

Selección de Comportamientos básicos de Conducción usando MDP-ProbLog y Webots¹

Introducción a Prolog y ProbLog

Héctor Avilés, Karina Arévalo

havilesa@upv.edu.mx

Universidad Politécnica de Victoria

Escuela de Invierno de Robótica 2022-2023

11-12 de enero del 2023

¹En trabajo conjunto con Marco Negrete (FI-UNAM), Rubén Machucho (UPV), Alberto Reyes (IEE)

Introducción

- Prolog² es uno de los lenguajes **declarativos**³ más utilizados en IA y tiene aplicaciones como:
 - Procesamiento del lenguaje natural
 - Construcción de sistemas expertos
 - Planeación de secuencias de acciones
 - Prueba automática de teoremas

²Programation en *Logique*, que en español es Programación Lógica

³En los lenguajes declarativos se “declara” *cuál* es el problema a resolver y no *cómo* se resuelve como en los lenguajes *imperativos/procedurales* (eg, C++ ó Python)


Instalación

- El intérprete para programación a usar es SWI-prolog (<https://www.swi-prolog.org/>) y su instalación en Ubuntu 20.04⁴ puede hacerse como sigue:
 - `sudo apt-add-repository ppa:swi-prolog/stable`
 - `sudo apt-get update`
 - `sudo apt-get install swi-prolog`

Un código en Prolog se puede cargar con `'swipl -s nomarch.pl'`^{5,6}

⁴Una versión en línea con tutorial y ejemplos es: <https://swish.swi-prolog.org/>

⁵El intérprete se puede ejecutar sin parámetros con `'swipl'`

⁶Dentro de la línea de comandos se puede escribir también `'?- [nomarc].'` ó `'?- [nomarc.pl].'`, siempre que se ejecute Prolog en el mismo subdirectorio que el archivo 

Componentes léxicos

- Componentes léxicos básicos de los programas en Prolog son:
 - **Número**: Secuencia de dígitos (posiblemente un número negativo que inicia con '-') y decimales
 - **Variable**: Cadena de caracteres que inicia con una letra mayúscula seguidos de guión bajo, números ó letras (X, X_123, Mi_variable); ó un guión bajo seguido o no de números y letras⁷ (_Y, _var, '_')
 - **Átomo**: Cadena de caracteres que inicia con una letra minúscula (amigo, juan, perro, x_abc, 'Hola mundo') seguido por 0 ó más argumentos⁸ entre paréntesis⁹ (ladra(perro), es_hermano(juan, maria))

⁷La variable '_' es una **variable anónima**

⁸ie, que a su vez son términos como átomos, números, variables

⁹No debe haber espacios en blanco entre el nombre del término y el primer paréntesis

Aridad de los átomos

- Los átomos¹⁰ representan propiedades o relaciones siendo las aridades 0, 1, 2 y 3 más comunes:
 - **Aridad 0:** Son proposiciones (eg, p : "Hoy llueve")
 - **Aridad 1:** Indican propiedades o características (eg, $\text{humano}(X)$: "X es humano")
 - **Aridad 2:** Describen una relación dos elementos (eg, $\text{padre}(X, Y)$: "X es padre de Y")

¹⁰Conocidos como predicados en lógica de predicados

Aridad de los átomos

- **Aridad 3:** Describen una acción o posición entre 3 elementos:
 - $\text{presenta}(X, Y, Z)$: “El sujeto X presenta a Y con Z”
 - $\text{enmedio}(X, Y, Z)$: “El punto X se ubica entre los puntos Y y Z”
 - $\text{obsequia}(X, Y, Z)$: “La persona X le obsequia Y a Z”
 - $\text{producto}(X, Y, Z)$: $X \times Y = Z$
- **Aridad 4:** Es una aridad posible (eg, $\text{regala}(X_1, X_2, X_3, X_4)$ son posibles ó “La persona X_1 regaló X_2 a X_3 en el lugar X_4 ”)

Tipos de cláusulas

- Un **programa lógico normal** está compuesto por **cláusulas** de la forma:

$$\underbrace{a}_{\text{cabeza}} \text{ :- } \underbrace{b_1, \dots, b_m, \backslash+ c_1, \dots, \backslash+ c_n}_{\text{cuerpo}}.$$

donde $m \geq 0, n \geq 0$, a, b_1, \dots, b_m son átomos (literales positivas) y $\backslash+ c_1, \dots, \backslash+ c_n$ son átomos negados (literales negativas) pudiendo ambos involucrar variables¹¹

¹¹A cada literal positiva o negativa en el cuerpo se le llama **sub-objetivo**

Tipos de cláusulas

- Las cláusulas pueden ser hechos, reglas y consultas:
 - Un **hecho**¹² es una declaración (afirmación) sobre lo que es verdad en el mundo
 - Una **regla**¹³ funciona como sentencia condicional para describir cómo se “comporta” el mundo
 - La **consulta**¹⁴ (ya sea un objetivo ó una pregunta¹⁵) es la conclusión que se quiere comprobar de hechos y reglas

¹²Una cláusula sin cuerpo, ie, $m = 0$ y $n = 0$, eg., ‘persona(maria).’, ‘hermano(luis, ana).’

