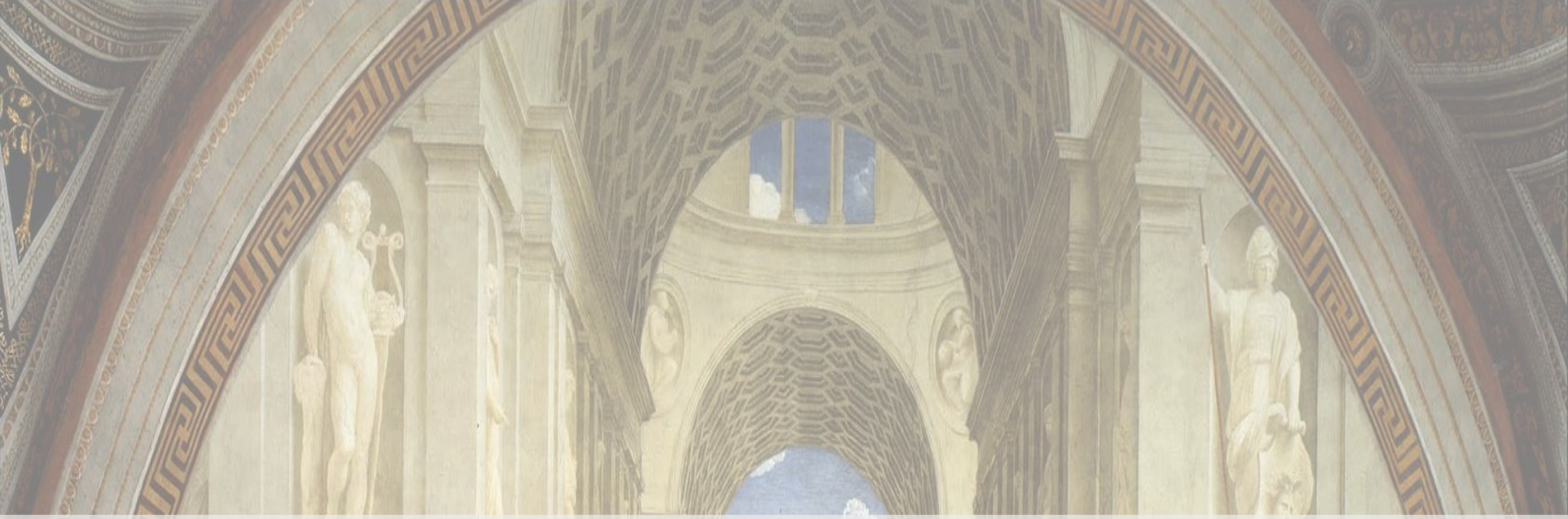

Desarrollo de la Lógica Proposicional y de Primer Orden bajo el paradigma funcional y la orientación Web.

Proyecto de Alumnía Interna 2019/20

Autor:
Ramos González, Víctor

Tutor:
Sancho Caparrini, Fernando
Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Desarrollo de la Lógica Proposicional y de Primer Orden bajo el paradigma funcional y la orientación Web.

Víctor Ramos González

Tutor: Fernando Sancho Caparrini
Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática
Universidad de Sevilla

Septiembre 2020

Resumen

El proyecto aborda los conceptos y algoritmos básicos de la Lógica Proposicional y la Lógica de primer, desde un punto de vista implementativo a través de un lenguaje encuadrado en el paradigma funcional (Elm).

El proyecto, basado en la asignatura de Lógica Informática, busca una doble finalidad, por un lado servir como una somera introducción a la programación declarativa al mismo que tiempo que proporcionar al alumnado herramientas intuitivas y de sencillo uso en la realización de los ejercicios que apoyen los contenidos teóricos que se desarrollan en dicha asignatura.

Índice general

Capítulo 1

Introducción. Objetivos y organización del proyecto

1.1. Introducción

Tras haber cursado la asignatura de Lógica Informática con el profesor D. Fernando Sancho Caparrini, director de este proyecto, se despertó mi gusto por la Teoría de la Lógica Matemática y Computacional. Aunque desde el punto de vista teórico la asignatura presentaba los contenidos de forma completa y con total formalismo y corrección, los profesores encargados de la misma pretendían darle a la misma un enfoque más práctico de manera que estos nuevos contenidos sirvieran como complemento a los conceptos formales presentados teóricamente.

