Lógica para Cs. de la Computación

Fabrega, Juan Ignacio
Schwerdt, Matías David
Comision 16

Proyecto 1: Teorema(f)

Primer Cuatrimestre 2020

Requerimientos de implementación

Se pide implementar en Prolog el mecanismo de computación para la teoría formal L visto en la materia, esto es, el procedimiento efectivo basado en Eliminación de Literales Complementarios que permite determinar si una fórmula bien formada (fbf) es o no teorema de la teoríia.

Metas

1. **Afianzar los conceptos teóricos acerca de la teoría L.**
2. **Mejorar el dominio del lenguaje Prolog.**

Implementación del predicado teorema/1.

El predicado teorema/1 recibe una fórmula bien formada (en adelante, FBF) de L y determina si ésta es o no un teorema, retornando *true* o *false*. Para llevar a cabo su funcionamiento, implementamos dos predicados: *fncr/2* y *refutable/1*.

```
teorema(F):- fncr(~F,FNCR), !, refutable(FNCR).
```

Implementación del predicado fncr/2.

El predicado fncr/2 recibe una FBF de L como primer argumento y retorna su *forma normal conjuntiva reducida* como segundo argumento. Para lograrlo implementamos una serie de predicados que detallaremos a continuación.

```
fncr(FBF,FNCR):- fBienEscrita(FBF,R1), fPaso1(R1,R2),
fPaso2Iterado(R2,FNC), eliminarParentesis(FNC,RTA),
guardarExpresion(RTA,RTA1), eliminarRepetidos(RTA1,RTA2),
generarTops(RTA2,RTA3), eliminarTops(RTA3,RTA44),crearTop(RTA44,RTA4),
generarBottoms(RTA4,RTA5), hayBottom(RTA5,RTA6),
borrarClausulasRepetidas(RTA6,RTA7), transformarExpresion(RTA7,FNCR).
```

fBienEscrita /2

El procedimiento de transformar una FBF en FNCR requiere que no haya implicaciones ni equivalencias. Es por ello que utilizamos éste predicado.

```
fBienEscrita(F,RTA):-
    transformarImplicaciones(F,Rta1),
    transformarEquivalencias(Rta1,RTA).
```

transformarImplicaciones/2: Recibe una FBF y la retorna sin implicaciones. Lo que hace exactamente el predicado es reemplazar $P \rightarrow Q$ por $\neg P \vee Q$.

transformarEquivalencias/2: Recibe una FBF y la retorna sin equivalencias. Lo que hace el predicado es reemplazar $P \leftrightarrow Q$ por $(\neg P \vee Q) \wedge (P \vee \neg Q)$.

fPaso1/2

Recibe una FBF sin implicaciones ni equivalencias, y la retorna luego de aplicarle las siguientes transformaciones:

- ¬¬P se reduce a P
- ¬(P v Q) se reduce a ¬P ∧ ¬Q
- ¬(P ∧ Q) se reduce a ¬P v ¬Q
- ¬⊥ se reduce a ⊤
- ¬⊤ se reduce a ⊥

fPaso2/2 y fPaso2Iterado/2

fPaso2Iterado/2 es una cáscara que sirve para iterar fPaso2/2 hasta que su llamado deje de tener efecto debido a que la fbf ya ha sido procesada por completo.

fPaso2/2 recibe una FBF, y la retorna luego de aplicarle las siguientes transformaciones:

- P v (Q ∧ R) se reduce a (P v Q) ∧ (P v R)

  Este parte del predicado lo que haces es distribuir las disyunciones entre las conjunciones, para así obtener distintos conjunto de conjunciones que se relacionan entre sí mediante disyunciones.

- P v ⊤ se reduce a ⊤
- P v ⊥ se reduce a P

eliminarParentesis/2

La función de este predicado es eliminar aquellos paréntesis de más que tiene nuestra fbf, para así obtener una fbf con la cual es más simple trabajar.

guardarExpresion/2

Este predicado se encarga de guardar una conjunción en una lista. La lista estará representada tal que, los elementos pueden representar una cláusula, que se guarda como una lista, o directamente un elemento de la lista puede ser un literal. Por ejemplo, la siguiente expresión: (a /\ (b\/c)) se guarda como [a, [b,c]].

guardarClausula/2

Se encarga de guardar una cláusula en una lista. Los distintos elementos de la lista representa los distintos literales que formar la cláusula.

