Programación Declarativa, 2023-2
Nota de clase 02: Fundamentos teóricos de la Programación Lógica

Manuel Soto Romero

Facultad de Ciencias UNAM

En el curso de Lógica Computacional, se estudia a la Programación Lógica desde un punto de vista teórico, en donde, se define este estilo de programación a partir del sistema de la Lógica de Primer Orden, también llamado, Lógica de Predicados. Este sistema es mucho más completo que el de la lógica proposicional pues permite, entre otras cosas, dar la relación que hay entre objetos del mundo real, por ejemplo personas, colores, automóviles, entre otras.

En esta nota revisaremos cómo son aplicados estos conceptos en la Programación Lógica, en particular, siguiendo el caso de PROLOG, haciendo especial énfasis en los métodos de unificación, resolución y retroceso.

Lógica de primer orden

Recordemos que la Lógica de Primer Orden de basa en el uso de predicados que al ser aplicados a ciertos objetos, resultan en una afirmación, es decir, pueden ser verdaderos o falsos. La Lógica de Primer Orden no tiene un lenguaje único, pues este depende el problema particular que se esté modelado, sin embargo, podemos destacar características en común:

- Fórmulas atómicas: Son las constantes lógicas ($\bot, \top$), y predicados aplicados a términos ($P(t_1, ..., t_n)$). Recordemos que existen tres tipos de *términos*: Constantes ($c_1, ..., c_n$), variables ($x_1, ..., x_n$) y funciones aplicadas a términos ($f(t_1, ..., t_n)$).
- Operadores aplicados a fórmulas: Negación ($\neg\varphi$), conjunción ($\varphi \wedge \psi$), disyunción ($\varphi \vee \psi$), implicación ($\varphi \rightarrow \psi$), equivalencia ($\varphi \longleftrightarrow \psi$).
- Cuantificadores: Universal ($\forall x \varphi$) y existencial ($\exists x \varphi$).

Usando este lenguaje podemos dar la especificación formal de enunciados de forma que podamos manipularlos más fácil, por ejemplo, para decidir si un argumento es correcto o no.

Ejemplo

Supongamos el siguiente enunciado:

Hay dos pizzas, la primera, es de pepperoni, la segunda es hawaiana. Sólo como pizza si hay hawaiana. ¿Puedo comer pizza?

Damos la traducción del enunciado anterior a lógica de predicados.

$$\exists x P(x), \exists y H\left(y\right), \forall z (H(z) \rightarrow C(z)), C(x)$$

Sin embargo, para Programación Lógica, no usaremos todos estos conceptos explícitamente para representar fórmulas, en su lugar, usaremos las fórmulas representadas mediante cláusulas (disyunción de fórmulas atómicas o su negación) de la siguiente manera:

- Tenemos con conjunto de cláusulas que representan lo que sabemos acerca del problema a modelar y al que llamamos *base de conocimientos*.
- Tenemos una cláusula a la que llamamos *cuestión* o *consulta*, es decir una pregunta que pueda responderse a través de lo que conocemos de la base de conocimientos.
- Para saber si la *consulta* es verdadera a partir de la base de conocimientos aplicamos el mecanismo de Resolución Binaria que a su vez usa unificación. De esta forma, si podemos llegar a la *cláusula vacía*, decimos que la consulta es verdadera y en caso contrario falsa.

A continuación analizamos estos conceptos con más detalle.

Cláusulas

Recordemos que una cláusula es simplemente una disyunción de fórmulas atómicas o su negación (literales).

$$\mathscr{C} = \ell_1 \vee \ell_2 \vee ... \vee \ell_n$$

Además de esta notación, existen otras formas de denotar a las cláusulas: [2]

- **Notación conjuntista**: Cada cláusula es representada como un conjunto, donde cada literal es un elemento de dicho conjunto. Esta notación aunque es empleada por diversos autores, no es la más adecuada en Programación Lógica.

