Representación del conocimiento

Representación lógica

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

19 de octubre de 2021

Repaso de lógica de predicados

1. Repaso de lógica de predicados
2. Algoritmo de sustitución
3. Prolog
4. Búsqueda de las demostraciones
5. Otras características

Temas

1. Repaso de lógica de predicados
   - Cálculo de predicados
   - Definición de “aterrizado”

Componentes

1. *Términos* $\mathbb{T} = (\mathbb{C} \bigcup \mathbb{V})_+$[1].
   1. Constantes $\mathbb{C} = \{a, b, c, a_1, b_1, c_1 ...\}$
   2. Variables $\mathbb{V} = \{x, y, z, x_1, ...\}$
   3. Símbolos funcionales $\mathbb{F} = f^n_m$ con $n, m \geqslant 1$, con n el número de argumentos, y m el identificador o $\mathbb{F} = \{f, g, h, ...\}$.
      Si $t_1, ..., t_n \in \mathbb{T}$ y $f^n_m \in \mathbb{F}$ con $n \geqslant 0 \Rightarrow f^n_m(t_1, ..., t_n) \in \mathbb{T}$
2. *Fórmulas* (Pred)
   1. $\bot$ y $\top$.
   2. Símbolos para predicados $\mathbb{P} = P^n_m$ con $n, m \geqslant 1$.
      Fórmulas atómicas (Pred_0). Si $t_1, ..., t_n \in \mathbb{T}$ y $P^n_m \in \mathbb{P} \Rightarrow P^n_m(t_1, ..., t_n) \in \mathrm{Pred}_0$.
   3. Funciones generadoras.
      1. Conectivas lógicas $= \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$.
      2. Cuantificadores $= \{\forall, \exists\}$.
         Si $x \in \mathbb{V}$ y $\alpha \in \mathrm{Pred}$[2] $\Rightarrow (\forall x.\alpha)$ y $(\exists x.\alpha) \in \mathrm{Pred}$.

El *cálculo de Predicados* Pred es $(\mathrm{Pred}_0)_+$ bajo funciones en 2.3.1 y 2.3.2.

[1] Cerradura inductiva

[2] α es una fórmula.

Ejemplos de fórmulas

Fórmulas atómicas:

$$P_1^2(a, x_1)$$
$$P_2^3(a, y, f_1^2(a_1, y_1))$$

Fórmulas compuestas:

$$\neg P_1^2(a, x_1)$$
$$P_1^2(a, x_1) \Rightarrow P_2^3(a, x_1, f_1^2(a, x_1))$$
$$\forall x.P_1^2(a, x)$$
$$P_3^1(x) \wedge P_1^2(x, y) \vee P_1^2(y, z)$$

Temas

1. Repaso de lógica de predicados
   - Cálculo de predicados
   - Definición de “aterrizado”

Definición de "aterrizado"

Definición

- Un **término** o **átomo** se encuentra *aterrizado* si y sólo si no contiene variables.
- Una **fórmula** se encuentra aterrizada si y sólo si no contiene cuantificadores ni variables.
- Una fórmula A es un *ejemplar aterrizado* de una fórmula libre de cuantificadores A si y sólo si puede ser obtenida de A sustituyendo términos aterrizados por las variables libres en A.

Algoritmo de sustitución

1. Repaso de lógica de predicados
2. Algoritmo de sustitución
3. Prolog
4. Búsqueda de las demostraciones
5. Otras características

Algoritmo de sustitución

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ *a una fórmula* $\varphi \in \mathrm{Pred}$, *denotada* $\varphi[\vec{x} := \vec{t}]$ *se define como la fórmula obtenida al reemplazar simultáneamente todas las presencias libres de* x_i *en* φ *por* t_i, *verificando que este proceso no capture posiciones de variables libres.*

Sustitución I

La aplicación de una sustitución a una fórmula φ se define recursivamente como sigue:
Casos base:

$$\bot[\vec{x} := \vec{t}] = \bot$$
$$\top[\vec{x} := \vec{t}] = \top$$
$$x[\vec{x} := \vec{t}] = t$$
$$y[\vec{x} := \vec{t}] = y \text{ si } y \neq x$$

