Computación en L: Implementación en Prolog

Integrantes:

- Rychert Alan
- Rios Matias

2020

Introducción

En el presente proyecto se implementó el mecanismo de computación de la teoría formal L, es decir, el procedimiento efectivo basado en la eliminación de literales complementarios que permite afirmar si una cierta fórmula bien formada es teorema o no utilizando el lenguaje de programación Prolog.

Objetivos de proyecto:

Son múltiples los objetivos por los cuales se desarrolló este proyecto. Uno de los más relevantes es aumentar la comprensión de los conceptos teóricos de la teoría L (lenguaje proposicional). Otro objetivo es aumentar la comprensión y manejo del lenguaje de programación Prolog, y por último desarrollar la habilidad de documentar las soluciones implementadas y ejercer la práctica del trabajo en equipo, situación recurrente en el ámbito laboral.

Representación de fbfs en Prolog

El siguiente fragmento de código Prolog, será incluído al comienzo del archivo .pl del proyecto, declarando los operadores lógicos que emplearemos para representar las fórmulas bien formadas de L: ~ (negación), /\ (conjunción), \/ (disyunción), => (implicación), y <=> (equivalencia).

```
:-op(300,fx,~). % negación, prefija, no asociativa.
:-op(400,yfx,(/\)). % conjunción, infija, asociativa a izquierda.
:-op(500,yfx,(\/)). % disyunción, infija, asociativa a izquierda.
:-op(600,xfx,=>). % implicación, infija, no asociativa.
:-op(650w,xfx,<=>). % equivalencia, infija, no asociativa.
```

La notación :- p es una forma de solicitar al SWI-Prolog que ejecute la consulta p, por única vez, al cargar el archivo .pl. Luego el fragmento de código anterior consiste de una secuencia de consultas al predicado predefinido op/3 para declarar al intérprete nuevos operadores. El primer argumento de op/3 establece la precedencia del operador, donde un menor valor significa una mayor precedencia. Para más información acerca de op/3 ver la documentación del predicado en el manual de referencia del SWI.

Una vez declarados, el SWI-Prolog "reconocerá" estos operadores como estructuras especiales, de forma análoga a cómo considera las estructuras aritméticas +, -, *, etc., que pueden notarse de manera infija, más cómoda, y que cuentan con reglas de precedencia y asociatividad que permiten evitar paréntesis en algunas situaciones (ver transparencia 14 de la clase práctica 3).

Además de los operadores lógicos, adoptaremos el uso de dos constantes especiales: **top**, para denotar τ (o verdadero) y **bottom**, para denotar □ (o falso). Los símbolos τ y □ son introducidas en la sección 2.4 de Davis, R. E. Truth, Deduction and Computation: logic and semantics for computer science. Computer Science Press, 1989.

Estrategias de resolución

Para poder resolver el problema de implementar el mecanismo de computación de la teoría formal L, primero quisimos resolver el problema de identificar si una fbf es o no teorema. Luego, dividimos en dos subproblemas la solución, pasar la fórmula bien formada a forma normal conjuntiva reducida e indicar si la fbf en fncr es refutable, es decir, si existe o no una derivación por resolución de bottom a partir de ella.

➔Teorema

La estrategia adoptada para resolver el problema de decidir si una fórmula bien formada en L es un teorema es la refutación por resolución. En la refutación por resolución, dada una fbf se debe negar la misma y expresarla en la forma normal conjuntiva reducida (conjunción de cláusulas), para luego poder aplicar el proceso de resolución, que consiste en aplicar iterativamente la reducción de literales complementarios entre cláusulas hasta que ya no es posible hallar más "resolventes", es decir, cláusulas que resultan de eliminar literales complementarios entre otras dos. Dada la complejidad de este procedimiento, el problema fue subdividido en dos problemas distintos, fncr y refutable.

➔Fncr

Para pasar una fórmula bien formada a la forma normal conjuntiva reducida se llevaron a cabo los siguientes pasos:

1. Se aplican las equivalencias (A=>B)=~A\/B y (A<=>B)=(~A\/B)/\(~B\/A) y luego se quitaron dobles negaciones, aplicaron leyes de De Morgan y redujo ~top a bottom y ~bottom a top, con el objetivo de que la fórmula resultante contenga solo ~, /\ y \/.
2. Se aplica iterativamente la propiedad distributiva de la disyunción (A\/(B/\C))=(A\/B)/\(A\/C)) hasta que ya no es posible volver a aplicarla lo cual significa que la fbf está en forma normal conjuntiva.
3. Se quitan todas las apariciones de bottom en las cláusulas aplicando la propiedad (Q\/bottom) = Q, se reduce a top toda cláusula que involucre a top aplicando la propiedad (Q\/top)=top y luego se aplican la propiedades (Q/\bottom)=bottom y (Q/\top) = Q.
4. Se aplican las propiedades (P\/P)=P, (P\/~P)=top y (~P\/~P)=P para eliminar las apariciones repetidas de un mismo literal (negado o no) en una cláusula, evitando que haya dos apariciones de un mismo literal al alcance de un \/.
5. Se vuelve a aplicar el paso 3 dado que el paso 4 puede haber generado alguna aparición de top y en ese caso es necesario volver a aplicar las propiedades correspondientes de ser posible.