¹³También llamada **cláusula predicativa** con $m > 0$ ó $n > 0$; para **satisfacer** la cabeza se deben satisfacer a cada átomo o **sub-objetivo** en el cuerpo (ya sean hechos o cabeceras de otras reglas)

¹⁴Es una cláusula sin cabeza

¹⁵El objetivo sólo requiere de verificar su veracidad y una pregunta puede tener variables como parámetros que deben ser instanciados

Definición de predicados

- La definición de un **predicado** es una o más cláusulas que comparten un mismo **functor** (nombre/aridad), eg:

hijo_de(pepe,luis).

hijo_de(pepe,maria).

donde el functor de 'hijo_de(pepe,luis)' y 'hijo_de(pepe,maria)' es 'hijo_de/2'

Verificación de átomos, números, variables y términos compuestos

- Verificación de términos en el *prompt* de SWI-prolog:

Archivo “comandos.script” (Sección 1)

- Predicados de utilidad en el *prompt* de SWI-prolog:

Archivo “comandos.script” (Sección 2)

Ejemplo de un programa en Prolog

- Ejemplo de la organización de hechos, reglas y consultas:

Archivo “nilss.pl”

Ejemplos de aritmética

- Ejemplos de predicados de la aritmética:

Archivo “**aritmetica.pl**”
(cargar con `['aritmetica.pl']`.)

Más funciones en:

<https://www.swi-prolog.org/pldoc/man?section=functions>)

Unificación

- La **unificación** es un procedimiento que busca hacer coincidir¹⁶ dos términos¹⁷; el resultado es una **sustitución**, eg, de una variable por una constante

Archivo “unificacion.script”

¹⁶Dos términos coinciden si son iguales o si hay variables involucradas éstas pueden igualarse mediante la instanciación

¹⁷Sirve también como paso de parámetros, recolectar datos (eg, de los hechos) y para construir datos (eg, para listas de elementos)

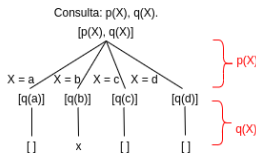
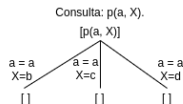
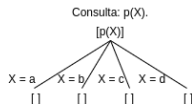
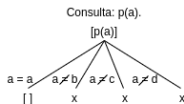
- El **retroceso** (*backtracking*) es el mecanismo de Prolog para recuperar las soluciones alternativas de una consulta¹⁸ a través de los **puntos de elección**¹⁹ verificando unificación de arriba hacia abajo y de izquierda a derecha

Ver diapositiva Árboles de búsqueda y archivo
“retroceso.pl”

¹⁸Prolog sigue la suposición de mundo cerrado

¹⁹(*Choice points* en inglés) son los sub-objetivos cuyas variables pueden ser instanciadas con diferentes valores

Árboles de búsqueda

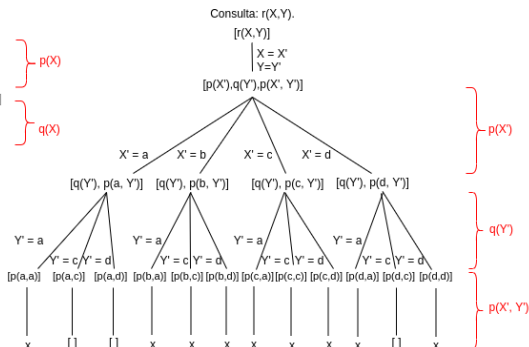


Código en Prolog:

```

p(a).
p(b).
p(c).
p(d).
q(a).
q(c).
q(d).
p(a, b).
p(a, c).
p(a, d).
p(d, c).
  
