Dpto. de Álgebra, Computación, Geometría y Topología
Universidad de Sevilla

Lógica computacional

José A. Alonso Jiménez
(jalonso@us.es)

Sevilla, 12 de Diciembre de 1991

Contenido

Capítulo 1

Preliminares

1.1 Cadenas

Definición 1.1.1 Sea Σ un conjunto no vacío.

1. Una **cadena de longitud** n en Σ es una aplicación $u : n \to \Sigma$.
2. La **longitud** de la cadena u se representa por $|u|$.
3. El conjunto de las cadenas de longitud n en Σ se representa por Σ^n.
4. $\Sigma^* = \bigcup_{n\geq 0} \Sigma^n$.
5. Las **cadenas** en Σ son los elementos de Σ^*.
6. La **cadena vacía** es el único elemento de Σ^0 y se representa por Λ.
7. Si $u \in \Sigma^*$ y $0 < i \leq |u|$, entonces $u_i = u(i-1)$.
8. Si $u \in \Sigma^n$, la cadena u se representa por $u_1 \dots u_n$.

Definición 1.1.2 Dadas dos cadenas $u \in \Sigma^m$ y $v \in \Sigma^n$, su **concatenación** (representada por $u.v$ ó uv) es la cadena $w \in \Sigma^{m+n}$ definida por

$$w(i) = \begin{cases} u(i), & \text{si } 0 \leq i < m; \\ v(i-m), & \text{si } m \leq i < m+n. \end{cases}$$

Lema 1.1.3

1. Λ es elemento neutro para la concatenación.

2. La concatenación es asociativa.

3. La concatenación no es conmutativa.

Nota 1.1.4 Para disminuir el número de paréntesis se usará el convenio de asociar por la derecha. Por ejemplo, se escribirá uvw en lugar de $u.(v.w)$.

Definición 1.1.5 Sean $u, v \in \Sigma^*$.

1. Se dice que u es un **prefijo** de v (ó v es un **sufijo** de u), y se representa por $u \leq v$, si existe $w \in \Sigma^*$ tal que $v = uw$.

2. Se dice que u es un **prefijo propio** de v (ó v es un **sufijo propio** de u), y se representa por $u < v$, si $u \leq v$ y $u \neq v$.

1.2 Inducción y recursión

Nota 1.2.1 En esta sección usaremos la siguiente notación:

1. A es un conjunto no vacío.

2. F es un conjunto de operaciones en A; es decir, para cada $f \in F$, existe un $n > 0$ tal que $f : A^n \to A$.

3. α es una aplicación de F en $\mathbb{N}$ tal que para cada $f \in F$, $f : A^{\alpha(f)} \to A$. Se dice que $\alpha(f)$ es la aridad de f.

4. X es un subconjunto no vacío de A.

Definición 1.2.2 Sea $Y \subseteq A$.

1. Y es **cerrado bajo** F si para todo $f \in F$ y todo $y_1, \ldots, y_{\alpha(f)} \in Y$, se tiene que $f(y_1, \ldots, y_{\alpha(f)}) \in Y$.

2. Y es **inductivo sobre** X si $X \subseteq Y$ e Y es cerrado bajo F.

Definición 1.2.3 La **clausura transitiva** de X es

$$X^+ = \bigcap \{Y \subseteq A : Y \text{ es inductivo sobre } X\}.$$

Lema 1.2.4 X^+ es el menor subconjunto inductivo de A que contiene a X y es cerrado bajo F.

Definición 1.2.5

1. La sucesión $(X_i)_{i\geq 0}$ está definida recursivamente por:

$$\begin{array}{ll} X_0 & = X \\ X_{i+1} & = X_i \cup \{f(x_1,\ldots,x_n) : f \in F, n = \alpha(f), (x_1,\ldots,x_n) \in X_i^n\} \end{array}$$

2. El conjunto **engendrado por X mediante F** es

$$X_+ = \bigcup_{i\geq 0} X_i$$

Lema 1.2.6 $X^+ = X_+$

Teorema 1.2.7 (**Principio de inducción para X_+**)
Si $Y \subseteq X_+$ es inductivo sobre X, entonces $Y = X_+$.

Definición 1.2.8 Se dice que X_+ está **libremente generado por X mediante** F si se verifican las siguientes condiciones:

1. Para toda $f \in F$, la restricción de f a $X_+^{\alpha(f)}$ es inyectiva.
2. Para toda $f, g \in F$, si $f \neq g$ entonces $rang(f) \cap rang(g) = \emptyset$.
3. Para toda $f \in F$, $rang(f) \cap X = \emptyset$.

Nota 1.2.9 En lo que sigue, $X_{-1} = \emptyset$.

Lema 1.2.10 Si X_+ está libremente generado por X mediante F, entonces para todo $i \geq 0$, se verifican:

1. Si $f \in F, n = \alpha(f)$ y $(x_1,\ldots,x_n) \in X_i^n - X_{i-1}^n$, entonces $f(x_1,\ldots,x_n) \notin X_i$.
2. $X_{i-1} \neq X_i$.

Teorema 1.2.11 (**de recursión**)
Sea B un conjunto no vacío, G un conjunto de operaciones en B, $\beta : G \to \mathbb{N}$ y $d : F \to G$ tales que

1. para toda $g \in G$, g es una aplicación de $B^{\beta(g)}$ en B.

