Lenguaje P-log: estructura y traducción a SMODELS

Daniel Rodríguez, Claudia Zepeda y José Luis Carballido

Benemérita Universidad Autónoma de Puebla (BUAP),
Facultad de Ciencias de la Computación, Puebla, México
rmdaniel.04@gmail.com, czepedac@gmail.com, jlcarballido7@gmail.com

Resumen En este artículo estudiaremos el lenguaje P-log, el cual posee una base lógica y una base probabilística. P-log es un lenguaje que sirve para modelar y resolver problemas de probabilidad basándose en la programación lógica. P-log tiene una base lógica que es soportado por *Answer Set Programming* y la base probabilística por Redes bayesianas. *Answer Set Programming* tiene diferentes implementaciones y P-log utiliza SMODELS. Nuestro artículo tiene como propósito analizar el lenguaje P-log y mostrar como se realiza una traducción a una implementación del lenguaje ASP, es decir, traducirlo a SMODELS.

Palabras clave: P-log, *answer set programming*, SMODELS.

1. Introducción

La programación lógica es un tipo de programación declarativa que se forma por un conjunto de reglas. Un programa lógico para que pueda ser entendido se le debe asignar una semántica. La semántica es la manera de determinar el tipo de conclusiones que se pueden establecer a partir del conjunto de reglas, una de éstas semánticas es la semántica estable o semántica de modelos estables (SE) [1, 6]. La SE permite modelar el concepto de negación como falla en los programas lógicos y es la base de los *Answer Set Programming* (ASP).

ASP es un lenguaje de representación de conocimiento basado en SE de programas lógicos con negación como falla [2-3]. En ASP los programas se reducen a calcular los modelos estables. ASP tiene diferentes implementaciones como: POTASSCO[1], DLV[2], SMODELS[3], etc., y diferentes aplicaciones como planificación [7], actualización, razonamiento sobre acciones y causalidad, modelado de problemas con probabilidad [4].

Entre las aplicaciones que hemos mencionado de ASP, se encuentra el modelado de problemas probabilísticos, y uno de los lenguajes desarrollados para modelar este problema es P-log [4]. P-log es un lenguaje declarativo que posee una base lógica y una base probabilística. La base lógica es soportada por ASP y la base probabilística por las redes bayesianas. Debido a que P-log está basado en ASP, debemos realizar una transformación de un programa en lenguaje P-log al lenguaje ASP y obtener los modelos estables gracias a una implementación de ASP que es SMODELS. Los mo-

[1] http://potassco.sourceforge.net/
[2] http://www.dlvsystem.com/
[3] http://www.tcs.hut.fi/Software/smodels/

delos estables que están asociados al programa ASP corresponden a los mundos posibles asociados a P-log, además P-log posee otra característica muy importante que es la de agregar información probabilística a los mundos posibles. Esta información probabilística es calculada por P-log, sin embargo, en este artículo no nos enfocaremos en saber cómo los asigna. La finalidad de P-log es saber que probabilidad hay de que ocurra un suceso, y dependiendo de esas probabilidades tomar decisiones. Para obtener las probabilidades en el programa P-log, se realiza una consulta al final del programa.

El propósito de este artículo es presentar de forma muy detallada y sencilla la traducción de un programa escrito en lenguaje P-log a un programa que entienda SMODELS. En estos tiempos, aún no existe un manual o articulo que hable de la traducción de un programa en P-log a SMODELS, nosotros estamos presentando esta traducción para entender como trabaja la semántica en P-log. Cómo habíamos dicho P-log modela y resuelve problemas de probabilidad, que pueden ser de ayuda al momento de tomar decisiones tomando en cuenta probabilidades.

En la sección 2 veremos algunos conceptos que utilizaremos durante este artículo. En la sección 3 mostraremos como se traduce un programa de lenguaje P-log a SMODELS. En la sección 4 daremos algunas conclusiones de este trabajo.

2. Preliminares

En esta sección vamos a familiarizarnos con algunos conceptos con los cuales trabajaremos durante este artículo.