Casos recursivos:
Símbolos funcionales

$$f(t_1, ..., t_n)[\vec{x} := \vec{t}] = f(t_1[\vec{x} := \vec{t}], ..., t_m[\vec{x} := \vec{t}]) \text{ con } n \geqslant 0$$

Sustitución II

Fórmulas atómicas

$$P(t_1, ..., t_n)[\vec{x} := \vec{t}] = P(t_1[\vec{x} := \vec{t}], ..., t_n[\vec{x} := \vec{t}]) \text{ con } n \geqslant 1$$
$$(t_1 = t_2)[\vec{x} := \vec{t}] = t_1[\vec{x} := \vec{t}] = t_2[\vec{x} := \vec{t}]$$

Fórmulas generadas

$$(\neg\varphi)[\vec{x} := \vec{t}] = (\neg\varphi[\vec{x} := \vec{t}])$$
$$(\varphi \wedge \psi)[\vec{x} := \vec{t}] = (\varphi[\vec{x} := \vec{t}] \wedge \psi[\vec{x} := \vec{t}])$$
$$(\varphi \vee \psi)[\vec{x} := \vec{t}] = (\varphi[\vec{x} := \vec{t}] \vee \psi[\vec{x} := \vec{t}])$$
$$(\varphi \Rightarrow \psi)[\vec{x} := \vec{t}] = (\varphi[\vec{x} := \vec{t}] \Rightarrow \psi[\vec{x} := \vec{t}])$$
$$(\varphi \Leftrightarrow \psi)[\vec{x} := \vec{t}] = (\varphi[\vec{x} := \vec{t}] \Leftrightarrow \psi[\vec{x} := \vec{t}])$$

Sustitución III

$$(\forall y \varphi)[\vec{x} := \vec{t}] = \forall y(\varphi[\vec{x} := \vec{t}]) \text{ si } y \notin \vec{x} \cup Var(\vec{t})$$
$$(\exists y \varphi)[\vec{x} := \vec{t}] = (\exists y \varphi[\vec{x} := \vec{t}]) \text{ si } y \notin \vec{x} \cup Var(\vec{t})$$

Si $y \in \vec{x}$ la variable ligada en la fórmula debe ser renombrada antes de realizar la sustitución.

Prolog

1. Repaso de lógica de predicados
2. Algoritmo de sustitución
3. Prolog
4. Búsqueda de las demostraciones
5. Otras características

Temas

3 Prolog
- Filosofía
- Instalación y uso básico
- Representación de la información
- Unificación

Prolog *(Programming in logic)*

- `Prolog` es un lenguaje de programación *declarativa*, que se resume en su eslogan: *"Dí en **qué** consiste el problema en lugar de **cómo** resolverlo."*
- Está hecho para computos simbólicos, no numéricos. Especialmente para resolver problemas que involucren **objetos** y **relaciones** entre objetos.
- Acepta *hechos* y *reglas* como un conjunto de **axiomas** y la *consulta* del usuario como un *teorema conjeturado*, entonces intenta demostrar este teorema. Bratko 1990
- El estilo de este lenguaje propicia que el programador piense en términos de *metas*.

Aplicaciones

- Este lenguaje es especialmente adecuado para aplicaciones que requieren la representación y análisis de estructuras ricas. Por ejemplo:
  - Lingüística computacional
  - Inteligencia artificial
  - Sistemas expertos
  - Biología molecular
  - La red semántica (*semantic web*)

Temas

3 Prolog
- Filosofía
- Instalación y uso básico
- Representación de la información
- Unificación

Instalación y uso

Para instalar en Ubuntu:

```
$ sudo apt install swi-prolog
```

Para ejecutar el intérprete:

```
$ swipl
```

Cargar un archivo llamdo `bloques.pl` con declaraciones:

```
1 ?- ['bloques'].
```

Listar lo que está en memoria:

```
1 ?- listing.
```

Para salir:

```
1 ?- halt.
```

Temas

3 Prolog
- Filosofía
- Instalación y uso básico
- Representación de la información
- Unificación

Cláusulas

- Mientras que los lenguajes **procedimentales** se componen de **enunciados** y **bloques**,
- este lenguaje **declarativo** se compone de *cláusulas*, que terminan en . .
  Las hay de tres tipos:
  - Hechos.
  - Reglas.
  - Consultas.