```

$r(X, Y) :- p(X), q(Y), p(X, Y).$



Árboles de búsqueda para la satisfacción de diferentes consultas (**búsqueda primero en profundidad** con retroceso; una trayectoria hasta un punto de elección implica conjunción y la bifurcación implica disyunción). SWI-Prolog utiliza indexación para la unificación, por lo que las comparaciones que no unifican son sólo ilustrativas.

Corte ‘!’

- Al ejecutarse el predicado corte (*cut*) ‘!’ se previene el retroceso a los sub-objetivos entre el inicio del cuerpo de la regla que llama al ‘!’ con lo cual se previene la búsqueda de más soluciones²⁰

Ver diapositiva Árbol de búsqueda para ‘!’ y archivo
“corte.pl”

²⁰ie, gráficamente, “poda” las bifurcaciones abiertas siguientes en los puntos de elección del árbol de búsqueda

Árbol de búsqueda para '!'

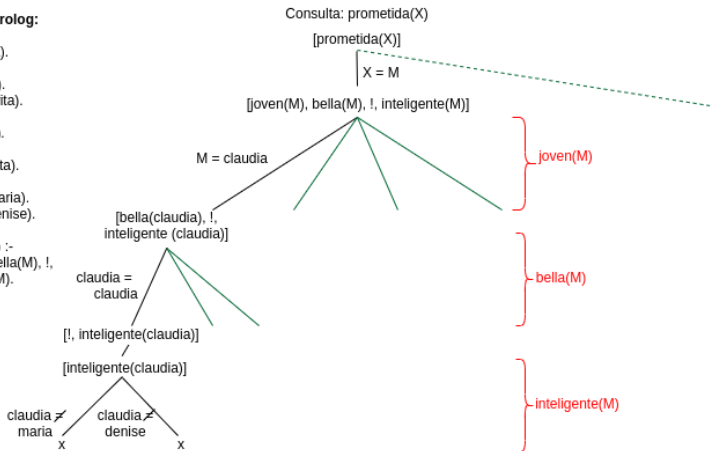
Código en Prolog:

```
joven(claudia).  
joven(maria).  
joven(denise).  
joven(margarita).
```

```
bella(claudia).  
bella(maria).  
bella(margarita).
```

```
inteligente(maria).  
inteligente(denise).
```

```
prometida(M) :-  
  joven(M), bella(M), !,  
  inteligente(M).
```



Árbol de búsqueda para el corte '!'. Las ramas en verde no se recorren (incluyendo la línea punteada que representa una segunda regla para la pregunta y que puede o no existir)

Ejemplo modificado ligeramente sin permiso de: Ulle Endriss. Lecture Notes: An Introduction to Prolog Programming. Institute for Logic, Language and Computation. University of Amsterdam. 2014.

Negación por falla

- La **negación por falla**^{21,22} es un mecanismo para probar en un número finito de pasos si la negación de un sub-objetivo en una regla es V ²³:

$\backslash + p(X)$ se evalúa como la cláusula ' $p(X), !, \text{false}$ '.

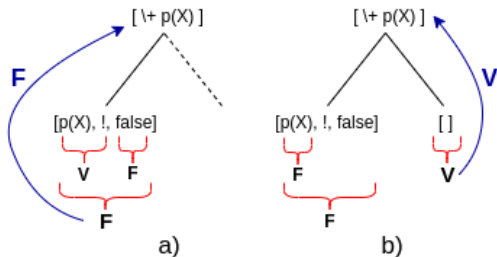
Diapositiva de Árbol de negación por falla y archivo
“negacion.pl”

²¹ Algunas veces son necesarias expresiones del tipo “No es cierto que ocurre p ”

²² Las expresiones $\backslash + p$ ó $\text{not}(p)$ son equivalentes

²³ En esencia, se intenta probar que su versión afirmada es verdadera y si se satisface de acuerdo al programa, entonces su negación es falsa, de lo contrario, la negación es verdadera

Árbol de negación por falla



El árbol de la negación por falla genera 2 ramas: a) si se comprueba que $p(X)$ es V entonces se ejecutan '!' (que poda la segunda rama) y 'false' con lo que ' $\backslash + p(X)$ ' evalúa a F, b) si $p(X)$ es F entonces la segunda rama evalúa a V al igual que ' $\backslash + p(X)$ ', de ahí el nombre negación por (la) falla (al intentar demostrar que $p(X)$ es verdad)²⁴

(Gráfica modificada sin permiso de: Drawing Prolog Search Trees: Johan Bos. A Manual for Teachers and Students of Logic Programming. University of Groningen. 2020. Disponible en: <https://www.arxiv-vanity.com/papers/2001.08133/>)

