 \mid X_1 \wedge X_2) \ldots p(X_n \mid X_1 \wedge \cdots \wedge X_{n-1})$$

Si todas las variables son independientes, esto se reduce a:

$$p(X_1 \wedge \cdots \wedge X_n) = p(X_1)p(X_2)p(X_3) \ldots p(X_n)$$

Teorema de Bayes

Finalmente, necesitaremos el **teorema de Bayes**:

$$p(X \mid Y) = p(X)\frac{p(Y \mid X)}{p(Y)}$$

que se sigue de la regla de la conjunción definida previamente. El teorema es útil para razonar sobre causas y efectos. Consideremos que un ladrón es una causa de que la alarma se encienda, es natural razonar en términos de que proporción de ladrones disparan alarmas, es decir $p(alarma \mid ladron)$. Pero cuando oímos la alarma, estamos interesados en saber la probabilidad de su causa, es decir $p(ladron \mid alarma)$. Aquí es donde entra el teorema de Bayes en nuestra ayuda:

$$p(ladron \mid alarma) = p(ladron)\frac{p(alarma \mid ladron)}{p(alarma)}$$

Una variante del teorema de Bayes, toma en cuenta el **conocimiento previo** B. Esto nos permite razonar acerca de la probabilidad de una hipótesis H, dada la evidencia E, bajo el supuesto del conocimiento previo B:

Conocimiento previo

$$p(H \mid E \wedge B) = p(H \mid B)\frac{p(E \mid H \wedge B)}{p(E \mid B)}$$

7.7.3 Implementación

En esta sección implementaremos un programa que compute las probabilidades condicionales de una red bayesiana. Dada una red, queremos que este intérprete responda a preguntas del estilo de: ¿Cual es la probabilidad de una proposición dada, asumiendo que otras proposiciones son el caso?

Ejemplo 7.14. *Algunas preguntas al intérprete de razonamiento Bayesiano, podrían ser:*

- $p(ladron \mid alarma) = ?$
- $p(ladron \wedge relampago) = ?$
- $p(ladron \mid alarma \wedge \neg relampago) = ?$
- $p(alarma \wedge \neg llamada \mid ladron) = ?$

El intérprete computará la respuesta a cualquiera de estas metas, aplicando recursivamente las siguientes reglas:

1. Probabilidad de una conjunción:

$$p(X_1 \wedge X_2 \mid Cond) = p(X_1 \mid Cond) \times p(X_2 \mid X_1 \wedge Cond)$$

2. Probabilidad de un evento cierto:

$$p(X \mid Y_1 \wedge \cdots \wedge X \wedge \ldots) = 1$$

3. Probabilidad de un evento imposible:

$$p(X \mid Y_1 \wedge \cdots \wedge \neg X \wedge \ldots) = 0$$

4. Probabilidad de una negación:

$$p(\neg X \mid Cond) = 1 - p(X \mid Cond)$$

5. Si la condición involucra un descendiente de X, usamos el teorema de Bayes. Si Y es un descendiente de X en la red, entonces:

$$p(X \mid Y \wedge Cond) = p(X \mid Cond)\frac{p(Y \mid X \wedge Cond)}{p(Y \mid Cond)}$$

6. Si la condición no involucra a ningún descendiente de X, puede haber dos casos:
   a) Si X no tiene padres, entonces $p(X \mid Cond) = p(X)$, donde $p(X)$ está especificada en la red.
   b) Si X tiene padres, entonces:

$$p(X \mid Cond) = \sum_{S \in estadosPosibles(Padres)} p(X \mid S)p(S \mid Cond)$$

