**PARCIAL LOGICO – PARADIGMAS DE PROGRAMACION - 18/10/2006**
**CURSO K2054 – HOJA 1 DE** **NOMBRE:**

---

Se nos pide hacer una aplicación Prolog que analice información acerca de las series infantiles. Tenemos la siguiente base de conocimientos:

```
infantil(scoobyDoo).
infantil(lazyTown).
infantil(barney).

violento(powerRangers).
violento(chicasSuperpoderosas).
violento(pokemon).

musical(frutillitas).
musical(chiquititas).

emite(discoveryKids, lazyTown).
emite(discoveryKids, barney).
emite(cartoonNetwork, chicasSuperpoderosas).
emite(cartoonNetwork, scoobyDoo).
```

**Se pide:**

1. agregar la siguiente información.
   - En Canal 9 emiten frutillitas y chiquititas
   - En Jetix dan todas las series violentas y frutillitas.
   - En Boomerang emiten repeticiones de todas las series violentas y musicales.
   - En Canal 11 emiten todas las repeticiones que da Boomerang y además Lazy Town.

2. resolver el predicado emiten/2, que relaciona una lista de canales con una lista de series e indica si *todos* los canales emiten *todas* las series que aparecen en la lista. Resolverlo utilizando:
   a) cláusulas recursivas **o**
   b) cláusulas forall

   Justificar la decisión tomada.

```
? emiten([jetix, cartoonNetwork], [chicasSuperpoderosas, scoobyDoo]).
No (jetix emite chicasSuperpoderosas pero no scoobyDoo)

? emiten([boomerang, canal11], [chicasSuperpoderosas, pokemon])
Yes
```

3. resolver el predicado canalParaChicos/1, que indica si un canal tiene al menos 5 series infantiles (de cualquier tipo). Resolverlo utilizando setOf / findall. Comentar dónde aparece la idea de declaratividad justificando su elección.

4. desarrollar el predicado puedeVer/2, que relaciona las distintas series de televisión que puede ver cada chico.
   - Chiara puede ver los programas que emite discoveryKids
   - Gabriel puede ver los programas musicales y los que emite jetix. **Tip:** usar el predicado append.

   Resolverlo utilizando setof / findall. El predicado puedeVer/2, ¿es inversible? Justificar.

5. Resolver el predicado canalTranqui/1, que indica si un canal no emite series violentas.

```
? canalTranqui(discoveryKids)          ? canalTranqui(cartoonNetwork)
Yes                                    No
```

a) Resolverlo utilizando forall.
b) ¿Cómo hay que definir la cláusula para que sea inversible? ¿Qué implica que sea inversible?
c) Si yo quiero generar el conjunto de canales tranquis, sin repetidos, ¿cómo hago?

6. Resolver el predicado programasEnComun/3, que relaciona las series en común que tienen dos chicos.

**PARCIAL LOGICO – PARADIGMAS DE PROGRAMACION - 18/10/2006**
**CURSO K2054 – HOJA 1 DE NOMBRE:**

---

**Soluciones**

1.

```
emite(discoveryKids, lazyTown).
emite(discoveryKids, barney).
emite(cartoonNetwork, chicasSuperpoderosas).
emite(cartoonNetwork, scoobyDoo).

emite(canal9, frutillitas).
emite(canal9, chiquititas).

emite(jetix, Serie):-violento(Serie).
emite(jetix, frutillitas).

emite(boomerang, Serie):-violento(Serie).
emite(boomerang, Serie):-musical(Serie).

emite(canal11, Serie):-emite(boomerang, Serie).
emite(canal11, lazyTown).
```

2. a)

```
emiten([], _).
emiten([Canal|RestoCanales], Series):-
   emiteTodasLasSeries(Canal, Series), emiten(RestoCanales, Series).

emiteTodasLasSeries(_, []).
emiteTodasLasSeries(Canal, [Serie|RestoSeries]):-emite(Canal, Serie),
   emiteTodasLasSeries(Canal, RestoSeries).
```

Necesito utilizar una cláusula recursiva ya que estoy relacionando dos listas (de canales y de series). Como la forma de separar una lista en cabeza y cola me lleva a una definición recursiva de lista, la forma de definir la relación que me dice si un canal emite todas las series es por ende recursiva.

**Nota para un alumno que lee la solución:** tratar de no decir “recorro” la lista está bueno para despegar la idea de recursividad respecto de un do/loop.

b) emiteTodos(Canal, ListaProgramas):- forall( member(Programa, ListaProgramas), emite(Canal,Programa)).

emiten(ListaCanales, ListaProgramas):- forall( member(Canal, ListaCanales), emiteTodos(Canal, ListaProgramas)).

3.

canalParaChicos(Canal):-setof(Serie, emite(Canal, Serie), Series), length(Series, Longitud), Longitud > 4.

La codificación del length es opcional y hasta diría que preferiría que no lo hicieran, para conservar el ejercicio lo más declarativo posible.

La declaratividad es propia del paradigma, donde sólo queda escrito qué condición tiene que cumplir (el canal tiene que emitir 5 o más series), en lugar de generar un algoritmo que realice la búsqueda de los canales en la base de conocimientos (algoritmo que queda a cargo del motor de inferencia Prolog).

4.

```
puedeVer(chiara, Series):-setof(Serie, emite(discoveryKids, Serie), Series).

puedeVer(gabriel, Series):-setof(Serie, emite(jetix, Serie), Jetixs),
   setof(Musical, musical(Musical), Musicales),
   append(Jetixs, Musicales, Series).
```

**Opcional (no pedido).** Para que no traiga duplicados, podría hacerse:

```
puedeVer(gabriel, Series):-setof(Serie, (emite(jetix, Serie), not(musical(Serie))), Jetixs),
   setof(Musical, musical(Musical), Musicales),
   append(Jetixs, Musicales, Series).
```

Como setof se aplica sobre un predicado inversible (emite, que es un hecho por lo tanto puedo hacer cualquier tipo de consulta) y append también es inversible, entonces puedeVer/2 también es inversible.

5.

a)
```
canalTranqui(Canal):- forall(emite(Canal, Serie), not(violento(Serie))).
```

b)
```
canalTranqui(Canal):-canal(Canal),forall(emite(Canal, Serie), not(violento(Serie))).
canal(Canal):-emite(Canal, _).
```
También vale generar “a mano” los canales en la base de conocimientos.

c) setof(Canal, canalTranqui(Canal), Canales).

6)
```
programasEnComun(Chico1, Chico2, Series):-
   puedeVer(Chico1, Series1),
   puedeVer(Chico2, Series2),
   interseccion(Series1, Series2, Series).
```

**Dos opciones para resolver intersección:**
```
interseccion([], _ , []).
interseccion([X|L1], L2, [X|L3]):-member(X, L2), interseccion(L1, L2, L3).
interseccion([_|L1], L2, L3):-interseccion(L1, L2, L3).

interseccion(A, B, C):-setof(X, (member(X, A), member(X, B)), C).
```

| Punto | Qué evaluamos |
|---|---|
| 1 | Uso de “y” , “o” castellano vs. and/or lógico. Definición de hechos y reglas en base de conocimientos. |
| 2 | Recursividad o Predicado forall. Declaratividad. |
| 3 | Setof / Findall. Declaratividad. |
| 4 | Setof / Findall. Inversibilidad. |
| 5 | Inversibilidad con un predicado forall. Generación. Uso de setof para eliminar duplicados. |
| 6 | Están todos los conceptos: declarativiad, puede estar recursividad (depende de cómo lo resuelvan), aparece la inversibilidad, pueden utilizar generación, etc. |