UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA EN INFORMÁTICA

EVALUADOR DE FÓRMULAS DE LA LÓGICA DE PREDICADOS

Caso de estudio 12

Grupo de Trabajo: Cas-ET2

Participantes:

Elena Ballesteros Morallón

Samuel Espejo Gil

Fecha: 28/04/2024

Tabla de contenido

1. INTRODUCCIÓN

Este trabajo consiste en desarrollar en Prolog un **programa capaz de evaluar fórmulas de la lógica de predicados** en el contexto de interpretaciones con dominios finitos.

Es decir, dada una fórmula bien formada, y una interpretación, que asigna significado a los diferentes símbolos, el programa debe ser capaz de expresar la verdad o falsedad de esta. Se limita a que las fórmulas estén cerradas.

El usuario puede usar diferentes interpretaciones y fórmulas y el programa podrá responder para cualquier fórmula, mientras que la entrada sea correcta.

Para entender la solución planteada para este programa, primero se explicarán los fundamentos de la lógica de predicados, necesarios para entender los diferentes elementos que se tratarán. Luego, se verá cómo se ha llevado a cabo la representación de los diferentes elementos en Prolog, además de los algoritmos que se han usado para poder implementar el programa. Seguiremos con la forma usada para poder ejecutar el programa para que un usuario cualquiera pueda utilizarlo, se tratará el cómo se ha implementado una interfaz básica junto a un pequeño manual de usuario. Para terminar, se verán ciertas conclusiones obtenidas de llevar a cabo este trabajo.

2. FUNDAMENTOS

En este apartado se incluye la explicación de los fundamentos teóricos sobre los que se basa la solución propuesta para el caso de estudio. Se introducirán los principales conceptos y notaciones sobre la lógica de predicados que permitirán comprender cómo se ha ido creando la solución paso a paso, desde el plano sintáctico al semántico.

2.1. Lógica de predicados

El lenguaje formal de la lógica de predicados, también llamada lógica de primer orden, permite traducir las frases del lenguaje natural a expresiones sintácticamente correctas en este lenguaje formal. A continuación vamos a describir sus características.

2.1.1. Alfabeto

a. Símbolos comunes a todos los formalismos.
   - Un conjunto infinito numerable de símbolos de variable $\mathcal{V} = \mathrm{X}, \mathrm{Y}, \ldots$. No designan a ningún objeto o individuo en particular, sino que representan elementos genéricos.
   - Un conjunto de conectivas $\{\neg,\ \wedge,\ \vee,\ \rightarrow,\ \leftrightarrow\}$
   - Cuantificadores $\{\exists, \forall\}$[1]

[1] Para diferenciar claramente el cuantificador $\forall$ de la conectiva $\wedge$, se ha decido utilizar esta notación en lugar de $\{\vee, \wedge\}$ que es la usada en la lógica clásica.

El generalizador, o cuantificador universal (∀), equivale a las expresiones "para todo" o "cada".

El particularizador, o cuantificador existencial (∃), equivale a las expresiones "existe" o "algún".

- Signos de puntuación: '(', ')', ','

b. Símbolos propios de un formalismo.
   - Conjunto de símbolos de constante $\mathcal{C} = \{a, b, c, \dots\}$. Se utilizan para identificar a un individuo.
   - Conjunto de símbolos de función de aridad n $\mathcal{F} = \{f, g, h, \dots\}$. Representan funciones que se aplican a las variables o las constantes.
   - Conjunto de símbolos de relación de aridad n $\mathcal{P} = \{P, Q, R, \dots\}$. Representan relaciones entre términos y pueden ser verdaderos o falsos en función del valor de verdad de sus argumentos.

La aridad indica el número de argumentos sobre los que se puede aplicar un símbolo, en el caso de las constantes, se consideran como símbolos de aridad cero.

2.1.2. Términos y fórmulas

Juntando los símbolos mencionados, podemos formar secuencias y aquellas que estén bien formadas darán lugar a términos o fórmulas. Estas son las llamadas expresiones de $\mathcal{L}$, nuestro lenguaje formal.

a. Términos. Un término de $\mathcal{L}$, representado por T. Es una expresión, en la que intervienen variables, constantes y símbolos de función con las siguientes reglas:
   1. t es un término si $t \in V \cup C$. Un término de $\mathcal{L}$ es una variable o una constante de $\mathcal{L}$.
   2. Si $t_1, t_2, \cdots, t_n$ son términos de $\mathcal{L}$ y f^n es un símbolo de función, n-ádico de $\mathcal{L}$ entonces $f^n(t_1, t_2, \cdots, t_n)$ es un término de $\mathcal{L}$.

b. Fórmulas.