2. para toda $f \in F$, $\beta(d(f)) = \alpha(f)$.

Si X_+ está generado libremente por X mediante F, entonces para cada aplicación $h : X \to B$ existe una única aplicación $\hat{h} : X_+ \to B$ tal que:

1. Para todo $x \in X$, $\hat{h}(x) = h(x)$.
2. Para todo $f \in F$, $(x_1, \ldots, x_n) \in X_+^n$,

$$\hat{h}(x)(f(x_1, \ldots, x_n)) = g(\hat{h}(x_1), \ldots, \hat{h}(x_n)),$$

donde $n = \alpha(f)$ y $g = d(f)$.

Capítulo 2

Sintaxis de la lógica proposicional

2.1 El lenguaje de la lógica proposicional

Definición 2.1.1 El **alfabeto proposicional** Σ_0 consta de:

1. Un conjunto numerable SP de **símbolos proposicionales**: $p_0, p_1, \ldots$.
2. Las **conectivas lógicas**: $\neg$ (negación) y $\vee$ (disyunción).
3. **Símbolos auxiliares**: "(" y ")".

Definición 2.1.2 El conjunto $PROP$ de las **fórmulas (proposicionales)** es el conjunto engendrado por SP mediante las aplicaciones $C_\neg : \Sigma_0^* \to \Sigma_0^*$ y $C_\vee : \Sigma_0^* \times \Sigma_0^* \to \Sigma_0^*$ definidas por:

$$C_\neg(F) = \neg F$$
$$C_\vee(F, G) = (F \vee G)$$

Lema 2.1.3 (Principio de inducción para fórmulas)
Sea $S \subseteq PROP$ tal que:

1. $SP \subseteq S$;
2. si $F \in S$, entonces $\neg F \in S$;
3. si $F, G \in S$, entonces $(F \vee G) \in S$.

Entonces $S = PROP$.

Ejemplo 2.1.4 El conjunto $PROP$ es el menor subconjunto de Σ_0^* tal que:

1. $SP \subseteq S$
2. Si $F \in S$, entonces $\neg F \in S$.
3. Si $F, G \in S$, entonces $(F \vee G) \in S$.

Ejemplo 2.1.5

1. Las siguientes cadenas son fórmulas:

$$p_1, \quad (p_1 \vee \neg p_0), \quad \neg(p_1 \vee p_1)$$

2. Las siguientes cadenas no son fórmulas:

$$(p_1), \quad p_1 \vee \neg p_0$$

Nota 2.1.6

1. Los símbolos $p, q, r, \ldots$ representarán símbolos proposicionales.
2. Los símbolos $F, G, H, \ldots$ representarán fórmulas.

Nota 2.1.7 Se usarán las siguientes abreviaturas:

1. $(F \wedge G)$ en lugar de $\neg(\neg F \vee \neg G)$.
2. $(F \to G)$ en lugar de $(\neg F \vee G)$.
3. $(F \leftrightarrow G)$ en lugar de $((F \to G) \wedge (G \to F))$.
4. $\left(\bigvee_{i=1}^{n} F_i\right)$ en lugar de $(F_1 \vee (F_2 \vee \ldots \vee (F_{n-1} \vee F_n)\ldots))$.
5. $\left(\bigwedge_{i=1}^{n} F_i\right)$ en lugar de $(F_1 \wedge (F_2 \wedge \ldots \wedge (F_{n-1} \wedge F_n)\ldots))$.

2.2 Libre generación del conjunto de las fórmulas

Lema 2.2.1

1. Todas las fórmulas tienen el mismo número de paréntesis izquierdos que derechos.
2. Si F' es un prefijo propio de una fórmula F, entonces F' es la cadena vacía, una cadena no vacía de símbolos de negación o contiene un exceso de paréntesis izquierdos.
3. Ningún prefijo propio de una fórmula es una fórmula.

Teorema 2.2.2 El conjunto $PROP$ de las fórmulas proposicionales está libremente generado por el conjunto de los símbolos proposicionales y las conectivas lógicas.

Corolario 2.2.3 (recursión sobre fórmulas)
Sea X un conjunto no vacío, $f : X \to X$ y $g : X^2 \to X$. Para cada función $h : SP \to X$, existe una única función $\hat{h} : PROP \to X$ tal que:

1. Para todo $p_i \in SP$, $\hat{h}(p_i) = h(p_i)$.
2. Para toda $F \in PROP$, $\hat{h}(\neg F) = f(\hat{h}(F))$.
3. Para toda $F, G \in PROP$, $\hat{h}((F \vee G)) = g(\hat{h}(F), \hat{h}(G))$.

Definición 2.2.4 El conjunto $Subf(F)$ de las **subfórmulas** de una fórmula F se define recursivamente por:

$$Subf(F) = \begin{cases} \{F\}, & \text{si } F \text{ es una variable proposicional;} \\ \{F\} \cup Subf(G), & \text{si } F = \neg G; \\ \{F\} \cup Subf(G) \cup Subf(H), & \text{si } F = G \vee H; \end{cases}$$

Nota 2.2.5 Para disminuir el número de paréntesis, se introducen los siguientes convenios:

1. Pueden eliminarse los paréntesis externos.
2. Las conectivas tienen una precedencia de asociación. De mayor a menor, están ordenadas por: $\neg, \wedge, \vee, \to, \leftrightarrow$.
3. Cuando una conectiva se usa repetidamente, se asocia por la derecha.