2.1. *Answer Set Programming*

ASP es un lenguaje de representación de conocimiento y fue introducido alrededor de 1990. ASP está basado en la semántica de modelo estable de programas lógicos con negación como falla [4].

Un programa en ASP corresponde a un conjunto de reglas, donde las reglas son de la forma: $L_0 \leftarrow L_1,\ldots, L_m$, not $L_{m+1}, \ldots$, not L_n, donde L_is son literales[4], $m \geq 0$, y $n \geq m$, en programación lógica cambiamos "$\leftarrow$" por el símbolo ":-". Las partes de la izquierda y derecha de "$\leftarrow$" se llaman la cabeza y el cuerpo de la regla, respectivamente. Cuando el cuerpo de la regla es vacío y en la cabeza hay una sola literal se llama un hecho, este se puede escribir sin el símbolo "$\leftarrow$" por ejemplo: "L_0". Cuando la cabeza es vacía, la regla se conoce como restricción, por ejemplo "$L_1,\ldots, L_m$, not $L_{m+1}, \ldots$, not L_n".

SMODELS, es un implementación de ASP que trabaja con SE para programas lógicos con negación como falla [5]. Los programas hechos en SMODELS son escritos usando notaciones de programación lógica básica, es decir, programas compuestos de átomos y reglas de inferencia.

[4] Literales son átomos o átomos negados (precedidos del símbolo "¬").

2.2. Lenguaje P-log

Un programa en P-log se conforma de las siguientes partes: declaraciones, reglas de selección random, información probabilística, observaciones y acciones. La manera más detallada de cada una de estas partes la encuentra en [4]. Vamos a describir cada una de las partes de manera breve y para que sea más fácil de entender a los lectores mostraremos un ejemplo sencillo. *El problema que se ilustra como ejemplo trata de 4 corredores que están compitiendo en las olimpiadas, y se asigna un juez de línea que está en la salida, éste indica si la salida fue buena o fue salida en falso, entonces habrá un ganador si hay una salida buena. La finalidad de este programa es saber que probabilidad hay de que un corredor llegue en primer lugar. Podemos decir que el corredor 2 tiene una probabilidad de ganar del 20%(0.20).*

Declaraciones: La declaración en un programa P-log puede tener una o todas de las siguientes formas: *definición de dominio de tipo, definición de tipos, declaración de tipos para atributos, declaración de la parte regular.*

Una *definición de dominio de tipo* en P-log tiene el siguiente formato:

```
#domain Tipo(Variable).
```

donde `Tipo` es un identificador para nombrar el tipo, el identificador debe ir con letras minúsculas. `Variable` es un identificador que puede ser una palabra que inicie con letra mayúscula, o simplemente, una letra mayúscula.

Tabla 1. En nuestro problema tenemos 2 dominios, uno es `corredor`, para designar a los corredores, y otro es `boolean` que designa al juez de línea.

`#domain corredor(C).`
`#domain boolean(B).`

La *definición de tipos* en P-log puede ser de 2 formas:

- en forma de lista y tiene el siguiente formato: `Tipo = {`X_1`, … , `X_n`}.`
- en forma de rango y tiene el siguiente formato `Tipo = {`X_1` .. `X_n`}.`

donde `Tipo` es un identificador con letras minúsculas, o bien, que pudo haber sido definido como dominio de tipo o ser definir como uno nuevo. X_i, con $1 \leq i \leq n$, es una lista de valores.

Tabla 2. Definimos los 2 tipos que habíamos declarado en la tabla 1, observamos que tenemos 4 corredores y 2 valores de boolean que indican si la salida es buena o falsa, cabe mencionar que se puede definir tipos sin declararlo en un dominio.

`corredor = {1 .. 4}.`
`boolean = {b, f}.`

La *declaración de tipos para atributos* en P-log puede ser de 2 formas:

- Atributo con dominio y rango, tiene el siguiente formato:
  `Atributo: Dominio -> Rango.`
- Atributo solo con rango, tiene el siguiente formato: `Atributo: Rango.`

donde `Dominio, Rango` son definiciones de tipo. `Atributo` es un identificador que debe ser escrita en letras minúsculas.