Ejemplo 7.15. *¿Cual es la probabilidad de un ladrón dado que sonó la alarma?* $p(ladron \mid alarma) = ?$. *Primero, por la regla 5:*

$$p(ladron \mid alarma) = p(ladron)\frac{p(alarma \mid ladron)}{p(alarma)}$$

y por la regla 6:

$$p(alarma \mid ladron) = p(alarma)p(sensor \mid ladron) + p(alarma \mid \neg sensor \mid ladron)$$

y por la misma regla 6:

$$\begin{aligned} p(sensor \mid ladron) \quad = \quad & p(sensor \mid ladron \wedge relampago)p(ladron \wedge relampago \mid ladron) + \\ & p(sensor \mid \neg ladron \wedge relampago)p(\neg ladron \wedge relampago \mid ladron) + \\ & p(sensor \mid ladron \wedge \neg relampago)p(ladron \wedge \neg relampago \mid ladron) + \\ & p(sensor \mid \neg ladron \wedge \neg relampago)p(\neg ladron \wedge \neg relampago \mid ladron) \end{aligned}$$

Aplicando las reglas 1,2,3 y 4 como corresponde, esto se reduce a:

$$\begin{aligned} p(sensor \mid ladron) \quad &= \quad 0{,}9 \times 0{,}02 + 0 + 0{,}9 \times 0{,}98 + 0 = 0{,}9 \\ p(alarma \mid ladron) \quad &= \quad 0{,}95 \times 0{,}9 + 0{,}001 \times (1 - 0{,}9) = 0{,}8551 \end{aligned}$$

Usando las reglas 1,4 y 6 obtenemos:

$$p(alarma) = 0{,}00467929$$

Finalmente

$$p(ladron \mid alarma) = 0{,}001 \times 0{,}8551 / 0{,}00467929 = 0{,}182741$$

El mecanismo de inferencia se implementa como sigue:

```
% Motor de inferencia para Red Bayesiana
:-  op(  900, fy, not).

prob( [X | Xs], Cond, P) :-  !,   % Prob de la conjunción
  prob( X, Cond, Px),
  prob( Xs, [X | Cond], PRest),
  P is Px * PRest.

prob( [], _, 1) :-  !.            % Conjunción vacía

prob( X, Cond, 1) :-
  miembro( X, Cond), !.            % Cond implica X

prob( X, Cond, 0) :-
  miembro( not X, Cond), !.        % Cond implica X es falsa

prob( not X, Cond, P) :-  !,      % Negación
  prob( X, Cond, P0),
  P is 1 - P0.

% Usa la regla de Bayes si Cond0 incluye un descendiente de X

prob( X, Cond0, P) :-
  borrar( Y, Cond0, Cond),
  pred( X, Y), !,           % Y es un descendiente de X
  prob( X, Cond, Px),
  prob( Y, [X | Cond], PyDadoX),
  prob( Y, Cond, Py),
  P is Px * PyDadoX / Py.         % Asumiendo Py > 0

% Casos donde Cond no involuchra a un descendiente

prob( X, _, P) :-
```

```
  p( X, P), !.                    % X una causa raíz - prob dada

prob( X, Cond, P) :-  !,
  findall( (Condi,Pi), p(X,Condi,Pi), CPlist),  % Conds padres
  suma_probs( CPlist, Cond, P).

% suma_probs( CondsProbs, Cond, SumaPond)
%   CondsProbs es una lista de conds y sus probs
%   SumaPond suma de probs de Conds dada0 Cond

suma_probs( [], _, 0).

suma_probs( [ (Cond1,P1) | CondsProbs], Cond, P) :-
  prob( Cond1, Cond, PC1),
  suma_probs( CondsProbs, Cond, PResto),
  P is P1 * PC1 + PResto.

% pred(var1,var2). var1 es predecesora de var2 en la red.

pred( X, not Y) :- !,         % Y negada
  pred( X, Y).

pred( X, Y) :-
  padre( X, Y).

pred( X, Z) :-
  padre( X, Y),
  pred( Y, Z).

% miembro(X,L). X es miembro de L.

miembro( X, [X | _]).

miembro( X, [_ | L]) :-
  miembro( X, L).
```