1.2. Objetivos del proyecto

El proyecto persigue un doble objetivo,

1.3. Estructura del proyecto

El proyecto se estructará en 2 partes fundamentales:

1.3.1. Módulos funcionales

Se implementan distintos módulos, que se describen detalladamente a lo largo del proceso, en el que se desarrollarán las estructuras y métodos necesarios para abordar los algoritmos vistos en la asignatura.

1.3.2. Interfaz gráfica

Capítulo 2

LP I. Sintaxis y Semántica

2.1. Introducción. Descripción general del capítulo

TO DO

2.2. Módulo SintaxSemanticsLP

En este primer capítulo vamos a estudiar, desde un punto de vista práctico, a través de las implementaciones, los distintos elementos que conforman la Lógica Proposicional, esto es la Síntaxis y la Semántica que se han presentado en los fundamentos teóricos.

2.2.1. Aspectos Sintácticos

Átomo y Fórmula Proposicional

Recordemos que las expresiones están formadas por símbolos proposicionales, organizados en átomos (expresiones básicas) y relaciones entre ellas (conectivas), de acuerdo a esto, podemos definir un símbolo proposicional como una cadena de texto:

```
type alias PSymb = String
```

Listing 2.1: Definición de Símbolo Proposicional como alias de String.

Y las conectivas lógicas como categorías del tipo recursivo *Prop*, de forma que:

```
type Prop = Atom PSymb
          | Neg Prop
          | Conj Prop Prop
          | Disj Prop Prop
          | Impl Prop Prop
          | Equi Prop Prop
```

Listing 2.2: Definición del tipo Prop (Proposición).

Esto nos permite definir todas las fórmulas proposicionales que se pueden formar, algunos ejemplos:

(a) $(p \wedge q) \vee (p \wedge r)$ (b) $(p \wedge r) \vee (\neg p \wedge q) \rightarrow \neg q$ (c) $(p \leftrightarrow q) \wedge (p \rightarrow \neg q) \wedge p$

```
a = Disj (Conj (Atom "p") (Atom "q")) (Conj (Atom "p") (Atom "r"))

b = Impl
        (Disj
            (Conj (Atom "p") (Atom "r"))
            (Conj (Neg (Atom "p")) (Atom "q"))
        )
        (Neg (Atom "q"))

c = Conj
        (Conj
            (Equi (Atom "p") (Atom "q"))
            (Impl (Atom "p") (Neg (Atom "q")))
        (Atom "p")
```

Listing 2.3: Ejemplos de definición de fórmulas proposicionales.

Como se puede apreciar, escribir las fórmulas de esta forma puede resultar una tarea ardua y propensa a errores, por eso, como mostraremos más adelante, se ha desarrollado un parser que nos permite escribir de forma más cómoda, sintética y visual las fórmulas. Aunque aquí no influyen las reglas de asociación vistas en los fundamentos teóricos, ya que dicho orden viene dado por el propio orden de aplicación, sí hemos de tenerlas en cuenta a la hora de realizar el parser.

Conjuntos de fórmulas

Definidas las fórmulas proposicionales, definiremos los conjuntos de fórmulas como listas de fórmulas proposicionales de manera que una misma fórmula puede aperecer varias veces en el conjunto. De esta forma:

```
type alias PropSet = List Prop
```

Listing 2.4: Definición de Conjunto de Fórmulas como Lista de Fórmulas Proposicionales

2.2.2. Aspectos Semánticos

Interpretaciones y Valor de verdad de fórmulas proposicionales.

Una vez provista la sintaxis, pasamos a desarrollar la semántica de la Lógica Proposicional. Aunque en secciones futuras mostraremos la implementación del parser, por ahora vasta conocer la estructura de las fórmulas para hacer abordable la implementación que se ha llevado a cabo de la semática del lenguaje de la lógica de proposiciones.

Como ya se ha comentado, y explicado en los fundamentos teóricos, hemos de abordar la interpretación de las fórmulas, y para ello hemos de crear las estructuras necesarias que nos permitan representar interpretaciones, esto es, definir los símbolos proposicionales con valor de verdad 1 y aquellos con valor 0. Para ello, hemos elegido una representación "dispersa"de manera que una interpretación corresponde a una lista de símbolos proposicionales que son los que son considerados verdaderos, los términos que no aparecen en la lista serán considerados como falsos.

```
type alias Interpretation = List PSymb
```

