Facultad de Ciencias - UNAM
Lógica Computacional 2025-1
Práctica 3: Resolución Binaria

Kevin Axel Prestegui Ramos
Eduardo Vargas Pérez
Carolina Stephanie Orea Romero

3 de Septiembre de 2024
Fecha de entrega: 17 de Septiembre de 2024

1 Introducción

A lo largo del curso, se ha revisado el concepto de **satisfacibilidad**, y el cómo a base de interpretaciones podemos saber si una fórmula es satisfacible o no.

Lamentablemente, el verificar por medio de interpretaciones, si una fórmula es satisfacible o no, no es factible a la hora de implementar un programa. Esto debido a que sería lo equivalente a crear una tabla de verdad completa para la fórmula, y verificar uno a uno si alguna de sus filas representa un modelo donde la evaluación de la fórmula sea verdadera. Esto tendría una complejidad de $O(n^2)$, donde n sería el número de variables dentro de la fórmula, lo cual no es lo más óptimo posible.

Debido a esto, introduciremos el concepto de **Resolución Binaria**, el cual por medio de una regla de inferencia, nos permite resolver de manera óptima el problema de satisfacibilidad para una fórmula proposicional.

2 Objetivo

El objetivo principal de esta práctica, es revisar el concepto de Resolución Binaria, y comprenderlo tal grado de poder realizar una implementación correcta de este. En este proceso, también se busca repasar el concepto de satisfacibilidad, comprender el concepto de las **formas normales**, y utilizar todos estos conceptos para implementar un **algoritmo de saturación**.

3 Justificación

En el estudio de la lógica proposicional, se podría decir que la satisfacibilidad es uno de los problemas más importantes, por lo cual es de suma importancia encontrar formas óptimas de poder automatizar el proceso para conocer si una fórmula es o no satisfacible.

4 Desarrollo de la práctica

4.1 Lógica proposicional

Para poder trabajar con la lógica proposicional en `Haskell`, hay que realizar una forma de representar las fórmulas. Para esto crearemos el siguiente tipo de dato.

```
 data Prop = Var String | Cons Bool | Not Prop
.            | And Prop Prop | Or Prop Prop
.            | Impl Prop Prop | Syss Prop Prop
.            deriving (Eq)
```

En este, por simplicidad representamos las variables como cualquier cadena, además damos soporte a las constantes Booleanas True y False y tenemos un constructor para cada uno de nuestros operadores proposicionales.

Adicionalmente, para facilitar la lectura de las fórmulas en la terminal, podemos agregar el siguiente código que le dice a *Haskell* cómo debe imprimir un tipo de dato Prop.

```
instance Show Prop where
.            show Cons True = "Verdadero"
.            show Cons False = "Falso"
.            show (Var p) = p
.            show (Not p) = "¬" ++ show p
.            show (Or p q) = "(" ++ show p ++ " ∨ " ++ show q ++ ")"
.            show (And p q) = "(" ++ show p ++ " ∧ " ++ show q ++ ")"
.            show (Impl p q) = "(" ++ show p ++ " → " ++ show q ++ ")"
.            show (Syss p q) = "(" ++ show p ++ " ↔ " ++ show q ++ ")"
```

4.2 Formas Normales

La resolución binaria no trata directamente con la fórmula proposicional, en cambio, trabaja con la forma clausular de esta. En Lógica Proposicional, una cláusula se define como una disyunción de literales, en dónde una literal es o una variable o la negación de una.

Forma normal negativa
La primera transformación que se le aplica a la fórmula es nombrada forma normal negativa (FNN). Una fórmula proposicional φ está en forma normal negativa si:

- φ no contiene equivalencias ni implicaciones,
- las negaciones solo afectan a variables proposicionales.

Cualquier fórmula proposicional puede ser convertida a forma normal negativa en tiempo lineal usando las siguientes equivalencias lógicas:

- Eliminación de la doble negación: $\neg\neg\varphi \equiv \varphi$
- Leyes de Morgan: $\neg(\varphi \wedge \rho) \equiv \neg\varphi \vee \neg\rho \mid \neg(\varphi \vee \rho) \equiv \neg\varphi \wedge \neg\rho$

Forma normal conjuntiva
La forma normal conjuntiva (FNC) de una fórmula es equivalente a la forma clausular de esta. Una fórmula proposicional φ está en formal normal conjuntiva si es una conjunción de disyunciones de literales, es decir, si es una conjunción de cláusulas.