unir/3

Lo que hace es unir dos listas sin importan su contenido, solo las une. Recibe la lista1, la lista2 y luego la retorna la unión de ambas listas.

eliminarRepetidos/2

Dada una lista recorre todos sus elementos y elimina los elementos repetidos dentro de cada unos de los elementos de la lista principal. Retorna una nueva lista sin elementos repetidos.

borrarTodas/3

Es predicado se encarga de borrar todas las apariciones de un elemento dado de una lista dada y retorna la nueva lista sin ningún tipo de aparición del elemento.

reducirClausula/2

Dada una cláusula, representada mediante una lista, elimina todos aquellos literales que se encuentran repetidos y no alteran el valor de la expresión.

esta/2

Dado un elemento y una lista, retorna true en caso de que el elemento se encuentre en la lista dada, y falso en caso contrario.

existeComplementario/2

Dado un elemento, que representa un literal, y dado una lista, retorna true en caso que en la lista se encuentre el complementario del literal, y falso en caso contrario.

tieneComplementario/1

Dado una cláusula, representada en forma de lista, retorna true en caso de que algún elemento de la cláusula tenga complementario, es decir si se encuentren dos literales ambos complementarios dentro de una misma cláusula.

tieneComplementario/2

Dado un elemento, y una lista devuelve true en caso de que en la lista exista un elemento complementario al pasado por parámetros, falso en caso contrario.

generarTops/2

Dado una lista que representa una expresión, recorre la lista para verificar si las distintas cláusulas tienen dentro elementos complementarios, en caso de que una cláusula tenga elementos complementarios, reemplaza a la cláusula por [top]. En caso contrario no modifica la cláusula. Retorna la lista luego de realizar este proceso por todos sus elementos.

eliminarTops/2

Recorrer una lista, que representa una expresión, para eliminar todas las apariciones de las cláusulas tops. Retorna la lista sin los tops.

crearTop/2

Dado una lista, si esta lista estuviera vacía retorna la lista con el elemento top, en caso que la lista no estuviera vacía la retorna sin ningún tipo de modificaciones.

primerElemento/1

Dado una lista, retorna su primer elemento.

tieneUnElemento/1

Dado una lista retorna true en caso que la lista tenga un solo elemento.

sonComplomentarios/2

Dado un elemento y una lista que tiene un único elemento, retorna true ambos elementos son complementarios, falso en caso contrario.

generarBottoms/2

Dado una lista que representa una conjunción, resuelve la expresión de literales complementarios eliminandolos de la lista y colocando un bottom. Devuelve la lista luego de realizar este proceso por todos los elementos que la componen.

borrarComplementario/3

Dado un elemento, y una lista, busca el complementario dentro de lisa y si lo encuentra lo reemplaza por bottom. En caso de que el complementario no exista, retorna la lista sin ninguna modificación.

hayBottom/2

Dado una lista, si en ella se encuentra un bottom devuelve un la lista con un único elemento: bottom. Si la lista no tienen ningún bottom devuelve la lista sin ningún tipo de modificación.

mismoLargoListas/2

Dado dos listas retorna true en caso que ambas listas tengan la misma cantidad de elementos, falso con caso contrario.

listasSonIguales/2

Devuelve true si una lista el igual a la otra, falso en caso contrario.

borrarClausula/3

Dado una cláusula, representada en forma de lista, y una conjunto de cláusulas, representadas en forma de lista, elimina la apariciones de la primer cláusula en el conjunto. Retorna un nuevo conjunto de cláusulas habiendo borrado la cláusula pasado por parámetro.

borrarClausulasRepetidas/2

Dado la conjunción de disyunciones , representadas en forma de listas, devuelve un nuevo conjunto de cláusulas sin repeticiones.

```
borrarClausulasRepetidas([],[]).
borrarClausulasRepetidas([X|Xs],Rta):-
    borrarClausula(X,Xs,Rta1),
    borrarClausulasRepetidas(Rta1,Rta2),
    Rta=[X|Rta2].
```

De esta manera se recorre las listas de conjunciones para eliminar las cláusulas repetidas. Se obtiene la primer cláusula, se busca y elimina todas sus repeticiones en el resto de la lista y se obtiene una nueva lista sin la repeticiones de la primer cláusula. Luego se llama recursivamente con esta nueva lista. En Rta se agrega la primer cláusula, seguido del resultado del llamado recursivo.

pasarClausula/2

Dado una lista, que representa una cláusula, devuelve esta expresión utilizando el operador “\/” entre literales.

pasarAfbf/2

Dado un expresión conjunción de disyunciones, representada mediante listas, devuelve la expresión utilizando el operador “/\” entre disyunciones.

transformarExpresion/2

Dado una conjunción de disyunciones, retorna la misma expresión pero eliminando los paréntesis de más y/o que no tienen ningún tipo de relevancia en la fbf.