Ejemplo 2.2.6

1. $F \wedge G \rightarrow \neg F \vee G$ es una abreviatura de $((F \wedge G) \rightarrow (\neg F \vee G))$
2. $F \vee G \vee H$ es una abreviatura de $(F \vee (G \vee H))$

Capítulo 3

Semántica de la lógica proposicional

3.1 La semántica de las proposiciones

Definición 3.1.1 Los elementos del conjunto $2 = \{0, 1\}$ se llaman **valores de verdad**. Se dice que 0 es el valor **falso** y el 1 es el valor **verdadero**.

Definición 3.1.2

1. La **función de verdad** de la negación es $H_\neg : 2 \to 2$ definida por:

$$H_\neg(i) = \begin{cases} 1, & \text{si } i = 0; \\ 0, & \text{si } i = 1. \end{cases}$$

2. La **función de verdad** de la disyunción es $H_\vee : 2 \times 2 \to 2$ definida por:

$$H_\vee(i, j) = \begin{cases} 0, & \text{si } i = j = 0; \\ 1, & \text{en otro caso.} \end{cases}$$

Definición 3.1.3 Una **valoración de verdad** es una aplicación $v : SP \to 2$.

Lema 3.1.4 Para cada valoración de verdad v existe una única aplicación $\hat{v} : PROP \to 2$ tal que:

1. Para todo $p_i \in SP$, $\hat{v}(p_i) = v(p_i)$.

2. Para toda $F \in PROP$, $\hat{v}(\neg F) = H_{\neg}(\hat{v}(F))$.

3. Para toda $F, G \in PROP$, $\hat{v}((F \vee G)) = H_{\vee}(\hat{v}(F), \hat{v}(G))$.

Se dice que $\hat{v}(F)$ es el **valor de verdad** de F respecto de v.

Lema 3.1.5 Sea F una fórmula y v, v' dos valoraciones. Si $v(p) = v'(p)$ para todos los símbolos proposicionales de F, entonces $\hat{v}(F) = \hat{v'}(F)$.

Definición 3.1.6

1. La **función de verdad** de la conjnción es $H_{\wedge} : 2 \times 2 \to 2$ definida por:
$$H_{\wedge}(i,j) = \begin{cases} 1, & \text{si } i = j = 1; \\ 0, & \text{en otro caso.} \end{cases}$$

2. La **función de verdad** del condicional es $H_{\to} : 2 \times 2 \to 2$ definida por:
$$H_{\to}(i,j) = \begin{cases} 0, & \text{si } i = 1, j = 0; \\ 1, & \text{en otro caso.} \end{cases}$$

3. La **función de verdad** del bicondicional es $H_{\leftrightarrow} : 2 \times 2 \to 2$ definida por:
$$H_{\leftrightarrow}(i,j) = \begin{cases} 1, & \text{si } i = j; \\ 0, & \text{en otro caso.} \end{cases}$$

Lema 3.1.7 Sea v una valoración de verdad.

1. $\hat{v}((F \wedge G)) = H_{\wedge}(\hat{v}(F), \hat{v}(G))$.

2. $\hat{v}((F \to G)) = H_{\to}(\hat{v}(F), \hat{v}(G))$.

3. $\hat{v}((F \leftrightarrow G)) = H_{\leftrightarrow}(\hat{v}(F), \hat{v}(G))$.

Ejemplo 3.1.8 Sea $F = p \vee q \to p$ y v una valoración de verdad tal que $v(p) = 0$ y $v(q) = 1$. Entonces $\hat{v}(F) = 0$.

3.2 Tautologías

Definición 3.2.1 Sea F una fórmula, Γ un conjunto de fórmulas y v una valoración de verdad.

1. La valoración v **satisface** a F (o es un **modelo** de F), y se representa por $v \models F$, si $\hat{v}(F) = 1$. En caso contrario, se dice que v **falsifica** a F y se representa por $v \not\models F$.

2. La valoración v **satisface** a Γ (o es un **modelo** de Γ), y se representa por $v \models \Gamma$, si $\hat{v}(F) = 1$ para toda $F \in \Gamma$. En caso contrario, se dice que v **falsifica** a Γ y se representa por $v \not\models \Gamma$.

Definición 3.2.2 Sea F una fórmula y Γ un conjunto de fórmulas.

1. F (resp. Γ) es **consistente** (o **satisfacible**) si tiene modelos. En caso contrario, es **inconsistente** (o **insatisfacible**).

2. F es una **tautología**, y se representa por $\models F$, si $\hat{v}(F) = 1$ para todas las valoraciones de verdad.

3. F es **consecuencia taulógica** de Γ, y se representa por $\Gamma \models F$, si $\hat{v}(F) = 1$ para todos los modelos de Γ.