De forma que las siguientes consultas son posibles:

```
?- prob(ladron,[alarma],P).
P = 0.182741321476588.

?- prob(alarma,[],P).
P = 0.00467929198.

?- prob(ladron,[llamada],P).
P = 0.23213705371651422.

?- prob(ladron,[llamada,relampago],P).
P = 0.008928571428571428.

?- prob(ladron,[llamada,not relampago],P).
P = 0.47393364928909953.
```

Aunque nuestra implementación es corta y concisa, es poco eficiente. No al grado de ser un problema, para redes pequeñas como las del ejemplo, peros si para redes más grandes. El problema es, a grandes rasgos, que la complejidad el algoritmo crece exponencialmente con respecto al número de padres de un nodo en la red. Esto se debe a la sumatoria sobre todos los estados posibles de los padres de un nodo.

7.8 REDES SEMÁNTICAS Y MARCOS

Representaciones estructuradas

En esta sección estudiaremos otros dos formalismos de representación de conocimiento, ampliamente usados en Inteligencia Artificial. Ambos difieren de las reglas de producción, en el hecho de que están diseñados para representar, de manera **estructurada**, grandes cantidades de hechos. Este conjunto de hechos es normalmente

compactado: Se obvian algunos hechos, cuando estos pueden ser inferidos. Ambos formalismos, redes semánticas y marcos, utilizan la **herencia**, de manera similar a como se utiliza en los lenguajes de programación orientados a objetos. *Herencia*

Ambos formalismos pueden implementarse fácilmente en Prolog, básicamente adoptando de manera disciplinada, una forma particular de estilo y organización de nuestros programas lógicos.

7.8.1 Redes semánticas

Una red semántica está compuesta por entidades y relaciones entre ellas. Normalmente, una red semántica se representa como un grafo. Existen varios tipos de redes, que siguen diferentes convenciones, pero normalmente, los **nodos** de la red representan entidades; mientras que las relaciones se denotan por medio de **ligas** etiquetadas. La figura 7.6 muestra un ejemplo de red semántica. *Nodos* *Ligas*

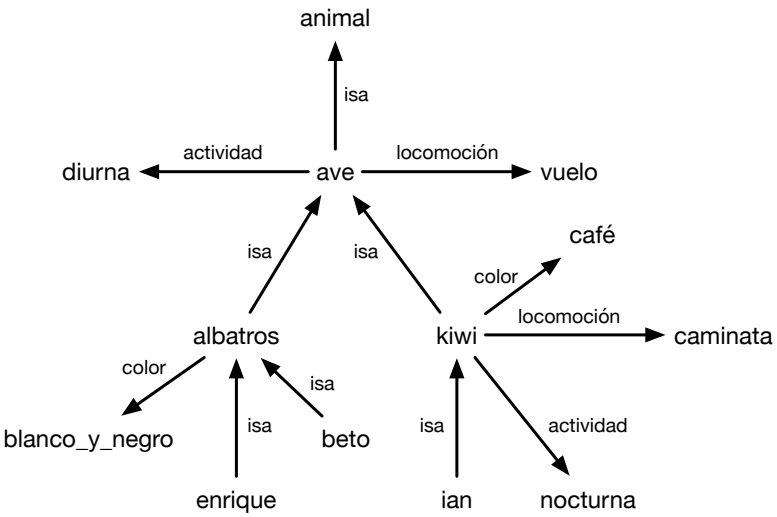

Figura 7.6: Una red semántica.

La relación especial `isa` denota la relación "es un". La red del ejemplo representa los siguientes hechos:

- Un ave es un tipo de animal.
- El vuelo es el medio de locomoción de las aves.
- Un albatros es un ave.
- Enrique y Beto son albatros.
- Un kiwi es un ave.
- La caminata es el medio de locomoción de un kiwi.
- etc.