   1. Fórmula atómica.

   Una fórmula atómica es una expresión, en la que intervienen términos y símbolos de relación, definida como:

   Si $t_1, t_2, \cdots, t_n$ son términos de $\mathcal{L}$ y R^n es un símbolo de relación n-ario de $\mathcal{L}$ entoces $R^n(t_1, t_2, \cdots, t_n)$ es una fórmula atómica de $\mathcal{L}$.

   2. Fórmula bien formada.

   Aquella expresión de $\mathcal{L}$ formada por fórmulas atómicas, conectivas y/o cuantificadores:

   - Fórmula atómica de $\mathcal{L}$.
   - Si A y B son fórmulas bien formadas de $\mathcal{L}$, entonces también son fórmulas bien formadas: $\neg A$, $\neg B$, $A \wedge B$, $A \vee B$, $A \rightarrow B$ y $A \leftrightarrow B$.

- Si A es una fórmula bien formada de $\mathcal{L}$ y $x \in V$ entonces también son fórmulas bien formadas: $\bigwedge xA$ y $\bigvee xA$

2.2. Interpretaciones

Las interpretaciones pertenecen al plano semántico y se encargan de asignar significado a los elementos del lenguaje formal, ya que sin este significado no se podría evaluar si una fórmula es verdadera o falsa. Hay que asignar significado tanto al dominio, o universo de discurso, como a los símbolos propios de cada formalismo.

Formalmente se define una interpretación $\mathcal{I}$ como un par $(\mathcal{D}_\mathcal{I}, \mathcal{J})$ tal que:

1. $\mathcal{D}_\mathcal{I}$ es un conjunto no vacío que pertenece al dominio de $\mathcal{I}$. Este dominio agrupa los individuos a los que se referirán las variables.
2. $\mathcal{J}$ es una aplicación que asigna:
   a. A cada símbolo de constante $a \in \mathcal{C}$, un elemento distinguido del dominio $\mathcal{D}_\mathcal{I}$; tal que $\boldsymbol{\mathcal{J}(a) = \bar{a}_i}$
   b. A cada símbolo de función f $\in \mathcal{F}$, una función $\boldsymbol{\mathcal{J}(f) = \bar{f}}$ que cuando se aplica a elementos del dominio $\mathcal{D}_\mathcal{I}$ nos devuelve otro elemento del dominio.
   c. A cada símbolo de relación R $\in \mathcal{P}$, una relación $\boldsymbol{\mathcal{J}(R) = \bar{R}} \subset \mathcal{D}_\mathcal{I}$ que es un conjunto de n-tuplas del dominio.

Algunos autores consideran una definición diferente para la aplicación a los símbolos de relación, dicen que por cada símbolo de relación, la interpretación asigna una aplicación $\mathcal{D}_\mathcal{I} \rightarrow \{V, F\}$

2.3. Valoraciones

En el apartado anterior se describían las interpretaciones, pero esto no es suficiente para hablar de verdad o falsedad de un enunciado. Primero hay que asignar valores a las ocurrencias de las variables, en el caso de que la fórmula bien formada no este cerrada.

Así, una valoración ϑ en una interpretación $\mathcal{I}$ es una aplicación que asigna a cada término de $\mathcal{I}$ un elemento de $\mathcal{D}_\mathcal{I}$. Una valoración también recibe el nombre de asignación.

$$\vartheta : \mathcal{T} \rightarrow \mathcal{D}$$

$$\vartheta(\mathrm{t}) = \begin{cases} (1)\ v(X) & si\ t \in \mathcal{V} \wedge t \equiv X \\ (2)\ \mathcal{J}(a) & si\ t \in \mathcal{C} \wedge t \equiv a \\ (3)\ \mathcal{J}(f)\big(\vartheta(t_1) \dots \vartheta(t_n)\big) & si\ f \in \mathcal{F} \wedge (t_1 \in \mathcal{T} \wedge \dots \wedge t_n \in \mathcal{T}) \\ & \quad \wedge \mathrm{t} \equiv \mathrm{f}(\mathrm{t}_1 \dots \mathrm{t}_\mathrm{n}) \end{cases}$$

(1) Si el término es una variable, se le asigna una función $v(x)$ que es una aplicación que asigna a cada variable un elemento del dominio $\mathcal{D}_\mathcal{I}$.