Nota 3.2.3 Si $\Gamma = \{F_1, \ldots, F_n\}$, se escribirá $F_1, \ldots, F_n \models G$ en lugar de $\{F_1, \ldots, F_n\} \models G$.

Ejemplo 3.2.4

1. $p \vee q$ y $\{p, q\}$ son consistentes.

2. $p \wedge \neg p$ y $\{p, p \to q, \neg q\}$ son inconsistentes.

3. $p \to p$ es una tautología y $p \to q$ no lo es.

4. $p, p \to q \models q$.

Nota 3.2.5

1. El **problema de la consistencia** (o de la satisfacibilidad) consiste en dada una fórmula determinar si es consistente.

2. El **problema de la tautología** consiste en dada una fórmula determinar si es una tautología.

Lema 3.2.6 Una fórmula F es una tautología syss $\neg F$ es inconsistente.

Lema 3.2.7 Dadas las fórmulas $F_1, \ldots, F_n, G$, son equivalentes:

1. $F_1, \ldots, F_n \models G$.
2. $\models \bigwedge_{i=1}^{n} F_i \to G$.
3. $\{F_1, \ldots, F_n, \neg G\}$ es inconsistente.

Lema 3.2.8 $\Gamma \models F$ syss $\Gamma \cup \{\neg F\}$ es inconsistente.

3.3 Métodos de verificación de tautologías

Definición 3.3.1 Sea F una fórmula y $p_{i_1}, \ldots, p_{i_n}$ los símbolos proposicionales que ocurren en F con $i_1 < \ldots < i_n$. La **función de verdad** de F es la aplicación $H_F : 2^n \to 2$ definida por:

$$H_F(j_1, \ldots, j_n) = \hat{v}(F),$$

donde v es una valoración de verdad tal que $v(p_{i_k}) = j_k$ para $k \in \{1, \ldots, n\}$.

Definición 3.3.2 La **tabla de verdad** de una fórmula F es el grafo de su función de verdad.

Ejemplo 3.3.3 La tabla de verdad de la fórmula

$$F = (p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_1)$$

es

p_1	p_2	F
0	0	0
0	1	1
1	0	1
1	1	0

Nota 3.3.4 Para facilitar los cálculos, suelen añadirse columnas para subfórmulas de F.

Nota 3.3.5 (Método de las tablas de verdad)

1. Una fórmula F es consistente syss $1 \in rang(H_F)$ (i.e., en la columna de F aparece algún 1).
2. Una fórmula F es una tautología syss $\{1\} = rang(H_F)$ (i.e., en la columna de F sólo aparecen 1).

Ejemplo 3.3.6 Comprobar que $p \wedge (p \rightarrow q) \rightarrow q$ es una tautología.

Nota 3.3.7 (Método de reducción)
Se supone que en una valoración de verdad desconocida, el valor de verdad de F es 0. Entonces, F es una tautología syss de este supuesto se sigue una contradicción.

Los valores supuestos o deducidos para F y sus subfórmulas se colocan debajo de sus conectivas principales.

Nota 3.3.8

1. El método de las tablas de verdad es exponencial, en el sentido de que la verificación de una fórmula con n símbolos proposicionales puede requerir la evaluación de 2^n valoraciones para F.
2. El problema de la consistencia es NP-completo.

Capítulo 4

Completitud funcional y compacidad

4.1 Completitud funcional

Definición 4.1.1 Una **función de verdad** n–aria es una aplicación de 2^n en 2.

Teorema 4.1.2 (de completitud funcional)
Para cada función de verdad n–aria g existe una fórmula F tal que $H_F = g$ y las únicas conectivas de F están en $\{\neg, \vee, \wedge\}$.

Corolario 4.1.3 Existen procedimientos de forma que para cada función de verdad g encuentran una fórmula F tal que $H_F = g$ y las únicas conectivas de F están en $\{\neg, \vee, \wedge\}$.

Ejemplo 4.1.4 Sea g la función de verdad definida por

i_1	i_2	g
0	0	1
0	1	0
1	0	1
1	1	1

Calcular una fórmula F tal que $H_F = g$.

4.2 Compacidad

Definición 4.2.1 Un conjunto de fórmulas Γ es **finitamente consistente** si cualquier subconjunto finito de Γ es consistente.

Lema 4.2.2 Si Γ es finitamente consistente, entonces $\Gamma \cup \{F\}$ ó $\Gamma \cup \{\neg F\}$ es finitamente consistente.

Definición 4.2.3 Un conjunto de fórmulas Γ es **completo** si para cualquier fórmula F se tiene $F \in \Gamma$ ó $\neg F \in \Gamma$.

Lema 4.2.4 Si Γ es un conjunto finitamente consistente, entonces existe un conjunto finitamente consistente y completo que contiene a Γ.

Teorema 4.2.5 (de compacidad)
Si un conjunto de fórmula es finitamente consistente, entonces es consistente.

Corolario 4.2.6 Un conjunto de fórmulas es consistente syss es finitamente consistente.

Corolario 4.2.7 Si $\Gamma \models F$, entonces existe un conjunto finito $\Gamma_0 \subseteq \Gamma$ tal que $\Gamma_0 \models F$.