Observen que algunas veces `isa` relaciona una clase con una super-clase, por ejemplo, `ave` con `animal`; y otras relaciona un miembro con su clase, por ejemplo: `enrique` con `albatros`. La red se puede implementar directamente en Prolog:

```
1 % Red semántica
2
3 isa(ave,animal).
4 isa(albatros,ave).
5 isa(kiwi,ave).
6 isa(enrique,albatros).
```

```
isa(beto,albatros).
isa(ian,kiwi).

actividad(ave,diurna).
actividad(kiwi,nocturna).

color(albatros,blanco_y_negro).
color(kiwi,cafe).

locomocion(ave,vuelo).
locomocion(kiwi,caminata).
```

Para poder inferir propiedades haciendo uso del mecanismo de herencia, definimos un predicado `fact/1`, que explore las propiedades de la super clase de una entidad, cuando ésta no pueda establecerse directamente. El predicado es como sigue:

```
% Interprete Redes Semánticas

fact(Hecho) :-
    Hecho,!.

fact(Hecho) :-
    Hecho =.. [Rel,Arg1,Arg2],
    isa(Arg1,SuperClaseArg1),
    SuperHecho =.. [Rel,SuperClaseArg1,Arg2],
    fact(SuperHecho).
```

Finalmente, definimos una interfaz `seRedSemantica.pl` para llamar nuestro método de inferencia basado en herencia y la base de conocimiento basada en redes semánticas:

```
% Sistema Experto basado en Frames

:- [inferenciaHerencia].
:- [kbRedSemantica].
```

Una consulta a este sistemas, es como sigue:

```
?- [seRedSemantica].
true.

?- fact(locomocion(ian,Loc)).
Loc = caminata.

?- fact(locomocion(enrique,Loc)).
Loc = vuelo.
```

7.8.2 Marcos

La representación de conocimiento basada en marcos, puede verse como un predecesor de la programación orientada a objetos. En este tipo de representación, los hechos se agrupan en objetos. Por objeto, entendemos una entidad física concreta o abstracta. Un **marco** (*frame*) es una estructura de datos, cuyos componentes se llaman **ranuras** (*slots*). Las ranuras pueden guardar información de diferentes tipos, por ejemplo:

Marco
Ranuras

- Valores.
- Referencias a otros marcos.
- Procedimientos para computar valores.

Es posible dejar vacía una ranura, en cuyo caso, ésta obtendrá su valor por medio de una inferencia. Como en el caso de las redes semánticas, la inferencia incluye herencia: Cuando un marco representa una clase de objetos, como `albatros`; y otra

representa una super clase, como ave, el marco clase hereda los valores del marco super clase.

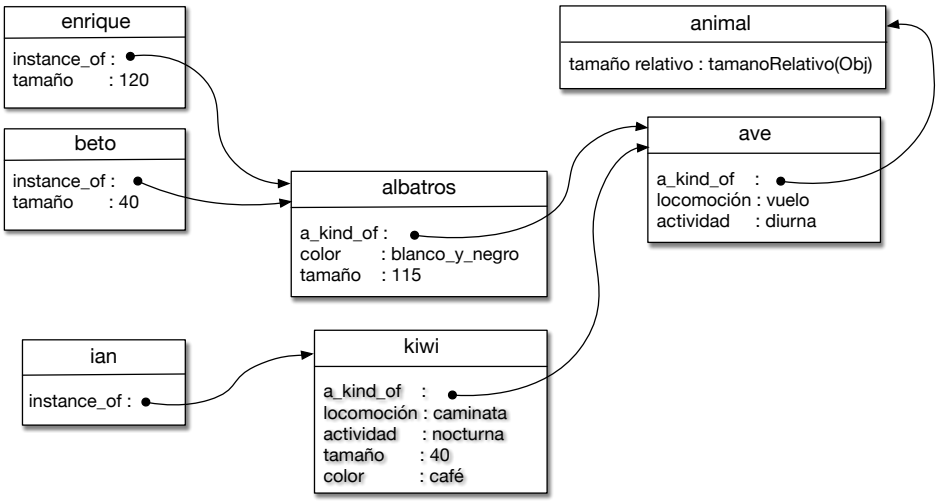

Figura 7.7: Una base de conocimiento basada en marcos. Los rectángulos son marcos y sus componentes son ranuras. que pueden contener valores, ligas a otros marcos y funciones.