Tabla 3. Declaramos un atributo `primero` de tipo `corredor`, un atributo `salida` de tipo `boolean` y un atributo `ganador` que mapea un tipo `corredor` a un tipo `boolean`, `primero` y `boolean` tendrán los valores que se les asignaron a los tipos.

`primero : corredor.`
`salida : boolean.`
`ganador : corredor -> boolean.`

La *declaración de la parte regular* de un programa P-log consiste en una colección de reglas formadas usando literales. Éstas reglas pueden ir o no en el programa.

Tabla 4. En esta parte se define que si la `salida` fue falsa no habrá ningún `ganador`, y si no hay evidencia de que haya una `salida` en falso, es decir, que la `salida` es buena, entonces habrá un `ganador` de la carrera.

`ganador(C) = b :- not ganador(C) = f.`
`ganador(C) = f :- salida = f.`

Reglas de selección random. Existen 2 formas de declarar reglas random en P-log: *declarando reglas random con un atributo y una función de condición, declarando reglas random un atributo y además agregando cuerpo a la regla*.

Las *reglas random con un solo atributo y una función de condición* tienen el siguiente formato:

```
[Nombre_regla] random(Atributo:{X:Función(X)}).
```

donde `Nombre_regla` es un identificador para la regla, `Atributo` fue declarado en la sección de declaraciones. `Función(X)` es una regla que debe cumplirse para que el valor obtenido de la selección aleatoria sea verdadero.

Tabla 5. En esta regla se define que se escogerá un valor aleatorio para el atributo primero, pero debe cumplir que la función que sea verdadera, y para eso se utilizan las reglas escritas en la tabla 4.

`[r1] random(primero:{X : ganador(X)=b}).`

Las *reglas random con un solo atributo con cuerpo en la regla* tienen el siguiente formato:

```
[Nombre_regla] random(Atributo) :- Cuerpo.
```

donde `Nombre_regla` es un identificador para la regla, `Atributo` fue declarado en la sección de declaraciones, `Cuerpo` es un hecho que debe cumplirse para que la regla random sea verdadera, este tipo de regla puede no tener cuerpo.

Tabla 6. Se escogerá un valor aleatorio para el atributo salida.

`[r2] random(salida).`

Información probabilística. En P-log tiene el siguiente formato:

```
[Nombre_regla] pr(Función | Condición) = Valor.
```

donde `Nombre_regla` es un identificador para la regla y debe ser igual al `Nombre_regla` que asignamos en la regla random, `Función` es un valor que puede ser obtenido en las reglas random, `Valor` es un número entre [0, 1] escrito de forma Y/X, con X,Y dos números cualesquiera, y $X \geq Y$, `Condición` es un valor que debe ser cierto en las reglas random, esta condición puede ir o no.

Tabla 7. La probabilidad de que llegue en primer lugar el corredor 2 es de 1/5 (0.20).

```
[r1] pr(primero = 2) = 1/5.
```

Observaciones y acciones. Las observaciones y acciones son de la forma: `obs(l).` `do(l).` donde `l` es una literal. Las observaciones son usadas para llevar registro de los resultados de los eventos random. Por otro lado, las acciones indican que se hace verdadero como resultado de una acción deliberada, es decir, hubo una intervención sobre algún atributo Por ejemplo, si decimos que primero = 1 dentro del programa, entonces primero siempre será 1 y no podrá tener ningún otro valor y solo tendrá un mundo posible, en donde primero es igual a 1. Agregaremos una observación a este ejemplo:

Tabla 8. Agregamos la observación que la salida sea buena.

```
obs(salida=b).
```

Ya hemos mencionado que la finalidad de este ejemplo, es saber que probabilidad tiene cada uno de los corredores de llegar en primer lugar. Para esto se realiza una consulta al final del programa que sería de la forma siguiente:

Tabla 9. En esta tabla hacemos la consulta para saber que probabilidad hay de que el corredor 2 llegue en primer lugar.

```
? {primero = 2}.
```