Capítulo 5

Equivalencia y formas normales

5.1 Equivalencia

Definición 5.1.1 Dos fórmulas F y G son equivalentes, y se representa por $F \equiv G$, si $\hat{v}(F) = \hat{v}(G)$ para toda valoración v.

Lema 5.1.2

1. $F \equiv G$ syss $\models F \leftrightarrow G$.
2. La relación $\equiv$ es de equivalencia en $PROP$.

Lema 5.1.3

1. Si $F \equiv F'$, entonces $\neg F \equiv \neg F'$.
2. Si $F \equiv F'$, $G \equiv G'$ y $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, entonces $F * G \equiv F' * G'$.

Teorema 5.1.4 (de sustitución)
Sea G una subfórmula de F y F' la fórmula obtenida sustituyendo una ocurrencia de G en F por G'. Si $G \equiv G'$, entonces $F \equiv F'$.

Lema 5.1.5 Las siguientes equivalencias se verifican para todas las fórmulas:

1. Idempotencia:
$$F \vee F \equiv F \qquad F \wedge F \equiv F$$
2. Conmutatividad:
$$F \vee G \equiv G \vee F \qquad F \wedge G \equiv G \wedge F$$

3. Asociatividad:

$$F \vee (G \vee H) \equiv (F \vee G) \vee H \qquad F \wedge (G \wedge H) \equiv (F \wedge G) \wedge H$$

4. Absorción:

$$F \wedge (G \vee H) \equiv F \qquad F \vee (G \wedge H) \equiv F$$

5. Distributividad:

$$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H) \qquad F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$$

6. Doble negación:

$$\neg\neg F \equiv F$$

7. Leyes de DE MORGAN:

$$\neg(F \wedge G) \equiv \neg F \vee \neg G \qquad \neg(F \vee G) \equiv \neg F \wedge \neg G$$

8. Leyes de tautologías: Si F es una tautología, entonces

$$F \wedge G \equiv G \qquad F \vee G \equiv F$$

9. Leyes de inconsistentes: Si F es inconsistente, entonces

$$F \wedge G \equiv F \qquad F \vee G \equiv G$$

Ejemplo 5.1.6 Usar las anteriores equivalencias y el teorema de sustitución para demostrar que

$$(F \vee (G \vee H)) \wedge (H \vee \neg F) \equiv (G \wedge \neg F) \vee H.$$

Lema 5.1.7

1. Leyes de DE MORGAN:

$$\neg\left(\bigwedge_{i=1}^{n} F_i\right) \equiv \bigvee_{i=1}^{n} \neg F_i \qquad \neg\left(\bigvee_{i=1}^{n} F_i\right) \equiv \bigwedge_{i=1}^{n} \neg F_i$$

2. Distributividad:

$$\left(\bigvee_{i=1}^{m} F_i\right) \wedge \left(\bigvee_{j=1}^{n} G_j\right) \equiv \bigvee_{i=1}^{m}\left(\bigvee_{j=1}^{n}(F_i \wedge G_j)\right) \qquad \left(\bigwedge_{i=1}^{m} F_i\right) \vee \left(\bigwedge_{j=1}^{n} G_j\right) \equiv \bigwedge_{i=1}^{m}\left(\bigwedge_{j=1}^{n}(F_i \vee G_j)\right)$$

5.2 Formas normales

Definición 5.2.1

1. Un **literal** es un símbolo proposicional o su negación.
2. Un literal es **positivo** si es un símbolo proposicional y **negativo**, en caso contrario.
3. Usaremos la letra L (posiblemente con índices) para representar literales.
4. El **complementario** de un literal L es
$$\overline{L} = \begin{cases} \neg p & \text{si } L = p; \\ p & \text{si } L = \neg p; \end{cases}$$
5. Los literales L y L' son **complementarios** si $L' = \overline{L}$.
6. Una **cláusula conjuntiva** (ó $\wedge$**–cláusula**) C es una conjunción de un conjunto finito de literales; i.e., $C = \bigwedge_{i=1}^{n} L_i$
7. Una **cláusula disjuntiva** (ó $\vee$**–cláusula**) D es una disyunción de un conjunto finito de literales; i.e., $D = \bigvee_{i=1}^{n} L_i$
8. Una fórmula F está en **forma normal conjuntiva (FNC)** si es una conjunción de un conjunto de cláusulas disjuntivas; i.e.,
$$F = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right)$$
9. Una fórmula F está en **forma normal disjuntiva (FND)** si es una disyunción de un conjunto de cláusulas conjuntivas; i.e.,
$$F = \bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{m_i} L_{i,j} \right)$$
10. F es una **forma normal conjuntiva de** G si F está en forma normal conjuntiva y $F \equiv G$.

11. F es una **forma normal disjuntiva de** G si F está en forma normal disjuntiva y $F \equiv G$.

Teorema 5.2.2 Para cada fórmula F existe una fórmula G_1 y una fórmula G_2 tal que G_1 es una forma normal conjuntiva de F y G_2 es una forma normal disyuntiva de F.

Nota 5.2.3 (Algoritmo de normalización)

Entrada: Una fórmula F.

Salida: Una forma normal conjuntiva de F, G.