(2) Si el término es una constate, se le asigna la función de interpretación para las constantes y se aplica.
(3) Si el término es un símbolo de función aplicado a términos, se aplica la función de interpretación para los símbolos de función y se hace la valoración de cada uno de los términos. Aquí encontramos una definición recursiva.

Además, una valoración que coincide exactamente con una valoración ϑ, salvo (tal vez) en el valor asignado a una variable $X \in \mathcal{V}$, se denomina valoración X-equivalente de ϑ.

2.4. Evaluación

Una evaluación consiste en asignar un valor de verdad (Verdadero o Falso) a una fórmula bien formada. Dada una fbf $\mathcal{A}$, una interpretación $\mathcal{I}$ y una valoración ϑ, la evaluación de $\mathcal{A}$ en este contexto se define como:

a. Si $\mathcal{A} \equiv R(t_1 \dots t_n)$, es decir una fórmula atómica, entonces

$$Eval(\mathcal{A}) = \mathcal{J}(R)\big(\vartheta(t_1) \dots \vartheta(t_n)\big) \; = V \; si \; y \; solo \; si \; \big(\vartheta(t_1) \dots \vartheta(t_n)\big) \in \mathcal{J}(R) = \overline{R}$$

Si es una fórmula atómica, se le aplica la interpretación del símbolo de relación y se valoran los términos. Si esta relación entre el símbolo y sus términos estaba definida en el dominio, entonces la fórmula tomará un valor de verdad V.

b. Si $\mathcal{A}$ está formada por conectivas y fbf y es de la forma:
- $\neg B$ entonces $Eval(\mathcal{A}) = f_{\neg}\big(Eval(B)\big)$
- $(B \wedge C)$ entonces $Eval(\mathcal{A}) = f_{\wedge}\big(Eval(B), Eval(C)\big)$
- $(B \vee C)$ entonces $Eval(\mathcal{A}) = f_{\vee}\big(Eval(B), Eval(C)\big)$
- $(B \rightarrow C)$ entonces $Eval(\mathcal{A}) = f_{\rightarrow}\big(Eval(B), Eval(C)\big)$
- $(B \leftrightarrow C)$ entonces $Eval(\mathcal{A}) = f_{\leftrightarrow}\big(Eval(B), Eval(C)\big)$

En el caso de que $\mathcal{A}$ este formada por conectivas, al ser interpretada y valorada devolverá un valor de verdad V o F en función de los valores de verdad que tomen B o C y según la interpretación que se haga de las funciones de las conectivas.

c. Si $\mathcal{A} \equiv (\forall X)B$ entonces $Eval(\mathcal{A}) = V$ si y solo si para **toda** valoración X-equivalente de ϑ, $Eval(B) = V$. También se puede entender como que no puede existir un caso en el que $Eval(B) = F$

En el caso de que la fórmula esté cuantificada universalmente, al sustituir todas las ocurrencias de X por elementos del dominio todas sus evaluaciones devuelvan el valor de verdad V.

d. Si $\mathcal{A} \equiv (\exists X)B$ entonces $Eval(\mathcal{A}) = V$ si y solo si para **alguna** valoración X-equivalente de ϑ, $Eval(B) = V$.

En el caso de que la fórmula esté cuantificada existencialmente, al sustituir todas las ocurrencias de X por elementos del dominio al menos una de ellas deberá devolver el valor de verdad V.

3. REPRESENTACIÓN DE LAS FÓRMULAS E IMPLEMENTACIÓN DE ALGORITMOS

Una vez explicados los fundamentos, veremos cómo se representa la sintaxis en el programa y cómo se ha implementado el algoritmo de evaluación.

El programa se encuentra dividido en dos módulos con la funcionalidad, uno contiene la interfaz para interactuar con el usuario y el otro la lógica principal del evaluador. También se incluyen dos ficheros con las interpretaciones, uno con el dominio (se explicará en el apartado 4.1) y otro con las interpretaciones de las conectivas y su definición de función de verdad.

El fichero de las conectivas, *"semantica_operadores*", permite adaptar las funciones de verdad de las conectivas en caso de que se quiera utilizar una interpretación diferente a la definida por la lógica clásica, que es la que se usará por defecto.

A continuación vamos a explicar con mayor detalle los módulos del programa y sus características.

3.1. Evaluador

3.1.1. Alfabeto, términos y fórmulas

Para la definición del alfabeto, las **variables**, se representan en Prolog con cadenas de símbolos formadas por letras, dígitos o '_' siempre que esta cadena comience por una letra mayúscula o '_'. Y las **constantes, símbolos de función y relación** se representan igual, excepto que esta cadena comenzará por una letra minúscula.