Hechos, reglas y consultas I

Las construcciones básicas en `Prolog` son:

1. Una *base de conocimiento* (o base de datos), especificada mediante dos tipos de *cláusulas*, que se almacenan en un archivo `.pl`:
   - *Hechos*. Proposiciones y predicados. También se les puede considerar reglas sin cuerpo.

Código 1: hechos.pl

```
1  piso.
2  bloque(a).
3  bloque(b).
4  encima(a, piso).    % a está directamente arriba del piso.
5  encima(b, a).       % b está directamente arriba de a.
6  encima(c, b).
```

   - *Reglas*. Compuestas por:

Hechos, reglas y consultas II

```
<regla>   ::= <cabeza>:-<cuerpo>
<cabeza>  ::= <hecho>
<cuerpo>  ::= <hecho> ( (,|;) (<hecho>))*
```

con el operador and (,) teniendo mayor precedencia que or (;).
Si las premisas en el cuerpo se cumplen dados los datos en la base de conocimiento, entonces la cabeza es verdadera también (**Modus ponens**).

que expresan **datos** y **relaciones** entre ellos.

Código 2: hechos.pl

```
1 debajo(X,Y) :- encima(Y,X).
2 arriba(X,Y) :- encima(X,Y).                % Caso base: justo encima
3 arriba(X,Y) :- encima(X,Z), arriba(Z,Y).   % Definición recursiva
```

Alternativamente, la segunda regla se puede escribir:

Hechos, reglas y consultas III

Código 3: hechos.pl

```
1 arriba(X,Y) :- encima(X,Y); encima(X,Z), arriba(Z,Y).
```

Nota: Para agregar hechos directamente en el intérprete se usa `assert`:

```
1 ?- assert(abajo(X,Y) :- arriba(Y,X)).
```

2. *Consultas* que plantean preguntas sobre la información almacenada.
   - Consisten en una o más *metas* donde `Prolog` busca los valores que se deben asignar a las variables para que las premisas planteadas se satisfagan, de otro modo se dice que hubo un **fallo**.
   - Si hay más de una solución, el usuario puede continuar pidiéndolas hasta que se hayan listado todas. (Por ejemplo, presionando la barra espaciadora hasta que se muestren todas.)

Hechos, reglas y consultas IV

- La respuesta a una consulta que no se siga directamente de los hechos y reglas en la base de conocimiento será considerada **falsa**.

```
1 ?- abajo(X,b).      % ¿Quién está abajo de b?
2 X = a.
```

Átomos

Los elementos más sencillos en `Prolog` son los *átomos*:

- Cadenas de caracteres, dígitos y algunos en `+-*/<>=:.&_~`, que **comienzan con minúsculas**, como `bloque`, `x`.
- Secuencias de caracteres entre comillas simples, como `'Daria'`, `'Louis Pasteur'`.
- Una cadena de caracteres especiales, como `@=`, `===>`, `.:.`, cuidado con los operadores `,,` `;` y `:-`.

Términos

Hay tres tipos de *términos*:

- *Constantes*: son **números** o **átomos**.
- *Variables*: Cadenas de caracteres, dígitos y guiones bajos, que **comienzan con mayúsculas** o con **_**. Más la **variable anónima**: **_**.
  El alcance de una variable es sólo una clásula.
- *Términos complejos*:

```
<término complejo> ::= <functor>(<parámetro>(,<parámetro>)*)
<functor>          ::= <átomo>
<parámetro>        ::= <constante> | <variable> | <término complejo>
```

  Por ejemplo: `parentesco(abuelito, padre(padre(X)))`.