Para terminar esta sección presentamos el programa completo escrito en P-log que modela el problema del corredor planteado al inicio de esta sección.

```
#domain corredor(C).
corredor= {1..4}.
#domain boolean(B).
boolean={b,f}.

salida : boolean.
primero : corredor.
ganador : corredor -> boolean.

ganador(C) = b :- not ganador(C) = f.
ganador(C) = f :- salida = f.

[r1] random(primero:{X: ganador(X)=b}).
[r2] random(salida).
```

```
[r1] pr(primero=2) = 1/5.

? {primero=2} | obs(salida=b).
```

Vemos la observación se agrega después de un signo "|" que va después de la consulta que se le hace al programa, en este caso preguntamos cual es la probabilidad que el ganador de la carrera sea el corredor 2 observando que la salida es buena.

3. Lenguaje P-log y su traducción a SMODELS

En esta sección vamos a mostrar la traducción de un programa escrito en lenguaje P-log a un programa SMODELS. Vemos importante realizar la traducción de un programa escrito en lenguaje P-log a un programa SMODELS porque nos facilitará entender la semántica del lenguaje P-log.

Retomaremos el ejemplo del corredor visto en la subsección 2.2 y explicaremos cómo se realiza la traducción a SMODELS, el orden de cómo explicar la traducción es conforme a las partes que estructuran un programa P-log.

Traducción de la parte de declaraciones: Como ya sabemos las declaraciones en un programa P-log puede tener una o todas de las siguientes formas: *definición de dominio de tipo, definición de tipos, declaración de tipos para atributos, declaración de la parte regular*.

Una *definición de dominio de tipo* en P-log tiene el siguiente formato: `#domain Tipo(Variable).` y su traducción a SMODELS tiene el mismo formato, es decir, este código es el mismo para P-log y SMODELS. Por lo tanto, para nuestro ejemplo la traducción queda igual que en la tabla 1.

La *definición de tipos* en P-log puede ser de 2 formas:

- en forma de lista y tiene el siguiente formato: `Tipo = {`X_1`, … , `X_n`}.`
- en forma de rango y tiene el siguiente formato `Tipo = {`X_1` .. `X_n`}.`

Su traducción a SMODELS sería lo siguiente:

- en forma de lista se traduciría a: `Tipo(`X_1`). Tipo(`X_2`). … Tipo(`X_n`).`
- en forma de rango se traduciría a: `Tipo(`X_1` .. `X_n`).`

Para nuestro ejemplo, veremos la traducción en la siguiente tabla.

Tabla 10. En SMODELS se escribe cada atributo por separado como un átomo con un solo argumento, resultaría más cansado si se tuvieran muchos numeros pues se tardaría mucho en escribirlo.

Lenguaje P-log	Traducción a SMODELS

`corredor = {1 .. 4}.`
`boolean = {b,f}.`
	`corredor(1). corredor(2). corredor(3).`
	`corredor(4).`
	`boolean(b). boolean(f).`

La *declaración de tipos para atributos* en P-log no tienen una traducción a SMODELS, sin embargo, son utilizados cuando se traducen las reglas random.

La *declaración de la parte regular* de un programa P-log consiste en una colección de reglas formadas usando literales. Éstas reglas pueden ir o no en el programa.

Para nuestro ejemplo, veremos la traducción en la siguiente tabla.

Tabla 11. En esta parte regular P-log la traducción se hace en átomos con 2 argumentos.

Lenguaje P-log	Traducción a SMODELS

`ganador(C) = b :-`
`not ganador(C) = f.`
`ganador(C) = f :- salida = f.`
	`ganador(C,b) :-`
	`not ganador(C,f).`
	`ganador(C,f) :- salida(f).`

Traducción de la parte de reglas de selección random. Como ya sabemos existen 2 formas de declarar una regla random en P-log: *declarando reglas random con un atributo y una función de condición, declarando reglas random un atributo y cuerpo en la regla.*

Las *reglas random con un solo atributo y una función de condición* tienen el siguiente formato:

```
[Nombre_regla] random(Atributo:{X:Función(X)}).
```

Su traducción a SMODELS sería lo siguiente:

```
1{Atributo(X_): Rango(X_)}1 :- not intervene(Atributo).
pd( Nombre_regla, Atributo) :- not intervene(Atributo),
                               Variable(X), Función(X).
:- not intervene(Atributo), Atributo(X), Función(X),  Rango(X).
show Atributo(X_).
```

Para nuestro ejemplo, veremos la traducción en la siguiente tabla.

