Deseño de Linguaxes de Programación

Programación Lógica

Rodríguez Arias, Alejandro
alejandro.rodriguez.arias@udc.es

Bouzas Quiroga, Jacobo
jacobo.bouzas.quiroga@udc.es

Thursday 26th October, 2017

Índice

1. Introducción

El presente trabajo tratará de ofrecer una aproximación al paradigma de la programación lógica y los diferentes lenguajes de programación que se encuadran en él, dando una perspectiva de las particularidades de este tipo de lenguajes respecto a los de otros paradigmas y ofreciendo una visión comparativa de las características propias de cada uno de los lenguajes estudiados.

La programación lógica, en su forma actual, surge alrededor de la década de los 70 en el marco del debate entre representaciones procedurales y declarativas del conocimiento en Inteligencia Artificial. La programación lógica se basa en la lógica formal.

Un programa en este paradigma se expresa como una serie de hechos y reglas que representan las relaciones entre ellos. De este modo, al escribir un algoritmo de estos lenguajes, no es necesario expresar el componente de control. El flujo del programa, el orden en que se ejecutan las instrucciones, se decidirá de forma automática y por tanto, la programación lógica es un subconjunto del paradigma de la programación declarativa.

La expresión de programas como una serie de cláusulas lógicas encuentra su utilidad fundamental en varios dominios en los que la lógica se presenta como la expresión natural de los problemas a abordar. Algunos de estos dominios son el razonamiento automático, tanto para la demostración automática de teoremas y la verificación formal como para la construcción de sistemas expertos; o el reconocimiento del lenguaje natural.

La programación lógica abarca varios tipos de lógicas, útiles en diferentes tipos de problemas. Además de la lógica proposicional y de primer orden, existen implementaciones de lógica difusa y lógica modal, como la temporal, para trabajar con la ambigüedad semántica propia de los lenguajes humanos o lógicas no monotónicas, para razonar bajo incertidumbre o información incompleta.

Los lenguajes elegidos en este trabajo para ejemplificar los diferentes usos de la programación lógica serán Prolog, para el caso más general, Datalog, como ejemplo de un lenguaje orientado a la consulta de bases de conocimientos, Maude system como ejemplo de sistema de reescritura y Clingo para el enfoque de programación de conjunto-respuesta (*answer set programming*).

El resto del trabajo está estructurado como sigue. La sección 2 es una brevísima introducción a la lógica matemática y algunas lógicas alternativas a la usual lógica de predicados, que incorporan artefactos adecuados para el razonamiento en dominios en los que la lógica estándar se queda corta. En la sección 3 introducimos el lenguaje Prolog, diseccionando su funcionamiento y características y comentando por encima algunas extensiones populares. La sección 4 comenta varios lenguajes de programación lógica al-

ternativos orientados a distintos dominios y finalmente, la sección 5 examina los dominios de aplicación más frecuentes para la programación lógica y presenta algunos ejemplos de casos de uso reales desarrollados en este paradigma.

2. Lógicas alternativas

2.1. Lógica proposicional

La lógica proposicional, o lógica de orden cero, es un sistema formal en el que sus elementos más simples son variables proposicionales cuyo significado no es importante. Solo interesa su valor de verdad («verdadero» o «falso»).

Las preposiciones están conectadas mediante conectivas lógicas que son tratadas como funciones de verdad; es decir, como funciones que toman conjuntos de valores de verdad y devuelven, también, valores de verdad. Por ejemplo, la conectiva «y» es una función que devolverá «verdadero» cuando tome dos valores de verdad verdaderos y, en el resto de casos, devolverá «falso». La diferencia entre los distintos conectores lógicos es el valor de verdad devuelto según las combinaciones de valores que recibe.

2.2. Lógica de predicados

La lógica de predicados, o lógica de primer orden, aumenta la capacidad de expresión de la lógica proposicional. Un predicado es una expresión lingüística que puede conectarse con una o varias expresiones para formar una oración, pero para la lógica de predicados es una función que recibe expresiones como argumentos. La figura 1 muestra algunas oraciones en lenguaje natural y su correspondencia en lógica de predicados.

