PROLOG

Inteligencia Artificial

Universidad de Talca, II Semestre 2005

Jorge Pérez R.

Introducción a PROLOG

- PROLOG es un lenguaje **interpretado** basado en la lógica de predicados de primer orden.
- Puede ser visto como un lenguaje de programación o como un demostrador mecánico de teoremas.
- Fue implementado a principios de los años 70 por Alain Colmerauer y junto con Lisp son los lenguajes que históricamente se han utilizado para construir aplicaciones en gran parte de las ramas de IA.
- Hoy existen muchas implementaciones de PROLOG, la que nosotros usaremos es SWI-Prolog desarrollado en la Universidad de Amsterdam. Es una versión libre y puede ser descargada para distintos ambientes.

  `http://www.swi-prolog.org`

LPOP a la rápida...

- (supondremos cierta familiaridad con el tema)
- A grandes razgos la lógica de predicados de primer orden (LPOP) es un lenguaje formal para expresar cierta parte del lenguaje matemático.
- Sus componentes principales son los *predicados* (atributos, relaciones), *constantes*, *variables*, y *cuantificadores* ($\forall$, $\exists$).
- Ejemplo:

  "Todos los hombres son mortales,
  y dado que Sócrates es un hombre,
  podemos concluir que Sócrates es mortal."

- Podemos modelarla con la siguiente fórmula:

  $$(\forall X(hom(X) \rightarrow mor(X)) \wedge hom(socrates)) \rightarrow mor(socrates)$$

  que es una fórmula de LPOP donde hom y mor son predicados, X es una variable cuantificada y $socrates$ es una constante... la fórmula resulta ser **siempre verdadera**.
- Desde otro punto de vista podríamos pensar en la misma fórmula anterior pero en un esquema de consulta donde:

  1. $\forall X(hom(X) \rightarrow mor(X))$
  2. $hom(socrates)$

  son *oraciones*, y

  Q: ¿$mor(socrates)$?

  es una consulta.
- Luego si suponemos las oraciones 1 y 2 como hechos verdaderos, y preguntamos por la consulta Q, la respuesta debiera ser SI.
- PROLOG sigue un esquema similar de hechos y consultas.

PROLOG

- PROLOG es un lenguaje interpretado que nos permite especificar hechos y reglas, similares a los de LPOP, y luego hacer consultas sobre ellos.
- El interprete presenta un *prompt* para iteractuar `?-`.
- La interacción con el interprete es a partir del ingreso de hechos y reglas de manera directa

  ```
  ?- [user].
  |: aquí se ingresan los hechos y reglas uno a uno...
  ```

  o cargando desde un archivo

  ```
  ?- ['ejemplo.pl'].
  ```

  Donde 'ejemplo.pl' es el nombre de un archivo con los hechos y reglas a cargar.
- El prompt siempre está a la espera de consultas a responder.
- Veremos con ejemplos las distintas formas de interacción y los conceptos fundamentales de PROLOG.

Hechos

- Los hechos son el tipo básico de oraciones de un programa PROLOG. Supongamos que queremos modelar información genealógica acerca de un grupo de personas. Para esto utilizaremos los predicados o relaciones `padre` y `madre` sobre ciertas constantes.

```
padre(juan, amanda).
madre(ximena, amanda).
madre(laura, juan).
padre(andres, juan).
padre(patricio, bonifacio).
padre(juan, patricio).
padre(juan, ana).
madre(ximena, ana).
```

- Esto lo podemos ingresar haciendo

```
?- [user].
|: padre(juan, amanda).
|: madre(ximena, amanda).
|: ....
```

  o cargando desde un archivo

```
?- ['genealogico.pl'].
```

- Los hechos anteriores representan un modelamiento del mundo en donde, por ejemplo "Juan es el padre de Amanda".
- Algunas observaciones de sintaxis: los hechos son predicados *instanciados* en valores constantes, en nuestro caso `juan` y `amanda` son valores constantes.
- En PROLOG siempre las constantes comienzan con minúscula, según esto la oración `padre(Juan, Ana)` no sería un hecho de PROLOG más adelante veremos qué significa.

Consultas

- Dados los hechos agregados al interprete PROLOG, podemos hacer consultas como por ejemplo

```
?- madre(ximena,ana).
Yes
?- padre(juan, ximena).
No
```

- Los `Yes` y `No` son las respuestas que entrega PROLOG luego de presionar [enter]. Note que son consistentes con los hechos agregados.
- Consultas más interesantes involucran el uso de variables (en el sentido de la lógica LPOP que es distinta al de una variable usual de un lenguaje de programación).
- Todas las variables en PROLOG comienzan con mayúsculas.
- ¿Cuál puede ser el significado de la siguiente consulta?

```
?- padre(X, juan).
```

- En la anterior consulta `X` representa a una variable y `juan` a un valor constante.
- Cuando aparece una variable en una consulta, PROLOG la interpreta cómo una pregunta *existencial*, o sea sería equivalente a la consulta en LPOP

$$¿\exists X\ padre(X, juan)?$$

- Entonces la respuesta de PROLOG debiera ser `Yes` según nuestros hechos, pero la respuesta va más allá...

```
?- padre(X, juan).
X = andres
```

Consultas (cont.)