$$\mathscr{C} = \{\ell_1, \ell_2, ..., \ell_n\}$$

- **Notación para Programación Lógica:**

  Consideremos una cláusula $\mathscr{C}$ de la forma:

$$\mathscr{C} = \ell_1 \vee \ell_2 \vee ... \vee \ell_n$$

  Usando la conmutatividad de la disyunción, podemos reescribir la cláusula agrupando las literales negativas y positivas. Las literales negativas P_i y positivas Q_i son predicados, omitimos sus argumentos pues de momento no son relevantes.

$$\neg P_1 \vee \neg P_2 \vee ... \vee \neg P_j \vee Q_i \vee Q_2 \vee ... \vee Q_k$$

  Aplicando las leyes de De Morgan:

$$\neg\big(P_1 \wedge P_2 \wedge ... \wedge P_j\big) \vee (Q_1 \vee Q_2 \vee ... \vee Q_k)$$

  Reescribimos como implicación:

$$P_1 \wedge P_2 \wedge ... \wedge P_j \rightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$$

  Sustituimos las disyunciones y conjunciones por comas. Recordando que las comas del antecedente son conjunciones y las del consecuentes disyunciones.

$$P_1, P_2, ..., P_j \rightarrow Q_1, Q_2, ..., Q_k$$

  Escribimos la implicación al revés:

$$Q_1, Q_2, ..., Q_k \leftarrow P_1, P_2, ..., P_j$$

  Llamamos a los átomos $Q_1, ..., Q_k$ el encabezado y a los átomos $P_1, P_2, ..., P_j$ el cuerpo de la cláusula.

Definición

Una Cláusula de Horn, también llamada Cláusula Definida, es una cláusula con a lo más una literal positiva. Las Cláusulas de Horn son alguna de las siguientes cuatro formas:

- *Hechos: una literal positiva y ninguna negativa.*

$$Q \leftarrow$$

- *Reglas: una literal positiva y al menos una negativa.*

$$Q \leftarrow P_1, ..., P_j$$

- *Metas: ninguna literal positiva y al menos una negativa.*

$$\leftarrow P_1, ..., P_j$$

- *Cláusula vacía: sin literales, también denotada* $\square$.

$$\leftarrow$$

Un programa lógico $\mathbb{P}$ es simplemente un conjunto de hechos y reglas (base de conocimientos). Las consultas se representan mediante metas y los resultados dependerán de llegar o no a la cláusula vacía. Es decir, dado un programa lógico $\mathbb{P}$ y una consulta $\mathscr{C}$ queremos verificar $\mathbb{P} \models \mathscr{C}$.

Ejemplo

Supongamos el enunciado del Ejemplo anterior:

Hay dos pizzas, la primera, es de pepperoni, la segunda es hawaiana. Sólo como pizza si hay hawaiana. ¿Puedo comer pizza?

1. Damos la traducción del enunciado anterior a lógica de predicados.

   $\exists x \mathrm{P}(x), \exists y H\left(y\right), \forall z\left(H\left(z\right) \rightarrow C\left(z\right)\right), C\left(x\right)$

2. Forma Clausular:

   $\{P\left(a\right), H\left(b\right), \neg H\left(z\right) \vee C\left(z\right), \neg C\left(x\right)\}$

3. Sintaxis de Programación Lógica

   Programa $\mathbb{P}$:

$$\begin{array}{lcll} P(a) & \leftarrow & & \textit{Hecho} \\ H(b) & \leftarrow & & \textit{Hecho} \\ C(z) & \leftarrow & H(z) & \textit{Regla} \end{array}$$

   Consulta $\mathscr{C}$:

$$\leftarrow \quad C(x) \quad \textit{Meta}$$

   Objetivo: $\mathbb{P} \models \mathscr{C}$

Observación

Para llegar a la Forma Clausular, deben realizarse distintas transformaciones, en el siguiente orden: (1) Rectificación, (2) Forma Normal Negativa, (3) Forma Normal Prenex, (4) Sokolemnización, (5) Forma Normal de Skolem, (6) Forma Normal Conjuntiva y finalmente (7) Forma Clausular.