- «Claire es una gata» $\rightarrow gata(Claire)$
- «Juan juega con Lorena» $\rightarrow juega(Juan, Lorena)$
- «Juan es rico» $\rightarrow rico(Juan)$
- «x es rico» $\rightarrow rico(x)$

Figura 1: Ejemplos de predicados, constantes y variables en lógica de predicados

Existen constantes de individuos que hacen referencia a entidades determinadas y variables de individuo que no tienen referencias determinadas; por tanto, su referencia

varía con el contexto (figura 1). Los cuantificadores son operadores sobre conjuntos de individuos que indican que una condición se verifica, por ejemplo: los cuantificadores universales y existenciales. Los predicados también pueden conectarse con las conectivas de la lógica proposicional.

2.3. Lógica no monotónica

Los sistemas lógicos tradicionales tienen una relación de consecuencia monotónica; es decir, las nuevas fórmulas añadidas a una teoría no reducen el número de consecuencias (lo conocido no se ve modificado por el nuevo conocimiento). Esto impide trabajar con razonamientos por incertidumbre, probabilidad o la contradicción de creencias. La lógica no monotónica permite trabajar con estos criterios, ya que el nuevo conocimiento puede contradecir al viejo y, por tanto, reducirlo. Esto permite suponer hechos y modifcarlos cuando se tiene el conocimiento.

2.4. Lógica modal

La lógica modal añade las expresiones «es necesario que» y «es posible que» a la lógica de predicados. Estas expresiones califican la verdad de los juicios. Extensiones de la lógica modal añaden nuevas expresiones para calificar, como la lógica temporal, en la que el valor de la verdad de las expresiones es dependiente del tiempo; o la deóntica, que agrega la lógica de las normas.

3. Prolog

El lenguaje de programación lógica más destacable es Prolog. Prolog fue creado por Alain Colmerauer y Phillippe Roussel a principios de los 70. Se trata de un lenguaje de programación lógica de propósito general cuya finalidad original era realizar procesamiento de lenguaje natural para el idioma francés.

3.1. Cláusulas de Horn

El Prolog puro estaba originalmente restringido al uso de cierto tipo de fórmulas lógicas conocidas como cláusulas de Horn, esto es, implicaciones lógicas con un único consecuente.

$$(p \wedge q \wedge r) \rightarrow s$$

Figura 2: Cláusula de Horn en forma de implicación lógica. A la derecha encontramos una única variable.

La particularidad de estas cláusulas es que expresan una relación de causalidad entre los antecedentes y el consecuente, permitiendo demostrar la veracidad lógica del consecuente si se verifica cada uno de los antecedentes. En el ejemplo de la figura 2 s se cumple si lo hacen p, q y r.

Programar en Prolog implica definir en primer lugar una base de hechos, que pueden ser átomos o proposiciones sobre estos, y reglas, las mencionadas cláusulas de Horn. Un átomo es un nombre definido en Prolog que no referencia ningún otro valor y sin significado propio, por ejemplo carlos, 'Pato' o 'universidade da Coruña'. Un ejemplo de una base de conocimiento se presenta en la figura 3.

```
% atomo victor: victor existe
victor.
% victor es alto
alto(victor).
% si llueve (antecedente), esta mojado (consecuente)
mojado :- llueve.
% si el dia X es lluvioso y soleado, habra arcoiris
arcoiris(X) :- lluvioso(X), soleado(X).
```

Figura 3: Definición de hechos atómicos, proposiciones y reglas en Prolog.

Una vez definida esta base de conocimiento, es posible plantear cuestiones sobre la misma al intérprete de Prolog. Las figuras 4 y 5 presentan ejemplos de ejecución de consultas sobre una base de hechos de Prolog.