Para la representación del conjunto de las **conectivas**: `{~, ^, \/, =>, <=>}`, se ha utilizado el predicado *op/3*, este predicado permite definir operadores indicando su precedencia, el tipo de notación y el nombre que se le asigna. Así podemos representar una fórmula igual que si la escribiéramos en un papel, con notación infija y para evitar ambigüedades hay que establecer una prioridad. Una menor precedencia hace que ese signo se evalúe antes que otro con mayor precedencia.

Los **cuantificadores** se representan con los predicados *exists/2*, para el cuantificador existencial, y *forAll/2*, para el cuantificador universal. Ambos siguen la misma estructura, *cuantificador(Variable_cuantificada, Formula)*.

Así, podemos definir algunas fórmulas bien formadas, usando la interpretación de ejemplo, como: par(doble(a)), exists(X, par(X)), par(a) <=> par(a).

3.1.2. Valoración

Se define la función de valoración, con el predicado *valoracion(Termino, Valor)*. Los casos para los que se define son:

- Para variables, en cuyo caso se obtiene el dominio, y a partir de este, un elemento.
- Para una constante, la valoración obtiene el valor del dominio asignado a esta constante por la interpretación.
- Para un término general:
  1. Se obtiene el número de términos que usa el functor, esto es la aridad.
  2. Se descompone en functor y argumentos.
  3. Se interpreta el functor, obteniendo su significado, se interpretan los argumentos, mediante un predicado auxiliar que, dada una lista, devuelve la misma interpretada.
  4. Una vez se tienen todos los símbolos interpretados, se llama al functor interpretado, que es un predicado, donde como último argumento, tiene el valor que debe devolver.

3.1.3. Evaluación

La evaluación obtiene un valor de la verdad {v, f} de una fórmula. Se realiza mediante el predicado *evaluacion(Formula, Valor)*. Se definen 4 casos, para fórmulas atómicas, fórmulas formadas por operadores y otras fórmulas, fórmulas con *forAll* y fórmulas con *exists*. Aunque la evaluación de fórmulas atómicas y fórmulas con conectivas es muy parecida, igual que en los casos de la evaluación del generalizador y el existencial.

A) Fórmulas atómicas
   1. Se comprueba que la fórmula no sea cerrada.
   2. Se obtiene el número de argumentos (aridad), el símbolo del relator y los argumentos.
   3. Se interpreta el relator y se valoran los argumentos, mediante el predicado auxiliar *valoracion_lista*, previamente tratado.
   4. Se forma la relación interpretada.
   5. Se llama a esta, obteniendo el valor de la verdad en el último argumento.

B) Fórmulas con conectivas
   1. Se obtiene el número de argumentos (aridad), el símbolo de la conectiva y las fórmulas involucradas.
   2. Se interpreta la conectiva y se valoran las fórmulas, mediante el predicado auxiliar *evaluacion_lista*, que dada una lista de fórmulas, devuelve sus respectivos valores de verdad. Se forma un predicado con el significado de la conectiva y los valores de verdad de entrada.
   3. Se llama al predicado, obteniendo el valor de verdad en el último argumento.

C) Fórmulas con *forAll*
   1. Se descompone en *forAll*, la variable cuantificada y la fórmula.
   2. Se comprueba si al menos una interpretación de la variable hace que fórmula evalúe a falso, en cuyo caso el valor será falso. Si no existe ninguna, será verdadero. Esta evaluación se hace a través del predicado *at_least_one*.

D) Fórmulas con *exists*
   1. Se descompone en *exists*, la variable cuantificada y la fórmula.
   2. Con *at_least_one* se comprueba si al menos una interpretación de la variable hace que la fórmula evalúe a verdadero, en cuyo caso el valor de *exists* será verdadero. Si no existe ninguna, será falso. Para mejorar la eficiencia, se ha usado el operador de corte que evitará que una vez obtengamos un valor de verdadero no tengamos que seguir.

Predicados auxiliares:

- *at_least_one/3*: dada una variable y una expresión devuelve el valor de la evaluación. Obtiene la valoración para la variable, copia la expresión cambiando la variable con *copy_term*. La copia de la fórmula se hace para evitar que se obtenga por pantalla el valor de la variable.
- *evaluación_lista/2:* predicado para evaluar una lista de términos, devuelve una lista con las evaluaciones de verdad de estos.
- *functor/3:* en este caso, dado un término lo descompone en su nombre y la aridad.
- *call/2:* como primer argumento tiene el objetivo que queremos lanzar y le sigue una lista de argumentos que se lanzarán con el objetivo.
- *ground/1*: comprueba que las formulas que vamos a evaluar sean cerradas, es decir que no tengan variables libres.