Revisar Nota adicional B.

Otra forma de representar estas cláusulas es mediante reglas de inferencia, de forma tal que el encabezado representa la conclusión y el cuerpo las premisas. Los hechos son reglas sin premisas (se consideran verdaderos).

Ejemplo

Reglas de inferencia para el Ejemplo anterior:

$$\frac{}{P(a)} \qquad \frac{}{C(z)} \qquad \frac{H(z)}{C(z)}$$

Resolución binaria con unificación

La resolución binaria es de gran importancia en los lenguajes de programación lógica, al proporcionar un método que permita decidir la consecuencia lógica $\Gamma \models \varphi$ (recordemos que este es el objetivo al tener una consulta sobre una base de conocimientos), al llegar a la cláusula vacía $\square$ desde las formas clausulares del conjunto $\Gamma \cup \{\neg\varphi\}$. [3]

Recordemos que la regla de resolución binaria en Lógica de Primer Orden se define como sigue: [3]

$$\frac{\mathscr{C} =_{def} \mathscr{C}_1 \vee \ell \qquad \mathscr{D} =_{def} \mathscr{D}_1 \vee \ell' \qquad Var(\mathscr{C}) \cap Var(\mathscr{D}) = \varnothing \qquad \sigma \text{ umg de } \{\ell^c, \ell'\}}{(\mathscr{C}_1 \vee \mathscr{D}_1)\sigma}$$

- Se tienen dos cláusulas $\mathscr{C}$ y $\mathscr{D}$ con variables ajenas.
- $\mathscr{C}$ tiene una literal ℓ y $\mathscr{D}$ una literal ℓ' de forma que ℓ^c es unificable con ℓ' es decir, existe un unificador más general σ.
- La regla nos devuelve una disyunción $(\mathscr{C}_1 \vee \mathscr{D}_1)\sigma$ resultado de eliminar las literales ℓ y ℓ'.
- Existen distintos algoritmos de unificación, por ejemplo, recordemos el siguiente: [3]

  Para este algoritmo tenemos una pila de restricciones y un conjunto de sustituciones.

  1. Si X y Y son constantes idénticas, no hagas nada.
  2. Si X y Y son identificadores idénticos, no hagas nada.
  3. Si X es un identificador, reemplaza, todas las apariciones de X por Y tanto en la pila como en las sustituciones, y añadimos $[X := Y]$ en las sustituciones.
  4. Si Y es un identificador, reemplaza, todas las apariciones de Y por X tanto en la pila como en la sustituciones, y añadimos $[Y := X]$ en las sustituciones.
  5. Si X es un constructor de la forma $C(X_1, ..., X_n)$ para algún constructor C, y Y es de la forma $C(Y_1, ..., Y_n)$,es decir tienen el mismo constructor y mismo número de argumentos, entonces, agrega $X_i = Y_i$ con $1 \leq i \leq n$ a la pila.
  6. En cualquier otro caso, X y Y no unifican y se reporta un error.

Ejemplo

Supongamos las cláusulas $\mathscr{C} = P(x, y) \vee \neg Q(a, x)$ y $\mathscr{D} = R(z) \vee Q(z, b)$

- Ambas cláusulas tienen variables ajenas.
- Tenemos en $\mathscr{C}$ una literal $\ell = \neg Q(a, x)$ y una literal en $\mathscr{D}$, $\ell' = Q(z, b)$. Busquemos su unificador más general:

Acción	Pila	Sustitución
Inicio	$Q(a,x) = Q(z,b)$	
5	$a = z$ $x = b$	
4	$x = b$	$[z := a]$
3		$[z := a]$ $[x := b]$

- Nuestro umg es $\sigma = [z := a, x := b]$
- El resultado de la resolución binaria es: $(P(x,y) \vee R(z))\sigma = P(b,y) \vee R(a)$.

Volvamos a nuestro ejemplo, queremos $\mathbb{P} \models \mathscr{C}$.