3.2. Unificación y backtracking

La ejecución de código en Prolog se guía por dos mecanismos, la unificación y el backtracking. Mediante la unificación de una cláusula se determinan los objetivos, las condiciones que forman el antecedente. Cada objetivo especifica un conjunto de cláusulas que podrían verificarlo, estas cláusulas se denominan puntos de elección. Si, tras su ejecución, uno de los puntos de elección prueba ser falso, se deshace la ejecución y se selecciona el siguiente

```
% base de hechos

% mujeres
mujer(sara).
mujer(selena).
mujer(silvia).
mujer(soraya).
mujer(susana).

% varones
hombre(jaime).
hombre(jeronimo).
hombre(jimeno).
hombre(jorge).
hombre(julian).

% parentescos explicitos: padres(hijo, progenitor1, progenitor2)
padres(susana, jaime, sara).
padres(julian, jeronimo, selena).
padres(silvia, julian, susana).
padres(jimeno, jorge, soraya).

% parentescos derivados (logicamente deductibles)
hija(X,Y) :- mujer(X), padres(X,Y,_).
hija(X,Y) :- mujer(X), padres(X,_,Y).
hijo(X,Y) :- hombre(X), padres(X,Y,_).
hijo(X,Y) :- hombre(X), padres(X,_,Y).
abuelx(X,Z) :- padres(Z,Y,_), padres(Y,X,_).
abuelx(X,Z) :- padres(Z,_,Y), padres(Y,X,_).
abuelx(X,Z) :- padres(Z,Y,_), padres(Y,_,X).
abuelx(X,Z) :- padres(Z,_,Y), padres(Y,_,X).
abuelo(X,Y) :- hombre(X), abuelx(X,Y).
abuela(X,Y) :- mujer(X), abuelx(X,Y).
```

Figura 4: Definición de una base de hechos.

```
% query: quien es abuela de silvia?
?- abuela(Abuela, silvia).
Abuela = sara ;
Abuela = selena ;
false.

% query: es jimeno hijo de susana?
?- hijo(jimeno, susana).
false.

% query: de quien es hijo jimeno?
?- hijo(jimeno, Progenitor).
Progenitor = jorge ;
Progenitor = soraya ;
false.

query: que hijos varones tiene selena?
?- hijo(Hijo, selena).
Hijo = julian.
```

Figura 5: Consultas sobre la base de hechos.

punto de elección. Este proceso, denominado backtracking, se repite hasta que una de estas cláusulas resulte ser cierta o todos los puntos de elección resultan falsos.

3.3. Prolog como lenguaje de programación

Prolog provee un sistema de tipos dinámico. Estrictamente hablando, el único tipo considerado en Prolog es el término, que puede representar constantes, variables o valores compuestos, cláusulas incluídas. El estándar de prolog divide las constantes en varios subtipos para restringir el uso de distintas operaciones a los datos correctos, a saber números, que pueden ser de punto flotante o enteros, y átomos, los valores simbólicos introducidos en la sección 3.1

Prolog trabaja únicamente con dos "scopes" posibles. Uno global, el de la base de reglas, accesible desde cualquier punto del programa y desde la línea de comandos del intérprete, y otro local a la cláusula en la que se llama un nombre.

A nivel de estructuras de datos, Prolog introduce listas recursivas, a la manera de Lisp; parejas de elementos y listas asociativas, implementadas con árboles binarios balanceados.

4. Alternativas a Prolog

Otros lenguajes ofrecen distintos enfoques para la programación lógica pero se basan aproximadamente en los mismos principios básicos que Prolog. No son infrecuentes versiones relajadas o restringidas de Prolog y extensiones al propio lenguaje con *syntactic sugar* destinado a facilitar el uso de uno u otro tipo de lógica. A continuación examinamos algunos lenguajes de mérito propio así como extensiones de Prolog de interés.