➔ Refutable

Determina si un conjunto de cláusulas es o no refutable, es decir, si existe o no una derivación por resolución de bottom a partir de ella.
Para resolver este problema se dividió en tres subproblemas:

1. Obtener una resolvente entre dos cláusulas.
2. Obtener todas las resolventes entre una cláusula y el resto de las cláusulas.
3. Obtener todas las resolventes posibles entre todas las cláusulas hasta encontrar la cláusula vacía o hasta que no se puedan obtener nuevas resolventes.

Implementación

Para implementar el mecanismo de computación de la teoría formal L se utiliza la técnica de refutación por resolución, por lo que fue implementado mediante un predicado teorema/1 que recibe como argumento una fbf de L y se implementa en términos de los predicados auxiliares fncr/2 que recibe una fbf de L como primer argumento y retorna como segundo argumento su forma normal conjuntiva reducida, y refutable/1 que recibe una fbf en fncr y determina si ésta es o no refutable, es decir, si existe o no una derivación por resolución de bottom a partir de ella.

Para resolver algunos problemas, en la implementación las fbf en forma normal conjuntiva se convirtieron en listas en las cuales sus elementos son las cláusulas de la fbf y estas a su vez también están en forma de lista de literales, o literales negados. Por ejemplo, la fbf (a\/b)/\c/\(d\/e) se verá representada como [[a,b],[c],[d,e]].
Este formato facilita recorrer la fbf y permite eliminar los paréntesis que no son necesarios pero que son generados por prolog por cuestiones de precedencia.
En **fncr/2** se aplica convierte a esta notación luego de aplicar el predicado **distributivaBucle/2**, y en **refutable/2** se aplica al principio antes de refutar/1 ya que como se mencionó anteriormente aunque impacte en el tiempo de ejecución del programa facilita la tarea de recorrer la fbf y disminuye la cantidad de código ya que usando este formato es posible usar predicados predefinidos para listas como **member/2**.

➔ aLista/2

Recibe en su primer argumento una fórmula bien formada la cual asume que estará en forma normal conjuntiva y devuelve en su segundo argumento el resultado de convertir esa fbf en una lista cuyos elementos son listas de disyunciones (cláusulas).

➔ fncr/2

Este predicado recibe como primer argumento una fórmula bien formada y retorna como segundo argumento la misma fórmula expresada en forma normal conjuntiva reducida, la cual se computa mediante los predicados nombrados debajo, en el orden en el que son nombrados, y finalmente utiliza por segunda vez el predicado **reducirTopBottom/2** dado que **reducirRepetidosYComplementarios/2** puede generar apariciones de top:

- **criterio1/2:** Recibe una fórmula bien formada y llama al predicado **desarmar/2**, que devuelve la fbf resultante de aplicar las propiedades de equivalencia de implicaciones y dobles implicaciones, es decir una fbf en cuyos operadores son solo ~,/\ y \/. Esta es la entrada al predicado **deMorgan/2**, que se aplica propiedades para eliminar dobles negaciones, aplica reglas de De

Morgan y transforma ~top a bottom y ~bottom a top, retornando la fórmula bien resultante de aplicar esas propiedades. **Criterio1/2** retorna el resultado de **deMorgan/2**.

- **distributivaBucle/2:** Recibe una fórmula bien formada en forma y llama iterativamente al predicado distributiva con ella hasta que no es posible volver a aplicar la propiedad. Retorna una fbf formada solo por conjunciones de cláusulas (forma normal conjuntiva).

- **aLista/2:** Explicado en la sección predicados generales.

- **reducirTopBottom/2:** Recibe una fbf en formato lista de listas y aplica las propiedades (1) Q\/Top=top, (2) Q\/bottom=Q, (3) Q/\top=Q y (4) Q/\bottom=bottom en ese orden. Para aplicar (1), por cada sublista que contenga top como elemento, la reemplaza por la sublista [top]. Para aplicar (2) y (3), elimina todas las apariciones de bottom de cada Sublista y todas las de [top] de la lista de listas. Para aplicar (4), si la lista contiene [bottom], deja [bottom] como único elemento. Estas propiedades son aplicadas en predicados auxiliares.

- **reducirRepetidosYComplementarios/2:** Recibe una fbf en formato lista de listas y utilizando dos predicados auxiliares recursivos: **paso4aCascara/2** y **paso4bCascara/2** en el orden en el que fueron nombrados, retorna una fbf en formato lista de listas que no contiene apariciones redundantes de literales o de literales negados (aplica las propiedades (P\/P=P, P\/~P=top y ~P\/~P=~P).