Ejemplo

Hay dos pizzas, la primera, es de pepperoni, la segunda es hawaiana. Sólo como pizza si hay hawaiana. ¿Puedo comer pizza?

Programa $\mathbb{P}$:

$$\begin{array}{lcll} P(a) & \leftarrow & & \textit{Hecho} \\ H(b) & \leftarrow & & \textit{Hecho} \\ C(z) & \leftarrow & H(z) & \textit{Regla} \end{array}$$

Consulta $\mathscr{C}$:

$$\leftarrow \quad C(x) \quad \textit{Meta}$$

Objetivo: $\mathbb{P} \models \mathscr{C}$

Para lograr este objetivo, debemos aplicar una resolución binaria a $\mathbb{P} \cup \{\neg\mathscr{C}\}$. En este caso debemos aplicar la resolución por cada par de cláusulas (todas contra todas) hasta encontrar la cláusula vacía. Omitimos los pasos de unificación, pero el lector los puede verificar. Se muestra en cada paso a qué cláusulas se les aplica la resolución y el umg de éstas.

$$\begin{array}{lllll} 1. & P(a) & \leftarrow & & Hip. \\ 2. & H(b) & \leftarrow & & Hip. \\ 3. & C(z) & \leftarrow & H(z) & Hip. \\ 4. & & \leftarrow & C(x) & Hip. \\ 5. & C(b) & \leftarrow & & Res(2,3,[z := b]) \\ 6. & & \leftarrow & & Res(4,5,[x := b]) \end{array}$$

Como llegamos a la cláusula vacía $\mathbb{P} \models \mathscr{C}$ y por lo tanto puede comer pizza y obtenemos cuál es esa pizza a partir de las sustituciones, en este caso:

$[z := b, x := b]$, la segunda pizza (hawaiana) es la que puedo comer.

Búsqueda de pruebas

Hemos visto cómo es que los programas lógicos resuelven las consultas mediante resolución binaria, sin embargo, para pasar realmente de la Lógica a la Programación Lógica, no podemos hacer la resolución de cláusulas todas contra todas, por tanteo, es necesario que el lenguaje implemente una búsqueda de pruebas de acuerdo a una estrategia particular, no basta saber que esa prueba existe, debemos construirla. Existen dos métodos, en general, para hallar estas pruebas: [2]

- Buscar hacia atrás a partir de la meta o que queremos probar. Este método se conoce como el método dirigido a metas.

- Buscar hacia adelante desde los axiomas o hechos aplicando reglas hasta llegar a la conclusión. Este método se conoce como razonamiento hacia adelante.

La Programación Lógica fue concebida como una búsqueda dirigida a metas. La idea del método de búsqueda es la siguiente: dada una meta, determinar qué regla de inferencia se pudo haber aplicado para llegar a dicha meta. Seleccionamos una de tales reglas y aplicamos recursivamente la estrategia de búsqueda a las premisas. Las cuales se convierten en submetas. Si no hay más premisas, decimos la prueba de la meta está terminada [2].

En la búsqueda de pruebas existen dos aspectos importantes a tomar en cuenta: [2]

- Retroceso (*backtracking*). Cuando en una búsqueda una meta corresponde a la conclusión de más de una regla decimos que tenemos un *punto de elección*. Cuando se llega a un punto de elección se elige la primera de las reglas posibles. Si la búsqueda falla al elegir dicha regla regresamos al primer punto de elección y cambiamos de regla.
- Orden de las submetas. Otro tipo de elección surge cuando una regla de inferencia tiene más de una premisa, a saber, el orden en que intentemos hallar una prueba de ellas. Por supuesto desde el punto de vista lógico formal este orden es irrelevante, pero operacionalmente el comportamiento del programa puede ser muy distinto, de hecho podríamos causar ciclos infinitos.

Ejemplo

Supongamos el siguiente programa que implementa la suma de números naturales y queremos hallar la suma de $1+1$.