Cualquier fórmula proposicional puede ser convertida a forma normal conjuntiva usando las siguientes equivalencias lógicas.

- Las mencionadas anteriormente para la forma normal negativa.
- Leyes distributivas: $\varphi \vee (\rho \wedge \psi) \equiv (\varphi \vee \rho) \wedge (\varphi \vee \psi) \mid \varphi \wedge (\rho \vee \psi) \equiv (\varphi \wedge \rho) \vee (\varphi \wedge \psi)$

Forma normal disyuntiva
Es importante mencionar que existe otra forma normal llamada forma normal disyuntiva, que consiste en la disyunción de conjunciones de literales, no obstante, no es de interés para esta práctica.

Ejercicios:

1. `fnn :: Prop → Prop`
   Implementar la función `fnn` que convierte una fórmula proposicional en su forma normal negativa. **[1 punto]**
2. `fnc :: Prop → Prop`
   Implementar la función `fnc`, que convierte una fórmula proposicional en su forma normal conjuntiva. Se recomienda usar la función `fnn`. **[1 punto]**

4.3 Resolución Binaria

La base de la resolución binaria es la regla de inferencia que se muestra a continuación. Dada una cláusula que contiene la literal χ y otra cláusula que contiene la literal $\neg\chi$, se puede inferir una nueva que consta de las literales de dichas cláusulas sin el par complementario, a la cual se le denomina **resolvente**.

$$\frac{\varphi_1, \varphi_2, ..., \chi, ..., \varphi_n \qquad \rho_1, \rho_2, ..., \neg\chi, ..., \rho_m}{\varphi_1, \varphi_2, ..., \varphi_n, \rho_1, \rho_2, ..., \rho_m} \tag{1}$$

Tenga en cuenta que, dado que las cláusulas son conjuntos de literales, no puede haber dos ocurrencias de ninguna literal en una cláusula. Por lo tanto, el resolvente debe seguir cumpliendo la definición de conjunto (no debe haber literales repetidas).

A las cláusulas que constan de una sola literal se les llama unitarias. De modo que si al aplicar resolución binaria una de las dos cláusulas involucradas es unitaria, entonces el resolvente contendrá las literales restantes de la otra cláusula. El resolvente de dos cláusulas unitarias es conocido como cláusula vacía, pues no contiene ninguna literal, sin embargo, obtener esta como resultado indica que el conjunto de cláusulas no es satisfacible.

$$\frac{\varphi \quad \neg\varphi}{\square} \tag{2}$$

Es importante destacar que la regla solo permite que se haga una resolución a la vez, en otras palabras, solo es posible aplicar la regla sobre un par de literales a la vez.

Ejercicios

Para poder implementar la resolución binaria realizaremos lo siguiente:

1. `type Literal = Prop`
   Crear un sinónimo `Literal`, que será igual a `Prop` por simplicidad, aunque solo deberían ser variables o negaciones de variables. **[0.5 puntos]**
2. `type Clausula = [Literal]`
   Crear un sinónimo `Clausula`, que representará las cláusulas como conjunto de literales. **[0.5 puntos]**
3. `clausulas :: Prop -> [Clausula]`
   Definir la función `clausulas` que dada una fórmula en FNC, devuelve una lista con las cláusulas que la forman. **[2 puntos]**
4. `resolucion :: Clausula -> Clausula -> Clausula`
   Definir la función `resolucion` que dadas dos cláusulas, devuelve el resolvente obtenido después de aplicar la regla de resolución binaria. Se puede asumir que se puede obtener un resolvente a partir de los argumentos. **[2 puntos]**

4.4 Algoritmo de saturación

La resolución binaría es de gran importancia, pues proporciona un método para decidir la satisfacibilidad de una fórmula, no obstante, por si sola esta no es un procedimiento algorítmico, es decir, se sabe que usando la resolución binaria es posible decidir la satisfacibilidad de una fórmula; sin embargo, no sé índica cómo hacerlo. Para ello se hace uso de un **algoritmo de saturación**.

