Introducción a la Lógica (para Informáticos)

Luis M. Pardo

22 de mayo de 2006

Índice general

Capítulo 1

Lógica en el Origen de la Informática

1.1. Introducción

En estas notas indicaremos solamente algunas de las plabaras claves y las nociones y notaciones que usaremos durante el curso. Evitaremos, en lo posible, las descripciones pormenorizadas que son las que se imparten en las clases teóricas. En particular, en estas notas (por ser un simple resumen) omitiremos en su mayor parte los ejemplos discutidos en clase. Dejamos al lector combinar las notas de clase con lo que aquí se recoge.

1.2. Algunos nombres fundamentales

La Lógica ha evolucionado a lo laro de la historia hasta establecerse, desde finales del diglo XIX hasta la actualidad siguiendo dos frentes de actuación.

1.2.1. Lógica como fundamento de la Matemática.

Es la motivaci1on principal, sobre todo desde finales del siglo XIX y principios del XX. Destacamos dos nombres:

- B. Russell (y su obra con Whitehead sobre los *Principa Mathematica*).
- D. Hilbert (y su programa de axiomatiación de la Matemática).

Estos aspectos interesan menos al curso que pretendemos impartir.

1.2.2. Lógica como Fundamento de la Informática.

En el año 1990, D. Hilbert imparte una conferencia plenaria en el ICM basada en 23 problemas que considera los problemas centrales que deben ser tratados por la matemática. Entre esos problemas destacamos el que más incide en el futuro desarrollo de la informática.

Problema 1 (Problema X de Hilbert) *Dar un algoritmo que resuelva el siguiente problema:*
Dado un polinomio univariariado $f \in \mathbb{Q}[X_1, \ldots, X_n]$, decidir si existe una solución en $\mathbb{Q}^n$, es decir, decidir si existe $x \in \mathbb{Q}^n$ tal que $f(x) = 0$.

Algoritmo=programa Es el término usado por los matemáticos para designar a los programas. En la primera mitad del siglo IX, el matemático uzbeko Muhammad ibn–Musa Al–Juaritzmi escribió su tratado *"Hisab al–jabr wa–al–muqabala"* (traducio libremente por Libro (o Tratado) sobre las operaciones "jabr" (restablecimiento) y "qabala" (reducción). Los científicos europeos comenzaron a conocer el álgebra a principios del siglo XII justamente a partir de una traducción al latín de este tratado. Obsérvese que aparecen conectados dos grupos de fonemas que hoy son de uso común "al–jabr" (álgebra) y "al–Juaritzmi" (algoritmo). Durante 800 años, los matemáticos habían diseñado multitud de algoritmos, pero nadie sabía qué era un algoritmo. Esa fué la principal dificultad del Problema de Hilbert.

Los Años 30 del siglo XX. Se produce la conjunción (independiente y original) de varios científicos, cada uno de los cuales define independientemente una noción de algoritmo:

1. En 1931, *K. Gödel* publica su tesis[1] (23 páginas) sobre la incompletitud de algunos sistemas formales. Volveremos a ello. Pero aquí Gödel usa por vez primera algo parecido a las funciones computables (él las llamó "rekursiv"), aunque no toma fondo en el asunto. Son las llamadas "primitivas recursivas", i.e. las que permiten hallar $f(n+1)$ de la información de $f(n)$ salvo composición y proyección.

2. A principios de los años 30, *A. Turing* estaba ya interesado en los trabajos sobre computacin y Algebra. En su trabajo de 1948 A. Turing [2] introducir la nocin de condicionamiento de los mtodos del Algebra Lineal Numrica, convirtindose en el fundador del Algebra Lineal Numrica moderna.
   A la sazón, A. Turing publicaba su modelo en su trabajo de 1936 [3] dedicado a caracterizar los nmeros reales computables (recursivamente enumerables) y ya hacía referencia al trabajo de Church. Probó en un apéndice la equivalencia entre su modelo y la $\lambda-$definibilidad. De hecho, dos son las aportaciones fundamentales de Turing en este artculo. De una parte, la introduccin de un nuevor modelo alternativo de algoritmo (mquinas de Turing) y el resultado de autoreducibilidad basado en la mquina Universal. De otro, el anlisis de los nmeros reales recursivamente enumerables y la demostracin de que $\mathbb{R}_{re}$ no es un cuerpo computable.

3. En Princeton, *A. Church* dirige la tesis de *S.C. Kleene* sobre una nocin de recursividad alternativa a la nocin introducida por Gödel. Se trata de la nocin de $\lambda-$*calculus* que ha caido relativamente en desuso. En los trabajos de Kleene [4] y Church [5] demuestran que su nocin de algoritmo y la nocin propuesta por K. Gödel son exactamente la misma. Est naciendo la **Tesis de Church**.

---

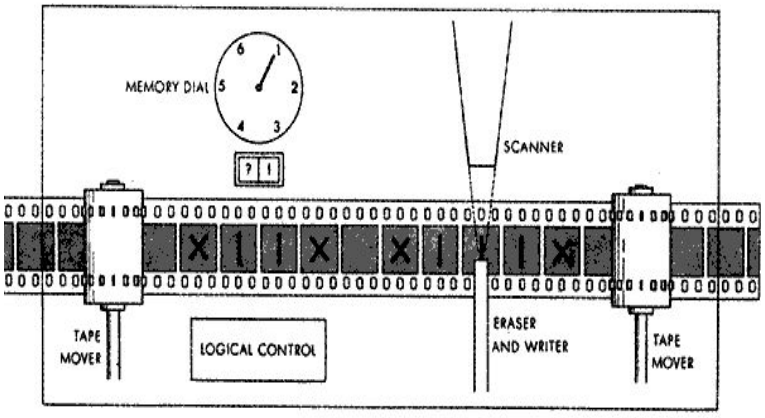
[1] K. Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I". *Monatsh. Math. Phys.* **38** (1931) 173–198.

[2] A. Turing."Rounding-off errors in matrix processes". *Quart. J. Mech. Appl. Math.*, **1** (1948) 287–308.

[3] A. Turing. "On computable numbers, with an application to the Enscheidungspoblem". *Proc. London Math. Soc., Ser. 2,* **42** (1936) 230–265.

[4] S.C. Kleene. "$\lambda-$definability and recursiveness". *Duke Math. J.* **2** (1936) 340–353.

[5] A. Church. "An unsolvable problem of elementary number theory". *Am. J. Math.* **58** (1936) 345–363.



Dibujo de Lee Manovich

La Tesis de Church. *Llamaremos algoritmo a cualquier formalismo equivalente a una máquina de Turing, a una función recursiva o al $\lambda-$calculus.*

El Problema X de Hilbert. Los trabajos de J. Robinson y J. Matijasievich finalmente demostraron que la respuesta al problema X de Hilbert es la siguiente:
No puede existir ningún algoritmo o programa que resuelva el problema propuesto por Hilbert en su Problema X.
En términos de informática hay una versión más importante ya descrita por Turing en su trabajo de 1936:

Teorema 1 (Halting Problem) *No puede existir ningún algoritmo que realice la siguiente tarea:*
Dado un programa P y un input x del programa, decidir si P finaliza sus computaciones sobre x.

Bletchley Park. (Muy brevemente) Durante la Segunda Guerra Mundial, Churchill decide poner en marcha un proyecto de inteligencia militar llamado Proyecto Ultra. Este proyecto tiene como objetivo interceptar las comunicaciones militares alemanas. Para ello establece, en un lugar llamado Bletchley Park, una amplísimo equipo de técnicos y científicos cuya principal misión consiste en decodificar los mensales cifrados de las comunicaciones alemanas. Al frente del equipo se encuentra el matemático A. Turing.
Al principio de la Guerra los alemanes usan un sistema de codificación basado en la máquina Enigma de tres rotores, que pasarán a ser cuatro durante la Batalla de Inglaterra. El sistema usado para descodificar e interceptar estas comunicaciones se basa en el uso de rgleas de probabilidad: dada la inmensa masa de comunicaciones por radio de la guerra, ciertas palabras son más repetidas que otras. Usano un amplio equipo de gente se pueden ir decantando las interpretaciones más probables y descifrando el mensaje.
Avanzada la guerra, los submarinos alemanes de Doenizt introducen la máquina Lorentz de 12 rotores. La cantidad de combinaciones posibles y la necesidad de agilizar los cálculos, exigen la creación de una máquina física que permita descodificar (o ayudar a los descodificadores) de la manera más eficiente. A partir de las ideas de Turing se crean los primeros dos ordenadores de la historia Colossus I y Colossus II.

AL final de la Seunda Guerra Mundial, los británicos imponen el secreto de sus proyectos militares con lo que nadie pudo hablar (ni nadie supo) de la existencia de estos ordenadores haasta mediados de los años 90. Los estadounidenses conocen de la existencia de esta experiencia cuando Churchill informa a Eisenhower. Aprovecharán de ellas para el diseño del ENIAC para cálculo de trayectorias balísticas. Los soviéticos harán lo propio dado que Stalin tiene el correspondiente espía en Bletchley Park

La máquina de Turing como patrón. Salvo variantes menores, la máquina de Turing sigue siendo el patrón de medida de la computación, al menos en los dos ingredientes fundamentales de la eficacia:

- *Tiempo* (o tiempo de ejecución de un programa sobre una máquina)
- *Espacio* (o memoria requierida para ejecutar una programa).

Nociones como *bit, byte* y sus variantes (Megabyte, Gigabyte. Terabyte) tienen su origen en esta noción de Turing.
El número de Mips (millions of instructions per second) sigue siendo el parámetro que mide la velocidad de un ordenador. Así en la lista top500 de *Supercomputing*, el ordenador más rápido del mundo parece ser el *Blue Gene/L* de IBM con una velocidad de 300 Teraips, es decir $3{,}10^8 Mips$.

1.3. Programación Lógica

El objetivo del curso será orientar a los alumnos hacia la comprensión de los principios del paradigma de la Programación Lógica.

Capítulo 2

Cálculo Proposicional

2.1. Introducción

Los sistemas lógicos (o teorías lógicas) son estructuras que se definen a través de los siguientes ingredientes:

2.1.1. Sintaxis

Fija dos elementos fundamentales: *el alfabeto* y *la clase de fórmulas bien formadas.*

Alfabeto. Es un conjunto finito o numerable de símbolos que se está autoriado a usar. Los símbolos que no aparecen en ese alfabeto no son admitidos ni utilizables.

Fórmulas. Son algunas listas de símbolos sobre el alfabeto (palabras sobre el alfabeto) que serán admitidas como bien formadas. El resto se descartan. Habitualmente, las fórmulas bien formadas se definen mediante un proceso recursivo (en el sentido goedeliano de computable). Estas reglas constituyen la gramática de nuestra teoría.

2.1.2. Reglas Deductivas.

Son las reglas de transformación de fórmulas. En términos matemáticos, son las reglas que permiten construir *Demostraciones* y, por tanto, permiten concluir *Teoremas.* Aunque no es el momento de entrar en detalles, el proceso de deducción es un proceso asociado a un sistema de transición y, por ende, indéntico en esquema al proceso de computación (eso sí, cuando la teoría sea decidible!.

2.1.3. Semántica.

Con la sola aparición de la sintaxis no podemos tener una teoría, del mismo modo que con el mero conocimiento de las reglas graaticales de la lengua castellana no tenemos literatura o conversación entre individuos. Para que todas las piezas encajen, se necesita poder asignar valores semánticos (esto es significados) a las fórmulas o frases aceptadas como "bien formadas". Las asignaciones de significado se llaman *interpretaciones* y las reglas que permiten usar las interpretaciones se llaman reglas semánticas.

Nota 2 *En las Teorías a tratar en este curso evitaremos la descripciónd e las correspondiente reglas deductivas.*

2.2. Sintaxis y Semántica del Cálculo Proposicional

Concepto:

Paradoja 1 *Esta frase es mentira.*

Objetivo: Separación de la sintaxis (lo que se escribe) de la semántica (lo que significa). Ideas de A. Tarski.

2.2.1. Sintaxis del Cálculo Proposicional

Alfabeto Son los símbolos que nos está permitido usar. En el Cálculo Prosicional aparecen los siguientes tipos:

- Fórmulas Atómicas:
  - *Variables:* $X_1, \ldots, X_n, \ldots ...$.
  - *Constantes:* $0, 1$
- *Conectivas:* Son los símbolos usados para relacionar los anteriores:
  - *Conjunción:* $\wedge$
  - *Disyunción:* $\vee$
  - *Negación:* $\neg$.
- *Delimitadores:* Usualmente los paréntesis. (,).

Reglas Sintácticas. Se definen de manera recursiva en los términos siguientes.

1. Las fórmulas atómicas son fórmulas[1].
2. Si F es una fórmula, también lo es $(\neg F)$.
3. Para cualesquiera fórmulas F y G, también son fórmulas $(F \vee G)$ y $(F \wedge G)$.

Definición 3 *El menor lenguaje (subconjunto) de palabras sobre el alfabeto anterior, conteniendo las fórmulas atómicas y cerrado ante las reglas sintácticas anteriores, se llama el conjunto de las fórmulas del cálculo proposional.*

Abreviaturas La presencia de un número tal de paréntesis hace que la lectura de algunas de esas fórmulas pueda parecer incómoda. En otras ocasiones, se usan medios de escritura más próximos a la semántica. Para ello se introducen ciertas abreviaciones de estas expresiones. Las más usadas son:

- *Implicación:* Se escribe $(F \to G)$ en lugar de $((\neg F) \vee G)$, para F y G.
- *Equivalencia:* Se escribe $(F \leftrightarrow G)$ en lugar de $(((\neg F) \vee G) \wedge (F \vee (\neg G)))$, para F y G[2].
- *O exclusivo:* Se escribe $(F \oplus G)$ en lugar de $((F \wedge (\neg G)) \vee (G \wedge (\neg F)))$

[1] Algunos autores usan el término fómulas ien formadas. Aquí usaremos esta término más corto.
[2] Nótese que equivalencia es una abreviación de $((F \to G) \wedge (G \to F))$.

Notaciones No son propiamente abreviaciones de fórmulas lógicas (dado que hacen intervenir muchos otros símbolos) pero simplifican la escritura de algunas de las fórmulas.

- *Conjunción de varias fórmulas.* Dadas n fórmulas $F_1, \ldots, F_n$, se usa

$$\left(\bigwedge_{i=1}^{n} F_i\right),$$

para denotar

$$(\cdots((F_1 \wedge F_2) \wedge F_3) \wedge \cdots).$$

- *Disyunción de varias fórmulas.* Dadas n fórmulas $F_1, \ldots, F_n$, se usa

$$\left(\bigvee_{i=1}^{n} F_i\right),$$

para denotar

$$(\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots).$$

- *O exclusivo de varias fórmulas.* Dadas n fórmulas $F_1, \ldots, F_n$, se usa

$$\left(\bigoplus_{i=1}^{n} F_i\right),$$

para denotar

$$(\cdots((F_1 \oplus F_2) \oplus F_3) \oplus \cdots).$$

Nota 4 *En la tradición de la Electrónica Digital se suelen usar las "puertas" NAND y NOR para denotar:*

$$NAND(F, G) \text{ denotando } (\neg(F \wedge G)).$$
$$NOR(F, G) \text{ denotando } (\neg(F \vee G)).$$

Definición 5 (Árbol de Formación de una fórmula) *Sea define mediante la siguiente regla recursiva. Sea F una fórmula y $T(F)$ su árbol de formación.*

- *Si F es una constante $F = c \in \{0, 1\}$, $T(F) = c$.*
- *Si F es una variable $F = X_i \in \{X_1, X_2, \ldots, X_n, \ldots\}$, $T(F) = X_i$.*
- *Si $F = (G \wedge H)$ con G y H fórmulas, entonces:*

$$T(F) = \begin{array}{ccc} & \uparrow & \\ & \boxed{\wedge} & \\ \swarrow & & \nwarrow \\ T(G) & & T(H) \end{array}$$

- *Si $F = (G \vee H)$ con G y H fórmulas, entonces:*

$$T(F) = \begin{array}{ccc} & \uparrow & \\ & \boxed{\vee} & \\ \swarrow & & \nwarrow \\ T(G) & & T(H) \end{array}$$

- *Si $F = (\neg G)$ con G fórmula, entonces:*

$$T(F) = \begin{array}{c} \uparrow \\ \boxed{\neg} \\ \uparrow \\ T(G) \end{array}$$

Definición 6 (Subfórmula de una fórmula) *Dadas dos fórmulas F y G, decimos que G es una subfórmula de F si existen palabras ω y ω' sobre el alfabeto del Cálculo Proposicional, tales que*

$$F = \omega G \omega'.$$

Nota 7 *El árbol de formación de una subfórmula es un sub-árbol del árbol de formación de la fórmula completa.*

2.2.2. Semántica del Cálculo Proposicional.

Interpretación. Una interpretación del Cálculo Proposicional es una sucesión $\mathcal{E}$ de dígitos en el conjunto $\{0, 1\}$, es decir,

$$\mathcal{E} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \ldots),$$

donde $\varepsilon_i \in \{0, 1\}$.