3.2. Interfaz

En este módulo se define la interfaz para que el usuario interactúe con el programa. Al principio se define el predicado *fichero_cargado/1* como dinámico, ya que la definición de este predicado se modificará en tiempo de ejecución. Se usarán los predicados *assertz/1*, para añadir a la base de datos, y *retractall/1*, para eliminar todas las cláusulas que unifiquen con su argumento. La funcionalidad de *fichero_cargado/1* será llevar un registro de cuál es el fichero de interpretación que se está usando.

Aparte de este predicado se definen por orden de aparición:

- main/0: predicado que muestra el menú de opciones y lee la que el usuario haya seleccionado.
- borrar_interpretación/0: busca el nombre del fichero que está cargado y borra todas las cláusulas que se han cargado desde él usando el predicado predefinido, *unload_file/1*.

- cargar_fichero/1: se encarga de comprobar que el fichero que ha introducido el usuario exista y si es así, lo carga con *consult/1*.
- procesar_opcion/1: es el predicado que ejecuta cada una de las opciones. Algunas solo muestran texto en pantalla y otras ya se han descrito anteriormente.
    - Opción 1, ver interpretación cargada, se hace uso del predicado *findall/3*, para crear una lista de instanciaciones que toma una variable.
    - Opción 2, evaluación, se lee la fórmula que ha introducido el usuario y se intenta ejecutar con *catch/3*, el predicado *evaluacion/2* y en caso de que devuelva alguna excepción se mostrará el mensaje de error si no mostrará el resultado de la evaluación.

4. JUNTANDO LAS PIEZAS: INTERFACES DE ENTRADA Y DE SALIDA

El usuario puede interactuar con el programa ejecutando el archivo *"interfaz.pl"*. Al iniciarlo se muestra en la terminal un menú con las siguientes opciones:

```
================================================
== Evaluador de formulas logicas de primer orden ==
================================================
0. Cargar una interpretacion
1. Ver interpretacion cargada
2. Evaluar una formula
3. Ayuda
4. Exit
Seleccione una opcion:
```

Para seleccionar una opción hay que introducir el número de la opción y terminar con un punto antes de pulsar Enter, ya que esta es la forma en la que Prolog lee de la entrada.

0. *Cargar una interpretación*: Permite al usuario cargar una interpretación desde un fichero y borrar la que hubiera antes.
1. *Ver interpretación cargada*: Muestra el nombre del fichero desde el que se ha cargado la interpretación, el dominio, las constantes y los símbolos de función y de relación.
2. *Evaluar una fórmula*: Esta opción es la que permite interactuar con el predicado de *evaluación*.
3. *Ayuda*: Se muestran los predicados del programa y los símbolos de las conectivas.
4. *Exit*: Sale del programa.

4.1. Interfaz de entrada

El formato del archivo donde se define la interpretación debe ser el siguiente:

- *dominio([$\overline{a_1}$, $\overline{a_2}$, $\cdots$, $\overline{a_n}$])* predicado con una lista donde $\overline{a_i}$ es un elemento del dominio, sin que se repita ningún par.
- *interpretación(símbolo, aridad, significado):* predicado auxiliar para definir la función de aplicación $\mathcal{J}$, define un símbolo, su aridad y un predicado de significado.
    - Constantes: *interpretacion(*a, 0, $\overline{a}$*)*, donde $a \in C$, elemento $\overline{a}$ distinguido del dominio D.
    - Función: *interpretacion(*f, n, $\overline{f^n}$*)*, $f \in F$
    - Relación: *interpretacion(*r, n, $\overline{r^n}$*)*, $r \in R$

Posteriormente, habría que definir los predicados de significado de las funciones y de las relaciones $\text{nombre\_funcion}(x_1, \ldots, x_n, valor\_retorno), x \in \mathcal{D}$. Teniendo en cuenta que las funciones devuelven un elemento del dominio y las relaciones un valor de verdad {v,f}.

Hay que tener en cuenta que este debe ser un archivo Prolog y por tanto seguir su sintaxis, como que todos los predicados tienen que acabar en un punto.