Listing 2.5: Definición de Interpretación como Lista de Símbolos Proposicionales

De esta forma podemos definir, de forma sencilla la evaluación de las fórmulas, de forma que el valor de verdad de un símbolo se reduce a la pertenencia del mismo a la lista de interpretación, de esta forma, podemos, dada la estructura recursiva definida para las fórmulas proposicionales, establecer una función, tambieén recursiva, que nos permite evaluar las mismas:

- (*Caso base*) Una fórmula atómica será verdadera si y sólo si el símbolo proposicional pertenece a la lista de interpretación.
- (Casos recursivos) Según la clase de fórmula:
  - La negación de una fórmula será verdadera respecto de una interpretación si y sólo si la evaluación de la fórmula es falsa.
  - La conjunción de dos fórmulas proposicionales será verdadera respecto de una interpretación si y sólo si la evaluación de ambas fórmulas respecto de dicha evaluación es verdadera.
  - La disyunción de dos fórmulas proposicionales será verdadera será verdadera respecto de una interpretación si y sólo si alguna de las evaluaciones de las dos fórmulas es evaluada verdadera respecto de dicha interpretación.
  - La implicación será verdadera respecto de una interpretación si y sólo si o la evaluación del antecedente es evaluado falso respecto de dicha interpretación o el consecuente es evaluado verdadero respecto de la misma.
  - La equivalencia será verdadera respecto de una interpretación si y sólo si la evaluación del antecedente coincide con la evaluación del consecuente.

```
valuation : Prop -> Interpretation -> Bool
valuation pr i =
    case pr of
        Atom p -> List.member p i
        Neg p -> not (valuation p i)
        Conj p q -> valuation p i &&  valuation q i
        Disj p q ->   valuation p i ||  valuation q i
        Impl p q ->   not (valuation p i) ||  valuation q i
        Equi p q ->   valuation (Impl p q) i &&  valuation (Impl q p) i
```

Listing 2.6: Función de evaluación de las fórmulas proposicionales

Modelos (tablas de verdad, satisfactibilidad y validez lógica)

Desde el punto de vista teórico las tablas de verdad corresponden a estructuras que reflejan el valor de verdad de una fórmula proposicional respecto de cada una de las posibles interpretaciones posibles para la fórmula. Entonces, para poder construir la tabla de verdad, primero hemos de calcular todas las interpretaciones posibles, que, dada la definición que hemos proporcionado para las interpretaciones, correspondería a todos los conjuntos posibles (*powerset*) que podríamos construir con los símbolos proposicionales que aparecen en la fórmula. Así:

```
symbInProp : Prop -> Set PSymb

symbInProp f=
    case f of
        Atom p -> Set.singleton p
        Neg p -> symbInProp p
        Conj p q -> Set.union (symbInProp p ) (symbInProp q)
        Disj p q -> Set.union (symbInProp p ) (symbInProp q)
        Impl p q -> Set.union (symbInProp p ) (symbInProp q)
        Equi p q -> Set.union (symbInProp p ) (symbInProp q)
```

Listing 2.7: Función para extraer los símbolos proposicionales que intervienen en una fórmula

```
allInterpretations : Prop -> List Interpretation
allInterpretations x =  Aux.powerset <| List.sort <| Set.toList
                                                    <| symbInProp x
```

Listing 2.8: Función para extraer las posibles interpretaciones para una fórmula propsicional

De esta forma podemos expresar la tabla de verdad como una lista de tuplas en las que el primer elemento corresponde a la interpretación y el segundo corresponde a la evaluación de la fórmula respecto de dicha valoración:

```
truthTable : Prop -> List (Interpretation, Bool)
truthTable x = List.map (\xs ->  (xs,valuation x xs)) <| allInterpretations x
```

Listing 2.9: Función para la construcción de la tabla de verdad de una fórmula

Una vez estudiado lo anterior, los modelos corresponden a las interpretaciones que son evaluadas verdaderas, esto es, de las posibles interpretaciones aquellas hacen la fórmula verdadera. Aquellas interpretaciones que hacen la fórmula falasa se denominan contramodelos. Así:

```
models : Prop -> List Interpretation
models x = List.filter (\y -> valuation x y) (allInterpretations x)

countermodels : Prop -> List Interpretation
countermodels x = List.filter (\y -> not (valuation x y))
                                              (allInterpretations x)
```

Listing 2.10: Función para el cálculo de los modelos de una fórmula proposicional

Definidos los modelos, podemos así mismo definir (funcionalmente) los conceptos de satisfactibilidad y validez, de forma que:

- Una fórmula es satisfactible si posee al menos un modelo.
- Una fórmula es lógicamente válida o tautología si toda interpretación es modelo de la fórmula.
- Una fórmula es insatisfactible o contradicción si no posee ningún modelo.