Programa $\mathbb{P}$:

$$\begin{array}{lcll} suma(0,N,N) & \leftarrow & & Hecho \\ suma(s(M),N,s(P)) & \leftarrow & suma(M,N,P) & Regla \end{array}$$

Consulta $\mathscr{C}$:

La consulta usa una variable de manera que la resolución binaria encuentre el valor de la misma, usando el umg.

$$\leftarrow \quad suma(s(0),s(0),X) \quad Meta$$

Objetivo: $\mathbb{P} \models \mathscr{C}$

Para lograr este objetivo, aplicamos los siguientes pasos:

1. Tenemos dos reglas de inferencia, las escogemos en el orden en que fueron definidas. En este caso, la primera regla $suma(0,N,N)$ no puede unificarse con la meta al tener dos constantes (0 y $s(0)$) como primer argumento. Por lo tanto elegimos la segunda regla:

   $$suma(s(M),N,s(P)) \leftarrow suma(M,N,P)$$

2. Continuamos con la premisa $suma(M,N,P)$, en este caso $M=0$, $N=s(0)$, $s(P)=X$ y $P=X'$, donde X' es el predecesor de X.

   $$suma(0,s(0),X')$$

3. En este caso corresponde con la primera regla, por lo que la usamos:

$$suma(0, N, N)$$

Por lo tanto:

$$suma(0, s(0), s(0))$$

y como $X' = s(0)$, entonces $X = s(s(0))$ que es la respuesta esperada.

O visto como regla de inferencia:

$$\frac{suma(0, s(0), s(0))}{suma(s(0), s(0), s(s(0))}$$

En este sentido, los lenguajes de programación lógicos, hacen uso del no determinismo para encontrar todas las posibles soluciones a un problema dado. La forma en que se realiza este proceso de retroceso es mediante lo que se conoce como Árbol SLD: [1]

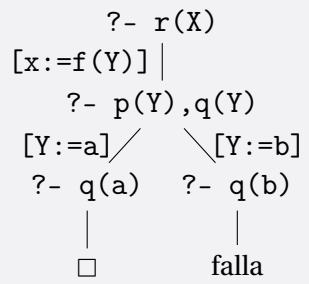
Definición

Sea $\mathbb{P}$ un programa lógico y $\mathcal{G}$ una meta. Un árbol SLD es un árbol n-ario posiblemente infinito cuya raíz es la meta $\mathcal{G}$ y cuyos nodos tienen metas exclusivamente; el hijo de un nodo $\mathcal{G}_i$ es la nueva meta $\mathcal{G}_j$ obtenida a partir de la resolución binaria de $\mathcal{G}_i$ y una cláusula del programa.

Ejemplo

Árbol de búsqueda para $\mathbb{P} = \{p(a), p(b), r(f(X)) \leftarrow p(X), q(X)\}$ con la meta $r(X)$.

1.	$p(a)$	$\leftarrow$		$Hip.$
2.	$p(b)$	$\leftarrow$		$Hip.$
3.	$q(a)$	$\leftarrow$		$Hip.$
4.	$r(f(Y))$	$\leftarrow$	$p(Y), q(Y)$	$Hip.$
5.		$\leftarrow$	$r(X)$	$Meta$
6.		$\leftarrow$	$p(Y), q(Y)$	$SLDRes(4,5,[X = f(Y)])$
7.		$\leftarrow$	$q(a)$	$SLDRes(6,1,[Y := a])$
8.	$\square$			$SLDRes(3,7)$



Respuesta: $[X := f(a)]$

Referencias

[1] Favio E. Miranda, A. Liliana Reyes, et.al. *Notas para el curso de Lógica Computacional,* Facultad de Ciencias UNAM, Revisión 2020-2.

[2] Favio E. Miranda, Lourdes del C. González, *Notas para el curso de Programación Funcional y Lógica,* Facultad de Ciencias UNAM, Revisión 2012-1.

[3] Shriram Krishnamurthi, *Programming Languages Applications and Interpretation,* Brown University, Primera Edición, 2007.