4.2. Interfaz de salida

Cuando seleccionamos la opción 2 del menú, *Evaluar una fórmula*, el programa pide al usuario que introduzca la fórmula que quiere evaluar y se devuelve como resultado:

- v, si la fórmula se evalúa a verdadero.
- f, si la fórmula se evalúa a falso.

5. CONCLUSIÓN

El objetivo de este trabajo es profundizar en el lenguaje de Prolog desarrollando un proyecto más complejo que un simple ejercicio. En este caso, el proyecto está relacionado con la asignatura de Lógica por lo que hemos repasado nuestros conocimientos y nos ha servido para aplicarlos en el ámbito de la programación.

Sobre Prolog, hemos aprendido a crear predicados que se modifican en tiempo de ejecución con *dynamic*, a descomponer términos en distintas partes y operar con ellos, a crear operadores y a crear una interfaz por terminal para interactuar con el usuario. Además, hemos investigado en la documentación sobre predicados específicos que necesitábamos como *copy_term* o *unload_file.*

Al principio la representación de los elementos de la lógica de predicados fue confusa pero después estructuramos el código de forma que se diferencien claramente los planos sintácticos y semánticos.

La implementación de la interfaz por terminal para permitir que los usuario interactúen con el programa ha hecho que la solución sea accesible y fácil de usar. Además, nos ha permitido ver otra forma distinta de interactuar con el intérprete de Prolog.

En resumen, este trabajo nos ha permitido poner en práctica los conceptos teóricos de la lógica de predicados y de la programación lógica en un proyecto real.

6. ANEXOS

A. Ejemplo de interpretación

```
 1 %% USUARIO
 2
 3 % dominio(Dominio)
 4 dominio([0, 1, 2, 3]).
 5
 6 % interpretacion(Simblo, Aridad, Elemento)
 7 interpretacion(a, 0, 1).
 8 interpretacion(b, 0, 2).
 9 interpretacion(doble, 1, significado_doble).
10 interpretacion(par, 1, significado_par).
11
12 % functores
13 significado_doble(0, 0).
14 significado_doble(1, 2).
15 significado_doble(2, 0).
16 significado_doble(3, 2).
17
18 % relatores
19 significado_par(0, v).
20 significado_par(1, f).
21 significado_par(2, v).
22 significado_par(3, f).
```

Listado 1: archivo *interpretacion.pl*

B. Ejemplo de uso

```
$ swipl interfaz.pl

?- ==================================================
== Evaluador de formulas logicas de primer orden ==
==================================================
0. Cargar una interpretacion
1. Ver interpretacion cargada
2. Evaluar una formula
3. Ayuda
4. Exit
Seleccione una opcion: 0.

========= INTERPRETACION =========
Introduce el nombre del fichero ("nombre.pl"):|: "interpretacion.pl".
Interpretacion cargada

?- ==================================================
== Evaluador de formulas logicas de primer orden ==
==================================================
0. Cargar una interpretacion
1. Ver interpretacion cargada
2. Evaluar una formula
3. Ayuda
4. Exit
```

```
Seleccione una opcion: |: 1.

===== INTERPRETACION CARGADA =====
Fichero: interpretacion.pl
Dominio: [0,1,2,3]
Constantes: [a,b]
Simbolos de funcion y de relacion: [doble,par]

==================================================
== Evaluador de formulas logicas de primer orden ==
==================================================
0. Cargar una interpretacion
1. Ver interpretacion cargada
2. Evaluar una formula
3. Ayuda
4. Exit
Seleccione una opcion: 2.

============ EVALUACION ============
Introduce la formula a evaluar|: par(b).
Resultado de la evaluacion: v
...
============ EVALUACION ============
Introduce la formula a evaluar|: exists(X, par(X)).
Resultado de la evaluacion: v
...
============ EVALUACION ============
Introduce la formula a evaluar|: forAll(X, par(doble(X))).
Resultado de la evaluacion: v
...
============ EVALUACION ============
Introduce la formula a evaluar|: forAll(X, exists(Y, par(doble(X)) => par(Y))).
Resultado de la evaluacion: f
```