- **convertir/2:** Recibe una fbf en formato lista de listas y retorna la misma fbf en notación tradicional, es decir, en términos de ~,\/ y /\.

➔ Refutable/1

Este predicado recibe una una fbf en forma normal conjuntiva, la convierte en una lista con **aLista/2** y luego intenta derivar [bottom] a partir de ella usando los siguientes predicados:

- **refutar/1:** Recibe una fbf en forma normal conjuntiva reducida en forma de lista, donde cada cláusula está en forma de lista de literales y determina si es refutable, esto lo logra obteniendo todas las resolventes posibles a partir del conjunto de cláusulas inicial usando el predicado **obtenerResolventes/2,**

luego se verifica que el resultado de unir las cláusulas con las resolventes sea distinto al conjunto original, si no son iguales entonces tal vez se puede obtener más resolventes por lo que se llama recursivamente a **refutar/1.** Si son iguales entonces no es posible obtener más resolventes y cómo **refutar/1** verifica en la primera parte del predicado si [bottom] pertenecía a la lista, en la segunda parte del predicado se puede asumir que [bottom] no se encuentra en la lista por lo tanto el conjunto de cláusulas no es refutable.

- **obtenerResolventes/2:** Este predicado recibe en su primer argumento un conjunto de cláusulas en forma de una lista cuyos elementos son listas de literales y obtiene si es posible todas las resolventes entre todas las cláusulas, esto se logra recorriendo recursivamente la lista de "cláusulas" y obteniendo con cada una de ellas la resolvente entre esa cláusula y el resto usando el predicado **unaConTodas/2,** luego se unen las cláusulas y las resolventes en una única lista usando el predicado **unir/3** y el resultado se guarda en el segundo argumento de **obtenerResolventes/2** para poder devolverlo.

- **unaConTodas/2:** Este predicado recibe en su primer argumento una cláusula en forma de lista de literales y en su segundo argumento una lista de cláusulas con el mismo formato, luego usando el predicado **resolvente/3** se obtiene la resolvente entre la cláusula recibida y la primer cláusula de la lista, si es posible obtener una resolvente el resultado se va anidando recursivamente en una lista y se llama a **unaConTodas/2** con la cláusula inicial y el resto de la lista. Si no es posible obtener una resolvente entonces simplemente no se almacena ningún resultado en la lista y se llama a **unaConTodas/2** de la misma manera.

- **resolvente/3:** Recibe como primer y segundo argumento dos cláusulas en forma de lista e intenta obtener una resolvente a partir de ellas, si es posible la devuelve en su tercer argumento.

Casos de prueba

Para comprobar que la implementación funcione correctamente se usaron los siguiente casos de prueba:

Casos de prueba para el predicado teorema/2:

Casos cuyo resultado esperado es true:
teorema(((a => b) /\ (a => c)) => (a => (b /\ c))).
teorema(((a => (b /\ (c \/ d))) /\ (~b \/ ~c)) => (a => d)).
teorema((((a => b) => c) => ((a => b) => (a => c)))).
teorema(((~a => b) => (~b => a))).
teorema((~b=> ~a)=>((~b=>a)=>b)).

Casos cuyo resultado esperado es false:
teorema(~(((a => b) /\ (a /\ c)) => (b \/ c))).
teorema(~((~(a \/ b)) <=> (~a /\ ~b))).
teorema(a\/(c=>a)/\(d\/c)).
teorema(a<=>b).
teorema(a\/b\/c\/(d/\e)).

Casos de prueba para fncr/2:

fncr(((a => b) /\ (a => c)) => (a => (b /\ c)),R).
Resultado esperado:
R=top.

fncr(~(((a => b) /\ (a => c)) => (a => (b /\ c))),R).
Resultado esperado:
R = (~c\/ ~b)/\a/\(c\/ ~a)/\(b\/ ~a).

fncr(a\/(b<=>(a/\d/\a)),R).
Resultado esperado:
R = (~b\/a)/\(d\/ ~b\/a)/\(~b\/a) .

fncr(~(a<=>(b<=>(c=>a))),R).
Resultado esperado:
R = (~b\/ ~a)/\(~a\/ ~b\/c)/\(b\/c\/a)/\(~c\/ ~b\/a) .

Casos de prueba para refutable:

refutable((~c\/ ~b)/\a/\(c\/ ~a)/\(b\/ ~a)).
Resultado esperado: true.

refutable((~c\/ ~a)/\(~d\/ ~a)/\(b\/ ~a)).
Resultado esperado: false.

refutable(~a/\ ~b/\(b\/a)).
Resultado esperado: true.

refutable((~c\/ ~a)/\(~d\/ ~a)/\(~d\/c)/\ ~a).
Resultado esperado: false.