²⁴En el inciso b) si la consulta es del tipo $[p(X), q(X)]$ entonces $q(X)$ sustituye a $[]$ en la segunda rama

- Una lista es una secuencia finita de términos de Prolog, eg:
 - [manzana, pera, melon, sandia]
 - [1,a,2,[],b,3,c, padre(juan, X)]
- Las listas pueden dividirse en [Head|Tail] donde Head es el primer elemento y Tail es la lista de los elementos siguientes²⁵

Archivos “listas.script”

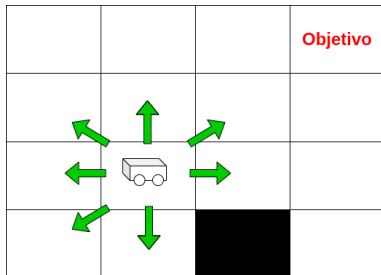
²⁵eg, en [1,3,5]=[Head|Tail], Head = 1 y Tail = [3,5]

- La recursión es un procedimiento donde una función se llama a si misma (llamada recursiva) para realizar alguna acción hasta que se cumpla una condición de paro (el caso base)

Archivo “recursion.pl”

Ejemplo

- Suponga que un robot debe desplazarse a través de un piso cuadriculado como se muestra en la imagen siguiente:



y para moverse desde un punto inicial a un punto final debe calcular las secuencias posibles de movimientos en el plano

Archivo "robot_path.pl"

¿Preguntas?

- ProbLog modela distribuciones de probabilidad discreta multi-variable usando programas lógicos similares a Prolog²⁶ extendidos con valores de probabilidad:
 - Esta revisión se centrará en la codificación de algunos modelos probabilísticos “estáticos” y “dinámicos” y cálculo de probabilidades de la evidencia $P(\text{Evidencia})$ y condicionales $P(\text{Consulta}|\text{Evidencia})$

²⁶Basado en el lenguaje Prolog, aunque no soporta algunas características como ‘!’, ver la lista: <https://problog.readthedocs.io/en/latest/prolog.html>

Introducción

- Los autores de ProbLog2 (implementación reciente) argumentan que es la única alternativa que incorpora:
 - Disyunciones anotadas para variables discretas de más de dos valores, eg, '0.3::p(1);0.45::p(2);0.25::p(3)' (sólo un valor verdadero a la vez) y reglas disyuntivas '0.3::a ; 0.7::b :- c, d, e.' (sólo una literal de la cabeza verdadera a la vez)
 - Cuerpos de reglas no mutuamente excluyentes (eg, 'a:-b. a:- c.') tal que $P(b, c) \neq 0$
 - Reglas cíclicas o recursivas (eg, 'a(X) :- b(X,Y), a(Y).')
 - Uso de cualquier átomo como evidencia
 - Múltiples consultas (eg, 'query(p). query(q).') resueltas a la vez
 - Aprendizaje paramétrico con datos faltantes mediante una variante del EM (no cubierto en esta plática)

Instalación y documentación

- **Instalación**²⁷: `$ pip3 install ProbLog v2.2.4` (probado en Ubuntu 20.04 y Python3)
- **Versión online (v2.1)**:
<https://dtai.cs.kuleuven.be/problog/editor.html>
- **Documentación**: <https://problog.readthedocs.io/en/latest/>
- **Tutorial**: <https://dtai.cs.kuleuven.be/problog/tutorial.html>

²⁷ Github: <https://github.com/ML-KULEuven/problog>

- La ejecución de un programa en ProbLog se puede hacer por:
 - Llamada a la librería desde un programa en Python3 ('python3 python_test.py')
 - Uso del editor en el sitio web
 - Línea de comandos en una terminal: `problog nomarch.pblg`²⁸

²⁸La extensión del archivo es una elección de los expositores y no es obligatoria

Elementos básicos de ProbLog

- A los hechos y reglas de Prolog se añaden los siguientes componentes:
 - **Hechos probabilísticos**, eg, `0.5::p.`
 - **Cláusulas probabilísticas**, eg, `0.6::q :- p.`
 - **Disyunciones anotadas**, eg, `.3::p(1); 0.7::p(2).`
(ó también, `0.9::p; 0.1::q :- r.`)

donde el operador `::` es indicador del valor de probabilidad, la aridad de cada átomo es ≥ 0 y las **consultas** se realizan como **query(p).** y la **evidencia** como **evidence(q).**