C. Código del programa

```
 1 % interpretacion_operadores(Operador, Aridad, Significado)
 2 interpretacion_operadores(~, 1, negacion).
 3 interpretacion_operadores(^, 2, conjuncion).
 4 interpretacion_operadores(\/, 2, disyuncion).
 5 interpretacion_operadores(=>, 2, condicional).
 6 interpretacion_operadores(<=>, 2, bicondicional).
 7
 8 negacion(v, f).
 9 negacion(f, v).
10
11 conjuncion(v, v, v).
12 conjuncion(v, f, f).
13 conjuncion(f, v, f).
14 conjuncion(f, f, f).
15
16 disyuncion(v, v, v).
17 disyuncion(v, f, v).
```

```
 18 disyuncion(f, v, v).
 19 disyuncion(f, f, f).
 20
 21 condicional(v, v, v).
 22 condicional(v, f, f).
 23 condicional(f, v, v).
 24 condicional(f, f, v).
 25
 26 bicondicional(v, v, v).
 27 bicondicional(v, f, f).
 28 bicondicional(f, v, f).
 29 bicondicional(f, f, v).
```

Listado 2: archivo *semantica_operadores.pl*

```
1 :- discontiguous interpretacion/3. % la definición de interpretación estará
separada (x-equivalente y la que introduzca el usuario)
  2 :- multifile interpretacion/3. % multifile porque queremos mantener la
interpretación x-equivalente
  3 :- dynamic interpretacion_operadores/3. % dynamic sólo se queda con la última
definición
  4 :- dynamic dominio/1.
  5
  6 %% DEFINICIÓN SINTÁCTICA DE LOS OPERADORES
  7 :-op(300, fy, [~]).
  8 :-op(400, yfx, [^]).
  9 :-op(450, yfx, [\/]).
 10 :-op(700, xfy, [=>]).
 11 :-op(700, xfy, [<=>]).
 12
 13 %% DEFINICIÓN SEMÁNTICA DE LOS OPERADORES
 14 :- [semantica_operadores].
 15
 16 %%
 17 interpretacion(X, 0, X). % x-equivalente
 18
 19 %% DEFINICIÓN DE VALORACIÓN
 20 % - Variables
 21 % - Constantes
 22 % - Término general (functor)
 23 valoracion(X, Valor) :- var(X), dominio(D), member(Valor, D).
 24 valoracion(X, Valor) :- atomic(X), interpretacion(X, 0, Valor).
 25 valoracion(X, Valor) :- compound(X), functor(X, Functor, NArgs), X =.. [Functor|
Args],
 26 interpretacion(Functor, NArgs, Functor_Interpretado), valoracion_lista(Args,
Args_Interpretados),
 27 Funcion =.. [Functor_Interpretado| Args_Interpretados], call(Funcion, Valor).
 28
 29 % valoracion_lista(+Terminos, -Terminos_Interpretados)
 30 valoracion_lista([], []).
 31 valoracion_lista([H|C], [H_interpretada|C_intepretados]) :- valoracion(H,
H_interpretada), valoracion_lista(C, C_intepretados).
```

```
 32
 33
34 %% DEFINICIÓN EVALUACIÓN
 35 % - Fórmulas atómicas
 36 % - Fórmulas con operador
 37 % - Fórmulas con "forAll"
 38 % - Fórmulas con "exists"
 39 evaluacion(Formula, _):-
 40     ((\+compound(Formula)) -> throw(error('El termino no es compuesto: ~w',
[Formula])), true).
 41
 42 % - Fórmulas atómicas
 43 evaluacion(Formula, Valor) :- functor(Formula, Relator, NArgs), Formula =..
[Relator| Args],
 44 interpretacion(Relator, NArgs, Relator_Interpretado), valoracion_lista(Args,
Args_Interpretados),
 45 Relation =.. [Relator_Interpretado| Args_Interpretados],
 46 ((\+ground(Formula)) -> throw(error('La formula no es cerrada: ~w', [Formula]));
true),
 47 call(Relation, Valor).
 48
 49 % - Fórmulas con operador
 50 evaluacion(Formula, Valor) :- functor(Formula, Operador, NArgs), Formula =..
[Operador| Args],
 51 interpretacion_operadores(Operador, NArgs, Operador_Interpretado),
evaluacion_lista(Args, Args_Evaluados),
 52 Relation =.. [Operador_Interpretado| Args_Evaluados],
 53 ((\+ground(Formula)) -> throw(error('La formula no es cerrada: ~w', [Formula]));
true),
 54 call(Relation, Valor).
 55
 56 % - Fórmulas con "forAll"
 57 evaluacion(Formula, Valor) :- Formula =.. [forAll, Variable, Expresion],
 58 ((at_least_one(Variable, Expresion, f), Valor = f, !); Valor = v, !).
 59
 60 % - Fórmulas con "exists"
 61 evaluacion(Formula, Valor) :- Formula =.. [exists, Variable, Expresion],
 62 (at_least_one(Variable, Expresion, v), Valor = v, !; Valor = f, !).
 63
 64 at_least_one(Variable, Expresion, Valor) :-
 65 valoracion(Variable, Elemento), copy_term([Variable], Expresion, [Copied_Var],
Copied_expresion),
 66 Copied_Var = Elemento, evaluacion(Copied_expresion, Valor), !.
 67
 68 % evaluacion_lista(+Terminos, -Terminos_Evaluados)
 69 evaluacion_lista([], []).
 70 evaluacion_lista([H|C], [H_evaluada|C_evaluados]) :- evaluacion(H, H_evaluada),
evaluacion_lista(C, C_evaluados).
```