Valor de Verdad de una Fórmula. Dada una fórmula F del Cálculo Proposicional y dada una interpretación

$$\mathcal{E} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \ldots),$$

definimos el valor de verdad de F en $\mathcal{E}$ $eval(F, \mathcal{E})$ de acuerdo a las reglas siguientes:

---

if $F = c \in \{0, 1\}$ es una constante, **then** $eval(F, \mathcal{E}) = c$,
elif $F = X_i$, **then** $eval(F, \mathcal{E}) = \varepsilon_i$
elif $F = (G \vee H)$, **then**

$$eval(F, \mathcal{E}) = \begin{cases} 1 & \text{si } eval(G, \mathcal{E}) = 1 \\ 1 & \text{si } eval(H, \mathcal{E}) = 1 \\ 0 & \text{en otro caso} \end{cases}$$

elif $F = (G \wedge H)$, **then**

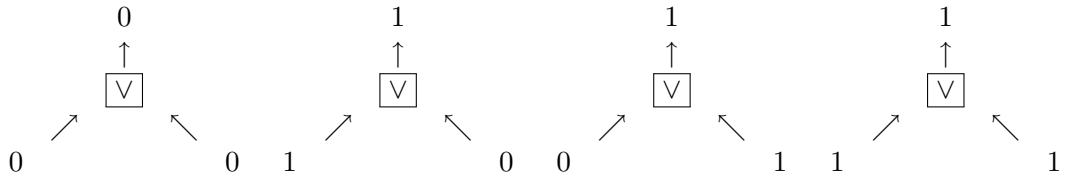
$$eval(F, \mathcal{E}) = \begin{cases} 1 & \text{si } eval(G, \mathcal{E}) = eval(H, \mathcal{E}) = 1 \\ 0 & \text{en otro caso} \end{cases}$$

elif $F = (\neg G)$, **then**

$$eval(F, \mathcal{E}) = \begin{cases} 1 & \text{si } eval(G, \mathcal{E}) = 0 \\ 0 & \text{si } eval(G, \mathcal{E}) = 1 \end{cases}$$

fi

Evaluación y Árbol de Formación. Reconstruir el efecto de la asignación de verdad de una fórmula a través del árbol de formación y de los distintos nodos/puertas que pueden aparecer. Como en el siguiente ejemplo:



Repetir con $\boxed{\wedge}, \boxed{\neg}, \boxed{\oplus}, \boxed{\rightarrow}, \boxed{\leftrightarrow}$.

Definición 8 (Términos Básicos) *Introducimos os siguientes términos:*

- *Una interpretación $\mathcal{E}$ se dice que es modelo para una fórmula F si $eval(F, \mathcal{E}) = 1$. Se suele usar también la expresión F satisface $\mathcal{E}$. La notación es $\mathcal{E} \models F$.*
- *Una fórmula se dice satisfactible si existe algún modelo $\mathcal{E}$ que la satisface. Aquellas fórmulas para las que no hay ningún modelo que la satisface, se llaman insatisfactibles.*
- *Una fórmula F se dice tautología si para cualquier interpretación $\mathcal{E}$, entonces, $eval(F, \mathcal{E}) = 1$.*
- *Dos fórmulas F, G se dicen tautológicamente equivalentes si para cualquier interpretación $\mathcal{E}$, $eval(F, \mathcal{E}) = eval(G, \mathcal{E})$. Denotaremos $F \equiv G$ para decir que F y G son tautológicamente equivalentes.*

Nota 9 *Un par de apreciaciones inmediatas:*

- *Dos fórmulas F y G son tautológicamente equivalentes si y solamente si para toda interpretación $\mathcal{E}$, $\mathcal{E}$ es modelo de F si y solamente si $\mathcal{E}$ es modelo de G.*
- *Dos fórmulas F y G son tautológicamente equivalentes si sy solamente si $(F \leftrightarrow G)$ es una tautología.*

Teorema 10 *Una fórmula F es tautológica si y solamente si $(\neg F)$ es insatisfactible.*

2.2.3. Tablas de Verdad, Funciones Booleanas.

Diremos que una fórmula F es expresable con variables en $\{X_1, \ldots, X_n\}$ si las variables usadas para escribir F están en el conjunto $\{X_1, \ldots, X_n\}$. En ocasiones utilizaremos la notación $F(X_1, \ldots, X_n)$ para decir que F es expresable con variables en $\{X_1, \ldots, X_n\}$.

Proposición 11 *Dadas dos fórmulas $F(X_1, \ldots, X_n)$ y $G(X_1, \ldots, X_m)$ entonces, las fórmulas $(F \vee G)$, $(F \wedge G)$, $(F \rightarrow G)$, $(F \oplus G)$ son expresables con variables en $\{X_1, \ldots, X_r\}$, donde $r = max\{n, m\}$.*

Proposición 12 *Dada una fórmula F expresable con variables en $\{X_1, \ldots, X_n\}$, entonces $(\neg F)$ también es expresable con variables en $\{X_1, \ldots, X_n\}$.*

Teorema 13 *Sea F una fórmula expresable sobre las variables $\{X_1, \ldots, X_n\}$. Sea $\mathcal{E} = (\varepsilon_1, \ldots, \varepsilon_n, \ldots)$ y $\mathcal{E}' = (\varepsilon'_1, \ldots, \varepsilon'_n, \ldots)$ dos interpretaciones.*
Si $\varepsilon_i = \varepsilon'_i$ para todo i, $1 \leq i \leq n$, entonces,

$$eval(F, \mathcal{E}) = eval(F, \mathcal{E}').$$

Nota 14 *En otras palabras, si F una fórmula expresable sobre las variables $\{X_1, \ldots, X_n\}$ el valor de cualquier interpretación $\mathcal{E}$ en F depende solamente de los n primeros términos de la suceión $\mathcal{E}$.*

El Conjunto $\{0,1\}^n$. Es el conjunto de las plabaras de longitud n sobre el alfabeto $\{0,1\}$. En otros términos:

$$\{0,1\}^n = \{(\varepsilon_1, \ldots, \varepsilon_n) \ : \ \varepsilon_i \in \{0,1\}\}.$$

Su cardinal (es decir, el número de sus elementos) viene dado por

$$\sharp(\{0,1\}^n) = 2^n.$$

El número de subconjuntos de $\{0,1\}^n$ viene dado por la siguiente expresión:

$$\sharp\left(\mathcal{P}(\{0,1\}^n)\right) = 2^{2^n}.$$

Definición 15 (Funciones Booleanas) *Se llaman función booleana a toda aplicación*

$$f : \{0,1\}^n \longrightarrow \{0,1\}.$$

Se llama aplicación booleana a toda lista finita de funciones booleanas, esto es, a toda aplicación

$$f : \{0,1\}^n \longrightarrow \{0,1\}^m.$$

La lista $f = (f_1, \ldots, f_m)$, donde cada f_i es una función boolena, se llama representación de f por sus coordenadas.
Deotaremos por $\mathcal{B}_n$ el conjunto de todas las funciones booleanas $f : \{0,1\}^n \longrightarrow \{0,1\}$.

Ejemplo 1 *Formas de obtener funciones booleanas.*

- *Sea F una fórmula del Cálculo Proposicional expresable con las variables $\{X_1, \ldots, X_n\}$. Entonces, F define una función booleana (que denotaremos con la misma letra por ahora) en la form siguiente:*

  $$F : \{0,1\}^n \longrightarrow \{0,1\}$$

  de tal modo que para cada $\underline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$,

  $$F(\underline{\varepsilon}) := eval(F, \mathcal{E}),$$

  para cualquier interpretación $\mathcal{E}$ tal que sus n primeros términos coincidan con los de $\underline{\varepsilon}$.

- *Para cada subconjunto $X \subseteq \{0,1\}^n$, definimos la función característica como sigue:*

  $$\chi_X : \{0,1\}^n \longrightarrow \{0,1\}$$

  De tal modo que para cada $\underline{\varepsilon} \in \{0,1\}^n$, se define

  $$\chi_X(\underline{\varepsilon} := \begin{cases} 1, & \text{si } \underline{\varepsilon} \text{ pertenece a } X \\ 0, & \text{en otro caso} \end{cases}$$

Teorema 16 *Se verifican las siguientes propiedades:*

1. *Toda función booleana es la función característica de un único subconjunto X de $\{0,1\}^n$. Por tanto, el número de funciones booleanas es igual al número de subconjuntos de $\{0,1\}^n$, esto es*
$$\sharp(\mathcal{B}_n) = 2^{2^n}.$$

2. *La función constante 1 es la función característica del subconjunto $\{0,1\}^n \subseteq \{0,1\}^n$, esto es*
$$1 = \chi_{\{0,1\}^n} : \{0,1\}^n \longrightarrow \{0,1\}..$$

3. *La función constante 0 es la función característica del subconjunto vacío $\emptyset \subseteq \{0,1\}^n$, esto es*
$$1 = \chi_{\emptyset} : \{0,1\}^n \longrightarrow \{0,1\}.$$

4. *Toda función booleana está definida por alguna fórmula del cálculo proposicional.*

5. *Dada una fórmula F expresable con las variables $\{X_1, \ldots, X_n\}$, F es una tautología si y solamente si la función booleana que define es la función constante 1, esto es F es tautología si y solamente si*
$$F = 1 : \{0,1\}^n \longrightarrow \{0,1\}.$$

6. *Dada dos fórmulas F y G expresables con las variables $\{X_1, \ldots, X_n\}$, F y G son tautológicamente equivalentes si y solamente si definen la misma función booleana, esto es F y G son tautológicamente equivalentes y solamente si*
$$F = G : \{0,1\}^n \longrightarrow \{0,1\}.$$

7. *Una fórmula F expresable con las variables $\{X_1, \ldots, X_n\}$, entonces, F es satisfactible si y solamente si 1 está en la imagen de F, esto es, si y solamente si $F^{-1}(\{1\}) \neq \emptyset$ o, equivalentemente si y solamente si se verifica*
$$\exists \underline{\varepsilon} \in \{0,1\}^n, \;\; F(\underline{\varepsilon}) = 1.$$

Nota 17 *Como indica el enunciado anterior, toda función booleana está definida por alguna fórmula del Cálculo Proposicional. Pero es fácil demostrar que el número de fórmulas que definen una función dada es infinito y, por tanto, un problema central consiste en detectar si dos fórmulas definen una misma función booleana, lo cual es lo mismo que preguntar si dos funciones del Cálculo Proposicional son tautológicamente equivalentes.*

Tablas de Verdad. La Tabla de Verdad de una fórmula expresable con las variables $\{X_1, \ldots, X_n\}$ es simplemente una representación en forma de Tabla de la función booleana asociada a la fórmula. Fácilemente se observa que esta manera de rpresentar funciones booleanas no es la más adecuada cuando el número de variables es relativamente grande. Construir la Tabla de Verdad de una fórmula corresponde a una de las estrategias (la obvia) de decidir si es o no es satisfactible. Sin embargo, el proceso es realmente caro.

Teorema 18 *Existe un algoritmo que realiza la siguiente tarea:*
Dada una fórmula F del Cálculo Proposicional expresable sobre las variables $\{X_1, \ldots, X_n\}$ y dado $\underline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n) \in \{0,1\}^n$, el algoritmo produce el valor de verdad $F(\underline{\varepsilon})$.
El tiempo requerido por el algoritmo está acotado por cL^2, donde c es una constante y L es la talla de F(es decir, el número de símbolos, contados con sus repeticiones, usados para escribir F).
Con este algoritmo el tiempo requerido para escribir la Tabla de Verdad de F es del orden

$$c2^n L^2.$$

Usaremos los siguientes elementos:

- *Profundidad de una conectiva en una fórmula.* Sea $op \in \{\vee, \wedge, \neg\}$ una conectiva. Llamaremos profundidad de op en F (lo denotaremos como $d(op, F)$ al número de paréntesis abiertos (a la izquierda de op en F menos el número de paréntesis cerrados) a la izquierda de op en F, i.e.

$$d(op, F) = \sharp\{\text{paréntesis (a la izqda.}\} - \sharp\{\text{paréntesis) a la izqda.}\}.$$

- *Profundidad de una fórmula.* Llamaremos profundidad de una fórmula al máximo de las profundidades de sus conectivas.

- Las fórmulas de profundidad 0 son las variables y las constantes (i.e. las fórmulas atómicas).

- Las fórmulas de profundidad 1 son de uno de los tipos siguientes:

$$(X_i \; op \; X_j), (X_i \; op \; c), (c \; op \; X_i), (\neg X_i), (\neg c),$$

  donde $c \in \{0,1\}$ es una constante y $op \in \{\vee, \wedge\}$.

- *Evaluar en profundidad 1.* Dada una fórmula de profundidad 1 y dado $\underline{\varepsilon} \in \{0,1\}^n$, $eval(F, \varepsilon)$ es el valor obtenido por las tablas de verdad de F aplicado directamente a la fórmula de profundidad 1 (ver item anterior).

INPUT:

- una fórmula F del Cálculo Proposicional expresable sobre las variables $\{X_1, \ldots, X_n\}$
- $\underline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n) \in \{0,1\}^n$

$\ell := d(F)$
while $\ell > 1$ **do**
Hallar la conectiva *op* de F más a la izquierda verificando $d(op, F) = d(F)$,
if $op = \neg$ *(Comentario: op va rodeada de una expresión de la forma* $(\neg H)$*, donde* $H = X_i$ *es una variable o* $H = c \in \{0,1\}$ *es una constante)*
then do Hallar ω y ω' tales que

$$F = \omega(\neg H)\omega',$$

$$F := \omega\ eval((\neg H), \underline{\varepsilon})\ \omega',\ \ \ell = d(F),$$

(Comentario: reemplazamos $(\neg H)$ *por su valor* $eval((\neg H), \underline{\varepsilon})$*, hallamos la nueva profundidad y volvemos al while)*
else $op \in \{\vee, \wedge\}$ *(Comentario: op va rodeada de una expresión de la forma* $(G\ op\ H)$*, donde* H *y* G *son variables o constantes)*
then do Hallar ω y ω' tales que

$$F = \omega(G\ op\ H)\omega',$$

$$F := \omega\ eval((G\ op\ H), \underline{\varepsilon})\ \omega',\ \ \ell := d(F),$$

(Comentario: reemplazamos $(G\ op\ H)$ *por su valor* $eval((G\ op\ H), \underline{\varepsilon})$*, hallamos la nueva profundidad y volvemos al while)*
fi
od
OUTPUT $eval(F, \underline{\varepsilon})$

2.2.4. Circuitos Booleanos.

Representan una alternativa a las fórmulas del Cálculo Proposicional en la representación de las funciones booleanas. No haremos una descripción muy detallada de las principales propiedades, aunque sí introduciremos las nociones esenciales. Las representaciones gráficas y los ejemplos serán aquellos expuestos en las clases de Teoría.

Definición 19 (Grafo) *Un grafo finito es un par* (V, E)*, donde:*

- V *es un conjunto finito, cuyos elementos se llaman vértices o nodos.*
- E *es un conjunto de subconjuntos* X *de* V *de cardinalidad a lo sumo* 2*. Los elementos de* E *se llaman aristas del grafo.*

Nota 20 *Como el conjunto de nodos de un grafo es un conjunto finito, el conjunto de nodos V suele identificarse con un conjunto finito de números naturales $V = \{1, 2, 3, \ldots, N\}$, siendo N el número de nodos del grafo. Así, los siguientes son ejemplos de grafos:*

- $\mathcal{G}_1 = (V_1, E_1)$ *donde:*

$$V_1 := \{a, b, c, d\}, \quad E_1 := \{\{a\}, \{a, b\}, \{c, d\}, \{c\}, \{a, d\}\}.$$

- $\mathcal{G}_2 = (V_2, E_2)$ *donde:*

$$V_2 := \{1, 2, 3, 4, 5, 6\}, \quad E_2 := \{\{1, 3\}, \{1, 4\}, \{1, 5\}, \{1, 2\}, \{1, 6\}, \{2, 5\}\}.$$

El alumno debería intentar la representación gráfica de estos dos grafos y de otros que se le ocurran.