Elementos básicos de ProbLog

- Un hecho ' $\alpha::p.$ ' puede verse como una VA de Bernoulli ($\{1, 0\}$) con la probabilidad de éxito $\alpha \in [0, 1]$
- Una cláusula probabilística del tipo ' $\alpha::q \text{ :- } p.$ ' es una conveniencia sintáctica para:
 $\alpha::\text{aux.}$
 $q \text{ :- } p, \text{aux.}$
donde 'aux' es un hecho "auxiliar"
- Las disyunciones anotadas emulan variables multi-valuadas discretas donde a lo más ocurre un valor a la vez (ó también disyunciones²⁹ en la cabeza de las reglas³⁰)

²⁹Posiblemente con diferentes probabilidades

³⁰Como una conveniencia sintáctica para representar que con un mismo antecedente pueden ocurrir dos o más consecuentes, uno a la vez

Elementos de un programa en ProbLog

- Más formalmente³¹, un programa en ProbLog es una tupla $PL = (A, F_p, R, C)$ donde:
 - A es un conjunto de los átomos que aparecen en PL
 - F_p es un conjunto de hechos probabilísticos aterrizados (eg., ' $\alpha_1::p(1). \alpha_2::q(1,2).$ ' donde $\alpha_1, \alpha_2 \in [0, 1]$)
 - R es un conjunto de **reglas lógicas normales** extendidas con un valor de probabilidad $\alpha \in [0, 1]$ con la forma:

$$\alpha::a \text{ :- } b_1, \dots, b_m, \backslash +c_1, \dots, \backslash +c_n$$

donde $m \geq 0, n \geq 0, a \in (A - F_p)$ (un átomo a no puede ser un hecho probabilístico) y $a, b_i, c_j \in A$ para $i = 1, \dots, m, j = 1, \dots, n$

- C es un conjunto de consultas del tipo: ' $\text{query}(p). \text{query}(q(X)).$ '

³¹Extracto de: Thiago P. Bueno, Denis D. Mauá, Leliane N. de Barros Fabio G. Cozman. Markov Decision Processes Specified by Probabilistic Logic Programming: Representation and Solution. 5th Brazilian Conference on Intelligent Systems. Págs. 337-342. 2016

Cálculo de probabilidades

- Cada posible interpretación para los hechos probabilísticos aterrizados (**elección total**) induce un programa lógico L ³² y sobre ellos se establece una distribución de probabilidad³³:

$$P(L|PL) = \prod_{f_i \in L} \alpha_i \prod_{f_i \notin (F_p - L)} (1 - \alpha_i)$$

donde $f_i \in F_p$ y además la probabilidad de una consulta $q \in A$ es

$$P(q|PL) = \sum_{L:L \models q} P(L|PL),$$

ie, la sumatoria de todos los programas lógicos L con los que q se vincula lógicamente

³²Reglas del programa original junto con sus hechos verdaderos en la interpretación

³³Lo cual se conoce como **semántica de distribución** propuesta en PRISM de T. Sato

Ejemplo

- Suponga el programa lógico probabilista *PL* en ProbLog³⁴:

0.9::r.

0.5::p:- r.

0.4::q:- r.

evidence(p).

evidence(q).

query(r).

PL original

0.9::r.

0.5::aux1.

0.4::aux2.

p:- r, aux1.

q:- r, aux2.

evidence(p).

evidence(q).

query(r).

PL sin conveniencia sintáctica

Aquí: $A = \{r, \text{aux1}, \text{aux2}, p, q\}$, $F_p = \{r, \text{aux1}, \text{aux2}\}$ y $R = \{p:- r, \text{aux1}, q:- r, \text{aux2}\}$. Las cláusulas para este programa son $(p \leftrightarrow r \wedge \text{aux1}) \wedge (q \leftrightarrow r \wedge \text{aux2})$ ³⁵

³⁴Ejemplo resuelto por Vincent Derkinderen (UK Leuven)

³⁵Debido a la **afinación de Clark** para capturar la suposición de un mundo cerrado (eg, la cabecera de una regla es verdad *ssi* el cuerpo correspondiente es verdad)