Procedimiento: Sustituir en F, mientras sea posible, las subfórmulas:

$(G \leftrightarrow H)$	por $((G \rightarrow H) \wedge (H \rightarrow G))$
$(G \rightarrow H)$	por $(\neg G \vee H)$
$\neg\neg G$	por G
$\neg(G \vee H)$	por $(\neg G \wedge \neg H)$
$\neg(G \wedge H)$	por $(\neg G \vee \neg H)$
$(G_1 \vee (G_2 \wedge G_3))$	por $((G_1 \vee G_2) \wedge (G_1 \vee G_3))$
$((G_1 \wedge G_2) \vee G_3))$	por $((G_1 \vee G_3) \wedge (G_2 \vee G_3))$

Nota 5.2.4 Intercambiando el papel de las conectivas $\vee$ y $\wedge$ en el algoritmo anterior se obtiene una forma normal disyuntiva de la fórmula.

Nota 5.2.5 A partir de la demostración del teorema de completitud funcional, puede diseñarse otro algoritmo para el cálculo de las formas normales.

5.3 Formas normales y validez

Lema 5.3.1 Sean $L_1, \ldots, L_n$ literales. Son equivalentes:

1. $\bigvee_{i=1}^{n} L_i$ es una tautología.
2. $\bigwedge_{i=1}^{n} L_i$ es inconsistente
3. $\{L_1, \ldots, L_n\}$ contiene un par de literales complementarios.

Teorema 5.3.2

1. Una fórmula en forma normal conjuntiva es una tautología syss cada una de sus cláusulas disjuntivas es una tautología.

2. Una fórmula en forma normal disjuntiva es inconsistente syss cada una de sus cláusulas conjuntivas es inconsistente.

Algoritmo 5.3.3 (de inconsistencia mediante FND)

Entrada: Una fórmula F.

Salida: `Consistente`, si F es consistente; `Inconsistente`, en caso contrario.

Procedimiento:

Sean G una forma normal disjuntiva de F
    $G_1, \ldots, G_n$ las $\wedge$–cláusulas de G
Desde $i = 1$ hasta n
    si en G_i no ocurren literales complementarios,
      entonces devolver `Consistente` y parar.
Devolver `Inconsistente` y parar.

Algoritmo 5.3.4 (de validez mediante FNC)

Entrada: Una fórmula F

Salida: `Tautologia`, si F es una tautología; `No-tautologia`, en caso contrario.

Procedimiento:

Sean G una forma normal conjuntiva de F
    $G_1, \ldots, G_n$ las $\vee$–cláusulas de G
Desde $i = 1$ hasta n
    si en G_i ocurren literales complementarios,
      entonces devolver `No-tautologia` y parar.
Devolver `Tautologia` y parar.

Capítulo 6

Cláusulas

6.1 Cláusulas

Definición 6.1.1

1. Una **cláusula** es un conjunto de literales.
2. La **cláusula vacía** es el conjunto vacío y se representa por $\square$.

Nota 6.1.2 Usaremos las siguientes variables sintácticas:

1. L para literales.
2. C, D para cláusulas.
3. S para conjuntos de cláusulas.

Definición 6.1.3 Sea C una cláusula.

1. El conjunto de los literales positivos de C es
$$C_+ = C \cap SP$$
2. El conjunto de los literales negativos de C es
$$C_- = C - C_+$$

Definición 6.1.4 Sea v una valoración de verdad.

1. Para cada literal L,

$$v'(L) = \begin{cases} v(p), & \text{si } L = p; \\ 1 - v(p), & \text{si } L = \neg p. \end{cases}$$

2. Para cada cláusula C,

$$v''(C) = 1 \text{ syss existe un } L \in C \text{ tal que } v'(L) = 1$$

3. Para cada conjunto de cláusulas S,

$$v'''(S) = 1 \text{ syss para todo } C \in S,\ v''(C) = 1$$

Lema 6.1.5 Sea v una valoración.

1. $v''(\square) = 0$.
2. $v'''(\emptyset) = 1$.

Nota 6.1.6 En lo que sigue, escribiremos $v(L), v(C)$ ó $v(S)$ en lugar de $v'(L), v''(C)$ ó $v'''(S)$

Definición 6.1.7 Sea v una valoración.

1. v es un **modelo** de C (ó C es **válida** en v), $v \models C$, si $v(C) = 1$.
2. v es un **modelo** de S (ó S es **válida** en v), $v \models S$, si $v(S) = 1$.

Definición 6.1.8

1. C (resp. S) es **consistente** si tiene un modelo. En caso contrario, es **inconsistente**.
2. C es una **tautología**, $\models C$, si $v(C) = 1$ para toda valoración v.
3. C es **consecuencia** de S, $S \models C$, si $v(C) = 1$ para todos los modelos v de S.

Lema 6.1.9 C es una tautología syss contiene un par de literales complementarios.

Nota 6.1.10

1. Si $S \subseteq S'$ y S' es consistente, entonces S es consistente.
2. Si $\square \in S$, entonces S es inconsistente.