Definición 21 (Grafo orientado) *Un grafo orietando es un par (V, E), donde V es un conjunto finito, y E es un subconjunto del producto cartesiano $E \times E$, es decir, E es una lista de pares de nodos en V.*

Nota 22 *De nuevo suele usarse el conjunto de nodos $V = \{1, 2, 3, \ldots, N\}$. El número total de nodos o vértices N se le denomina talla del grafo. A diferencia de los grafos en los que ambos sentidos son aceptables, un grafo orientado es un grafo en el que se asgina un único sentido entre dos nodos como aceptable. Algunos ejemplos son:*

- $\mathcal{G}_1 = (V_1, E_1)$ *donde:*

$$V_1 := \{1, 2, 3, 4\}, \quad E_1 := \{(1, 1), (1, 2), (3, 4), (4, 3), (3, 3), (4, 1)\}.$$

- $\mathcal{G}_2 = (V_2, E_2)$ *donde:*

$$V_2 := \{1, 2, 3, 4, 5, 6\}, \quad E_2 := \{(1, 3), (4, 1), (1, 5), (2, 1), (1, 6), (5, 2)\}.$$

Terminología Básica sobre Grafos. Usaremos los siguientes términos de uso común en la Teoría de Grafos. Sea $\mathcal{G} := (V, E)$ un grafo orientado:

- *Camino en un Grafo.* Es una sucesión finita de nodos (no todos ellos distintos) representados del modo siguiente:

$$a_1 \to a_2 \to a_3 \to \cdots \to a_n,$$

  con la propiedad de que para cada i, $(a_i, a_{i+1}) \in V$ son aristas del grafo. Un camino como el anterior se dice de longitud $n - 1$.

- *Ciclo en un Grafo.* Un ciclo en un grafo es un camino que empieza un termina en el mismo nodo. Un grafo orientado se llama acíclico si no contiene ningún ciclo.

$$a_1 \to a_2 \to \cdots \to a_n = a_1.$$

- *Profundidad de un Grafo.* Si $\mathcal{G}$ es un grafo orientado, se llama profundidad a la mayor de las longitudes de sus caminos. Obsérvese que un grafo tiene profundidad finita si y solamente si es acíclico. Se llaman grafos bien paralelizables a aquellos grafos en los que la profundidad es logarítimica (o poli-logarítimca) en la talla.

- *Abanico de entrada (fan–in) y Abanico de Salida (fan–out).* Se llama Abanico de entrada de un nodo al número de aristas que terminan en él. Se llama abanico de salida de un nodo al número de aristas que salen de él.
- *Nodos fuente, nodos output, nodos intermedios.* Se llaman nodos fuente (o nodos de input) a los nodos de fan–in 0, y nodos output a los nodos de fan–out 0. El resto se llaman nodos intermedios.

Definición 23 (Circuito Booleano) *Llamaremos circuito booleano a todo par $(\mathcal{G}, \varphi)$ donde $\mathcal{G} :_(V, E)$ es un grafo orientado y acíclico en el que el fan–in de todo nodo es a lo sumo 2 (fan–out no acotado) y φ es un proceso de etiquetado de los nodos que verifica las propiedades siguientes:*

$$\varphi : V \longrightarrow \{\vee, \wedge, \neg\} \cup \{0, 1\} \cup \{X_i \ : \ i = 1, 2, \ldots\}.$$

- *El grafo $\mathcal{G}$ tiene $n + 2$ nodos fuente que son numerados como $\{1, 2, \ldots, n, n + 1, n + 2\}$. Además,*

$$\varphi(i) := X_i, \text{ para cada } i,\ 1 \leq i \leq n,$$
$$\varphi(n + 1) = 0, \varphi(n + 2) = 1,$$

- *Para un nodo $\nu \in V$ de fan–in 1, $\varphi(\nu) = \neg$,*
- *Para un nodo $\nu \in V$ de fan–in 2, $\varphi(\nu) \in \{\vee, \wedge\}$.*

Nota 24 *Los circuitos booleanos pueden interpretarse como evluadores de aplicaciones booleanas.*
En los nodos fuente se introducen valores booleanos $\underline{\varepsilon} := (\varepsilon_1, \ldots, \varepsilon_n) \in \{0, 1\}^n$.
A través de los nodos intermedios vamos avanzando conforme a las reglas marcadas en los nodos (es decir, aplicando las evaluaciones de las conectivas $\{\vee, \wedge, \neg\}$).
Al final, en los nodos de output, detenemos los cálculos dando el resultado que es una lista en $\{0, 1\}^m$, donde m es el número de nodos de output del grafo.
Así, si un circuito Γ tiene un sólo nodo de output y $n+2$ nodos fuente, entonces, es una evaluador de una función booleana $f \in \mathcal{B}_n$, es decir, $f : \{0, 1\}^n \longrightarrow \{0, 1\}$.

Definición 25 *Se llama complejidad de una fórmula booleana $f \in \mathcal{B}_n$ al mínimo de las tallas de los circuitos booleanos que evalúan f.*

Teorema 26 (Teorema de Shannon–Lupanov) *Toda función booleana $f \in \mathcal{B}_n$ puede evaluarse con circuitos booleanos de talla $\frac{2^n}{n}$.*
Además, hay funciones booleanas de complejidad $\frac{2^n}{n}$, es decir, que no admiten circuitos booleanos que las evalúen con menos de $\frac{2^n}{n}$ nodos.

Codificación de circuitos booleanos. Comencemos con las codificaciones de grafos orientados. Hay dos modelos usuales: codificación como lista y codificación mediante matriz de adyacencia.

Codificación como lista. Es la codificación más próxima a la definición. Tiene dos componentes:

$$\mathcal{G} := (N, L),$$

donde N es el número de nodos y L es una lista de listas,

$$L := [[a_1, b_1], [a_2, b_2], \ldots, [a_L, b_L]],$$

siendo $[a_i, b_i]$ la lista de uno de los pares $(a_i, b_i) \in V$.

Codificación mediante matriz de Adyacencia. Dado un grafo orientado $\mathcal{G} = (V, E)$, donde $V = \{1, 2, \ldots, N\}$ es el conjunto de nodos. Codificaremos el grafo mediante una matriz $N \times N$

$$G := (a_{i,j})_{1 \leq i,j \leq N}$$

donde las coordenadas son dadas mediante:

$$a_{i,j} := \begin{cases} 1 & \text{si } (i,j) \in V \\ 0 & \text{en caso contrario} \end{cases}$$

Codificación de Circuitos Booleanos. Usaremos una codificación que sigue el esquema de la codificación como lista. Así, sea Γ un circuito booleno. Su codificación como lista de listas $\Gamma := [A_1, \ldots, A_L]$, donde L es el número de nodos, tendrá la forma siguiente:

- Si i es un nodo fuente de variable $A_i := [i, X_i]$,
- Para los nodos asociados a constantes,

$$A_{n+1} := [n+1, 0], A_{n+2} := [n+2, 1].$$

- Si i es un nodo de fan–in 1, entonces, existe un nodo $j < i$ tal que $(j, i) \in V$ y $\varphi(i) = \neg$. Escribiremos:

$$A_i := [i, \neg, j].$$

- Si i es un nodo de fan–in 2, hay dos nodos $i_1 < i, i_2 < i$ tales que $(i_1, i) \in V$ y $(i_2, i) \in V$. Si, además, $op = \varphi(i) \in \{\vee, wedge\}$, escribiremos:

$$A_i := [i, op, i_1, i_2].$$

Teorema 27 *Existe un algoritmo que realiza la tarea siguiente:*
Dado un circuito booleano Γ con un sólo nodo de output, por su codificación, involucrando n variables $\{X_1, \ldots, X_n\}$ y dado un valor booleano $\underline{\varepsilon} \in \{0,1\}^n$, el algoritmo develve el valor $f(\underline{\varepsilon})$, es decir, el valor de la función booleana evaluada por Γ en el valor $\underline{\varepsilon}$ elegido.
El tiempo requerido por el algoritmo es del orden $Llog_2L$, donde L es la talla del circuito.

2.3. Equivalencia Tautológica y Formas Normales

Antes de comenzar, introduzcamos el siguiente resultado técnico de gran utilidad para probar equivalencas tautológicas.

Teorema 28 *Sean F y G dos fórmulas tautológicamente equivalentes (i.e. $F \equiv G$). Entonces, para cualquier fórmula H que contiene a F como subfórmula, sea H' la fórmula obtenida reemplazando cada aparición de F en H por G. Sea tendrá que $H \equiv H'$.*

En otras palabaras, dos fórmulas tautológicamente equivalentes se pueden reemplazar unas por otras y no cambian el valor semántico de la fórmula obtenida.

Teorema 29 (Principales Equivalencias Tautológicas) *Para cuaesquiera tres fórmulas F, G, H las siguientes equivalencias tautológicas se verifican:*

- **Idempotencia**

$$(F \vee F) \equiv F, \quad (F \wedge F) \equiv F.$$

- **Elemento Neutro**
$$(F \vee 0) \equiv F, \ \ (F \wedge 1) \equiv F.$$
- **Doble negación**
$$(\neg(\neg F)) \equiv F.$$
- **Conmutatividad**
$$(F \vee G) \equiv (G \vee F), \ \ (F \wedge G) \equiv (G \wedge F).$$
- **Asociatividad**
$$(F \vee (G \vee H)) \equiv ((F \vee G) \vee H), \ \ (F \wedge (G \wedge H)) \equiv ((F \wedge G) \wedge H).$$
- **Distributivas**
$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H)), \ \ (F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H)).$$
- **Leyes de Morgan**
$$(\neg(F \vee G)) \equiv ((\neg F) \wedge (\neg G)), \quad (\neg(F \wedge G)) \equiv ((\neg F) \vee (\neg G)).$$
- **Leyes de Tautología.** *Si F es una tautología,*
$$F \equiv (F \vee G), \ \ G \equiv (F \wedge G).$$
- **Leyes de Insatisfactibilidad.** *Si F es una insatisfactible,*
$$G \equiv (F \vee G), \ \ F \equiv (F \wedge G).$$

Definición 30 (cláusulas) *Se llaman literales a las fórmulas atómicas y a las negaciones de fórmulas atómicas. Se llaman literales negativos a las negaciones de fórmulas atómicas (es decir, a los del conjunto $\{(\neg X_i), (\neg 1), (\neg 0)\}$) y literales positivos al resto.*
Se llaman calusulas, a toda disyunción de un número finito de literales. Esto es, son cláusulas las fórmulas del tipo
$$F = \left(\bigvee_{i=1}^{r} A_i \right) = (A_1 \vee A_2 \vee \cdots \vee A_r),$$
donde $A_1, A_2, \ldots, A_r$ son literales positivos o negativos.

Nota 31 (Varias Notaciones para las cláusulas) *Algunos autores usan una notación de lista para las cláusulas, dando por entendido el símbolo $\vee$ de la disyunción. Así, se tienen las siguientes notaciones para las cláusulas:*
$$F = (X_1 \vee (\neg X_2) \vee 0 \vee (\neg X_3) \vee X_5),$$
o también
$$F = \{X_1, (\neg X_2), 0, (\neg X_3), X_5\},$$
o incluso
$$F = [X_1, (\neg X_2), 0, (\neg X_3), X_5].$$
Usaremos indistintamente cualquiera de ellas, indicando en cada caso el tipo de notación que usamos.

Definición 32 (Formal Normal Conjuntiva CNF) *Decimos que una fórmula está en forma normal conjuntiva si es una conjnción de varias cláusulas, esto es, son las fórmulas que tienen la forma:*

$$F = \left(\bigwedge_{i=1}^{r} \left(\bigvee_{j=1}^{s_i} A_{i,j} \right) \right),$$

donde los $A_{i,j}$ son literales.

Nota 33 *A veces se usa el término forma normal disyuntiva (DNF) para designar las fórmulas que vienen dadas como disyunción de conjunciones de literales, esto es, las fórmulas que tienen la forma:*

$$F = \left(\bigvee_{i=1}^{r} \left(\bigwedge_{j=1}^{s_i} A_{i,j} \right) \right),$$

donde los $A_{i,j}$ son literales. Pero su uso está menos extendido y es menos relevante.

Teorema 34 *Toda fórmula es tautológicamente equivalente a una fórmula en forma normal conjuntiva.*
Toda fórmula es tautológicamente equivalente a una fórmula en forma normal disyuntiva.

Demostración.– El siguiente algoritmo calcula la fórmula en forma normal conjuntiva equivalente a una fórmula dada.

INPUT: Una fórmula F en forma cualquiera.
while F no esté en forma normal conjuntiva **do**

- Reemplaza toda aparición de subfórmulas de los tipos descritos en la columna de la derecha por la fórmula correspondiente descrita en la columna de la izquierda, hasta que no quede ninguno de los tipos que aparecen e la columna izquierda:

Reemplaza $(\neg(\neg G))$	por G
Reemplaza $(\neg(G \wedge H))$	por $((\neg G) \vee (\neg H))$
Reemplaza $(\neg(G \vee H))$	por $((\neg G) \wedge (\neg H))$

- Reemplaza toda aparición de subfórmulas de los tipos descritos en la columna de la derecha por la fórmula correspondiente descrita en la columna de la izquierda, hasta que no quede ninguno de los tipos que aparecen e la columna izquierda:

Reemplaza $(R \vee (G \wedge H))$	por $((R \vee G) \wedge (R \vee H)$
Reemplaza $((R \wedge G) \vee H)$	por $((R \vee G) \wedge (R \vee H))$

od

2.4. Cláusulas de Horn, Fórmulas de Horn

Definición 35 (Cláusulas de Horn) *Una cláusula se dice clásula de Horn si tiene a los sumo un literal positivo.*

Nota 36 *Una cláusula de Horn sin ningún literal positivo de llama clásula objetivo. Por ejemplo,*

$$((\neg X_1) \vee (\neg X_3) \vee (\neg X_5)),$$

es una clásula de Horn sin literales positivos. Nótese que la anterior cláusula es tautológicamente equivalente a

$$(\neg(X_1 \wedge X_3 \wedge X_5)).$$

Una cláusula de Horn con un único literal positivo se llama hecho (fact).

Nota 37 *Una clásula de Horn con exactamente un literal positivo es una implicación. Por ejemplo,*

$$((\neg X_1) \vee (\neg X_3) \vee (\neg X_5) \vee X_2),$$

es una cláusula de Horn tautológicamente equivalente a:

$$((X_1 \wedge X_3 \wedge X_5) \rightarrow X_2)\,.$$

Nota 38 *Una cláusula que no es cláusula de Horn es la siguiente:*

$$(X_1 \vee X_2 \vee (\neg X_3)).$$

Definición 39 *Una fórmula de Horn es una fórmula en forma normal conjuntiva (CNF) tal que cada una de las clásulas que aparecen en ella es una cláusula de Horn.*

Lema 40 *Sea F una fórmula de Horn que no contiene cláusulas de los siguientes dos tipos:*

$$((1 \wedge \cdots \wedge 1) \rightarrow B),$$

donde B es variable o

$$((1 \wedge \cdots \wedge 1) \rightarrow 0).$$

Entonces, F es satisfactible

Demostración.– Daremos la demostración[3]. En la fórmula F en forma normal conjuntiva, todas las clásulas son de la forma:

$$((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow B),$$

donde los A_i's y B son variables o constantes en $\{0, 1\}$.
Los casos excluidos son esencialmente aquellos casos en los que todas las A_i's son constantemente 1 y B es variable o B es 0. Supongamos que la fórmula F es expresable con las variables en $\{X_1, \ldots, X_n\}$ (es decir, las únicas variables que aparecen están en ese conjunto. Consideremos una interpretación tal que los n primeros términos de la sucesión son cero, es decir, consideremos $\underline{0} := (0, 0, \ldots, 0) \in \{0, 1\}^n$. Veamos que si F verifica las hipótesis, entonces es satisfactible porque $F(\underline{0}) = 1$.
Para ello, veamos que todas las clásulas C de F verifican $C(\underline{0}) = 1$.
Para ello, veamos qué forma tienen que tener las cláusulas C de F.

[3]Damos la demostración porque en ninguna de las referencias del curso aparece de modo completo.

- Si la cláusula C de F es de la forma:

$$((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow 0),$$

entonces, nuestra hipótesis indica que no todos los A_i's son constantemente igual a 1. Por tanto,para algún i, o bien existe una variable X_j tal que $A_i = X_j$ o bien $A_i = 0$. en ambos casos (reemplazando X_j por 0), tendremos que

$$eval((A_1 \wedge \cdots \wedge A_r), \underline{0}) = 0,$$

y $eval(C, \underline{0}) = eval((0 \rightarrow 0), \underline{0}) = 1$, con lo que es satisfactible.

- Si la cláusula C de F es de la forma:

$$((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow B),$$

con B variable, entonces existe j tal que $B = X_j$ y nuestra cláusula C tiene la forma:

$$C = ((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow X_j).$$

Por tanto, sustituyendo X_j por 0,

$$eval(C, \underline{0}) = eval(((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow 0), \underline{0}),$$

y se aplica el argumento anterior con lo que es satisfactible.

- Finalmente, si la fórmula tiene la forma:

$$((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow 1),$$

entonces es una tautología y, por tanto, es satisfactible para cualquier valor elegido. ■

Este sencillo resultado conduce al siguiente importante resultado.

Teorema 41 *Existe un algoritmo que en tiempo polinomial en el tamao de la fórmula decide si una fórmula de Horn es satisfactible.*

Demostración.– El algoritmo trata de eliminar la presencia de clásulas de las consideradas en el Lema anterior.
Así, dada una fórmula de Horn F, denotaremos por

$$M(F) := \sharp \text{ cláusulas en } F \text{ de la forma}$$

$$((1 \wedge \cdots \wedge 1) \rightarrow B),$$

con B variable o $B = 0$.