Ejemplo

Elección total			Evidencia		Reglas		Probabilidad
r	aux1	aux2	p	q	$((r \wedge \text{aux1}) \leftrightarrow p)$	$((r \wedge \text{aux2}) \leftrightarrow q)$	$P(L PL)$
V	V	V	V	V	V	V	$.9*.5*.4=0.18$
V	V	F	V	F	V	V	$.9*.5*.6=0.27$
V	F	V	F	V	V	V	$.9*.5*.4=0.18$
V	F	F	F	F	V	V	$.9*.5*.6=0.27$
F	V	V	F	F	V	V	$.1*.5*.4=0.02$
F	V	F	F	F	V	V	$.1*.5*.6=0.03$
F	F	V	F	F	V	V	$.1*.5*.4=0.2$
F	F	F	F	F	V	V	$.1*.5*.6=0.3$

$$\sum_L P(L|PL) = 1$$

Modelos del PL^{36} y la probabilidad $P(r, \text{aux1}, \text{aux2}, p, q)$ del respectivo programa L ; p y q pueden o no ser conclusiones de cada programa lógico L . El resto de interpretaciones (24) no son modelos y tienen probabilidad 0 de acuerdo a ProbLog (eg, $P(\neg r, \text{aux1}, \text{aux2}, p, q) = 0$ donde $\neg r$ es una notación corta para $r = F$)

³⁶ProbLog sigue la **semántica bien fundada** (*well-founded semantics*, en inglés) que para programas definidos equivale a la semántica de modelo mínimo único de Herbrand

Ejemplo - $P(r|p,q)$

- El cálculo de probabilidad para 'evidence(p). evidence(q). query(r).' corresponde a:

$$P(r|p,q) = \frac{P(r,p,q)}{P(p,q)}$$

donde considerando los 32 programas L ,

$$\begin{aligned} P(r,p,q) &= P(r,aux1,aux2,p,q) + P(r,aux1,\neg aux2,\neg p,q) \\ &\quad + P(r,\neg aux1,aux2,p,q) + P(r,\neg aux1,\neg aux2,p,q) \\ &= .18 + 0 + 0 + 0 \\ &= .18 \end{aligned}$$

y $P(p,q) = \sum_r P(r,p,q) = .18$ y así,

$$P(r|p,q) = \frac{P(r,p,q)}{\sum_{\forall r} P(r,p,q)} = \frac{.18}{.18} = 1$$

Ejemplo - $P(q)$

- Suponga que se quiere calcular $P(q)$:

Elección total		Consulta	Reglas	Probabilidad
r	aux2	q	$((r \wedge \text{aux2}) \leftrightarrow q)$	$P(L PL)$
V	V	V	V	$.9 * .4 = .36$
V	V	F	F	0
V	F	V	F	0
V	F	F	V	$.9 * .6 = .54$
F	V	V	F	0
F	V	F	V	$.1 * .4 = .04$
F	F	V	F	0
F	F	F	V	$.1 * .6 = .06$

$$\sum_L P(L|PL) = 1$$

Es posible considerar sólo aquellas reglas involucradas en la consulta, (note que hay interpretaciones que no son modelos del programa lógico probabilista)

Ejemplo - $P(q)$

- El cálculo de probabilidad para 'query(q).' es:

$$\begin{aligned} P(q) &= \sum_{r, \text{aux1}, \text{aux2}, p, q \in \{V, F\} \mid \{r, \text{aux1}, \text{aux2}, p, q\} \mid : q=V} P(r, \text{aux1}, \text{aux2}, p, q) \\ &= \sum_{r, \text{aux2}, q \in \{V, F\} \mid \{r, \text{aux2}, q\} \mid : q=V} P(r, \text{aux2}, q) \\ &= .36 \end{aligned}$$

(Sólo el primer renglón de la tabla anterior aporta, el resto de las interpretaciones no son modelos de PL y su probabilidad es 0)

Implementación

- La implementación ProbLog2 realiza los siguientes pasos:
 - 1 Aplicar resolución SLD para hallar los átomos y reglas aterrizadas de PL relevantes en la derivación lógica de la(s) consulta(s) y evidencia(s) formando nuevo programa lógico probabilista aterrizado “relevante”³⁷ PL_g (probado que $P(Q|E; PL_g) = P(Q|E; PL)$)
 - 2 Usar reglas $R_g \in PL_g$ para crear una expresión Booleana φ_r en CNF aplicando la afinación de Clark cuando no hay ciclos positivos o técnicas alternativas para deshacer los ciclos positivos ($SAT(\varphi_r) = MOD(PL_g)$)
 - 3 Incluir evidencia $\varphi = \varphi_r \wedge \varphi_e$ y construir un **fórmula Booleana pesada** que añade un mapeo $w(\cdot) \in [0, 1]$ para cada átomo en φ ; sólo los átomos en F_p aportan probabilidades diferentes a 1 ($SAT(\varphi) = MOD_E(PL_g)$ y $\forall \omega \in SAT(\varphi) : w(\omega) = PL_g(\omega)$, donde ω es una interpretación que satisface a φ y al mismo tiempo es un modelo para PL_g)

