Lógica

Práctica 2: Tablas Semánticas

Enunciado

En esta práctica construiremos un **probador de tablas semánticas** en Prolog que permitirá decidir si cualquier fórmula arbitraria en lógica proposicional es inconsistente o no, y en caso de no serlo, devolverá las distintas ramas abiertas obtenidas, expresadas como listas de literales. La práctica la dividermos en dos apartados.

Las **fórmulas de entrada** admitirán los siguientes operadores en Prolog:

```
:- op(1060, yfx, <->).  % doble implicación
:- op(1050, yfx, <-).   % implicación hacia la izquierda
:- op(800, yfx, xor).   % disyunción exclusiva
:- op(600, yfx, v).     % disyunción
:- op(400, yfx, &).     % conjunción
:- op(200, fy, ¬ ).     % negación
```

Apartado 1: operadores derivados (predicado `unfold`)

En un primer paso, implementaremos un predicado `unfold(F,G)` que toma de entrada una fórmula cualquiera usando los operadores de arriba y devuelve una fórmula equivalente pero que sólo utiliza la conjunción `&`, la disyunción `v` y la negación `¬`. Para eliminar todos los demás operadores, nuestro programa usará un un predicado que llamaremos `define(F,G)` que nos indica a qué corresponde cada operador:

```
define((F xor G), ¬ (F <-> G)).
define((F <-> G), (F -> G) & (F <- G)).
define((F <- G), (G -> F)).
define((F -> G), ¬ F v G).
```

Nótese que un operador derivado, por ejemplo el `xor`, puede estar definido en términos de otro operador derivado, en este caso la doble implicación `<->`, que a su vez depende de la definición de `->` y `<-`, etc.

Un par de ejemplos de ejecución serían:

```
?- unfold( p <-> q & ¬ r, G ).
G =  (¬p v q& ¬r)&(¬ (q& ¬r)v p).

?- unfold( (a -> b) xor c, G ).
G = ¬ ((¬ (¬a v b)v c)&(¬c v (¬a v b))).
```

aunque el resultado que obtengas puede variar ligeramente, por ejemplo, en la forma de anidar los paréntesis.

Ejercicio optativo (se valorará más): procura que el predicado `unfold` esté implementado de modo que, en cualquier momento, podamos definir nuevos operadores derivados con la simple adición de un par de líneas (una para crear el operador y otra con su `define`). Por ejemplo, para definir un nuevo operador `nand` (la conjunción negada), tu programa debería funcionar **tan sólo** con añadir las dos líneas:

```
:- op(800, yfx, nand).

define((F nand G), ¬ (F & G)).
```

Apartado 2: Tablas Semánticas

En el segundo apartado, implementaremos un predicado `tab(F,R)` que toma de entrada una fórmula proposicional cualquiera, la desdobla usando `unfold` para que sólo contenga `&`, `v`, `¬` y átomos, y por último, ejecuta sobre ella un algoritmo de tabla semántica. El predicado devolverá tantos valores de `R` como ramas abiertas encontremos en la tabla semántica. Cada rama viene expresada como una lista de literales.

La definición de este predicado `tab(F,R)` nos la dan ya hecha del siguiente modo:

```
tab(F,R) :- unfold(F,G), tab( [G], [], R).
```

pero, como vemos, depende de un nuevo predicado `tab(Fs,Lits1,Lits2)` de **tres argumentos** que es el que realmente implementaremos. Este predicado será recursivo y funciona del siguiente modo: dada una lista `Fs` de fórmulas que aún tenemos que desdoblar en la tabla semántica, y dado el conjunto `Lits1` de literales que llevamos calculados en la rama actual hasta el momento, si llegamos al final (no quedan fórmulas por desdoblar) la lista `Lits2` devuelve la rama que hemos encontrado abierta. Si todas las ramas están cerradas (fórmula inconsistente) tanto `tab/3` como, por tanto, `tab/2` deberán ser false.

Un ejemplo de ejecución, sería el siguiente:

```
?- tab( a -> ¬ ( b -> c), R).
R = [¬a] ;
R = [b, ¬c] ;
false

?- tab( a <-> ¬ a , R).
false.
```

donde, en el último caso, hemos intentado probar una inconsistencia y, por tanto, no hemos encontrado ninguna rama abierta.

Evaluación y Entrega

La nota máxima para este ejercicio es de **1,5 puntos = 15% del curso**. Entrega: usar tarea MOODLE correspondiente. Fecha límite: **Lunes 24 de abril, 23:59**. Las prácticas se pueden realizar por parejas o de forma individual. Si se realiza por parejas, es suficiente con que uno de los dos componentes realice la entrega en moodle, pero el fichero fuente debe contener en comentarios en las primeras líneas los nombres de los dos alumnos que formen la pareja. Se debe entregar **un único fichero, "practica2.pl"** con todos los predicados que se requieren en el enunciado.

Maintained by Pedro Cabalar