INPUT: Una fórmula de Horn F en forma normal conjuntiva de tal modo que todas sus cláusulas sean de la forma:

$$((A_1 \wedge A_2 \wedge \cdots \wedge A_r) \rightarrow B),$$

donde $A_1, \ldots, A_r, B$ son o bien variables o bien constantes en $\{0, 1\}$.

G:=F
$n := M(G)$
while $n \geq 1$ **do**
  if existe una cláusula en G de la forma:

$$((1 \wedge \cdots \wedge 1) \rightarrow 0),$$

    then OUTPUT *Insatisfactible*
  else hallar una cláusula en G de la forma

$$((1 \wedge \cdots \wedge 1) \rightarrow B),$$

  con B variable.
    then escribir una nueva fórmula G_1 obtenida mediante:
      * Suprimir la cláusula $((1 \wedge \cdots \wedge 1) \rightarrow B)$ de G
      * Reemplazar toda aparición de la variable B en G por 1.
      * $G := G_1$
      * $n := M(G)$
  fi
od
OUTPUT *Satisfactible*

En cada paso por el "while" el algoritmo o bien termina o bien suprime una de las cláusulas que aparecen en la fórmula. Así, el número de pasos por el "while" está acotado por el número de cláusulas de F y, por tanto, por la talla de F. ∎

2.5. Resolución

La alternativa a la evaluación como medio de testar satisfabilidad de fórmulas. La idea original parece deberse a J.A. Robinson.

Definición 42 *Sean C_1, C_2 dos cláusulas del cálculo proposicional, escritas en forma clausal. Diremos que C_1 y C_2 son cláusulas conflictivas (clashing clauses) si existe un literal ℓ tal que $\ell \in C_1$ y $\ell^c \in C_2$. Llamaremos resolvente de las cláusulas C_1 y C_2 a la cláusula dada mediante la siguiente igualdad:*

$$Res(C_1, C_s) := (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\ell^c\}).$$

Las clásulas C_1 y C_2 se llaman cláusulas parentales o antecedentes de $Res(C_1, C_2)$.

Nota 43 *De hecho, la Definición anterior es incompleta aunque es la más común en la Bibliografía usual de estas asignaturas. Nótese que dos cláusulas C_1, C_2 pueden estar en conflicto en más de un literal. Por ejemplo,*

$$C_1 := [X_1 \vee (\neg X_2)], \;\; [(\neg X_1) \vee X_2].$$

En este caso, el término resolución no precisa cuál de los literales X_1 o $(\neg X_2)$ es el que domina a la hora de hacer la resolución. La razón última es que tanto la terminología como la notación es incorrecta. Debe decirse que dos cláusulas C_1 y C_2 están en conflicto "con respecto al literal ℓ". Debe denotarse la "resolvente de C_1 y C_2 con respecto a ℓ mediante:

$$Res(C_1, C_2, \ell) := (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\ell^c\}) .$$

Así, el anterior ejemplo resulta tener la forma siguiente:

$$Res(C_1, C_2, X_1) = [(\neg X_2) \vee X_2],$$

mientras que

$$Res(C_1, C_2, (\neg X_2)) = [X_1 \vee (\neg X_2)].$$

Sin embargo, en evitación de conflictos supondremos que la notación $Res(C_1, C_2)$ se refiere a $Res(C_,C_2, \ell)$ para algún literal ℓ que presenta conflicto entre ambas. S'olo en caso de duda haremos un esfuerzo en destacarlo.

Lema 44 *Sean C_1, C_2 dos clásulas conflictivas con respecto al literal ℓ. Entonces, si $\mathcal{A}$ es un modelo que satisface $C_1 \wedge C_2$, $\mathcal{A}$ también satisface $Res(C_1, C_2)$.*

Demostración.– Sea $\mathcal{A}$ el modelo y supongamos que $\mathcal{A}(\ell) = 1$ (la prueba será igual en el otro caso). Entonces,

$$1 = \mathcal{A}(C_2) = \mathcal{A}\left((C_2 \setminus \{\ell^c\}) \vee \{\ell^c\}\right) = \mathcal{A}(C_2 \setminus \{\ell^c\}).$$

Por tanto,

$$\mathcal{A}(Res(C_1, C_2)) = \mathcal{A}(C_2 \setminus \{\ell^c\}) = 1,$$

y se sigue el resultado. ∎

Lema 45 *Sean $C_1, \ldots, C_r$ cláusulas que contienen un literal ℓ y no contienen el literal ℓ^c. Sean $C_{r+1}, \ldots, C_s$ clásulas que contienen el literal ℓ^c y no contienen el literal ℓ. Consideremos la fórmula F del Cálculo Proposicional dada como la conjunción de todas las resolventes de las cláusulas conflictivas anteriores. Esto es,*

$$F := \bigwedge \{Res(C_i, C_j) \;:\; 1 \leq i \leq r, r+1 \leq j \leq s\}.$$

Entonces, si $\mathcal{A}$ es un modelo que satisface F, entonces, existe una extensión $\widetilde{\mathcal{A}}$ de $\mathcal{A}$ que satisface la fórmula dada por las conjunciones

$$C_1 \wedge \cdots \wedge C_r \wedge C_{r+1} \wedge \cdots \wedge C_s.$$

En particular, si F es satifacible, también lo es la anterior conjunción de cláusulas.

Demostración.– Sea $\mathcal{A}$ un modelo satisfecho por F. Tenemos que

$$\mathcal{A}(Res(C_i, C_j)) = 1, \;\; 1 \leq i \leq r, r+1 \leq j \leq s.$$

Obsérvese que F no contiene al literal ℓ ni a su negación. Supongamos que existe algún j, $r+1 \leq j \leq s$ tal que $\mathcal{A}(C_j \setminus \{\ell^c\}) = 0$.
Entonces, para todo i, $1 \leq i \leq r$, tendremos

$$\mathcal{A}(Res(C_i, C_j)) = \mathcal{A}(C_i \setminus \{\ell\}) = 1.$$

Pogamos $\widetilde{\mathcal{A}}$ un modelo definido del modo siguiente: $\widetilde{\mathcal{A}}(\tau) = \mathcal{A}(\tau)$ para todo literal τ en F (que es distinto de ℓ y ℓ^c. Definamos $\widetilde{\mathcal{A}}(\ell^c) = 1$ y, por consiguiente, $\widetilde{\mathcal{A}}(\ell) = 0$. Tendremos que para cada i, $1 \leq i \leq r$ se tendrá:

$$\widetilde{\mathcal{A}}(C_i) = \widetilde{\mathcal{A}}(C_i \setminus \{\ell\}) = \mathcal{A}(C_i \setminus \{\ell\}) = 1.$$

De otro lado, para cada j, $r+1 \leq j \leq s$, tendremos

$$\widetilde{\mathcal{A}}(C_j) = \widetilde{\mathcal{A}}\left((C_j \setminus \{\ell^c\}) \cup \{\ell^c\}\right) = \widetilde{\mathcal{A}}(\{\ell^c\}) = 1,$$

y la conjunción de todas ellas satisface $\widetilde{\mathcal{A}}$ y es, por tanto, satisfactible.
Supongamos que, por el contrario, para cada j, $r+1 \leq j \leq s$, se tiene $\mathcal{A}(C_j \setminus \{\ell^c\}) = 1$. entonces, basta definir la extensión $\widetilde{\mathcal{A}}$ de $\mathcal{A}$ con los valores: $\widetilde{\mathcal{A}}(\ell) = 1$ y $\widetilde{\mathcal{A}}(\ell^c) = 0$. Esta extensión será satisfecha por la cojuntción de todas las clásulas. ∎

Lema 46 *Sea F una fórmula del Cálculo Proposicional en forma normal conjuntiva y denotada en forma clausal. Sea ℓ un literal que aparece en F. Supongamos que las cláusulas de F se pueden clasificar en los tres grupos siguientes:*

- *$C_1, \ldots, C_r$ son las cláusulas que contienen ℓ y no contienen ℓ^c.*
- *$C_{r+1}, \ldots, C_s$ son las cláusulas que contienen ℓ^c y no contienen ℓ.*
- *$C_{s+1}, \ldots, C_t$ son las cláusulas que contienen a ℓ y a ℓ^c.*
- *$C_{t+1}, \ldots, C_m$ son las clásulas que no contienen ni a ℓ ni a ℓ^c.*

Sea F_1 la fórmula en forma normal conjuntiva dada mediante:

$$\left(\bigwedge_{1 \leq i \leq r, r+1 \leq j \leq s} Res(C_i, C_j) \right) \wedge (C_{t+1} \wedge \cdots \wedge C_m).$$

Entonces, F es satisfactible si y solamente si F_1 es satisfactible.

Demostración.– Es análoga a la prueba del Lema 44 anterior.

∎

Definición 47 *Sea F una fórmula del Cálculo Proposicional en forma normal conjuntiva, escrita en forma clausal. Sea $\kappa(F)$ el conjunto de todos los pares conflictivos de F y definamos la resolvente de F mediante:*

$$Res(F) := F \cup \{Res(C_1, C_2) \;:\; (C_1, C_2) \in \kappa(F)\}.$$

Proposición 48 *Para toda fórmula en forma normal conjuntiva F, F y $Res(F)$ son tautológicamente equivalentes.*

Demostración.– Basta con usar el Lema 46. Si un modelo $\mathcal{A}$ satisface F, entonces también satisface $Res(F)$. El recíproco es obvio. ∎

Teorema 49 *Sea F una fórmula del Cálculo Proposicional en forma normal conjuntiva. Supongamos que $F = Res(F)$ y que la cláusula vacía $\square$ no está en F. Entonces, F es satisfactible. El recíproco también es cierto.*

Demostración.– Por inducción en el número n de literales que aparecen en la fórmula.
Caso $n = 1$. Supongamos que la fórmula F sólo contiene un literal ℓ y/o su negación ℓ^c. Entonces, las clásulas de F sólo pueden ser de los tipos siguientes:

$$\{\ell\}, \{\ell^c\}, \{\ell, \ell^c\}.$$

o la cláusula vacía $\square$. Si, además, la fórmula coincide con su resolvente, entonces no puede ser nada más que una de ellas y, por tanto, satisfactible o tautológica.
Caso $n > 1$. Consideremos F involucrando n literales y supongamos que involucra el literal ℓ. Distinguiremos los siguientes tipos de cláusulas en F:

- $C_1, \ldots, C_r$ son las cláusulas que contienen ℓ y no contienen ℓ^c.
- $C_{r+1}, \ldots, C_s$ son las cláusulas que contienen ℓ^c y no contienen ℓ.
- $C_{s+1}, \ldots, C_t$ son las cláusulas que contienen a ℓ y a ℓ^c.
- $C_{t+1}, \ldots, C_m$ son las clásulas que no contienen ni a ℓ ni a ℓ^c.

Consideremos las clásulas $Res(C_i, C_j)$ con $1 \leq i \leq r$, $r+1 \leq j \leq s$. Como $F = Res(F)$,

$$Res(C_i, C_j) \in \{C_{t+1}, \ldots, C_m\}.$$

Consideremos la fórmula F_1 dada mediante

$$\left(\bigwedge_{1 \leq i \leq r, r+1 \leq j \leq s} Res(C_i, C_j) \right) \wedge (C_{t+1} \wedge \cdots \wedge C_m).$$

Aplicando Lema 46, concluimos que si F_1 es satisfactible, también lo es F.
Además, $F_1 = Res(F_1)$. Para verlo, sean C_1, C_2 dos clásulas conflictivas de F_1. Por tanto son cláusulas conflictivas de F y no contienen ni al literal ℓ ni su negación ℓ^c. entonces, $Res(C_1, C_2)$ está en F y no contiene ni ℓ ni ℓ^c. Por tanto, $Res(C_1, C_2) \in \{C_{t+1}, \ldots, C_m\}$ y, por tanto, es una cláusula de F_1. Pero, además, F_1 contiene a lo sumo $n-1$ literales y no contiene la cláusula vacía. Por tanto, F_1 es satisfactible y el Teorema se sigue. ∎

Definición 50 *Sea F una fórmula del cálculo proposicional en forma normal conjuntiva. Definamos la sucesión definida por las iteraciones del operador Res sobre F del modo siguiente:*

$$Res^0(F) := F,$$

$$Res^1(F) := Res(F),$$

$$Res^{k+1}(F) := Res\left(Res^k(F)\right).$$

Finalmente, definamos

$$Res^*(F) := \bigcup_{k\in\mathbb{N}} Res^k(F).$$

Lema 51 *Con las anteriores notaciones, Existe $m \in \mathbb{N}$ tal que*

$$Res^*(F) = Res^m(F) = Res^k, \forall k \in \mathbb{N}.$$

Es decir, la sucesión de resolventes se estabiliza.

Demostración.– Hay dos maneras de hacer la prueba, dependiendo de cómo vemos las fórmulas. Supongamos que F es una fórmula en cuyas cláusulas aparecen solamente literales en el siguiente conjunto:

$$L(F) := \{1, 0, X_1, \ldots, X_n, (\neg X_1), \ldots, (\neg X_n)\}.$$

Si F es vista en forma clausal, las cláusulas de F son simplemente subconjuntos del conjunto anterior. La aplicación de la Resolvente genera nuevas cláusulas que no involucran nuevos literales. Luego $Res(F)$ genera nuevos subconjuntos del conjunto $L(F)$. Como el conjunto $L(F)$ es finito (de cardinal $2n+1$) el número de subconjuntos de $L(F)$ es finito (acotado por $2^{2(n+1)}$). Por tanto, la sucesión creciente de subconjuntos de $L(F)$:

$$Res^0(F) \subseteq Res^1(F) \subseteq Res^2(F) \subseteq \cdots \subseteq Res^k(F) \subseteq \cdots,$$

no puede ser una sucesión infinita y, en algún momento, se estabiliza. ∎

Teorema 52 *Dada una fórmula F del cálculo proposicional en forma normal conjuntiva F, se tiene que F es satisfactible si y solamente si $\square \notin Res^*(F)$.*

Demostración.– Claramente, como F y $Res(F)$ son tautológicamente equivalentes, F es satisfactible si y solamente si lo es $Res^*(F)$. Finalmente, como

$$Res(Res^*(F)) = Res(Res^m(F)) = Res^{m+1}(F) = Res^*(F),$$

para algún m, concluiremos que $Res^*(F)$ verifica las hipótesis de Lema 46 y concluiremos que $Res^*(F)$ es satisfactible si y solamente si no contiene la cláusula vacía $\square$. ∎

Esto nos permite generar el siguiente Algoritmo:

INPUT: F (*Una fórmula en CNF*)
$S := F$
while $S \neq Res(S)$ **do**
  if $\square \in Res(F)$ **then** OUTPUT: Insatisfactible
  else do $S := Res(S)$
  fi
od
OUTPUT: Satisfactible
end

Teorema 53 *El anterior Algoritmo decide si una fórmula del Cálculo Proposicional dada en forma normal conjuntiva es satisfactible. EL número de iteraciones está acotado por* 2^M*, donde* M *es el número de literales que aparecen el el input.*

2.6. Eliminación

En realidad es una variante de la resolución que trataremos de definir a partir del uso de la resolvente de una fórmula con respecto a una variable. A través de los ejercicios, el alumno observará que la aplicación del anterior algoritmo de resolución (como operador iterado) puede hacer crecer muy rápidamente el número de cláusulas que intervienen a cada iteración. Una manera de corregir este efecto es el uso de la Eliminación Iterada que no es otra cosa que la resolución eliminando, a cada etapa, un literal.
Para ello, definiremos la noción siguiente:

Definición 54 *Se llama fórmula vacía a la fórmula del Cálculo Proposicional en forma normal conjuntiva que no contiene ninguna cláusula. Se denota por* $\emptyset$*.*

Nota 55 *No confudir la cláusula vacía* $\square$*, la fórmula* $\{\square\}$ *cuya única cláusula es la clásula vacía y la fórmula vacía* $\emptyset$*.*

- *La cláusula vacía* $\square$ *es insatisfactible.*
- *La fórmula* $\{\square\}$ *cuya única cláusula es la clásula vacía es insatisfactible.*
- *La fórmula vacía* $\emptyset$ *es tautológica.*

Definición 56 *Sea* $F = \{C_1, \ldots, C_m\}$ *una fórmula del cálculo Proposicional en CNF. Sea* ℓ *un literal que aparece en* F*. Definiremos* $Res(F, \ell)$ *en los términos y casos siguientes:*

- *Caso I.– Supongamos que se verifica:*
  - $C_1, \ldots, C_r$ *son las cláusulas que contienen* ℓ *y no contienen* ℓ^c*.*
  - $C_{r+1}, \ldots, C_s$ *son las cláusulas que contienen* ℓ^c *y no contienen* ℓ*.*
  - $C_{s+1}, \ldots, C_t$ *son las cláusulas que contienen a* ℓ *y a* ℓ^c*.*
  - $C_{t+1}, \ldots, C_m$ *son las clásulas que no contienen ni a* ℓ *ni a* ℓ^c*.*