- La respuesta fue `X = andres` porque PROLOG determinó que si la variable `X` se reemplazaba por el valor constante `andres` la consulta era verdadera a partir de los hechos. En este caso PROLOG responde más que sólo `Yes` o `No`.
- En el ejemplo anterior después de responder `X = andres` PROLOG espera una acción del usuario, este puede preionar `[enter]` o digitar `;` obteniendo las siguientes respuestas:

```
?- padre(X, juan).
X = andres
Yes
```

si presiona `[enter]`, o

```
?- padre(X, juan).
X = andres;
No
```

si digita `;`.

- Para responder consultas PROLOG realiza un proceso de búsqueda entre los valores constantes y los predicados.
- Si digitamos `;` se le pide a PROLOG que siga buscando, si en el proceso de búsqueda PROLOG no encuentra un valor constante para satisfacer la consulta la respuesta final es `No`, por ejemplo

```
?- madre(ximena, X).
X = amanda;

X = ana;
No
```

Consultas (cont.)

- También se pueden hacer consultas que involucren más de una variable.

```
?- madre(X,Y).
X = ximena,
Y = amanda
Yes
```

- Del mundo modelado por nuestros hechos es fácilmente deducible que "el abuelo paterno de Ana es Andrés". ¿cómo podemos consultar esto en PROLOG?
- Primero, sabemos que "Andrés es el abuelo paterno de Ana" porque entre nuestros hechos existen los hechos

```
padre(andres,juan).
padre(juan,ana).
```

- La consulta a PROLOG debiéramos hacerla pensando en preguntar si existe un abuelo paterno para `ana`, dado que si esto efectivamente ocurre PROLOG responderá `Yes` y además entregará la constante que hace verdadera la consulta.
- Si lo miramos en LPOP la consulta debiera ser algo como

$$¿\exists Z(\exists X(padre(Z,X) \wedge padre(X,ana)))?$$

- En PROLOG se sigue exactamente la misma idea:

```
?- padre(Z,X), padre(X,ana).
Z = andres
X = juan
Yes
```

- En PROLOG el símbolo ',' representa al conector de conjunción lógico $\wedge$.
- Si no nos interesa el valor de `X` podemos usar la variable `_X`.

Reglas

- Las reglas en PROLOG son oraciones que nos permiten deducir información.
- Su correspondencia en LPOP son las **cláusulas de Horn definitivas**.
- En nuestro primer ejemplo de LPOP teníamos

  1. $\forall X(hom(X) \rightarrow mor(X))$
  2. $hom(socrates)$.

- La oración 2 puede representarse por `hom(socrates)`, ¿y 1?
- Un tipo especial de oración en PROLOG son las **reglas** que sirven para representar consecuencia lógica.
- En PROLOG se usan los caracteres '`:-`' en correspondencia con el $\leftarrow$ lógico.
- La regla

  ```
  mor(X) :- hom(X).
  ```

  es la representación en PROLOG de 1.
- PROLOG implícitamente supone cuantificación universal ($\forall$) en todas las variables utilizadas en una regla particular
- Así, si ingresamos el programa PROLOG

  ```
  hom(socrates).
  mor(X) :- hom(X).
  ```

  obtenemos respuestas como

  ```
  ?- mor(socrates).
  Yes
  ```

Predicados a partir de Reglas

- En nuestro ejemplo de información genealógica nos gustaría contar con información de los abuelos (diferenciando el sexo).
- Queremos tener un predicado `abuelo(X,Y)` que sea verdadero cuando `X` sea abuelo de `Y`.
- Esta información puede ser deducida desde los hechos, lo menos práctico sería agregar nuevos hechos, mucho mejor agregamos reglas...

```
abuelo(X,Y) :- padre(X,Z), padre(Z,Y).
abuelo(X,Y) :- padre(X,Z), madre(Z,Y).
```

- Ahora podemos preguntar quién es abuelo de quién

```
?- abuelo(X,Y).
X = andres,                X = andres,
Y = amanda ;               Y = ana ;

X = andres,                X = juan,
Y = patricio ;             Y = bonifacio ;
                           No
```

- La regla anterior se podría haber definido también como

```
abuelo(X,Y) :- padre(X,Z), (padre(Z,Y); madre(Z,Y)).
```

- El símbolo ';' representa la disyunción $\vee$, el "o" lógico.
- Ejercicio: defina la relación `herm(X,Y)` para representar que `X` e `Y` son hermanos.