4.1. Maude System

Maude es un lenguaje de programación lógica de propósito general y software libre desarrollado por el SRI International: un instituto de investigacion de California. Provee un compilador conocido como Core Maude. Maude es un lenguaje de alto rendimiento que utiliza lógica ecuacional y de rescritura y un sistema de tipado estático. Gracias a la lógica de reescritura Maude es naturalmente reflexivo, haciendo sencillo implementar diferentes tipos de lógica con los artefactos del lenguaje.

```
fmod NUMBERS is
...
endfm
```

Figura 6: Declaración de un módulo en Maude.

Maude contempla identificadores, que son cadenas de caracteres usadas para nombrar *modules* y *sorts*. Hay varios caracteres especiales, entre ellos el espacio en blanco, que no forman parte de los caracteres disponibles pero pueden ser usados mediante secuencias de escape. Las unidades básicas de especificación y programación son los módulos, que consisten en *sorts* para nombrar los tipos, *subsorts* y *kinds* para relacionar estos tipos y los operadores. La definición de un módulo de Maude se muestra en la figura 6. Hay dos tipos de módulos:

Módulo funcional Contiene los tipos de datos y las ecuaciones sobre ellos.

Módulo de sistema Contienen las reglas de transición entre estados, además de las ecuaciones y tipos de datos.

La forma de declarar un tipo en Maude es usar la palabra clave `sort` seguida de un identificador, un espacio en blanco y un punto. Los operadores en Maude se declaran con la palabra clave `op` seguida de su identificador y una especificación de la lista de tipos de los argumentos de entrada y los resultados. Opcionalmente pueden tener una

declaración de atributo. Además de *sorts*, Maude también permite especificar *kinds*, una clase de equivalencia que engloba a los *sorts* relacionados.

```
*** declaracion de tipos
sort Nat .
sort Zero .

*** se puede utilizar la palabra sorts para hacer
*** varias declaraciones en la misma linea.
sorts Nat Zero .

*** para declarar subtipos existe la palabra clave
*** subsort con la siguiente sintaxis.
subsort Zero < Nat .
subsorts NzNat Zero < Nat .

*** Nat es un kind de Maude
*** Nat NzNat y Zero forman parte de
[Nat]
```

Figura 7: *Sorts*, *subsorts* y *kinds* en Maude.

El tipado en Maude es estático, por lo que la declaración de la variable debe estar relacionada con un *sort* en concreto. Este tipo de declaración solo afecta al lugar donde es declarada; en cambio, si se utiliza la palabra clave `var` para declarar la variable, esta afectaría a todo el módulo.

```
*** declaracion de operadores
op zero : -> Zero .
op length : List -> Nat .
op _+_: Nat Nat -> Nat

*** declara una variable con identificador N del sort Nat
N : Nat
*** declara una variable con identificador N del kind Nat
N : [Nat]

*** el scope alcanza todo el modulo
var N : Nat .
var X : [Nat] .
vars M N : nat .
```

Figura 8: Especificación de operadores y variables en Maude.

La figura 9 muestra la implementación de la función factorial sobre el conjunto de los números naturales en Maude, mediante la declaración de un módulo y la definición del operador apropiado.

```
*** importar un modulo en modo protecting impide definir
*** nuevos terminos irreducibles a los tipos definidos en
*** el modulo importado y agregar reglas de reduccion que
*** hacen que terminos que eran distintos ahora no lo sean
fmod FACTORIAL is
protecting NAT .

op _! : Nat -> NzNat .
var N : Nat .

*** eq permite definir reglas de reescritura
eq 0 ! = 1 .
eq (s N) ! = (s N) * N ! .
endfm
```

Figura 9: Ejemplo de la implementación de la función factorial sobre los números naturales.

4.2. Datalog

Datalog es un subconjunto del lenguaje Prolog empleado como lenguaje de consulta en bases de datos deductivas. A este efecto, Datalog restringe varias de las características de Prolog con el fin de explotar mejores algoritmos de evaluación de estas consultas.

