Proyecto LCC 2020

Computación en $\mathcal{L}$

Aitor Ortuño Rosseto & Lucas Cervelli Haderne

Objetivo

El propósito principal del presente proyecto es afianzar los conocimientos de la teoría forma $\mathcal{L}$ y afianzar los conocimientos del lenguaje Prolog.

Para determinar si una fórmula bien formada (fbf) es o no teorema de la teoría se utiliza el mecanismo de computación basado en eliminación de literales complementarios.

¿Qué hace el sistema?

El sistema cuenta con un predicado principal ***teorema/1,*** el cual deriva su funcionalidad a dos predicados internos, ***fncr/2*** y ***refutable/1,*** determinando si la fbf recibida es o no un teorema en la teoría.

- ***fncr/2*** recibe una fbf de $\mathcal{L}$ como primer argumento y retorna como segundo argumento su formal normal conjuntiva reducida (fncr).
- ***refutable/1*** recibe una fbf en fncr y determine si existe o no derivación alguna por resolución de bottom a partir de ella, es decir, si es refutable.

Fncr

El primer paso para determinar si una fbf es o no teorema en la teoría, es obtener la forma reducida de dicha fbf por cuestiones de simplicidad computacional de cálculo lógico.

fncr/a recibe una fbf en y retorna su equivalente en forma normal conjuntiva reducida.

Pasos previos

Al recibir una fbf lo primero que debemos realizar es sustituir las implicaciones y equivalencias que presenta la misma para poder trabajar sólo con conjunciones (/\), disyunciones (\/) y negaciones (~).

sustituirImplicaciones/2 recibe una fbf y en caso de contar con implicaciones (=>) o equivalencias (<=>) retorna su expresión equivalente sin las mismas.

- P => Q == ¬P \/ Q
- P <=> Q == (¬P \/ Q) /\ (P \/ ¬Q)

Distribuir Negaciones

Para satisfacer la primer condición de forma normal conjuntiva entonces ninguna subfórmula cuyo conectivo principal es ¬ contiene otra aparición de ¬, /\, \/, top o bottom.

distribuirNegaciones/2 recibe una fbf y en caso de contar con negaciones que afecten a una subfórmula que tiene más de sólo un literal entonces se distribuye entre todos los literales que la componen.

- ¬¬Q == Q
- ¬(Q \/ P) == ¬Q /\ ¬P
- ¬(Q /\ P) == ¬Q \/ ¬P
- ¬ top == bottom
- ¬ bottom == top

DistribuirAND

Para satisfacer la segunda condición de forma normal conjuntiva entonces ninguna subfórmula cuyo conectivo principal es \/ contiene otra aparición de /\, top o bottom.

distribuirAND/2 recibe una fbf y en caso de contar con disyunciones que afecten a una subfórmula que tiene más de sólo un literal entonces se distribuye entre todos los literales que la componen.

- P \/ top == top
- P \/ bottom == P
- P \/ (Q /\ R) == (P \/ Q) /\ (P \/ R)
- (P /\ Q) \/ R == (P \/ R) /\ (Q \/ R)

reducir y sacarComplementos

Para satisfacer la tercer condición de forma normal conjuntiva entonces los símbolos top o bottom no aparecen dentro del alcance de ningún /\ .

reducir/2 recibe una fbf y en caso de contar con apariciones de top o bottom simplifica las expresiones.

- P /\ top == P
- P /\ bottom == bottom

reducir y sacarComplementos

Si en el paso anterior, se reemplazó alguna disyunción por un top, se procede a eliminarlo de las lista de disyunciones.

sacarComplementos/2 recibe una fbf y en caso de contar con apariciones de un literal y también su negado o más de una aparición de un mismo literal se sustituye por lo que corresponda según el caso, lo retorna en forma de lista de listas de disyunciones.

- P \/ P == P
- ¬P \/ ¬P == ¬P
- P \/ ¬P == top

sacarTops

Si existe alguna aparición de tops en la lista de listas de disyunciones, se eliminan.

sacarTops/2 recibe una fbf en forma de lista de listas de disyunciones y en caso de contar con apariciones de un literal top, se borran de la lista.

Ejemplo:

[[~a, b], [~a, c], [a], [~b, ~c], [top]] —> [[~a, b], [~a, c], [a], [~b, ~c]]

armarFNCR

Como último paso, procedo a rearmar la fbf recibida inicialmente como fncr, es decir, como conjunciones de términos disyunciones.

armarFNCR/2 recibe una lista de listas de disyunciones, las cuales retorna unidas por conjunciones. El resultado es la fbf inicial en forma normal conjuntiva reducida.

Ejemplo:

[[~a, b], [~a, c], [a], [~b, ~c]] —> (~c\/~b)/\a/\(c\/~a)/\(b\/~a)

Refutable

El último paso para demostrar que P es un teorema en la teoría, es demostrar que ¬P es insatisfacible y por esta razón ¬P es refutable.

refutable/1 recibe una fbf en fncr y determina si es o no refutable, es decir, si existe o no una derivación por resolución de bottom a partir de ella.

desarmarConjunciones

Para facilitar el mecanismo de computación basado en eliminación de literales complementarios procedemos a listar las conjunciones en distintas sublistas, dentro de una lista que cada elemento es un término disyunción.

desarmarConjunciones/2 recibe una fbf en fncr y retorna una lista de listas cuyos elementos de estas sublistas son los literales de cada término de conjunciones.

Ejemplo:

((~c\/ ~b)/\a/\(c\/ ~a)/\(b\/ ~a) —> [[~c, ~b], [a], [c, ~a], [b, ~a]]

Resolventes

Para computar el mecanismo de computación basado en eliminación de literales complementarios buscamos en cada una de las sublistas el complemento de los literales que se encuentran en las otras sublistas.

resolventes/2 recibe una lista de listas de conjunciones. Busca literales complementarios en distintas listas para formar los nuevos resolventes de cada una de las instancias y los devuelvo en una lista.

Ejemplo:

[[~c, ~b], [a], [c, ~a], [b, ~a]] —> [[~b, ~a], [~c, ~a], [c], [b], [~a]]

Literales

Por facilidad del cómputo del mecanismo de computación basado en eliminación de literales complementarios separamos las resolventes en listas con sólo un literal y listas con más de un literal.

literales/2 recibe una lista de listas de resolventes y retorna una lista con aquellas resolventes compuestas por más de un literal.

Ejemplo:

[[~b, ~a], [~c, ~a], [c], [b], [~a]] —> [[~b, ~a], [~c, ~a]]

intentarRefutar

Como última instancia se procede a verificar si alguna de las listas con sólo un literal es el negado de otra lista, obteniendo así bottom y dando por finalizado el procedimiento ***refutar/1.***

intentarRefutar/1 recibe una lista de listas de resolventes y retorna una lista con aquellas resolventes compuestas por sólo un literal y devuelve verdadero en caso de encontrar el complemento de un literal en otra lista.

Casos de Prueba

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