Tabla 12. Este tipo de reglas random es más complicado de traducir, pues una línea de código en P-log tiene 4 líneas de código en SMODELS.

Lenguaje P-log	Traducción a SMODELS

`[r1] random (primero:`
`{X: ganador(X) = b}).`
	`1{primero(X) : corredor(X)}1 :-`
	`not intervene(primero).`
	`pd(r1,primero(X)) :-`
	`not intervene (primero), corredor(X),`
	`ganador(X,b).`
	`:-not intervene(primero), primero(X),`
	`not ganador(X,b), corredor(X).`
	`show primero(X).`

Las reglas random no existen en SMODELS, pero podemos intuirla gracias a otras reglas que nos ayudan a realizar la misma función, la primera línea de código nos indica que primero solo tendrá un valor de tipo corredor siempre y cuando no haya intervención sobre primero. En la segunda línea de código ponemos `pd` para indicar que estamos hablando de probabilidad también no debe haber una intervención sobre primero, el valor que se asigne a primero debe ser de tipo corredor, además que la condición de ganador debe ser verdadera, `show` es una palabra reservada en SMODELS que muestra el valor de primero.

Las *reglas random con un solo atributo con cuerpo en la regla* tienen el siguiente formato:

```
[Nombre_regla] random(Atributo) :- Cuerpo.
```

Su traducción a SMODELS sería lo siguiente:

```
1{Atributo(X_): Rango(X_)}1: not intervene(Atributo).
pd( Nombre_regla, Atributo) :- not intervene(Atributo),
                               Rango(X_), Cuerpo.
show Atributo(X_).
```

Para nuestro ejemplo, veremos la traducción en la siguiente tabla.

Tabla 13. En esta traducción la regla random no tiene cuerpo, y solo se crean 3 reglas más para SMODELS, mencionaremos que si la regla tuviera cuerpo, se agregaría al final en el cuerpo de la regla que incia con pd.

Lenguaje P-log	Traducción a SMODELS

`[r2] random(salida).`	`1{salida(X_) : boolean(X_)}1:-`
	`not intervene(salida).`
	`pd(r2,salida(X_)) :-`
	`not intervene(salida),boolean(X_).`
	`show salida(X_).`

Traducción de la parte de información probabilística. En P-log tiene el siguiente formato:

```
[Nombre_regla] pr(Función | Condición) = Valor.
```

Su traducción a SMODELS sería lo siguiente:

```
pa (Nombre_regla, Función, di_(Y,X)) :- Condición.
```

Tabla 14. Resulta sencilla la traducción porque es una línea, sólo es un átomo con 3 argumentos, primero el nombre de la regla, segundo la función y tercero la probabilidad.

Lenguaje P-log	Traducción a SMODELS
`[r1] pr(primero = 2) = 1/5.`	`pa(r1, primero(2), di_(1,5)).`

Podemos deducir que se utiliza `pa` para denotar probabilidad y `di_` para denotar el valor de la probabilidad.

Traducción de la parte de observaciones y acciones. Su traducción a SMODELS es la misma `obs(l).` `do(l).` donde `l` es una literal.

Para nuestro ejemplo, veremos la traducción en la siguiente tabla.

Tabla 15. La traducción de una observación genera una restricción en SMODELS, y 4 reglas más que irán en todos los programas.

Lenguaje P-log	Traducción a SMODELS

`? {primero=2} |`
`obs(salida=b).`
	`:- not salida(b).`
	`hide. show pd(X,Y). show pa(X,Y,Z).`
	`show primero(2).`

La observación se traduce en una restricción en la cual se pide que la salida sea buena, hide es una palabra reservada que oculta valores de salida, y solo pide mostrar donde estén las palabras pd, pa y que muestre cuando el ganador sea el corredor 2.