1. Variables negadas en el cuerpo de una cláusula deben aparecer también sin negar en la propia cláusula.

2. Cualquier variable en la cabeza de una cláusula debe figurar también, sin negar, en el cuerpo de la misma.

3. Solo términos simples pueden aparecer como consecuente (cabecera) de una cláusula.

4. No exite operador de corte para interrumpir la ejecución de una consulta en determinado punto.

Debido a estas restricciones Datalog no es Turing-completo, pero sí es puramente decla-

```
% permitido en datalog
p(a) :- q(a), not r(a).
p(a, b, c) :- q(a), r(b), s(c).
p (a, b).

% prohibido en datalog
p(a) :- not r(a).                    % restriccion 1
p(a, b, c) :- r(b), s(c).            % restriccion 2
r(p(a, b), q(a, b)).                 % restriccion 3
```

Figura 10: Restricciones de Datalog ejemplificadas.

rativo. La ausencia de un operador de corte, junto con la independencia del orden de declaración de las cláusulas, tiene como consecuencia la completa abstracción del control de flujo en la ejecución de consultas de Datalog. Otra propiedad interesante derivada de las restricciones de Datalog es que la ejecución siempre se puede realizar de forma bidireccional.

Datalog goza de cierta popularidad a día de hoy. Existen proyectos, tanto de software libre como comercial/privativo, que lo utilizan como lenguaje para numerosas tareas relacionadas con la consulta de bases de conocimiento. Entre los más reseñables encontramos Datomic, una base de datos transaccional y distribuída construída sobre Clojure y Datalog; SociaLite, un dialecto de Datalog de alto rendimiento para análisis de grafos en minería de datos diseñado en la universidad de Stanford; SecPAL, un lenguaje de políticas de seguridad de Microsoft Research y una amplia gama de dialectos con diferentes adiciones al lenguaje original.

4.3. Clingo

Clingo, es un lenguaje y herramienta de la suite Potassco, Postdam Answer Set Solving Collection, desarrollada por la universidad de Postdam. Integra un *grounder*, software que toma conjuntos de expresiones en lógica de predicados y las traslada a la lógica proposicional, eliminando las variables; y un *solver*, que calcula los conjuntos respuesta que satisfacen las proposiciones ofrecidas por el grounder basándose en una técnica conocida como *answer set programming*. La sintaxis de Clingo se basa en la de Prolog. A continuación veremos cómo resolver un problema en Clingo.

Clingo requiere dos especificaciones para funcionar. Por un lado una base de hechos que describa el estado inicial del problema a resolver, conocida como instancia. Se muestra la instancia del estado inicial de las torres de Hanoi permitiendo 15 movimientos en la figura 11. Por otro, y en un archivo independiente, la especificación del problema en la

```
peg(a;b;c).                 % valores aceptables para las varas
disk(1..4).                 % valores aceptables para los discos
init_on(1..4,a).            % estado inicial del problema
goal_on(1..4,c).            % estado final
moves(15).                  % numero maximo de movimientos
```

Figura 11: Instancia (*instance*) del problema de las Torres de Hanoi en Clingo.

forma de reglas que permiten navegar el espacio de estados, llamada codificación. Para el mismo problema, las torres de Hanoi, se muestra su codificación en la figura 12.

```
% Generate : descripcion de soluciones candidatas
{ move(D,P,T) : disk(D), peg(P) } = 1 :- moves(M), T = 1..M.

% Define : propiedades auxiliares de una solucion candidata
move(D,T) :- move(D,_,T).
on(D,P,0) :- init_on(D,P).
6 on(D,P,T) :- move(D,P,T).
on(D,P,T+1) :- on(D,P,T), not move(D,T+1), not moves(T).
blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T) :- blocked(D,P,T), disk(D).

% Test : reglas para verificar la validez de una s. candidata
:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- goal_on(D,P), not on(D,P,M), moves(M).
:- { on(D,P,T) } != 1, disk(D), moves(M), T = 1..M.

% Display : predicados que interesa mostrar a la salida
#show move/3.
```

Figura 12: Codificación (*encoding*) del problema de las Torres de Hanoi en Clingo.