  Entonces definimos:

$$Res(F, \ell) := \left(\bigwedge_{1 \leq i \leq r, r+1 \leq j \leq s} Res(C_i, C_j) \right) \wedge (C_{t+1} \wedge \cdots \wedge C_m) .$$

- *Caso II.– Supongamos que se verifica:*
  - $C_1, \ldots, C_r$ *son las cláusulas que contienen* ℓ *y no contienen* ℓ^c*.*
  - *No hay cláusulas que contienen* ℓ^c *y no contienen* ℓ*.*
  - $C_{r+1}, \ldots, C_t$ *son las cláusulas que contienen a* ℓ *y a* ℓ^c*.*
  - $C_{t+1}, \ldots, C_m$ *son las clásulas que no contienen ni a* ℓ *ni a* ℓ^c*.*

Entonces definimos:

$$Res(F, \ell) := C_1 \wedge C_2 \wedge \cdots \wedge C_r \wedge (C_{t+1} \wedge \cdots \wedge C_m)\,.$$

- *Caso III.– Supongamos que se verifica:*
  - *No hay cláusulas que contienen ℓ y no contienen ℓ^c.*
  - *No hay cláusulas que contienen ℓ^c y no contienen ℓ.*
  - *$C_{r+1}, \ldots, C_t$ son las cláusulas que contienen a ℓ y a ℓ^c.*
  - *$C_{t+1}, \ldots, C_m$ son las clásulas que no contienen ni a ℓ ni a ℓ^c.*

  Entonces definimos:

$$Res(F, \ell) := (C_{t+1} \wedge \cdots \wedge C_m)\,.$$

Proposición 57 *Con las notaciones de la Definición anterior, se tiene, para una fórmula F en forma normal conjuntiva:*

1. *Si $F \neq \emptyset$ y $F \neq \{\,\square\,\}$, entonces, hay al menos un literal que aparece en F.*
2. *Para cualquier literal ℓ que aparece en F, F es satisfactible si y solamente si $Res(F, \ell)$ es satisfactible.*

Con estos instrumentoes deinimos el sguiente algoritmo, que llamaremos algoritmo de eliminación:

INPUT: F (*Una fórmula en CNF*)
while $F \neq \emptyset$ **do**
  if $\square \in F$ **then** OUTPUT: Insatisfactible
  else do $F := Res(F, \ell)$, donde ℓ es algún literal que aparece en F.
  fi
od
OUTPUT: Satisfactible
end

Teorema 58 *El algoritmo de eliminación decide si una fórmula del Cálculo Proposicional es satisfactible o no en tiempo exponencial en el número de literales involucrados.*

2.7. Ejercicios

Problema 2 *Reconstruir la demostración del Lema 46, interpretando la extensión del modelo que se satisface en ambos casos.*

Capítulo 3

Lógica de Predicados

3.1. Introducción

Los ejemplos siguientes muestran la aparición de fórmulas de primer orden de la Lógica de Predicados en diversos ámbitos conocidos por los alumnos.

3.1.1. El Registro Civil.

Se trata de la base de datos más antigua que existe. En ella constan, de cada individuo (en España), la siguiente información:

- Nombre y Apellidos (dos)
- Nombre la madre (sin apellidos)
- Nombre del padre (sin apellidos)
- Fecha y Lugar de Nacimiento.

Cada individuo que nace en el Estado español debe inscribirse en el registro Civil con esa información.
Esta Base de datos clásica (e histórica) puede usarse para responder a preguntas como:

1. Dados X e Y, decidir sin son primos.
2. Dados X e Y, decidir si son hermanos. O hermanastros.
3. Dados X e Y, decidir si poseen un antecesor común hasta la cuarta generación.

El número de preguntas y sus utilizaciones puden ser muy variadas. ¿Cómo interpretar esas preguntas que se hacen a la base de datos?.

Relación "Ser Hermanos". Usaremos nombres con un sólo apellido por simplicidad.
Así, tenemos las relaciones:

$$madre(X,Y) := \begin{cases} 1 & \text{Si X es la madre de Y} \\ 0 & \text{en caso contrario} \end{cases}$$

$$padre(X,Y) := \begin{cases} 1 & \text{Si X es el padre de Y} \\ 0 & \text{en caso contrario} \end{cases}$$

Podemos definir la relación "Ser hermanos" mediante:

$$Hermano(X,Y) := \exists Z \exists T \left([madre(Z,X) \wedge madre(Z,Y)] \wedge [padre(T,X) \wedge padre(T,Y)] \right).$$

Relación "Ser Primos" Añadimos la relación:

$$Primo(X,Y) := \exists Z \exists T \, [Hermano(Z,T)] \wedge [padre(Z,X) \vee madre(Z,X)] \wedge [padre(T,Y) \vee madre(T,Y)] \, .$$

Relación "Ser Parientes". Entendiendo que parientes son los que tienen alg1'un antepasado en común, la dificultad de definir la fórmula estriba en la "distancia" entre el "pariente común" y los individuos involucrados. Por eso se definen versiones de primer orden como "Grados de Consanguineidad" (esto es, se acota la distancia entre los individuos y el antecesor común).

3.1.2. Ejemplos del Cálculo (Análisis Matemático Elemental)

Las siguientes propiedades de una función $f : \mathbb{R} \longrightarrow \mathbb{R}$ son definibles mediante fórmulas de primer orden:

- f es contínua en todo $\mathbb{R}$:

$$\forall x \forall \varepsilon \exists \delta [\varepsilon > 0] \wedge [\delta > 0] \wedge [\forall y | x - y| < \delta \rightarrow |f(x) - f(y)| < \varepsilon].$$

- f es diferenciable con derivada contínua (i.e. $f \in \mathcal{C}^1(\mathbb{R})$) también es primer orden.
- $f \in \mathcal{C}^k(\mathbb{R})$ (con $k \in \mathbb{N}$, finito) es también primer orden.
- En cambio, $f \in \mathcal{C}^\infty(\mathbb{R})$ pierde esa cualidad de primer orden.

Del mismo modo, las propiedades de sucesiones y series, salen de primer orden puesto que combinan variables cuantificadas en $\mathbb{N}$ y en $\mathbb{R}$:

- $f : \mathbb{N} \rightarrow \mathbb{R}$, $f(n) := a_n$, una sucesión de números reales. Existencia de límite:

$$\exists x \in \mathbb{R}, \forall \varepsilon, \exists n_\varepsilon \in \mathbb{N}, [\varepsilon > 0] \wedge [\forall n \geq n_\varepsilon, |a_n - x| < \varepsilon,$$

- Del mismo modo, existencia suma de la serie...

3.1.3. Teoría Elemental de Números (ENT).

Es el clásico del Teorema de Gödel. Bastará con que los alumnos escirban las fórmulas relativas a "primo", "divisible", "ser una terna Pitagórica", existencia de infinidad de primos o existencia de infinidad de terna Pitagóricas.

3.1.4. Geometría Elemental "a la Tarski".

Simplemente recordar que:

- Las rectas planas son el subconjunto de $\mathbb{R}^3$ dado por:

$$\{(a,b,c) \in \mathbb{R}^3 \ : \ a^2 + b^2 \neq 0\}.$$

- Las circunferencias son
$$\{(a,b,c) \in \mathbb{R}^3 \ : \ r \neq 0\}.$$

- Los triángulos son los elementos de $\mathbb{R}^6$ dados por
$$\{(a,b,c,d,e,f) \ : \text{ tales que } (a,b),\ (c,d) \text{ y } (e,f) \text{ no están alineados}\}.$$

Escríbanse las fórmulas de:

- Una recta corta a una circunferencia en a lo sumo 2 puntos.

- Las tres alturas de un triángulo se cortan en un punto.

3.1.5. Relaciones, funciones.

Recordar las definiciones de ambas nociones en Teoría Intuitiva de Conjuntos.

3.2. El lenguaje : sintaxis

3.2.1. El alfabeto

Usaremos los ímbolos siguientes:

1. **Variables.** Se trata de un conjunto numerable de variables como en el Cálculo Proposicional $\{X_1, \ldots, X_n, \ldots\}$. Sin embargo, por comodidad de la escritura, usaremos en mucos casos las últimas variables del alfabeto (y en minúsculas) para indicar variables, esto es, x, y, z. El lector deberá en cada caso no indicado discernir cuales son las variables involucradas en cada fórmula.

2. **Símbolos de Relación (o relacionales)**. Se trata de una cantidad numerable de símbolos de relación $R_i^{k_i}$, $i \in \mathbb{N}$, de aridad k_i, esto es, afectando a k_i "objetos a definir". Por simplicidad usaremos letras mayúsculas entre P, Q, R, S, T evitando los sub–índices siempre que esto no induzca confusión.

3. **Símbolos de función (o funcionales)**. De nuevo, funciones en cantidad a lo sumo numerable $f_i^{k_i}$, $i \in \mathbb{N}$, de "aridad" k_i lo que quiere decir que afecta a k_i "objetos a definir". USaremos, por comodidad, los símbolos f, g, h para denotar funcionales, siempre que la evitación de los subíndices no induzac confusión.

4. **Conectivas.** Los símbolos de las conectivas del Cálculo Proposicional $\{\vee, \wedge, \neg, \rightarrow\}$.

5. **Constantes.** Se trata de una conjunto nuemrable de constantes $\{a_1, \ldots, a_n, \ldots\}$. De nuevo por comodidad de la escritura, tenderemos a usar las primeras variables del alfabeto ltino (en minúsculas) para denotar constantes, esto es, $a, b, c, ...$

6. **Cuantificadores.** Los cuantificadores *existencial* $\exists$ y *universal* $\forall$.

3.2.2. Fórmulas bien formadas.

Términos.

Son las palabras sobre el anterior alfabeto, definidas mediante la siguiente regla recursiva:

- Las variables y las constantes son términos.
- Si f es un fucional de aridad k y $t_1, \ldots, t_k$ son términos, entonces $f(t_1, \ldots, t_k)$ es también un término.

Obsérvese que los términos no han de ser fórmulas dado que no hay "predicado".

Fórmulas Atómicas.

Son fórmulas atómicas todas aquellas expresiones sobre el anterior alfabeto que tienen la forma siguiente:

$$R_i^{k_i}(t_1, \ldots, t_{k_i}),$$

donde:

- $R_i^{k_i}$ es un símbolo de relación de aridad k_i,
- $t_1, \ldots, t_{k_i}$ son términos.

Fórmulas.

Es la clase de palabras sobre el alfabeto anterior, cerrada ante las siguientes propiedades:

- Las fórmulas atómicas son fórmulas.
- Si F es una fórmula, entonces también lo es $\neg F$.
- Si F y G son fórmulas, entonces también lo son $(F \vee G)$ y $(F \wedge G)$.
- Si F es una fórmula, entonces también lo son

$$\exists x F,$$

$$\forall x F.$$

3.2.3. Algunos conceptos importantes.

Subfórmula. Una subfórmula de una fórmula de la Lógica de Predicados es un "trozo" de la fórmula que también es fórmula.

Matriz de una fórmula. Dada una fórmula F, denotaremos por F^* (y llamaremos matriz de F) a la fórmula obtenida suprimiendo todos los cuantificadores en F.

Variables libres y ligadas. Una variable en una fórmula F se dice ligada si aparece afectada en alguna subfórmula de F por un cuantificador. Una variable se dice que aparece de forma libre en F si aparece en alguna subfórmula de F no afectada por un cuantificador. *Nótese que una variable puede aparecer en forma libre y ligada simultáneamente dentro de una fórmula.*

Fórmulas Rectificadas. Una fórmula se dice rectificada si ninguna variable posee apariciones libres y ligadas dentro de F.

Ejemplo 2 *En la siguiente fórmula, la variable x tiene apariciones libres y ligadas:*

$$((\forall x\,(P(x,z) \to (\exists y Q(z,y)))) \vee (R(x,z)))\,.$$

En la subfórmula $(\forall x\,(P(x,z) \to (\exists y Q(z,y))))$ tiene una apración ligada, mientras que en la subfórmula $(R(x,z))$ tiene una aparición libre (es decir, el cuantificador no la afecta). Una fórmula se dice rectificada si no hay casos como éste.

Fórmula cerrada. *Es una fórmula sin apariciones libres de ninguna variable.*

Fórmula libre de cuantificadores. *Es una fórmula en la que toda aparición de variables es libre (o sea, no hay cuantificadores).*

3.3. Semántica

Definición 59 (Estructura) *Una estructura es un par $\mathcal{A} := (U_\mathcal{A}, I_\mathcal{A})$ donde:*

1. *$U_\mathcal{A}$ es un conjunto que se denomina Universo de la estructura,*
2. *$I_\mathcal{A}$ es una transformación (interpretación) que realiza la siguiente tarea:*
   - *A cada variable x_i le asigna una valor $x_i^\mathcal{A} \in U_\mathcal{A}$ en el universo.*
   - *A cada símbolo de relación R_i le asigna una relación (de la misma aridad) sobre el universo $U_\mathcal{A}$:*
   $$R_i^\mathcal{A} \subseteq U_\mathcal{A}^{k_i}.$$
   - *A cada símbolo de función f le asigna una aplicación de la misma aridad sobre el universo $U_\mathcal{A}$.*
   $$f^\mathcal{A} : U_\mathcal{A}^{k_i} \longrightarrow U_\mathcal{A},$$
   de la misma aridad que f.
   - *A cada símbolo de constante a le asigna un valor $a^\mathcal{A} \in U_\mathcal{A}$ en el universo.*

Definición 60 *Una estructura se dice adecuada para una fórmula si todos los símbolos de la fórmula tienen interpretación.*

Ejemplo 3 *Tomemos la siguiente fórmula:*

$$F := \forall x P(x, f(x)) \wedge Q(g(a,z)),$$

donde:

- *P es un símbolo de relación de aridad 2.*
- *f es un símbolo de función de aridad 1.*
- *g es un símbolo de función de aridad 2.*
- *Q es un símbolo de relación de aridad 1.*

- *a es un símbolo de constante (o función de aridad 0).*

Una estructura adeucada para F sería, por ejemplo, la siguiente:

$$\mathcal{A} := (U_{\mathcal{A}}, I_{\mathcal{A}}),$$

donde

- $U_{\mathcal{A}} = \{0, 1, 2, \ldots\} = \mathbb{N}$.
- *La asignación $I_{\mathcal{A}}$ se comporta del modo siguiente:*
  - $I_{\mathcal{A}}(P) := P^{\mathcal{A}} := \{(m, n) \in \mathbb{N} \ : \ m < n\} \subseteq \mathbb{N}^2$.
  - $I_{\mathcal{A}}(Q) := Q^{\mathcal{A}} := \{n \in \mathbb{N} \ : \ n \text{ es primo}\} \subseteq \mathbb{N}^1$.
  - $I_{\mathcal{A}}(a) := a^{\mathcal{A}} = 2$.
  - $I_{\mathcal{A}}(f) = f^{\mathcal{A}} : \mathbb{N} \longrightarrow \mathbb{N}$ *es la función sucesor, esto es,*

$$f(n) := n + 1, \text{ para todo } n \in \mathbb{N}.$$

  - $I_{\mathcal{A}}(g) = g^{\mathcal{A}} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ *es la función de aridad 2 dada coko la suma, esto es,*

$$f(n, m) := n + m, \text{ para todo } (n, m) \in \mathbb{N}^2.$$

  - $I_{\mathcal{A}}(z) = z^{\mathcal{A}} = 3$.

Nótese que no usamos las asignaciones de variables ligadas. Así, la fórmula se lee:

- *Todo número natural es menor que su sucesor y*
- *La suma de 2 y 3 es primo.*

Nótese que si cambiamos la asignación $z^{\mathcal{A}} = 4$, la "interpretación" de la fórmula deja de ser cierta.