6.2 Cláusulas y fórmulas

Definición 6.2.1

1. El conjunto de fórmulas correspondiente a la cláusula $C \neq \square$ es

$$Form(C) = \left\{ \bigvee_{i=1}^{n} L_i : C = \{L_1, \ldots, L_n\} \right\}$$

2. El conjunto de fórmulas correspondiente al conjunto de cláusulas $C \neq \emptyset$ es

$$Form(S) = \left\{ \bigwedge_{i=1}^{n} F_i : S = \{C_1, \ldots, C_n\}, F_i \in Form(C_i) \right\}$$

Lema 6.2.2

1. Si $F, G \in Form(C)$, entonces $F \equiv G$.
2. Si $F, G \in Form(S)$, entonces $F \equiv G$.

Definición 6.2.3

1. C y F son **equivalentes**, $C \equiv F$, si $v''(C) = \hat{v}(F)$ para toda valoración v.
2. S y F son **equivalentes**, $S \equiv F$, si $v'''(S) = \hat{v}(F)$ para toda valoración v.
3. S y S' son **equivalentes**, $S \equiv S'$, si $v'''(S) = v'''(S')$ para toda valoración v.

Lema 6.2.4

1. Si $F \in Form(C)$, entonces $C \equiv F$.
2. Si $F \in Form(S)$, entonces $S \equiv F$.

Teorema 6.2.5 Para cada fórmula F existe un conjunto de cláusulas S tal que $S \equiv F$.

Definición 6.2.6 Un conjunto de cláusulas S es una **forma clausal** de la fórmula F si $S \equiv F$.

Nota 6.2.7 Existe un algoritmo que para cada fórmula F devuelve una forma clausal de F.

Teorema 6.2.8 (de compacidad para conjuntos de cláusulas)
Un conjunto de cláusulas S es inconsistente syss existe $S' \subseteq S$ finito e inconsistente.

Capítulo 7

El algoritmo de Davis–Putnam

7.1 Las reglas de Davis–Putnam

Teorema 7.1.1 (Regla de tautología)
Si $C \in S$ es una tautología, entoces S es consistente syss $S - \{C\}$ es consistente.

Definición 7.1.2

1. $S_{L^+} = \{C \in S : L \in C\}$
2. $S_{L^-} = \{C \in S : \overline{L} \in C\}$
3. $S_{L^0} = \{C \in S : L \notin C, \overline{L} \notin C\}$.
4. $S_{L=0} = S_{L^0} \cup \{C - \{L\} : C \in S_{L^+}\}$
5. $S_{L=1} = S_{L^0} \cup \left\{C - \{\overline{L}\} : C \in S_{L^-}\right\}$

Nota 7.1.3

1. $S = S_{L^+} \cup S_{L^-} \cup S_{L^0}$
2. $S_{L^-} = S_{\overline{L}^+}$
3. $S_{L=1} = S_{\overline{L}=0}$

Lema 7.1.4 Sea v una valoración.

1. Si $v(L) = 0$, entonces $v(S) = v\left(S_{L=0}\right)$

2. Si $v(L) = 1$, entonces $v(S) = v\left(S_{L=1}\right)$

Teorema 7.1.5 (Regla de bifurcación)
S es consistente syss $S_{L=0}$ ó $S_{L=1}$ es consistente.

Definición 7.1.6 L es un **literal puro** de S si $L \in \bigcup S$ y $\overline{L} \notin \bigcup S$.

Teorema 7.1.7 (Regla de los literales puros)
Si L es un literal puro de S, entonces S es consistente syss S_{L^0} es consistente.

Definición 7.1.8 Una **cláusula unitaria** es una cláusula con un elemento.

Teorema 7.1.9 (Regla de las cláusulas unitarias)
Si $\{L\} \in S$, entonces S es consistente syss $S_{L=1}$ es consistente.

7.2 El algoritmo de Davis–Putnam

Algoritmo 7.2.1 (de Davis–Putnam)

Entrada: Un conjunto finito de cláusulas, S.

Salida: `Consistente`, si S es consistente; `Inconsistente`, en caso contrario.

Procedimiento:

Hacer $T := \emptyset$
$S := S - \{C : C$ es una tautologia$\}$
mientras que $S \neq \emptyset$ **y** $(\square \notin S$ ó $T \neq \emptyset)$
si $\square \in S$
entonces $S := \mathrm{car}(T)$
$T := \mathrm{cdr}(T)$
e.o.c. si existe un literal puro L en S
entonces $S := S_{L^0}$
e.o.c. si existe una cláusula unitaria $\{L\} \in S$
entonces $S := S_{L=1}$
e.o.c $S := S_{L=0}$
$T := \mathrm{cons}(S_{L=1}, T)$
fin de mientras que
si $S = \emptyset$
entonces devolver `Consistente`
e.o.c devolver `Inconsistente`

Ejemplo 7.2.2 Utilizar el algoritmo anterior para comprobar que:

1. $\{\{\neg p, q\}, \{p\}, \{\neg q\}\}$ es consistente.
2. $\{\{p, q, \neg r, s\}, \{\neg q, \neg p, \neg r, s\}, \{\neg q, \neg p, \neg r\}\}$ es consistente.
3. $\{\{\neg q, p\}, \{r, p\}, \{\neg p, \neg q\}, \{\neg p, s\}, \{q, \neg r\}, \{q, \neg s\}\}$ es inconsistente.

Teorema 7.2.3 El algoritmo anterior es correcto.