Gráficamente, nuestra base de conocimientos basada en marcos, se muestra en la figura 7.7. Su implementación en Prolog es la siguiente:

```
% Base de conocimiento basada en Marcos
% Los hechos se representan como marco(ranura,valor).
% El valor puede ser un procedimiento para su cálculo.

% Marco ave: un ave prototípica

ave(a_kind_of,animal).
ave(locomocion,vuelo).
ave(actividad,diurna).

% Marco albatros: un ave prototípica con hechos extra:
% es blanco y negro; y mide 115cm

albatros(a_kind_of,ave).
albatros(color,blanco_y_negro).
albatros(tamano,115).

% Marco kiwi: un ave prototípica con hechos extra:
% camina, es nocturno, café y mide unos 40cm

kiwi(a_kind_of,ave).
kiwi(locomocion,caminata).
kiwi(actividad,nocturna).
kiwi(tamano,40).
kiwi(color,cafe).

% Marco enrique: es un albatros

enrique(instance_of,albatros).
enrique(tamano,120).

% Marco beto: es un albatros bebé

beto(instance_of,albatros).
beto(tamano,40).

% Marco ian: es un kiwi.

ian(instance_of,kiwi).

```

```
% Marco animal: su tamaño relativo se calcula ejecutando un
% procedimiento

animal(tamanoRelativo,
       execute(tamanoRelativo(Obj,Val), Obj, Val)).

% tamanoRelativo(Obj,TamRel): El tamaño relativo TamRel de Obj

tamanoRelativo(Obj,TamRel) :-
    value(Obj,tamano,TamObj),
    value(Obj,instance_of,ClaseObj),
    value(ClaseObj,tamano,TamClase),
    TamRel is (TamObj/TamClase) * 100.
```

Para usar esta base de conocimientos, es necesario definir un procedimiento para consultar los valores de las ranuras. Tal procedimiento se define como `value/3` en el motor de inferencia correspondiente:

```
% Motor de inferencia para Marcos

value(Frame,Slot,Value) :-
    Query =.. [Frame,Slot,Value],
    Query, !. % valor encontrado directamente.

value(Frame,Slot,Value) :-
    parent(Frame,ParentFrame), % Un marco más general
    value(ParentFrame,Slot,Value).


parent(Frame,ParentFrame) :-
(    Query =.. [Frame, a_kind_of, ParentFrame]
    ;
    Query =.. [Frame, instance_of, ParentFrame]
)    ,
    Query.
```

Si la relación `value(Frame,Slot,Value)` se satisface, debe ser el caso que la relación $Frame(Slot, Value)$ esté incluída en nuestra base de conocimientos. De otra forma, el valor de esa ranura deberá obtenerse recurriendo a la herencia. El predicado `parent/2` busca si hay un padre del marco que recibe como primer argument. Este predicado se satisface si el marco en cuestión es una sub clase o un caso de otro marco. Esto es suficiente para consultas del tipo:

```
?- value(enrique,actividad,Act).
Act = diurna

?- value(kiwi,actividad,Act).
Act = nocturna.
```

Consideremos ahora un caso más complicado de inferencia, cuando el valor de una ranura se obtiene computando una función. Por ejemplo, el marco `animal` tiene la ranura correspondiente a tamaño relativo, que se computa con la función `tamanoRelativo/2`. Su computo es un radio, expresado como porcentaje, entre el tamaño particular de un espécimen y el de la especie en general. Por ejemplo, el tamaño de relativo de `beto` sería $(40 \div 115) \times 100 = 34{,}78\,\%$. Para ello, es necesario utilizar la herencia y detectar cuando un valor se puede computar directamente o mediante el llamado a una función. El motor de inferencia para esto es como sigue:

```
% Motor inferencial para marcos V2: Incluye funciones

value(Frame,Slot,Value) :-
    value(Frame, Frame, Slot, Value).

value(Frame, SuperFrame, Slot, Value) :-
    Query =.. [SuperFrame, Slot, ValAux],
    Query,
    process(ValAux, Frame, Value), !.