Definición 61 (Semántica de la Lógica de Predicados) *Sea F una f'ormula y sea $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ una estructura adecuada para F. Definiremos $\mathcal{A}(F)$ mediante el siguiente proceso recursivo:*

- *Si t es un término definimos $\mathcal{A}(t)$ mediante:*
  - *Si $t = x$ es una variable $\mathcal{A}(t) := x^{\mathcal{A}}$,*
  - *si $t := f(t_1, \ldots, t_k)$ donde f es un funcional y $t_1, ldots, t_k$ son términos, definimos*

$$\mathcal{A}(F) := f^{\mathcal{A}}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_K)).$$

- *Si $F := R(t_1, \ldots, t_k)$ es una fórmula atómica, donde R es una símbolo de relación de aridad k y $t_1, \ldots, t_k$ son términos,*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in R^{\mathcal{A}} \\ 0, & \text{en otro caso} \end{cases}$$

- *Si* $F := \neg G$ *es una fórmula*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si } \mathcal{A}(G) = 0 \\ 0, & \text{en otro caso} \end{cases}$$

- *Si* $F := (G \wedge H)$ *es una fórmula*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si } \mathcal{A}(G) = 1 \text{ y } \mathcal{A}(H) = 1 \\ 0, & \text{en otro caso} \end{cases}$$

- *Si* $F := (G \vee H)$ *es una fórmula*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si } \mathcal{A}(G) = 1 \text{ o } \mathcal{A}(H) = 1 \\ 0, & \text{en otro caso} \end{cases}$$

- *Si* $F = \forall x G$*, definiremos la siguiente estructura a partir de* $\mathcal{A}$*:*

$$\mathcal{A}_{[x/u]},$$

Tiene el mismo universo, las mismas identificaciones de relacionales, funcionales y constantes, excepto que la variable x *se asigna a* $u \in U_{\mathcal{A}}$*. Entonces,*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si para todo } u \in U_{\mathcal{A}},\ \mathcal{A}_{[x/u]}(G) = 1, \\ 0, & \text{en otro caso} \end{cases}$$

- *Si* $F = \exists x G$*, consideramos la estructura* $\mathcal{A}_{[x/u]}$ *definida a partir de* $\mathcal{A}$ *como antes. Entonces,*

$$\mathcal{A}(F) := \begin{cases} 1, & \text{si existe algún } u \in U_{\mathcal{A}},\ \mathcal{A}_{[x/u]}(G) = 1, \\ 0, & \text{en otro caso} \end{cases}$$

Definición 62 (Satisfactibilidad, Tautología) *Sea* F *una fórmula de la Lógica de predicados.*

1. *Si* $\mathcal{A}$ *es una estructura adecuada para* F *tal que* $\mathcal{A}(F) = 1$*, diremos que* $\mathcal{A}$ *es un modelo para* F *y lo denotaremos por* $\mathcal{A} \models F$*.*
2. *Decimos que* F *es satisfactible si existe algún modelo de* F*.*
3. *Decimos que* F *es válida (o tautológica) si para toda estructura* $\mathcal{A}$ *adecuada para* F*, se tiene* $\mathcal{A} \models F$*.*

Definición 63 *Dos fórmulas del Cálculo de Predicados se dicen tautológicamente equivalentes (y se denote mediante* $F \equiv G$*) si para toda estructura* $\mathcal{A}$ *adaptable a* F *y a* G *se tiene* $\mathcal{A}(F) = \mathcal{A}(G)$*.*

Nota 64 *Obsérvse que una fórmula* F *es válida si y solamente si* $\neg F$ *es insatifsactible.*

3.4. Formas Normales

En esta Sección procedremos a desarrollar una serie de reducciones de las fórmula del Cálsulo de Predicados.

3.4.1. Reducción a RPF

Definición 65 *Una fórmula del Cálculo de Predicados se dice en forma prenexa si tiene la forma:*

$$F = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n G,$$

donde G es una fórmula del Cálculo de Predicados libre de cuantificadores.

Definición 66 *Una fórmula del Cálculo de Predicados se dice en forma rectificada si no existe ninguna variable que sea a la vez cuantificada y libre en F.*

Definición 67 *Una fórmula del Cálculo de Predicados se dice en forma* **RPF** *(rectified prenex form) si es a la vez rectificada y prenexa.*

Unas pocas equivalencias tautológicas preliminares (que no demostraremos):

Teorema 68 *Sean F y G dos fórmulas. Se tienen los siguientes grupos de equivalencias tautológicas:*

1. $\begin{cases} (\neg(\forall x F)) \equiv (\exists x(\neg F)) \\ (\neg(\exists x F)) \equiv (\forall x(\neg F)) \end{cases}$
2. $\begin{cases} ((\forall x F) \wedge (\forall x G)) \equiv (\forall x(F \wedge G)) \\ ((\exists x F) \vee (\exists x G)) \equiv (\exists x(F \vee G)) \end{cases}$
3. $\begin{cases} \forall x \forall y F \equiv \forall y \forall x F \\ \exists x \exists y F \equiv \exists y \exists x F \end{cases}$
4. *Si x no tiene apariciones libres en G*[1]*, entonces:*

$$\begin{array}{rcl} (\forall x F \wedge G) & \equiv & \forall x(F \wedge G) \\ (\forall x F \vee G) & \equiv & \forall x(F \vee G) \\ (\exists x F \wedge G) & \equiv & \exists x(F \wedge G) \\ (\exists x F \vee G) & \equiv & \exists x(F \vee G) \end{array}$$

Nota 69 *Salvo casos excepcionales, se tiene la no equivalencia tautológica siguiente:*

$$((\forall x F) \vee (\forall x G)) \not\equiv (\forall x(F \vee G))$$

$$((\exists x F) \wedge (\exists x G)) \not\equiv (\exists x(F \wedge G)).$$

Es decir, las equivalencias del item 2 exigen que los cuantificadores estén ligados al tipo de conectiva del Cálculo Proposicional usada.
También, salvo casos excepcionales, los cuantificadores $\exists$ y $\forall$ no conmutan manteniendo equivalencia tautológica. Dejamos para las clases los ejemplos que explican estos fenómenos.

Definición 70 (Sustitución) *Sea F una fórmula de la Lógica de Predicados, x una variable y t un término. Denotaremos por*

$$F[x/t]$$

a la fórmula obtenida reemplazando cada aparición libre de x en F por el término t. Llamaremos sustitución a la operación $[x/t]$

[1] Sin esta condición serían falsas las equivalencias

Lema 71 (Renombrar variables ligadas) *Sea $F = QxG$, una fórmula de la Lógica de Predicados, donde $Q \in \{\forall, \exists\}$ es una cuantificador. Sea y una variable nueva que no aparece como variable libre en G. Entonces,*

$$F \equiv QyG[x/y].$$

Nota 72 *El significado del Lema anterior es el siguiente: si en una fórmula aparece una variable cuantificada x, y si elegimos una variable nueva y que no está entre las variables libres del resto de la fórmula, la fórmula obtenida reemplazando cada aparición de x por y es tautológicamente equivalente a la incial. En particular,*

Proposición 73 *Toda fórmula es equivalente a una fórmula rectificada.*

Demostración.– Aplicando recursivamente el Lema anterior. ∎

Finalmente tendremos:

Teorema 74 *Toda fórmula es equivalente tautológicamente a una fórmula en forma RPF (prenexa y rectificada). Además, podemos suponer que la matriz F^* están en forma normal conjuntiva.*

Demostración.– Definiremos un proceso recursivo en la longitud de la fórmula, que realiza la tarea indicada. Lo llamamos RPF

- Si F es una fórmula atómica, entonces, $RPF(F) := F$. (Ya está en forma RPF).
- Si F no es una fórmula atómica, puede ser de los tipos siguientes:
  - Si $F = \neg F_1$, siendo $RPF(F_1) := Q_1 y_1 \cdots Q_n y_n G$, con $Q_i \in \{\forall, \exists\}$ entonces, definiremos
    $$RPF(F) := \overline{Q_1} y_1 \cdots \overline{Q_n} y_n \neg G,$$
    donde:
    $$\overline{Q_i} := \begin{cases} \forall & \text{si } Q_i = \exists \\ \exists & \text{si } Q_i = \forall \end{cases}$$
  - Si $F = (F_1 \circ F_2)$, con $\circ \in \{\vee, \wedge\}$ y supongamos
    $$RPF(F_1) := Q_1 y_1 \cdots Q_n y_n G_1,$$
    $$RPF(F_2) := Q'_1 z_1 \cdots Q'_n z_n G_2.$$
    Mediante el Lema para Renombrar variables ligadas, podemos suponer que F_1 y F_2 no comparten variables liagadas (esto es, $z_i \neq y_j, \forall i, j$) Entonces,
    $$RPF(F) := Q_1 y_1 \cdots Q_n y_n Q'_1 z_1 \cdots Q'_n z_n (G_1 \circ G_2).$$
  - Si, finalmente, $F = QxF_1$, con $Q \in \{\exists, \forall\}$, meintras que
    $$RPF(F_1) := Q_1 y_1 \cdots Q_n y_n G,$$
    bastará con que $RPF(F_1)$ no contenga como variable cuantificada a x, para que
    $$RPF(F) := QxQ_1 y_1 \cdots Q_n y_n G.$$

∎

3.4.2. Skolemización

Hasta aquí las reducciones han guardado la equivaencia tautológica, a partir de ahora nos conformamos con que la transformada de una fórmula sea satisfactible si y solamente si la fórmula original lo era y sin compartir necesariamente modelos. Denotaremos $F \sim G$, si se verifica que F es satisfactible si y solamente si G es satisfactible.
Aunque la Skolemización es, ante todo, un proceso más que una forma normal, diremos que una fórmula de la lógica de predicados está en *forma de Skolem* si está en forma RPF y no contiene cuantificadores existenciales, esto es, si es de la forma:

$$\forall x_1 \cdots \forall x_n G,$$

Teorema 75 *Para toda fórmula F de la Lógica de Predicados, en forma RPF, existe una fórmula $Skolem(F)$ tal que*

$$F \sim Skolem(F),$$

y, obviamente, $Skolem(F)$ no contiene cuantificadores existenciales y está en forma RPF.

Demostración.– La demostración es el procedimiento ideado por Skolem.
INPUT F una fórmula en RPF
while F contiene cuantificadores existenciales, **do**
Supongamos

$$F := \forall x_1 \forall x_2 \cdots \forall x_n \exists y G,$$

Sea f un símbolo de función que no aparece en F ni en G,
Escribamos

$$F := \forall x_1 \forall x_2 \cdots \forall x_n G[y/f(x_1, \ldots, x_n)],$$

od
OUTPUT F

■

3.4.3. Fórmulas Cerradas.

Desde ahora en adelante nos ocuparemos solamente de fórmulas cerradas de la Lógica de Predicados, esto es, fórmulas sin variables libres. Adicionalmente podemos suponer que la fórmula F está en forma de Skolem, esto es, en RPF y sólo con cuantificadores universales ($\forall$). .Podemos mostrar el siguiente resultado:

Teorema 76 *Para cada fórmula de la Lógica de Peedicados F, existe una fórmula G (calculable mediante el siguiente algoritmo) verificando:*

- *G es satisfactible si y solamente si F es satisfactible.*
- *G está en forma rectificada y prenexa.*
- *G es una fórmula cerrada (i. e. no posee variables libres)*
- *G sólo posee cuantificadores universales (está en forma de Skolem).*
- *La matriz de G están en forma CNF.*

Demostración.– Bastará con que se aplique la siguiente secuencia de procesos.

- Usando "Renombrar Variables" obtener una versión rectificada F_1 de F.
- Si $y_1, \ldots, y_m$ son las variables libres de F_1, escribir

$$F_2 := \exists y_1 \cdots \exists y_m f_1.$$

  La fórmula F_2 es satisfactible si y solamente si F_1 y F lo son.
- Sea F_3 la fórmula obtenida mediante aplicación de RPF a F_2. De nuevo, F_3 es satisfactible si y solamente si F lo es.
- Sea F_4 el resultado de aplicar Skolemización a F_3.
- Aplicar la transformación a CNF del C'alculo Proposocional de la matriz de F_4.

■

Nota 77 *Obsérvese que la fórmula obtenida es "equivalente salvo satisfacibilidad", esto es, no es equivalencia tautológica.*

3.4.4. Teoría de Herbrand

La idea de Herbrand consiste en hallar, para cada fórmula F, un modelo contable $\mathcal{A}$ tal que si F es satisfactible, entonces satisface ese modelo contable (i.e. $\mathcal{A} \models F$). Lo que haremos es construir ese modelo, conocido como estructura de Herbrand.

Definición 78 (Universo de Herbrand) *Sea F una fórmula cerrada de la Lógica de Predicados en forma Skolem. Llamaremos universo de Herbrand al conjunto $D(F)$ definido mediante las reglas siguientes:*

- *Todas las constantes que aparecen en F están en $D(F)$. Si F no poseyera ninguna constante introduciremos una constante nueva (que denotaremos por a).*
- *Para cada símbolo de función f de aridad k que aparece en F y para cualesquiera elementos $(t_1, \ldots, t_k) \in D(F)^k$, se tiene que $f(t_1, \ldots, t_k) \in D(F)$.*

Ejemplo 4 *Consideremos las fórmulas*

$$F := \forall x \forall y \forall z P(x, f(x), g(z, x)),$$

$$G := \forall x \forall y Q(c, f(x), h(y, b)).$$

En F no hay constantes, en G hay constantes que son b y c. Entonces,

$$D(F) := \{a, f(a), g(a, f(a)), g(f(a), a), f^2(a), f(g(a, f(a))), g(a, f^2(a)), ...\},$$

$$D(G) := \{b, c, f(b), f(c), h(b, c), h(c, b), h(b, f(c)), ...\}.$$

Definición 79 (Estructura de Herbrand) *Sea F una fórmula en las condiciones anteriores. Una estructura de Herbrand para F, es una estructura $\mathcal{A} := (U_{\mathcal{A}}, I_{\mathcal{A}})$, verificando:*

- $U_{\mathcal{A}} = D(F)$,
- *Para cada función f de aridad k que aparece en F, y para todos los términos $t_1, \ldots, t_k \in D(F)$*
$$f^{\mathcal{A}}(t_1, \ldots, t_k) = f(t_1, \ldots, t_k).$$

Es claro que podemos construir una estructura de Herbrand a partir de $D(F)$. Obsérvese que no se incluyen relacionales, es decir, evitamos dar, valor semántico a los relacionales de F, lo que justamente nos permitirá una reducción al Cálculo Proposicional (infinitas fórmulas).

Definición 80 (Expansión de Herbrand) *Sea F una fórmula en las condiciones anteriores y supongamos F^* la matriz de F, siendo*

$$F = \forall y_1 \cdots \forall y_n F^*(y_1, \ldots, y_n).$$

Llamaremos expansión de Herbrand al conjunto:

$$E[F] := \{F^*[y_1/t_1] \cdots [y_n/t_n] \ : \ t_i \in D(F)\}.$$

En otras palabras, la Expansión de Herbrand consiste en reemplazar en la matriz F^* de F, toda aparición de variables por elementos de $D(F)$.
Ahora realizamos la siguiente construcción:
En las notaciones anteriores, consideremos el conjunto de todos los relacionales que aparecen en la fórmula F (sólo son unos pocos, esto es, un conjutno finito)

$$\mathcal{R} := \{R_1^{k_1}, \ldots, R_m^{k_m}\}.$$

Supondremos que $R_i^{k_i}$ tiene aridad k_i.
Ahora consideremos un conjunto infinito (numerable) de variables nuevas (que consideraremos variables booleanas):

$$\mathcal{B} := \{X_1, \ldots, X_n, \ldots\}.$$

Dado que el conjunto $D(F)$ es numerable, el conjunto de todas las posibles sustituciones de todos los posibles relacionales también es numerable, esto es,

$$\mathcal{R}[F] := R_i^{k_i}(t_1, \ldots, t_{k_i}) \ : \ 1 \leq i \leq m, t_j \in D(F)\}.$$

Ahora podemos definir una aplicación biyectiva (computable) entre $\mathcal{R}[F]$ y $\mathcal{B}$. Podemos usar, por ejemplo, un orden dado por *longitud + lexicográfico* que permite esa identificación.
Finalmente, obsérvese que cada elemento de $E[F]$ es, en realidad, una combinación usando los operadores booleanos $\{\vee, \wedge, \neg\}$ de elementos de $\mathcal{R}[F]$.
Así, para cada elemento $H \in E[F]$ podemos asociar, una fórmula $\varphi(H)$ del Cálculo Proposicional, involucrando las variables en $\mathcal{B}$. Decimos que $E[F]$ es satisfactible si lo es la colección infinita de fórmulas del Cálculo Proposicional

$$\{\varphi(H) \ : \ H \in E[F]\}.$$

Teorema 81 (Gödel–Herbrand–Skolem) *Para cada fórmula F en forma de Skolem, F es satisfactible si y solamente si es satisfactible sobre un universo de Skolem, si y solamente si la expansión $E[F]$ es satisfactible.*

Teorema 82 (Herbrand) *Una fórmula F en forma de Skolem, es insatisfactible si y solamente si existe un subconjunto finito $\mathcal{I} \subset E[F]$, tal que $\{\varphi(H) \ : \ H \in \mathcal{I}\}$ es insatisfactible.*

Obsérvese que, en principio, no podría decidirse la satisfacibilidad de $E[F]$ salvo cotejando una infinidad de fórmulas, lo cual no es factible computacionalmente.
En cambio, si F es insatisfactible, probando con todos los subconjuntos de $E[F]$ uno podría dar con ese conjunto finito de fórmulas booleanas del Cálculo Proposicional que se construyen a partir de $E[F]$ y que son insatisfactibles. Este es el concepto de problema recursivamente enumerable o, si se prefiere, semi–decidible, que representa el algoritmo siguiente:

Gilmore

INPUT Una fórmula F en forma de Skolem,
$\mathcal{I}$ un subconjunto de $E[H]$,
$\varphi(\mathcal{I}) := \{\varphi(H) \ : \ H \in \mathcal{I}\}$
while $\varphi(\mathcal{I})$ es satisfactible, **do**
$\mathcal{I}$:= next finite subset of $E[H]$
od
OUTPUT "INSATISFACTIBLE"

Teorema 83 (Church) *La Lógica de Predicados es indecidible, esto es, no puede existir ningún algoritmo o programa que decida si una cierta fórmula es satisfactible o no.*
Al mismo tiempo, el resultado anterior nos garantiza que la Lógica de Predicados es semi–decibile, esto es

- *Insatisfacibilidad es recursivamente enumerable, pero no es recursivo.*
- *La fórmulas válidas (tautologías) son recursivamente enumerables, pero no forman un lenguaje recursivo.*

Nota 84 *Habitualmente, los autores suelen dar una demostración de un enunciado que consiste en lo siguiente:*
"Reducir" Insatisfiacibilidad de la Lógica de Predicados a cualquier lenguaje recursivamente enumerable que no es recursivo (es decir, que alguien ha demostrado que no es recursivo)
Usando la Teoría de Herbrand uno concluye que Insatisfactibilidad es recursivamente enumerable (es el lenguaje aceptado por una máquina de Turing).
Sin embargo, demostrar que no es recusivo obliga a demostrar (o suponer que se ha demostrado) que alguno de los siguientes no es recursivo:

1. **[Halting Problem]** *El lenguaje*

$$\{(C_M, x) \ : \ C_M \text{ es el código de una máquina de Turing}, x \in L(M)\}.$$

   esrecursivamente enumerable y no es recursivo (A. Turing).