Temas

3 Prolog
- Filosofía
- Instalación y uso básico
- Representación de la información
- Unificación

Algoritmo de unificación de Prolog

Casos base:

1. Si `term1` y `term2` son **constantes**, se unifican si y sólo si son la misma constante.
2. Si `term1` es una **variable** y `term2` es un término cualquiera se unifican y se realiza la asignación ¡sin excepciones!:

$$\mathrm{term1} := \mathrm{term2}$$

Si `term2` es una **variable** y `term1` es un término cualquiera se unifican y se realiza la asignación:

$$\mathrm{term2} := \mathrm{term1}$$

Si ambas son **variables** se dice que son iguales.

Casos recursivos:

3. Si `term1` y `term2` son **términos complejos** se unifican si y sólo si:
   1. Tienen el mismo **functor** y la misma *aridad*, es decir, número de parámetros.
   2. Todos su argumentos correspondientes se unifican.
   3. Las asignaciones a variables son compatibles.
4. Dos términos se unifican si y sólo si se unifican en alguno de los casos anteriores.

Por ejemplo:

Código 4: Verificando unificación mediante los operadores: *unifica* = y *no unifica* \=

```
?- k(s(g),Y) = k(X,t(X)).
Y = t(s(g)),
X = s(g).

?- g(a,B,c) \= g(A,b,C).
false.
```

Unificación estándar vs unificación en Prolog

Código 5: Unificación en Prolog

```
% No revisa ocurrencias de la variable en el término con el que la unifica.
?- father(X) = X.
X = father(X).
```

Código 6: Unificación estándar en Prolog

```
% Revisa ocurrencias de la variable en el término con el que la unifica.
?- unify_with_occurs_check(father(X), X).
false.
```

Búsqueda de las demostraciones

1. Repaso de lógica de predicados
2. Algoritmo de sustitución
3. Prolog
4. Búsqueda de las demostraciones
5. Otras características

Búsqueda en profundidad con retroceso

Dada una consulta p, prolog intentará lo siguiente:

1. Unificar p con un **hecho** o la **cabeza de una regla** de **arriba** a **abajo**.
2. Si logró unificarla con un hecho, termina.
3. Si la unifica con la cabeza de una regla, ahora busca satisfacer una **lista de metas** conformadas por el cuerpo de la regla.
   1. La búsqueda es de izquierda a derecha.
   2. Renombra las variables en la regla, de modo que ninguna se llame igual que las variables en p.
   3. Reemplaza p por el cuerpo renombrado de la regla en la lista de metas a satisfacer.

Es posible revisar el proceso paso por paso:

```
1 ?- trace.
2 true.
3
4 ?- notrace.
5 true.
```

Otras características

1. Repaso de lógica de predicados
2. Algoritmo de sustitución
3. Prolog
4. Búsqueda de las demostraciones
5. Otras características

Assert

Es posible resolver un problema y almacenar la solución en la base de datos, de modo que la próxima vez se devuelva la solución encontrada, en lugar de volverla a calcular:

Código 7: Regla para resolver algún problema.

```
1 resuelve(Problema, Solución).
```

- En la línea siguiente el problema se resuelve al atender la primer premisa.
- El valor calculado se sustituye en la segunda y se almacena en la base de datos.

Código 8: Agrega al inicio.

```
1 ?- resuelve(Problema, Solución), asserta(resuelve(Problema, Solución)).
```

Referencias I

- Ayala-Rincón, Mauricio y Flávio L. C. de Moura (2017). «Derivations and Proofs in the Predicate Logic». En: *Applied Logic for Computer Scientists : Computational Deduction and Formal Proofs*. Cham: Springer International Publishing, págs. 43-72. ISBN: 978-3-319-51653-0. DOI: 10.1007/978-3-319-51653-0_2. URL: https://doi.org/10.1007/978-3-319-51653-0_2.
- Blackburn, Patrick, Johan Bos y Kristina Striegnitz (2014). *Learn Prolog Now!* URL: https://github.com/LearnPrologNow/lpn,http://www.let.rug.nl/bos/lpn//index.php (visitado 22-03-2021).
- Bratko, Ivan (1990). *Prolog Programming for Artificial Intelligence*. Addion-Wesley Pusblishing Company. 423 págs. URL: https://silp.iiita.ac.in/wp-content/uploads/PROLOG.pdf (visitado 24-03-2021).

Licencia