```

```
value(Frame, SuperFrame, Slot, Value) :-
    parent(SuperFrame, ParentSuperFrame),
    value(Frame, ParentSuperFrame, Slot, Value).

parent(Frame,ParentFrame) :-
(    Query =.. [Frame, a_kind_of, ParentFrame]
    ;
    Query =.. [Frame, instance_of, ParentFrame]
)    ,
    Query.

process(execute(Proc, Frame, Value), Frame, Value) :- !,
    Proc.

process(Value,_,Value).  % un valor, no un procedimiento.
```

Ahora podemos realizar las siguientes consultas:

```
?- value(enrique,tamanoRelativo,Tam).
Tam = 104.34782608695652

?- value(beto,tamanoRelativo,Tam).
Tam = 34.78260869565217

?- value(ian,tamanoRelativo,Tam).
Tam = 100

?- value(beto,color,C).
C = blanco_y_negro.
```

La base de conocimientos basada en marcos y su motor de inferencia, incluyendo funciones, se puede llamar con el *script* `seMarcos.pl`:

```
% Sistema Experto basado en marcos.

%:- [inferenciaMarcos].
:- [inferenciaMarcosFunc].
:- [kbMarcos].
```

si se desea usar la versión sin funciones, basta cambiar los comentarios en el *script*. Evidentemente, el predicado `value/3` puede usarse en las reglas de producción definidas al principio de este capítulo.

7.9 LECTURAS Y EJERCICIOS SUGERIDOS

El contenido de este capítulo está basado en el libro de Bratko [19] (cap. 15), haciendo énfasis en el uso de la programación lógica para implementar los mecanismos usados en los sistemas expertos: bases de conocimiento, motores de inferencia, heurísticas, e interfaz. El libro de Negrete-Martínez, González-Pérez y Guerra-Hernández [84] hace una revisión en el mismo sentido, pero basada en la programación funcional. La revisión de los componentes de un sistema experto, es más profunda en este texto. Un ejercicio interesante, sería programar los mecanismos propuestos en el "Pericia Artificial" usando Prolog. El libro provee además una revisión exhaustiva de las herramientas y aplicaciones de los sistemas expertos. Los ejercicios de este capítulo se complementan con el uso de `clips` [3] un *shell* de sistemas expertos, producido por la NASA. Los capítulos 9 y 10 de la revisión de técnicas avanzadas de Prolog de Covington, Nute y Avellino [32], presentan otra implementación de un `shell` de sistema experto y su manejador de incertidumbre. Un texto obligado en este tema es la revisión de los experimentos entorno a MYCIN, propuesto por Buchanan y Shortliffe [23]. Erman y col. [43] discuten la integración del conocimiento y el manejo de la incertidumbre en Hersay-II, un sistema para el

3 http://clipsrules.sourceforge.net

entendimiento del discurso hablado. Ligeza [75] ofrece una revisión exhaustiva de los fundamentos lógicos de los sistemas basados en reglas.

Nuestro colega Sucar [116], Premio Nacional de Ciencia 2016, nos ofrece una introducción generalista a los modelos gráficos probabilísticos. Bolstad [8] nos presenta una introducción a la estadística bayesiana, donde el énfasis es más matemático y menos computacional. Un complemento ideal para este capítulo es el artículo de Poole [100], donde se aborda la integración de la lógica y la teoría de decisión bayesiana, desde una perspectiva basada en la representación del conocimiento. Recientemente, Pearl, Glymour y Jewell [94] nos ofrecen un libro sobre la estadística bayesiana, con énfasis en la inferencia causal.

Ejercicios

Ejercicio 7.1. *Implemente un mecanismo de razonamiento como el sugerido en la figura 7.4.*

Ejercicio 7.2. *Implemente los ejemplos de este capítulo usando* `clips`*.*

Ejercicio 7.3. *Use las técnicas bayesianas aquí presentadas, como alternativa al sistema experto que contiende con incertidumbre mediante factores de certeza.*