2. **[ENT]** *La Teoría Elemental de Números es indecidible (K. Gödel).*

3. **Word Problem for Semi-groups, groups, etc...** *El problema de Palabra es recursivamente enumerable pero no es recursivo para semi-grupos, grupos finitamente presentados y finitamente generados..etc.*

4. **[PCP]** *Post Correspondence Problem....*

Si lo demostramos nos metemos en un esfuerzo considerable que, aunque vale la pena, no se corresponde con el curso. Si, por el contrario, no demostramos que alguno de éstos es indecidible, qué más da suponer también la veracidad de la indecidibilidad de Insatisfacibilidad?...

3.5. Unas Palabras sobre Indecidibilidad

3.6. Resolución

Lo que sigue pretende refinar un poco el procedimiento que se sigue de las ideas de Herbrand y Skolem. Trataremos de hacer este refinamiento relacionándolo con el Procedimiento de Resolución ya visto para el C'alculo Proposicional.

Definición 85 *Una cláusula de la Lógica de Predicados es una fórmula F, libre de cuantificadores, dada como una disyunción finita de fórmulas atómicas (literales positivos) o de negaciones de fórmulas atómicas (litelares negativos).*

Definición 86 (Clásulas de Horn de la Lógica de Predicados) *Una clásula de Horn de la Lógica de Predicados es una cláusula en la que hay, a lo sumo, un literal positivo. Son cláusulas de Horn:*

- *Las fórmulas atómicas (que llamaremos Hechos o "Facts"), es decir las fórmulas con sólo un literal positivo*
- *La fórmulas sin literales positivos (que llamaremos "query o goal formulae" o fórmulas objetivo), es decir, fórmulas del tipo:*

  $$((\neg C_1) \vee (\neg C_2) \vee \cdots (\neg C_r)),$$

  que denotaremos mediante:

  $$(C_1 \wedge \ldots \wedge C_r) \rightarrow 0, \text{ o simplemente mediante } (C_1 \wedge \ldots \wedge C_r) \rightarrow .$$

  donde $C_1, \ldots, C_r$ son fórmulas atómicas.
- *Las Reglas ("rules") o fórmulas donde hay al menos un literal positivo y al menos un literal negativo, esto es*

  $$(C_1 \wedge \ldots \wedge C_r) \rightarrow G,$$

  donde $C_1, \ldots, C_r, G$ son fórmulas atómicas.

3.6.1. Ground Resolution.

El Input. A partir de ahora tratamos de hallar una refutación de una fórmula F de la Lógica de Predicados de la que supondremos:

- F está en forma Prenexa y Rectificada (RPF)
- F es cerrada (i.e. no posee varables libres)
- $F = Skolem(F)$, es decir sólo posee cuantificadores universales.

- La matriz F^* de F es dada en forma CNF (forma normal conjuntiva de cláusulas de la Lógica de Predicados).

Definición 87 (Instancia) *Si una fórmula resulta de hacer ciertas sustituciones en otra fórmula, diremos que G es una instancia de F.*
Las sustituciones "sub" que hacen que una fórmula F se transforme en una fórmula G libre de variables se llaman "ground sustitutions" o sustituciones de base.
Las instancias de una fórmula F obtenidas tras hacer una sustitución de base se llaman instancias de base (o ground instances).

Nótese que tras obtener una instancia de base de una fórmula F, es decir, una expresión $G = F[sub]$, con G libre de variables, lo que tendremos se evalúa como una fórmula del cálculo proposicional. Si, además, la instancia de base G se obtiene mediante sustituciones en el Universo de Herbrand, identificando las apariciones de instancias de fórmulas atómicas con variables booleanas, la instancia de base sobre el universo de Herbrand es identificable con una fórmula en forma CNF del Cálculo Proposicional.

Ejemplo 5 *Consideremos la fórmula*

$$F := \forall x(P(x) \wedge \neg(P(f(x))).$$

Constuyamos el Universo de Herbrand:

$$\{a, f(a), f(f(a)), ...\}.$$

donde a es una constante introducida por no haber constantes en F.
Ahora consideremos todas las instancias de base obtenibles mediante sustituciones en el Universo de Herbrand:

$$F_1 := (P(a) \wedge \neg P(f(a))) \, , F_2 := \left(P(f(a)) \wedge \neg P(f^2(a))\right) \, , F_3 := \left(P(f^2(a)) \wedge \neg P(f^3(a))\right) \ldots$$

Ahora, nuestra fórmula sería satisfactible si todas las sustituciones en el Universo de Herbrand fueran satisfactibles a la vez. Es insatisfactible si hay un conjunto finito de instancias que son insatisfactibles.
En el ejemplo tratado, las fórmulas F_1 y F_2 son simultáneamente insatisfactibles o, si se prefiere,

$$F_1 \wedge F_2,$$

es insatisfactible co lo que F es insatisfactible.

Ejemplo 6 *Se sugiere al alumno hacer el mismo procedimiento con la fórmula:*

$$G := \forall x \forall y \left((\neg P(x) \vee \neg P(f(x)) \vee Q(y)) \wedge (P(y)) \wedge (\neg P(g(b, x)))\right).$$

Obsérvese que ahora aparecen dos constantes $\{a, b\}$ y dos funcionales f (unario) y g (binario), lo que fuerza a una cosntrucción del Universo de Herbrand en forma especial.

La reflexión de los ejemplos anteriores indica que, sabiendo que la matriz de nuestra fórmula está en forma CNF, bien podríamos trabajar directamente sobre las clásulas de F^*.
Esto conduce al siguiente enunciado:

Teorema 88 (Ground Resolution Theorem) *Sea F una fórmula de la Lógica de Predicados en forma de Skolem con matriz en CNF.Entonces, F es insatisfactible si y solamente si existe una sucesión finita de cláusulas del Cálculo Proposicional*

$$C_1, \ldots, C_r,$$

verificando las siguientes propiedades:

- $C_n = \square$ *es la cláusula vacía.*
- *Para cada i, $i \leq i \leq n$, la cláusla C_i verifica:*
  - *O bien, la cláusulas C_i se obtiene como instancia de base de una cláusula que aparece en la matriz F^*,*
  - *O bien, $C_i := Res(C_a, C_b, \ell)$ es la resolvente de dos cláusulas precedentes (esto es $a, b < i$).*

Demostración.– Véase [Schöning, 1989], p. 82 y anteriores. ∎

Nota 89 *Otra opción podría ser la versión "brutalizada" de Gilmore en la forma siguiente:*

INPUT: *Una fórmula F de la Lógica de Predicados en forma de Skolem con matriz F^* en CNF.*
Sean $E[F] := \{F_1, F_2, \ldots, F_n, \ldots\}$
Sea $B[F] := \{G_i = \varphi(F_i) \ : \ i \in \mathbb{N}\}$.
$i := 0$
$M := \emptyset$
while $\square \notin M$ **do**
    $i := 1 + 1$
    $M := M \cup \{F_i\}$
    $M := Res^*(M)$
od
OUTPUT: *"insatisfactible" y parar.*

3.6.2. Unificación: Unificador más general.

De nuevo son ideas de J.A. Robinson. La estrategia de Resolución basada en Herbrand, requiere de algunas posibilidades y estrategias que permitan no pasar por el inmenso árbol de comparaciones entre las instancias. Esto no quiere decir que lo que sigue es a opción algorítmica definitiva para detectar insatisfactibilidad. Lo que sigue quiere simplemente mostrar una estrategia un poco mejor que la anterior (aunque ingualmente define un algoritmo para un lenguaje semi–decidible).

Definición 90 (Unificador) *Se llama unificador de un conjunto para un conjunto de f'ormulas atómicas $\{L_1, \ldots, L_n\}$ a toda substitución* sub *tal que*

$$L_1[sub] = L_2[sub] = \cdots = L_r[sub].$$

En otras palabras, una unificación es una sustitución que hace que las instancias de varias fórmulas atómicas coincidan. Si existe algún Unificador, se dice que el conjunto de fórmulas es *unificable*. En caso contrario, se dice que es *no unificable*.
Intentaremos detectar cuándo se presenta la posibilidad de unificar. Para ello buscaremos el *unificador más general*.

Definición 91 *Sea* $\{L_1, \ldots, L_r\}$ *un conjunto unificable de fórmulas atómicas. Se llama unificador más general a todo unificador* sub *tal que para cualquier otro unificador* sub', *existen unas sustituciones* $[s]$ *tales que*

$$sub' = sub[s].$$

Notación 92 *Dado un conjunto finito* $\mathcal{L} = \{L_1, \ldots, L_r\}$ *de fórmulas atómicas de la Lógica de Predicados y dada una sustitución* sub, *escribiremos*

$$\mathcal{L}sub,$$

para denotar el conjunto

$$\mathcal{L}sub = \{L_1 sub, \ldots, L_r sub\},$$

obteniendo realizando la substitución sub *en todas las fórmulas atómicas de la Lógica de Predicados que están en* $\mathcal{L}$.
Denotaremos por $\sharp(\mathcal{L}sub)$ *el cardinal de ese conjunto. El lector observará que un conjunto* $\mathcal{L}$ *de fórmulas atómicas de la Lógica de Predicados es unificable si existe una sustitución* sub *tal que el cardinal de* $\mathcal{L}sub$ *es* 1, *es decir sii* $\sharp(\mathcal{L}sub) = 1$.

Teorema 93 (Teorema de Unificación de Robinson) *Todo conjunto unificable de fórmulas atómicas de la Lógica de Predicados posee un unificador más general. Además, el siguiente procedimiento decide si un conjunto finito de fórmulas atómicas es unificable y en caso de que lo sea clacula un unificador más general:*

INPUT: *un conjunto finito* $\mathcal{L} := \{L_1, \ldots, L_r\}$ *de fórmulas atómicas de la Lógica de Predicados.*
$sub := [\,]$ *(la sustitución vacía).*
while $sharp(\mathcal{L}sub) > 1$ **do**
Leer de izquierda a derecha las fórmulas de $\mathcal{L}sub$ hasta dar con dos distintas L_1 y L_2.
if *ninguno de los símbolos en* L_1 *y en* L_2 *es una variable,*
then *output "No Unificable" y* **end.**
else *Sea* x *una variable en* L_1 *y* t *el término en* L_2
que ocupa el mismo lugar que x.
if x *aparece en* t, **then** *output "No Unificable" y* **end.**
else $sub := sub[x/t]$
fi
fi
od
OUTPUT *sub.*

Demostración.– Véase [Nerode & Shore,1993] Sección II.11 o [Schöning, 1989], p.84. ∎

Ejemplo 7 *Apliquemos el algoritmo anterior a la colección de fórmulas atómicas*

$$\mathcal{L} := \{\neg P(f(z, g(a, y), h(z)), \neg P(f(f(u, v), w), h(f(a, b))\}.$$

Como primera observación, el relacional tiene que ser el mismo en todas las fórmulas atómicas de la lista que tratamos.
Además, recordemos que $\{a, b\}$ *son constantes, mientras que* $\{u, v, w, x, y, z\}$ *son variables.*
Comenzamos con la sustitución vacía $sub := [\,]$ *y miramos la lista inicial. Tenemos*

$$L_1 := \neg P(f(z, g(a, y), h(z)), L_2 := \neg P(f(f(u, v), w), h(f(a, b)).$$

son distintas y contienen variables, así que podemos empezar a comparar.
En L_1, *la variable* z *ocupa el lugar del término* $f(a, b)$ *que aparece en* L_2. *Por tanto, comenzamos con*

$$sub := [z/f(a, b)]$$

y calculamos

$$\mathcal{L}[z/f(a, b)] := \{L_1[z/f(a, b)], L2[z/f(a, b)]\}.$$

Tendremos:

$$L_1[z/f(a, b)] := \neg P(f(f(a, b), g(a, y), h(f(a, b))),$$

mientras que

$$L_2[z/f(a, b)] := L_2,$$

porque L_2 *no contiene ninguna vez a la variable* z.

Al final, habremos obtenido:

$$\mathcal{L}[z/f(a,b)] := \{\neg P(f(f(a,b),g(a,y),h(f(a,b))),\neg P(f(f(u,v),w),h(f(a,b))\}.$$

Volviendo al ciclo **while** *hagamos*

$$\mathcal{L} := \mathcal{L}[z/f(a,b)], L_2 = L_1[z/f(a,b)], L_1 = L_2[z/f(a,b)],$$

y nos ocupamos de la variable w y del término $g(a,y)$.
Tendremos la nueva sustitución:

$$sub := [z/f(a,b)][w/g(a,y)]$$

$$\mathcal{L} := \mathcal{L}[z/f(a,b)][w/g(a,y)] := \{\neg P(f(f(a,b),g(a,y),h(f(a,b))),\neg P(f(f(u,v),g(a,y)),h(f(a,b))\}.$$

Siguiendo con el proceso, $\mathcal{L}$ saldrá unificable y el unificador más general será:

$$sub := [z/f(a,b)][w/g(a,y)][u/a][v/b].$$

3.6.3. Resolvente en Lógica de Predicados

Aprovechando la calculabilidad del Unificador más general, podemos introducir la noción de resolvente en Lógica de Predicados y aplicarla para generar un procedimiento similar al desarrollado en el caso del Cálculo Proposicional (eso sí, con las particularidades del caso).

Definición 94 (Resolvente en Lógica de Predicados) *Sean C_1, C_2 y R tres cláusulas de la Lógica de Predicados. Diremos que R es una resolvente de C_1 y C_2 si se verifican las propiedades siguientes:*

- *Existen dos sustituciones s_1 y s_2 que son meras aplicaciones del proceso de "Renombrar Variables", de tal modo que $C_1 s_1$ y $C_2 s_2$ no comparten variables.*
- *Existe un conjunto de fórmulas atómicas $L_1, \ldots, L_m \in C_1 s_1$ y otro conjunto $L'_1, \ldots, L'_r \in C_2 s_2$ tales que el conjunto:*

  $$\{\neg L_1, \ldots, \neg L_m, L'_1, \ldots, L'_r\}.$$

  en unificable. Sea sub *el unificador más general de ese conjunto.*
- *Entonces, la resolvente R tiene la forma:*

  $$R := (C_1 s_1 \setminus \{L_1, \ldots, L_m\}) \bigcup (C_2 s_2 \setminus \{L'_1 \ldots, L'_r\})\,\text{sub}.$$

Ejemplo 8 *Consideremos las clásulas*

$$C_1 := \{P(f(x)), \neg Q(z), P(z)\}, \quad C_2 := \{\neg P(x), R(g(x), a)\}.$$

Como comparten la variable x, hagamos las sustituciones:

$$s_1 := [\,], s_2 := [x/u].$$

En totras palabras, dejamos C_1 como está y en C_2 reemplazamos x por u.

$$C_1 s_1 = C_1 := \{P(f(x)), \neg Q(z), P(z)\}, \quad C_2 s_2 := \{\neg P(u), R(g(u), a)\}.$$

Elejimos

$$L_1 := P(f(x)),\ \ L_2 := P(z),\ \ L_1' := \neg P(u).$$

Aplicamos el algoritmo de unificación a

$$\{\neg L_1, \neg L_2, L_1'\} = \{\neg P(f(x)), \neg P(z), \neg P(u)\}.$$

El conjunto es unificable y el unificador más general es

$$\text{sub} := [x/u][z/f(x)][u/f(x)].$$

Finalmente, aplicamos

$$Res(C_1, C_2) := (C_1 s_1 \setminus \{L_1, L_2\}) \bigcup \left(C_2 s_2 \setminus \{L_1'\}\right) \text{sub} = \{\neg Q(z), R(g(u), a)\}\text{sub}.$$

Por tanto,

$$Res(C_1, C_2) = \{\neg Q(f(x)), R(g(x), a)\}.$$

Definición 95 (Resolvente de una fórmula) *Sea F una conjunción de cláusulas de la Lógica de Predicados. Llamamos resolvente de F a*

$$Res(F) := F \cup \{Res(C_1, C_2) \ :\ (C_1, C_2) \in F\}.$$

Recursivamente,

$$Res^n(F) := Res(Res^{n-1}(F)).$$

$$Res^*(F) := \bigcup_{n \in \mathbb{N}} Res^n(F).$$

Proposición 96 *En las notaciones de la anterior Definición, la clásula vacía $\square$ está en $Res^*(F)$ si y solamente si existe una sucesión finita de cláusulas $C_1, \ldots, C_n$ tales que:*

- $C_n = \square$,
- *Para cada i, $1 \leq i \leq n$, se ha de tener:*
  - *O bien C_i está en F*
  - *O bien $C_i = Res(C_a, C_b)$, con $a, b < i$.*

Un resultado clave, que permite relacionar esta Resolución refinada con lo hecho en Secciones anteriores es el llamado **Lifting Lemma**.