A continuación mostramos el programa completo escrito P-log y su traducción a SMODELS.

Lenguaje P-log	Traducción a SMODELS
`#domain corredor(C).`	`#domain corredor(C).`
`#domain boolean(B).`	`#domain boolean(B).`

`corredor = {1 .. 4}.`
`boolean = {b,f}.`
	`corredor(1). corredor(2).`
	`corredor(3). corredor(4).`
	`boolean(b). boolean(f).`

`primero : corredor.`
`salida : boolean.`
`ganador : corredor -> boolean.`	

`ganador(C) = b :-`
`  not ganador(C) = f.`
`ganador(C) = f :- salida = f.`
	`ganador(C,b) :- not ganador(C,f).`
	`ganador(C,f) :- salida(f).`

`[r1] random (primero:`
`{X:ganador(X)=b}).`
	`1{primero(X):corredor(X)}1 :-`
	`  not intervene(primero).`
	`pd(r1,primero(X)):-`
	`  not intervene(primero),`
	`  corredor(X), ganador(X,b).`
	`:-not intervene(primero), primero(X),`
	`  not ganador(X,b),corredor(X).`
	`show primero(X).`

`[r2] random (salida).`	`1{salida(X_) : boolean(X_)}1:-`
	`  not intervene(salida).`
	`pd(r2,salida(X_)) :-`
	`  not intervene(salida),`
	`  boolean(X_).`
	`show salida(X_).`

`[r1] pr(primero=2) = 1/5.`	`pa(r1,primero(2),di_(1,5)).`

`? {primero=2} | obs(salida=b).`	`:- not salida(b).`
	`hide.`
	`show pd(X,Y).`
	`show pa(X,Y,Z).`
	`show primero(2).`

Los mundos posibles que se obtienen son:

```
Mundo posible 1: primero(4) salida(b)
Mundo posible 2: primero(2) salida(b)
Mundo posible 3: primero(3) salida(b)
Mundo posible 4: primero(1) salida(b)
```

La respuesta a la consulta de que probabilidad hay de que el corredor 2 sea el primero es de 0.20.

El programa de P-log se ejecutó en el sistema P-log diseñado por Weijun [4].

4. Conclusiones

Durante este artículo se explicó de forma muy detallada cómo se estructura P-log y cómo se traduce a SMODELS una implementación de ASP.

Los programas hechos en el lenguaje P-log se pueden ejecutar en el sistema P-log[5], un sistema creado por Weijun Zhu (2008) para obtener los mundos posibles.

Recordar que podemos encontrar más ejemplos de programas hechos en lenguaje P-log y su traducción a un programa SMODELS, en la siguiente dirección web:

https://sites.google.com/site/rmdaniel04/

Podemos ver que la traducción se realiza de un programa P-log a un programa SMODELS, pero un trabajo a futuro se podría realizar la traducción a otra implementación ASP como Potassco o DLV.

Agradecimiento. Este trabajo ha sido apoyado por el Proyecto de Ciencia Básica del Fondo Sectorial SEP-CONACyT con número de registro 101581.

Referencias

1. M. Gelfond, V. Lifschitz: The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editores, 5th Conference og Logic Programming. MIT Press (1988)
2. Answer Set Programming, http://en.wikipedia.org/wiki/Answer_set_programming
3. David Pearce: Introducción a ASP (Answer Set Programming - programación con conjuntos respuestos) (2009)
4. C. Baral, M. Gelfond, N. Rushton: Probabilistic reasoning with answer set. Cambrigde University Press (2008)
5. Tommi Syrjänen: Manual de usuario de Lparse, http://www.tcs.hut.fi/Software/smodels/index.html#downloads
6. Stable model semantics, http://en.wikipedia.org/wiki/Stable_model_semantics

M. Gelfond, V. Lifschitz: Action Languages. In: Electronic Transactions on Artificial Intelligence, vol. 2, pp 193-210 (1998)

[5] http://www.cs.ttu.edu/~wezhu/