Predicados Recursivos

- Las reglas también pueden ser recursivas... tendrán la forma de una definición recursiva usual de matemáticas.
- Queremos definir la relación `ancestro(X,Y)` con el sentido genealógico usual.
- Para esto definimos un predicado para la relación `ancestroDirecto`.

```
ancestroDirecto(X,Y) :- (padre(X,Y); madre(X,Y)).
```

que representa el caso base de la definición recursiva.

- Luego definimos la relación `ancestro` a parir del caso base y una recursión:

```
ancestro(X,Y):- ancestroDirecto(X,Y).
ancestro(X,Y):- ancestroDirecto(X,Z), ancestro(Z,Y).
```

- Así obtenemos

```
?- ancestro(X,Y).
X = juan,                X = andres,
Y = amanda ;             Y = juan ;
......
```

- Ejercicio: modele un grafo dirigido usando constantes para los nodos (`u`,`v`, etc.) y un predicado `e(U,V)` para indicar las aristas. Un grafo particular resultará de los hechos que agregue a PROLOG.
  - Defina un predicado `camino(U,V)` para determinar si existe un camino dirigido entre dos nodos `U` y `V`.
  - Defina un predicado `ciclo(U)` para determinar si en el grafo hay un ciclo dirigido que contenga al nodo `U`.
  - ¿Puede definir un predicado `ciclo` (sin argumentos)? ¿Y un predicado `caminoNoDirigido(U,V)` para determinar si existe un camino entre `U` y `V` que no toma en cuenta la dirección de las aristas?

Sintaxis y Semántica

- Los eleméntos sintácticos básicos de un programa PROLOG son *constantes*, *variables*, *functores* y *predicados*.
- Los predicados son los elementos más importantes y ha sido el centro de nuestra discusión anterior. Cada predicado tiene cierta *aridad* (cantidad de argumentos). Si el predicado `p` tiene aridad `m` escribiremos `p/m`. A cada predicado en PROLOG se le asocia un valor de verdad `Yes` o `No`.
- Las constantes pueden ser:
  - Símbolos que comienzan con letra minúscula, por ej. `x`, `y`, `abc`, etc., símbolos encerrados en comillas simples, por ej. `'X'`, `'5'`, `'x23'`, `'a'`, etc.
  - Números: enteros y/o reales, por ej. `4`, `4.5`, etc.
  - Caracteres y textos (strings): Los caracteres y los strings se encierran entre comillas dobles, por ej. `"f"`, `"fam"`, etc.
- Las variables son símbolos que comienzan con mayúsculas o un *underscore*, por ej. `X`, `_x23`, etc. La variable representada por el símbolo `_` es especial, ya veremos su significado.
- Los functores representan a funciones, son similares sintácticamente a los predicados con la diferencia de que a ellos se les asocia un objeto en vez de un valor de verdad (no se consulta por un functor). Si `f` es un functor y `m` es su aridad escribiremos `f/m`. Veremos que los functores nos sirven por ejemplo para crear estructuras de datos en PROLOG.
- Estos tres últimos tipos de elementos se llaman *términos*.
- Diremos que un *literal* es un elemento de la forma

  ```
  p(t1,t2,...,tm)
  ```

  donde `p/m` es un predicado y cada `ti` es un término.

Programas y Consultas

- Un programa PROLOG está compuesto por un conjunto de reglas de la forma

  ```
  l0 :- l1, l2, ..., ln.
  ```

  donde cada `li` es un literal.

- Los hechos pueden verse como reglas donde sólo existe el literal de más a la izquierda (`l0:-.` o simplemente `l0.`).

- Por ejemplo un programa sería:

  ```
  padre(juan, amanda).
  padre(andres, juan).
  abuelo(X,Y) :- padre(X,Z), padre(Z,Y).
  bisabuelo(X,Y) :- abuelo(X,Z), padre(Z,Y).
  ```

- Una consulta PROLOG es simplemente una conjunción de literales (literales separados por comas)

  ```
  g1, g2, ..., gm.
  ```

- Los literales que aparecen en la consulta se llaman *objetivos*

- El interprete de PROLOG recibe como entrada un programa más una consulta. La salida será `Yes` si a partir del programa fue posible *demostrar* la consulta haciendo verdadero simultáneamente todos sus objetivos, y `No` en el caso contrario.

- Al demostrar un objetivo el interprete usa los conceptos de *sustitución* y *unificación*.