Lema 97 (Lifting Lemma) *Sean C_1, C_2 dos clásulas de la Lógica de Predicados. Sea C_1', C_2' dos instancias, respectivamente de C_1 y C_2, pertenecientes al Cálculo Proposicional. Supongamos que C_1' y C_2' con conflictivas con respecto a un literal ℓ y sea $R' := Res(C_1', C_2', \ell)$ la resolvente correspondiente del Cálculo Proposicional.*
Entonces, existe una resolvente $R := Res(C_1, C_2)$ tal que R' es la instancia de R.

Demostración.– Véase [Schöning, 1989], p.90 o [Nerode & Shore,1993], Lemma II.13.8, p. 136 ■

Teorema 98 *Sea F una fórmula de la Lógica de Predicados en forma Skolem con matriz F^* en CNF. Entonces, F es insatisfactible si y solamente si $\square \in Res^*(F^*)$.*

Demostración.– Basta con combinar el Lifting Lemma anterior con el Ground Resolution Theorem (Teorema 88) y la Proposición 96 anterior, ■

Capítulo 4

Programación Lógica

4.1. Introducción

A partir de este Capítulo trataremos los fundamentos lógicos de PROLOG. Esto supone restringirnos a la Lógica de predicados con fórmulas cuyas matrices son fórmulas de Horn. Esto es,

$$F := \forall x_1, \ldots, \forall x_n F^*,$$

donde F^* es una conjunción finita de cláulas de Horn de la Lógica de Predicados.

4.2. Programas mediante Cláusulas de Horn

Retomamos la Definición 86

Definición 99 *Una clásula de Horn de la Lógica de Predicados es una cláusula en la que hay, a lo sumo, un literal positivo. Son cláusulas de Horn:*

- *Las fórmulas atómicas (que llamaremos Hechos o "Facts"), es decir las fórmulas con sólo un literal positivo*
- *La fórmulas sin literales positivos (que llamaremos "query o goal formulae" o fórmulas objetivo), es decir, fórmulas del tipo:*

  $$((\neg C_1) \vee (\neg C_2) \vee \cdots (\neg C_r)),$$

  que denotaremos mediante:

  $$(C_1 \wedge \ldots \wedge C_r) \rightarrow 0, \text{ o simplemente mediante } (C_1 \wedge \ldots \wedge C_r) \rightarrow .$$

  donde $C_1, \ldots, C_r$ son fórmulas atómicas.
- *Las Reglas ("rules") o fórmulas donde hay al menos un literal positivo y al menos un literal negativo, esto es*

  $$(C_1 \wedge \ldots \wedge C_r) \rightarrow G,$$

  donde $C_1, \ldots, C_r, G$ son fórmulas atómicas.

Definición 100 (Logic Programming) *Un Programa Lógico (o Programa con clásulas de Horn) es una lista finita de hechos y reglas, es decir, es una conjunción de clásulas de Horn de los tipos Hecho o Regla.*

Notación 101 *En PROLOG, por limitaciones pasadas de símbolos utulizables, no se usan las notaciones estándard de la Lógica sino sus "versiones" limitadas y poco estéticas. De todas formas, las introducimos.*

- *Los hechos son fórmulas atómicas, esto es realcionales aplicados a términos, luego usan la sintaxis correspondiente sin intervenciones de símbolos especialmente notables.*

$$R(x, f(a)) \text{ se escribe } \mathsf{R(x,f(a))}.$$

- *Para las reglas, el símbolo de implicación es reemplazado pro un símbolo de implicación "hacia atrás" :-. Así,*

$$(Q_1 \wedge Q_2 \wedge \cdots \wedge Q_r) \to P \text{ se escribe } \mathsf{P{:-}\ Q_1,Q_2,\ldots,\ Q_r}.$$

Aquí P se llama la cabeza de procedimiento y $Q_1, \ldots, Q_r$ se llaman el cuerpo del procedimiento. Cada Q_i se denomina llamada de procedimiento.

Notación 102 *Un programa Lógico $\mathcal{P}$ se activa mediante una cláusula objetivo. La cláusula objetivo se representa mediante:*

$$(\neg Q_1 \vee \neg Q_2 \vee \cdots \vee \neg Q_r) \to P \text{ se escribe } \mathsf{?{-}\ Q_1,Q_2,\ldots,\ Q_r}.$$

4.3. Resolución SLD

La cláusula objetivo activará el programa PROLOG, así, un programa $\mathcal{P}$ será activado por una clásula objetivo G y el intéprete de PROLOG tratará de decidir si la cláusula vacía $\square$ es deducible desde $\mathcal{P} \wedge G$. Para definir este concepto, introduciremos las siguientes nociones. Para empezar, usaremos notación de lista para representar nuestras cláusulas. Esto quiere decir, que supondremos que las fórmulas atómicas aparecen en un cierto orden. Así, escribiremos

$$G = [\neg A_1, \neg A_2, \ldots, \neg A_k], \text{ para representar } (\neg A_1) \vee (\neg A_2) \vee \ldots \vee (\neg A_k),$$

en el caso de las clásulas objetivo. Y para las clásulas de programa o reglas,

$$C = [B, \neg C_1, \neg C_2, \ldots, \neg C_n] \text{ para representar } (C_1 \wedge C_2 \wedge \cdots \wedge C_n) \to B.$$

Es decir, las fórmulas atómicas están "ordenadas".
Se trata ahora de "entender" el funcionamiento de un Programa Lógico y sus principales propiedades. Para ello, los autores difieren en su formalización del proceso, aunque el fondo es el mismo en todos los casos. Se usa o bien la terminología de Resolución SLD o bien la interpretación procedural. En lo que todos est'an más mo menos conformes es en la noción de Resolvente Ordenada:

Definición 103 (Resolvente Ordenada de dos Cláusulas de Horn) *Sean G una clásula objetivo y C una regla o un hecho, esto es, supongamos*

$$G = [\neg A_1, \neg A_2, \ldots, \neg A_k] \qquad (k \geq 1),$$

y

$$C = [B, \neg C_1, \neg C_2, \ldots, \neg C_n] \qquad (n \geq 0).$$

Supongamos que B y A_i son unificables y sea s su unificador más general. Llamaremos resultante ordenada de G y C a la clásula

$$Res_O(G, C, A_i) = [\neg A_1, \ldots, \neg A_{i-1}, \neg C_1, \ldots, \neg C_n, \neg A_{i+1}, \ldots, \neg A_k]s.$$

A la sustitución s la llamaremos respuesta a esta selección de unificables y la denotaremos por

$$s = Answ_O(G, C, A_i).$$

Obsérvese que ha intervenido un proceso de selección de litelares potencialmente unificables.

Nota 104 *Nótese que la resolvente ordenada es como la resolvente de la Lógica de Predicados sólo que en lugar de resolver en un sólo paso todos los literales unificables, se resuleven los literales unificables uno tras otro.*

Definición 105 (Resolución LD) *Sea $\mathcal{P}$ un programa Lógico y G una cláusula objetivo. Se denomina refutación LD*[1] *de $P \cup [G]$ a una sucesión finita de cláusulas ordenadas*

$$(G_0, C_0), (G_1, C_1), \ldots, (G_n, C_n),$$

donde

- $G_0 = G$,
- *cada C_i es una clásula de $\mathcal{P}$ (salvo renombrar variables),*
- *cada G_i es una clásula ordenada.*
- *Para cada i, $0 \leq i \leq n-1$, $G_{i+1} = Res_O(C_i, G_i, A_i)$ con respecto a algún literal A_i de G_i,*
- *Para algún literal A de G_n se tiene*

$$Res_O(C_n, G_n, A) = \square\,.$$

A la sucesión se la denomina también LD-resolución de $\mathcal{P} \cup [G]$.

Proposición 106 (Completitud de la resolución LD) *Si $CP \cup [G]$ es un conjunto de clásuluas ordenadas insatisfactibles, entonces existe una $LD-$refutación de $\mathcal{P} \cup [G]$ comenzando con G.*

Demostración.– Véase [Nerode & Shore,1993] Lemma III.1.4, p. 146. ∎

Definición 107 *Se llama Regla de Selección (selection rule) a toda aplicación R definida en el conjunto de cláusulas objetivo y con rango el conjunto de literales, esto es, para cada cláusula objetivo G, $R(G)$ es un literal en G.*

Ejemplo 9 *Una de las más simples reglas de selección de literales en clásulas objetivo ordenadas es la selecci'on por la posici'on relativa como elemento de las lista:*

- *Leftmost : elegir el literal más a la izquierda.*

[1] Linear Definite

- *Rightmost: elegir el literal más a la derecha.*

Dejamos al lector el diseño de variantes de la regla de selección.

Definición 108 (Resolución SLD) *Se llama resolución SLD*[2] *o refutación SLD con regla de selección R a toda refutación LD donde el literal eliminado a cada paso está deterninado por la regla R. Esto es, Sea $\mathcal{P}$ un programa Lógico y G una cláusula objetivo. Se denomina refutación SLD (linear definite) de $P \cup [G]$ con regla de selección R a una sucesión finita de cláusulas ordenadas*

$$(G_0, C_0), (G_1, C_1), \ldots, (G_n, C_n),$$

donde

- $G_0 = G$,
- *cada C_i es una clásula de $\mathcal{P}$ (salvo renombrar variables),*
- *cada G_i es una clásula ordenada.*
- *Para cada i, $0 \leq i \leq n-1$, $G_{i+1} = Res_O(C_i, G_i, R(G_i))$ con respecto a algún literal $A_i = R(G_i)$ de G_i, seleccionado por la regla R,*
- $$Res_O(C_n, G_n, R(G_n)) = \square\,.$$

A la sucesión se la denomina también LD-resolución de $\mathcal{P} \cup [G]$.

Definición 109 *Una regla de selección se dice invariante si para cada clásula C y cada sustitución θ,*

$$R(C)\theta = R(C\theta).$$

Es decir, si elegir conmuta con sustituir.

Nota 110 *Leftmost y Rightmost son reglas de selcción invariantes.*

Proposición 111 (Completitud de la resolución SLD) *Si $CP \cup [G]$ es un conjunto de clásuluas ordenadas insatisfactibles, entonces existe una $SLD-$refutación de $\mathcal{P} \cup [G]$ con cualquier regla de selección R comenzando con G.*

Demostración.– Véase [Nerode & Shore,1993] Theorem III.1.8, p. 148 para el caso de reglas invariantes. ∎

4.4. Programación Lógica

Para poder interpretar razonablemente lo que hace un intérprete de PROLOG al uso, es conveniente utilizar un lenguaje alternativo que reescribe lo anterior en términos de computación.

Definición 112 (Consequencia Lógica) *Sea A una fórmula cerrada y $\mathcal{M} = \{A_1, \ldots, A_n\}$ un conjunto finito de fórmulas cerradas. Decimos que A es consecuencia lógica de $\mathcal{M}$ si para toda estructura $\mathcal{A}$ tal que $\mathcal{A}(A_i) = 1$, para todo i, se tiene $\mathcal{A}(A) = 1$. Denotaremos $\mathcal{M} \models A$, para decir A es consecuencia lógica de $\mathcal{M}$.*

[2]Linear Definite with Selection rule

Nota 113 *Nótese que $\mathcal{M} \models A$ es equivalente a decir que la fórmula cerrada (i.e. sin variables libres)*

$$F := (A_1 \wedge \cdots \wedge A_n) \rightarrow A,$$

es una tautología ($\mathcal{A}(F) = 1$, para toda estructura adecuada a F).

Definición 114 (Correct Answer Sustitution) *Sea $\mathcal{P}$ un programa lógico y G una cláusula objetivo. Llamaremos substitución de respuesta correcta a toda sustitución θ tal que $\mathcal{P} \models \forall(\neg G)\theta$, donde $\forall$ representa todas las variables libres que puedan quedar aún en $\neg G$ tras hacer la sustitución θ.*

El objetivo de la programación lógica (a través de cláusulas de Horn) es deducir si una conjunción $H = H_1 \wedge H_2 \wedge \cdots \wedge H_r$ de propiedades posee una instancia que es consecuencia de un programa $\mathcal{P}$ o no. Y, en caso de que lo fuera, hallar una sustitución correcta.
Para ello, se aplica el programa lógico $\mathcal{P}$ a la cláusula obtenida tras aplicar negación $G = \neg H = \neg H_1 \vee \neg H_2 \vee \cdots \vee \neg H_r$.
Veamos cómo funciona ese proceso indeterminista.

Definición 115 (El Sistema de Transición de un Programa Lógico) *Las configuraciones de la Programación Lógica son pares (G, sub) donde G es una clásula objetivo y sub es una sustitución.*
Sea $\mathcal{P}$ un programa lógico. La función de transición asociada a $\mathcal{P}$ es una relación entre dos configuraciones

$$(G_1, sub_1) \rightarrow_{\mathcal{P}} (G_2, sub_2),$$

definida mediante:

- *Existe una cláusula C del programa $\mathcal{P}$*

  $$C = \{B, \neg C_1, \neg C_2, \ldots, \neg C_n\} \qquad (n \geq 0),$$

  tal que, salvo renombrar variables, B y una clásula A_i de G_1 son unificables. Además su unificador más general es $s = mgu(B, A_i)$.

- $$G_2 = Res_O(C, G, A_i),$$

- $$s := Answ_O(C, G, A_i) \qquad \text{y} \qquad sub_2 = sub_1 s.$$

Se dice que (G_1, sub_2) es computable, deducible, en un sólo paso del programa $\mathcal{P}$.

Definición 116 (Computación) *Una computación de un Programa Lógico $\mathcal{P}$ (también una Derivación o Deducción LD) sobre una cláusula objetivo G es una sucesión finita o infinita de pares de clásulas objetivos y sustituciones comenzando en $(G, [\,])$.*

$$(G, [\,]) = (G_1, sub_1) \rightarrow_{\mathcal{P}} (G_2, sub_2) \rightarrow_{\mathcal{P}} \cdots \rightarrow_{\mathcal{P}} (G_n, sub_n) \rightarrow_{\mathcal{P}} \cdots .$$

Una computación se llama de éxito si es finita y coincide con una refutación LD de $\mathcal{P} \cup [G]$.

Teorema 117 (Respuesta Correcta y SLD-refutación) *Sea $\mathcal{P}$ un programa lógico, sean $H_1, \ldots, H_r$ fórmulas atómicas y sea G la cláusula objetivo dada mediante:*

$$G := \{\neg H_1, \ldots, \neg H_r\}.$$

Sea, finalmente,

$$(G, [\,]) = (G_1, sub_1) \rightarrow_{\mathcal{P}} (G_2, sub_2) \rightarrow_{\mathcal{P}} \cdots \rightarrow_{\mathcal{P}} (\square, sub_n),$$

una computación de éxito. Entonces, sub_n es una sustitución correcta y si $x_1, \ldots, x_t$ son las variables libres entre todas las $H_i sub_n$, se tendrá:

$$\models \mathcal{P} \rightarrow \forall x_1 \cdots \forall x_t H_1 sub_n \wedge \cdots \wedge H_r sub_n.$$

El recíproco también es válido, esto es, toda sustitución de respuesta correcta puede obtenerse de una computación/refutación SLD por el anterior método.

Demostración.– Véanse [Ben Ari, 1993], p.178; [Lloyd, 1987], Section 2.8; o [Schöning, 1989]. ■

4.4.1. Indeterminismo?

Bibliografía

[Ben Ari, 1993] M. Ben–Ari. *Mathematical Logic for Computer Science*. Springer verlag, 1993.

[Lloyd, 1987] J. lloyd. *foundations of logic Programming*. Springer Verlag, 1987.

[Nerode & Shore,1993] A. Nerode, R.A. Shore. *Logic for Applications*.Springer Verlag, 1993.

[Schöning, 1989] U. Schöning. *Logic for Computer Scientist*. Birkhäuser Verlag, 1989.