Guardando el código de los archivos con extensión `.pl` e introduciéndolos en clingo para su procesamiento obtenemos el contenido de la figura 13. Para obtener este resultado Clingo trabaja generando soluciones plausibles mediante la descripción del bloque *generate* y contrastando su validez con los predicados decapitados del bloque *test*, en los que el consecuente suprimido sería la solución candidata que se está probando. De este modo es posible realizar una exploración del espacio de estados hasta alcanzar uno o más modelos que satisfaga las restricciones del problema.

```
clingo version 4.4.0
Reading from toh_ins.lp ...
Solving...
Answer: 1
move(4,b,1) move(3,c,2) move(4,c,3) move(2,b,4) \
move(4,a,5) move(3,b,6) move(4,b,7) move(1,c,8) \
move(4,c,9) move(3,a,10) move(4,a,11) move(2,c,12) \
move(4,b,13) move(3,c,14) move(4,c,15)
SATISFIABLE

Models : 1+
Calls : 1
Time : 0.017s (Solving: 0.01s 1st Model: 0.01s \ Unsat: 0.00s)
CPU Time : 0.010s
```

Figura 13: Solución proporcionada por Clingo al problema de las torres de Hanoi.

Clingo añade sintaxis y funcionalidades apropiadas para resolver problemas combinatorios complejos. Incluye, por ejemplo, un modo de especificar heurísticas para guiar la exploración del espacio de soluciones candidatas; la posibilidad de definir reglas con consecuentes disyuntivos, que permiten trabajar en situaciones en las que existen dependencias cíclicas entre proposiciones y muchos otros constructos útiles.

5. Dominios de aplicación

Si bien los lenguajes de programación lógica no son de uso común en proyectos de ingeniería de software y están más vinculados a la investigación y la academia, en algunos campos las particularidades de estos lenguajes los convierten en idóneos para resolver problemas. A continuación se describen algunos de estos dominios de aplicación.

5.1. Procesamiento de lenguaje natural

El primero de ellos, el procesamiento del lenguaje natural, es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. Se ocupa de la formulación e investigación de mecanismos computacionalmente eficaces para la comunicación entre personas y máquinas por medio de la lengua hablada o escrita. Como ejemplo tenemos Clarissa, un navegador

de procedimientos aeroespaciales completamente operado por voz que permite a los astronautas ser más eficientes con sus manos y ojos y prestar toda su atención a las tareas concretas mientras interactúan con la el sistema mediante el lenguaje hablado. Clarissa ha sido desarrollado en Prolog por la NASA.

5.2. Demostración automática, verificación formal y programación concurrente

Los lenguajes lógicos también son útiles en la demostración automatizada de teoremas, permitiendo una expresión de los mismos cercana a su formulación matemática. La verificación formal del software, basada en el uso de la lógica para proporcionar una demostración de que el programa realiza la tarea esperada sin posibilidad de error, y la programación concurrente, debido al aspecto declarativo de la programación lógica; son otros dos importantes campos de aplicación. El autor de Erlang escribió el primer intérprete de su lenguaje utilizando Prolog.

5.3. Inteligencia artificial y sistemas expertos

En el campo de la inteligencia artificial, Prolog es un lenguaje importante para la elaboración de sistemas expertos. Un sistema experto es un sistema computacional que emula la habilidad de toma de decisiones de un experto humano. Los sistemas expertos están diseñados para resolver problemas complejos mediante razonamiento sobre el conocimiento, representado mayormente como reglas `if-then` y no como instrucciones convencionales de código. Ejemplos de este tipo de aplicación son DealBuilder, un constructor automático de documentos legales, Arezzo, un sistema experto para apoyar la toma de decisiones clínicas, e InFlow, un analizador de grafos sociales para la detección de terroristas.