Implementación del predicado refutable/1.

El predicado refutable/1 recibe una FNCR , la cual convierte en una lista usando el predicado guardarExpresion/2. Usamos !/0 para quedarnos con el primer resultado obtenido y descargar los demás. Finalmente llamamos al predicado refutarLista/1 para intentar derivar por resolución de bottom a partir de ella.

```
refutable(FNCR):- guardarExpresion(FNCR, Claus), !, refutarLista(Claus).
```

refutarLista/1

Éste predicado retorna true o false según si la FNCR es refutable o no.
Recibe una FBF en FNCR, cuyo representación es en una lista de cláusulas, que a su vez se representan en una lista de literales ([[Cláusula], [Cláusula], … , [Cláusula]]). A partir de ello obtiene las resolventes, y genera un nuevo conjunto uniendo las cláusulas originales con las resolventes. Se llama así mismo recursivamente hasta que el conjunto generado sea igual al conjunto de cláusulas anterior, o en su defecto, hasta encontrar un bottom. Para lograrlo, se hace uso de los predicados recorrerListaDeClausulas/2, unir/2, borrarClausulasRepetidas/2 y cascaraListasIguales/2, las cuales detallaremos a continuación.

recorrerListaDeClausulas/2 y cadaClausulaConElResto/2

Recorre una lista de cláusulas, y para cada una de ellas llama al predicado cadaClausulaConElResto/2. Éste último lo que hace es buscar resolventes entre la primer cláusula y el resto de las cláusulas de la lista, haciendo uso del predicado resolventeEntreDosClausulas/3. Una vez que el predicado recorrerListaDeClausulas/2 termina de recorrer toda la lista, se llamar al predicado unir/3 y borrarClausulasRepetidas/3, obteniendo así el conjunto de resolventes sin que haya repetidos.

resolventeEntreDosClausulas/3 y resolventeEntreDosClausulasAux/3

Recibe dos cláusulas, y por cada literal de la primer cláusula, busca un literal complementario en la segunda cláusula. Si lo encuentra, devuelve la unión entre ambas cláusulas sin el par de literales complementarios. En caso de no encontrar literales complementarios, retorna falso.

Predicados auxiliares:

- borrarComplementarios/3: recibe un literal y una cláusula. Busca el literal complementario en la cláusula, si lo encontra lo elimina.
- unirSinRepeticiones/3: une dos listas sin elementos repetidos
- insertar/3: recibe un elemento y una lista. Inserta el elemento al principio de la lista y la retorna.
- cascaraSonIguales/2: recibe dos listas y retorna verdadero o falso según sin las listas son iguales o no. Una de las listas es la de cláusulas y la otra es la de cláusulas unida a resolventes. Se utiliza en el predicado refutarLista/1.

Tests teorema:

```
?- teorema( (a => (b => c)) => ((a => b) => (a => c)) ).
true
```

```
?- teorema(p\/ ~q\/r\/ ~p).
true
```

```
?- teorema(a\/ ~a).
true
```

```
?- teorema(a\/ a).
false
```

```
?- teorema(a=>(b/\c)/\b/\c).
false
```

Tests fncr:

```
?- fncr( ~((a => (b => c)) => ((a => b) => (a => c))), FNCR ).
FNCR = (~a \/ ~b \/ c) /\ (~a \/ b) /\ a /\ ~c
```

```
?- fncr(a=>b\/c/\d, FNCR).
FNCR = (~a\/b\/c)/\(~a\/b\/d)
```

```
fncr((a/\b\/c\/((a<=>b)<=>c)=>a), FNCR).
FNCR = (~c\/a)/\(~b\/a\/c)/\(~c\/a\/b)
```

Tests refutable:

```
?- refutable( (~a \/ ~b \/ c) /\ (~a \/ b) /\ a /\ ~c ).
true
```

```
refutable(~a/\a).
true
```

```
refutable(a\/a).
false
```

```
refutable((a\/b)/\(~a\/ ~b)).
false
```