Para describir el algoritmo, necesitamos definir algunas cosas:

Función R: Sea S un conjunto de cláusulas. Sea R(S) el conjunto resultado de la unión de S con el conjunto de todos los posibles resolventes de las cláusulas de S.

$$R(S) = S \bigcup \{res(c_n, c_m) | c_n, c_m \in S\} \tag{3}$$

Función Res_n : Se define recursivamente como la n-ésima resolución de S

$$Res_0 = S \tag{4}$$

$$Res_{n+1} = R(Res_n(S)) \tag{5}$$

Con estos conceptos, podemos describir fácilmente el algoritmo, diciendo que existe una $n \in N$ tal que $Res_n(S)$ cumple alguna de las siguientes condiciones:

- $\square \in Res_n(S)$, implica que la fórmula es insatisfacible.
- $Res_{n-1}(S) = Res_n(S)$, implica que ya no se pueden obtener más resoluciones, y que la fórmula es satisfacible.
- se agotan los recursos computacionales, es indeterminada la satisfacibilidad o insatisfacibilidad de nuestra fórmula

Con esto en mente, enumeramos una serie de pasos para nuestro algoritmo de saturación:

1. Convertimos φ a su fnc, y la representamos como el conjunto de cláusulas S.
2. Encontramos $Res_n(S)$ tal que se cumplen una de las condiciones de paro mencionadas anteriormente.
3. Dependiendo de la condición cumplida, sabremos si φ es o no es satisfacible.

Ejercicios

1. `hayResolvente :: Clausula → Clausula → Bool`
   Definir la función `hayResolvente`, que determina si es posible obtener un resolvente a partir de dos cláusulas. **[1 punto]**
2. `saturacion :: Prop → Bool`
   Definir la función `saturacion`, que dada una fórmula proposicional, determina si esta es satisfacible o no usando el algoritmo de saturación. **[2 puntos]**

5 Especificaciones de entrega

- **Equipos:** La práctica debe realizarse en equipos de cuatro personas.
- **Google Classroom:** Para realizar la entrega de la práctica se utilizará la plataforma de Google Classroom. Sólo un integrante del equipo deberá entregar una carpeta comprimida como .zip que contenga el archivo correspondiente a la práctica y un archivo ReadMe con las especificaciones del siguiente punto.
- **Datos personales:** Se debe agregar un documento *ReadMe* en el que se incluya el nombre de todos los participantes del equipo. Si así lo desean, pueden agregar un comentario sobre la práctica, ya sea especificar si algo se complicó demasiado o si algo no les llega a funcionar y no saben exactamente por qué. En caso de no entregar este archivo, la calificación podría no ser asignada.
- **Fecha de entrega:** La fecha de entrega será la indicada al principio de este documento. No se recibirá ninguna práctica en fechas posteriores a la fecha indicada al menos que el profesor indique una prórroga.
- **Flujo de trabajo:** Como recomendación, hagan los ejercicios en el orden que aparecen en este documento pues es posible que para realizar alguno de los últimos ejercicios, se necesite alguno de los primeros.
- **Evaluación:** Para la evaluación, la práctica se someterá a un conjunto de pruebas, y además se realizará una revisión del código. Comprobándose que se cumplan las condiciones indicadas para cada ejercicio.
- **Compilado:** Cualquier práctica que al ser descargada no compile, **será evaluada con una calificación de 0**.
- **Limpieza y estructura:** Se debe especificar a que ejercicio pertenece cada función. Si es que implementan funciones auxiliares, agregar un comentario con una breve descripción de lo

que hace la función y para qué ejercicio fue implementada. La calificación podría ser afectada negativamente de no seguir este punto.

6 Sugerencias y Notas

- **Dudas**: Pueden preguntar sus dudas a través del correo o por Classroom. Pero les recomendamos preguntar a través del Classroom para que sus demás compañeros también puedan aclarar dudas similares a la suya.
- **Limitaciones**: Pueden utilizar funciones predefinidas siempre y cuando su utilización no derive en que se pierda el objetivo académico del ejercicio que se está realizando. Si se tiene duda acerca del poder utilizar algo pueden preguntarlo.

Buena suerte a todos! ☺☺☺