De esta forma:

```
satisfactibility : Prop -> Bool
satisfactibility x = List.any (\xs-> valuation x xs) (allInterpretations x)

validity : Prop -> Bool
validity x = List.all (\xs-> valuation x xs) (allInterpretations x)

insatisfactibility : Prop -> Bool
insatisfactibility x = not (satisfactibility x)
```

Listing 2.11: Funciones de Satisfactibilidad, Validez e Insatisfactibilidad

Conjuntos de Fórmulas. Modelos, Consistencia, Validez y Consecuencia Lógica

Vista la satisfactibilidad, modelos, etc. aplicadas a una fórmula, pasamos a desarrollar los métodos necesarios para el estudio de la satisfactibilidad (consistencia) , inconsistencia en conjuntos de fórmulas. Ahora los modelos del conjunto de fórmulas corresponden a las interpretaciones tales que hacen verdaderas todas las fórmulas del conjunto. Para obtener los modelos hemos, al igual que en el caso de las fórmulas, obtener el conjunto de símbolos proposicionales y a partir de estos el conjunto de todas las posibles interpretaciones. De forma que:

```
setSymbols : List Prop -> Set PSymb
setSymbols xs =
    List.foldr (\x acc -> Set.union acc (symbInProp x)) Set.empty xs

allSetInterpretations : List Prop -> List Interpretation
allSetInterpretations xs = Aux.powerset <| Set.toList <| setSymbols xs

isSetModel : List Prop -> Interpretation -> Bool
isSetModel xs i =  List.all (\x -> valuation x i) xs

allSetModels : List Prop -> List Interpretation
allSetModels xs = List.filter (isSetModel xs) (allSetInterpretations xs)

allSetCounterModels : List Prop -> List Interpretation
allSetCounterModels xs =
    List.filter (\x -> not(isSetModel xs x)) <| allSetInterpretations xs
```

Listing 2.12: Modelos y contramodelos en conjuntos de fórmulas proposicionales

De forma que ahora, es sencillo, comprobar la consistencia de un conjunto a partir de la definición: *'Un conjunto es consistente si posee, al menos, un modelo. En caso contrario es inconsistente'*

```
isConsistent : List Prop -> Bool
isConsistent xs =
    List.any (\x -> isSetModel xs x) <| allSetInterpretations xs

isInconsistent: List Prop -> Bool
isInconsistent xs = not(isConsistent xs)
```

Listing 2.13: Consistencia e Inconsistencia en Conjuntos Proposicionales

Por último nos queda definir el concepto de consecuencia lógica. Acudiendo a la definición: *'Una fórmula es consecuencia lógica de un conjunto de fórmulas si y sólo si todo modelo del conjunto es también modelo de la fórmula'*, pero también *'Una fórmula es consecuencia lógica de un conjunto de fórmulas si la unión del conjunto y el conjunto formado por la negación del la fórmula es inconsistente.'*. De esta forma podemos plantear dos desarrollos alternativos:

```
isConsecuence : List Prop -> Prop -> Bool
isConsecuence xs x = List.all (\y -> valuation x y) <| allSetModels xs

isConsecuence : List Prop -> Prop -> Bool
isConsecuence xs x = isInconsistent (xs ++ [Neg x])
```

Listing 2.14: Consecuencia Lógica

2.2.3. Código del módulo y resumen de funciones

2.3. Módulo LP_toString

2.4. Módulo A_Expressions (Expr. Aritméticas)

2.5. Módulo B_Expressions (Expr. Booleanas)

2.6. Módulo LP_Parser

2.7. Módulo LPBig_Parser

Capítulo 3

LP II. Tableros Semánticos

Capítulo 4

LP III. Formas Normales y DPLL

Capítulo 5

LP IV. Sistemas Deductivos

Capítulo 6

LP V. Algoritmo de Resolución

Capítulo 7

LP VI. Modelado y Resolución de PSR a través de la LP

Capítulo 8

LPO I. Sintaxis y Semántica

Capítulo 9

LPO II. Tableros Semánticos

Capítulo 10

LPO III. Forma Prenex, Skolem y Teorema de Herbrand

Capítulo 11

LPO V. Algoritmos de Unificación y Resolución

Capítulo 12

LPO VI. Modelado y Resolución de PSR a través de la Lógica de Primer Orden