Listado 3: archivo *evaluador.pl*

```
:- dynamic fichero_cargado/1.

:- [evaluador].
:- initialization(main).
:- [interpretacion].
fichero_cargado("interpretacion.pl").

main :-
    menu,
    read(Opcion),
    procesar_opcion(Opcion).

menu :-
    write('\n================================================='),nl,
    write('== Evaluador de formulas logicas de primer orden =='), nl,
    write('================================================='),nl,
    write('0. Cargar una interpretacion'), nl,
    write('1. Ver interpretacion cargada'), nl,
    write('2. Evaluar una formula'), nl,
    write('3. Ayuda'), nl,
    write('4. Exit'), nl,
    write('Seleccione una opcion: ').

borrar_interpretacion:- fichero_cargado(Fichero), retractall(fichero_cargado(_)), unload_file(Fichero).

mostar_error(error(Msg, Args)):-
    (Msg=existence_error(_, F) -> format('Error, no existe definicion para: ~w',[F]),nl);
    format(Msg, Args), nl.

cargar_fichero(Fichero):- exists_file(Fichero), !, borrar_interpretacion,
    consult(Fichero),
    assertz(fichero_cargado(Fichero)),
    write('Interpretacion cargada').
cargar_fichero(_):- write('El fichero no existe').

procesar_opcion(Opcion):- Opcion=0, !,
    write('\n========= INTERPRETACION ========='), nl,
    write('Introduce el nombre del fichero ("nombre.pl"):'),
    read(Fichero), cargar_fichero(Fichero), nl,
    main.

procesar_opcion(Opcion):- Opcion = 1, !,
write('\n===== INTERPRETACION CARGADA ====='),nl,
fichero_cargado(Fichero), write('Fichero: '), write(Fichero), nl,
dominio(D), write('Dominio: '), write(D), nl,
findall(X, (interpretacion(X, 0,_), nonvar(X)), Xs), write('Constantes: '), write(Xs), nl,
findall(Y, (interpretacion(Y, N,_), N\=0), Ys), write('Simbolos de funcion y de relacion: '), write(Ys), nl,
main.

procesar_opcion(Opcion):- Opcion=2, !,
write('\n============ EVALUACION ============'), nl,
```

```
 52 write('Introduce la formula a evaluar'), read(Formula),
 53     (catch(evaluacion(Formula, V), Error, mostar_error(Error)),
 54         write('Resultado de la evaluacion: '), (nonvar(V), write(V) ;
write('ERROR')), nl % Imprime el valor de V
 55     ;
 56         write('Consulta no valida'), nl
 57     ),
 58     main.
 59
 60 procesar_opcion(Opcion):- Opcion = 3, !,
 61 write('\n============ AYUDA ============'),nl,
 62 write('Para introducir informacion al programa: Acabar las sentencias con "."'),
nl, nl,
 63 write('CUANTIFICADORES:'), nl,
 64 tab(4),write('forAll(Variable, Formula)    --> Predicado para definir el
cuantificador universal'), nl,
 65 tab(4),write('exists(Variable, Formula)    --> Predicado para definir el
cuantificador existencial'), nl,
 66 write('CONECTIVAS:'), nl,
 67 tab(4), write('~   negacion'), nl,
 68 tab(4), write('^   conjuncion'), nl,
69 tab(4), write('\\/  disyuncion'), nl,
 70 tab(4), write('=>  condicional'), nl,
 71 tab(4), write('<=> bicondicional'), nl,
 72 write('Introduce cualquier letra para volver al menu '), read(_),
 73 main.
 74
 75
 76 procesar_opcion(Opcion):- Opcion = 4,  write('\nSaliendo...'), halt.
 77
 78 procesar_opcion(_):-
 79     write('Opcion no valida'),nl, main.
```

Listado 4: archivo *interfaz.pl*