³⁷Eliminando reglas con literales en el cuerpo cuya negación o afirmación es contradictoria con la evidencia

Implementación

- (Cont.):

- 4 Obtener la probabilidad de la evidencia:

$$P(\text{Evidencia}) = \sum_{\omega \in \text{MODE}(PL)} P(\omega; PL) = \sum_{\omega \in \text{SAT}(\varphi)} w(\omega)$$

donde $\sum_{\omega \in \text{SAT}(\varphi)} w(\omega)$ se calcula transformando φ en una estructura *sd-DNNF* y evaluando su correspondiente *circuito aritmético* (que representa a los pesos y permite un número tractable de operaciones)

- 5 $P(\text{Consulta}|\text{Evidencia})$ se calcula utilizando el mismo circuito aritmético añadiendo la consulta a la evidencia ($\text{Consulta} \wedge \text{Evidencia}$) dado que:

$$P(\text{Consulta}|\text{Evidencia}) = \frac{P(\text{Consulta} \wedge \text{Evidencia})}{P(\text{Evidencia})}$$

Lanzamientos de monedas

- Modelado de lanzamientos independientes de 2 monedas distinguibles para el cálculo de la probabilidad de que resulte una o más ‘caras’

Archivos “coins_propositions.pblg” y “coins_predicates.pblg”

Noisy-Or

- La compuerta **noisy-OR** modela a un conjunto de n VA binarias³⁸ indexadas $\{X_1, \dots, X_n\}$ que son causas para un efecto Y y donde para una realización $X_1 = x_1, \dots, X_n = x_n$ de las VAs:

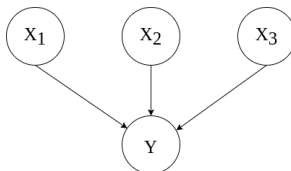
$$P(Y|X_1 = x_1, \dots, X_n = x_n) = \begin{cases} \prod_{\forall_i X_i=1}^n q_i & \text{si } Y = 0 \\ 1 - \left(\prod_{\forall_i X_i=1}^n q_i \right) & \text{de lo contrario} \end{cases}$$

donde $q_i = P(Y = 0|X_i = 1)$ es la probabilidad de que el efecto X_i no produzca a Y ³⁹

³⁸0 y 1 si la VA ocurre o no ocurre, respectivamente

³⁹Si hay algún mecanismo inhibitor para que la causa produzca al efecto, este inhibitor es independiente de los inhibidores del resto de las VA (esto habilita a la multiplicación de las probabilidades q_i); en la práctica, $q_i < 1$, ya que si $q_i = 1$ con toda seguridad X_i no produciría a Y

Noisy-Or



Descripción gráfica del modelo noisy-OR con tres causas (X_1, X_2, X_3) y su efecto Y


Table 7.1 Conditional probability table for a Noisy OR variable with three parents and parameters $q_1 = q_2 = q_3 = 0.1$

C_1	0	0	0	0	1	1	1	1
C_2	0	0	1	1	0	0	1	1
C_3	0	1	0	1	0	1	0	1
$P(E = 0)$	1	0.1	0.1	0.01	0.1	0.01	0.01	0.001
$P(E = 1)$	0	0.9	0.9	0.99	0.9	0.99	0.99	0.999

Tabla de valores de probabilidad para un efecto E dada ocurrencia de las causas C_1, C_2, C_3
(Tabla tomada sin permiso de: Luis Enrique Sucar. Probabilistic Graphical Models: Principles and Applications, 2nd ed. Springer International Publishing. 2021)

Archivo “noisy_or.pblg”

- A partir de lo anterior:
 - $P(Y = 0 | X_1 = 0, \dots, X_n = 0) = 1$: Con seguridad el efecto Y no ocurrirá dado que ninguna causa X_i ha ocurrido
 - $P(Y = 1 | X_1 = 0, \dots, X_n = 0) = 0$: No hay posibilidad de que el efecto Y ocurra si ninguna causa X_i ha ocurrido⁴⁰
 - $P(Y = 0 | X_1 = 1, \dots, X_n = 1) \approx 0$: Es poco probable que el efecto Y no ocurra si toda causa X_i ha ocurrido
 - $P(Y = 1 | X_1 = 1, \dots, X_n = 1) \approx 1$: Es muy probable que el efecto Y ocurra dado que toda causa X_i ha ocurrido

⁴⁰Responsabilidad, ie, si ninguna X_i ocurre no debe suceder Y 

Clasificador Bayesiano simple

- El **clasificador Bayesiano simple** es un clasificador probabilístico *supervisado*⁴¹ para estimar la probabilidad de una VA de clase $C \in \{c_1, \dots, c_m\}$ dado un vector $(A_1 = a_1, \dots, A_n = a_n)$ de n atributos:

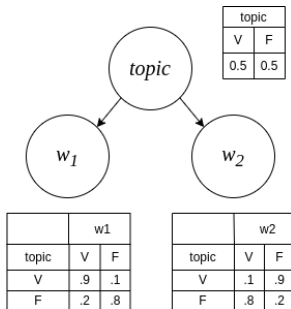
$$P(C = c_j | A_1 = a_1, \dots, A_n = a_n) = \frac{P(C = c_j) \prod_{i=1}^n P(A_i = a_i | C = c_j)}{\sum_{k=1}^m \left(P(C = c_k) \prod_{i=1}^n P(A_i = a_i | C = c_k) \right)},$$

para $k = 1, \dots, m$

y donde se supone que cada atributo es independiente del resto dada la clase

⁴¹ie, para su construcción requiere de ejemplos etiquetados por la clase a la que pertenecen

Clasificador Bayesiano simple



Clasificador Bayesiano simple para la probabilidad de que un texto pertenezca a un t3pico. La probabilidad de *topic* dado que hay dos palabras en un texto *w₁* y *w₂* es:

$$P(topic|w_1, w_2) = \frac{P(w_1|topic)P(w_2|topic)P(topic)}{P(w_1|topic)P(w_2|topic)P(topic) + P(w_1|\neg topic)P(w_2|\neg topic)P(\neg topic)}$$

Archivo "simple_bayes_classifier.pblg"

Redes Bayesianas

- Una **red Bayesiana** es un marco de trabajo (teoría y algoritmos) para representar distribuciones de probabilidad conjunta mediante grafos y hacer inferencia y aprendizaje

Redes Bayesianas

- Más formalmente, una red Bayesiana $B_n = (F, G)$ es un par ordenado donde:
 - F es un conjunto de funciones de probabilidad $P(X_i|Pa(X_i))$, cada una asociada a una VA X_i en $X = \{X_1, X_2, \dots, X_n\}$ tal que:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i|Pa(X_i))$$

y donde $Pa(X_i)$ son las VA “padre” de X_i (estructura **cuantitativa**)

- $G = (V, E)$ es un grafo acíclico dirigido donde V es un conjunto de n vértices cada uno asociado a una VA en X y $E \subset V \times V$ es un conjunto arcos dirigidos tal que si $X_j \in Pa(X_i)$ entonces existe un arco $(X_j, X_i) \in E$ (estructura **cualitativa**)

Redes Bayesianas

```
person(john).
person(mary).
```

```
0.7::burglary.
```

```
/* Disyunciones anotadas */
```

```
0.01::earthquake(heavy);
```

```
0.19::earthquake(mild);
```

```
0.8::earthquake(none).
```

```
0.90::alarm :- burglary, earthquake(heavy).
```

```
0.85::alarm :- burglary, earthquake(mild).
```

```
0.80::alarm :- burglary, earthquake(none).
```

```
0.10::alarm :- \+burglary, earthquake(mild).
```

```
0.30::alarm :- \+burglary, earthquake(heavy).
```

```
0.8::calls(X) :- alarm, person(X).
```

```
0.1::calls(X) :- \+alarm, person(X).
```

```
/* Introducción de evidencia */
```

```
evidence(calls(john),true).
```

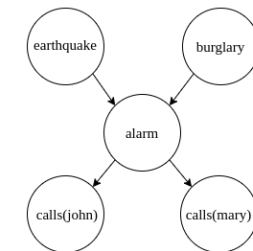
```
evidence(calls(mary),true).
```

```
query(burglary).
```

```
query(earthquake(_)).
```

earthquake		
heavy	mild	none
0.01	0.19	0.8

burglary	
V	F
0.7	0.3



		alarm	
earthquake	burglary	V	F
heavy	V	.9	.1
heavy	F	.3	.7
mild	V	.85	.15
mild	F	.1	.9
none	V	.80	.20
none	F	0	1

	calls(john)	
Alarm	V	F
V	.8	.2
F	.1	.9

	calls(mary)	
Alarm	V	F
V	.8	.2
F	.1	.9

Distribución de probabilidad conjunta $P(e, b, a, cj, cm) = P(cj|a)P(cm|a)P(a|e,b)P(e)P(b)$

Archivo "bayesian_network.pblg"

¿Preguntas?