7.3 Inconsistencia minimal y subsunción

Definición 7.3.1 La cláusula C **subsume** a la cláusula D si $C \subset D$.

Teorema 7.3.2 (Regla de subsunción)
Sean $C, D \in S$. Si C subsume a D, entonces S es consistente syss $S - \{D\}$ es consistente.

Nota 7.3.3 El algoritmo de Davis–Putnam puede modificarse para utilizar la regla de subsunción.

Definición 7.3.4 Un conjunto finito de cláusulas S es **inconsistente minimal** si S es inconsistente y todo $S' \subset S$ es consistente.

Lema 7.3.5 Si S es inconsistente, entonces existe un $S' \subseteq S$ inconsistente minimal.

Lema 7.3.6 Si S es inconsistente minimal, entonces

1. no existen $C, D \in S$ tales que $C \subset D$.
2. S no contiene literales puros.

Capítulo 8

Resolución proposicional

8.1 Sistema de resolución

Definición 8.1.1 *Sean C, D dos cláusulas y $L \in C_1$ tal que $\overline{L} \in C_2$, la* **resolvente** *de C y D respecto de L es*

$$res_L(C, D) = (C - \{L\}) \cup (D - \{\overline{L}\})$$

El **conjunto de resolventes** *de C_1 y C_2 es*

$$Res(C_1, C_2) = \{res_L(C_1, C_2) : L \in C_1, \overline{L} \in C_2\}$$

Definición 8.1.2 *Sea S un conjunto de cláusulas.*

1. *La sucesión $(C_1, \ldots, C_n)$ es una* **deducción por resolución** *a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:*
   (a) *$C_i \in S$.*
   (b) *existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$.*
2. *La cláusula C es* **deducible por resolución** *a partir de S, $S \vdash C$, si existe una deducción por resolución a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.*
3. *La sucesión $(C_1, \ldots, C_n)$ es una* **refutación por resolución** *de S si es una deducción por resolución a partir de S y $C_n = \square$.*
4. *S es* **refutable** *si $S \vdash \square$.*

Nota 8.1.3 *La refutabilidad de un conjunto de cláusulas puede ilustrarse mediante grafos de refutación.*

Definición 8.1.4 *Sean C, D dos cláusulas. El* **conjunto de resolventes** *de C y D es*

$$Res(C, D) = \{res_L(C, D) : L \in C, \overline{L} \in D\}$$

Definición 8.1.5 *Sea S un conjunto de cláusulas.*

1. $Res(S) = S \cup (\bigcup\{Res(C_1, C_2) : C_1, C_2 \in S\})$.
2. *La sucesión $(Res^n(S))_{n\geq 0}$ está definida por:*

$$\begin{aligned} Res^0(S) &= S \\ Res^{n+1} &= Res(Res^n(S)) \end{aligned}$$

3. $Res^*(S) = \bigcup_{n\geq 0} Res^n(S)$

Lema 8.1.6 *$S \vdash C$ syss $C \in Res^*(S)$.*

8.2 Correción y completitud de la resolución

Lema 8.2.1 (de resolución)
Si R es una resolvente de C_1 y C_2, entonces los conjuntos $\{C_1, C_2\}$ y $\{C_1, C_2, R\}$ son equivalentes.

Corolario 8.2.2 *$S \equiv Res^*(S)$.*

Teorema 8.2.3 (de corrección)
Si $S \vdash \square$, entonces S es inconsistente.

Teorema 8.2.4 (de completitud)
Si S es inconsistente, entonces $S \vdash \square$.

Corolario 8.2.5 *S es inconsistente syss $S \vdash \square$.*

8.3 Estrategias de resolución

Algoritmo 8.3.1 (de resolución por saturación)

Entrada: Un conjunto finito de cláusulas, S.

Salida: `Consistente`*, si S es consistente;* `Inconsistente`*, en caso contrario.*

Procedimiento:

Hacer $S' := \emptyset$
mientras que $(\square \notin S)$ **y** $(S \neq S')$
  hacer $S' := S$
    $S := Res(S)$
si $(\square \in S)$
  entonces devolver `Inconsistente`
  e.o.c devolver `Consistente`

Teorema 8.3.2 *El algoritmo anterior es correcto.*

Ejemplo 8.3.3 *Aplicar el algoritmo anterior al conjunto*

$$S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$$

Definición 8.3.4 *El* **simplificado** *de un conjunto finito de cláusulas S es el conjunto obtenido de S suprimiendo las tautologías y las cláusulas subsumidas por otras; es decir,*

$Simp(S) = S - \{C \in S : (C$ *es una tautología) ó (existe* $D \in S$ *tal que* $D \subset C)\}$

Algoritmo 8.3.5 (de resolución por saturación con simplificación)

Entrada: Un conjunto finito de cláusulas, S.

Salida: `Consistente`*, si S es consistente;* `Inconsistente`*, en caso contrario.*

Procedimiento:

Hacer $S' := \emptyset$
mientras que $(\square \notin S)$ **y** $(S \neq S')$
  hacer $S' := S$
    $S := Simp(Res(S))$
si $(\square \in S)$
  entonces devolver `Inconsistente`
  e.o.c devolver `Consistente`

Teorema 8.3.6 *El algoritmo anterior es correcto.*

Capítulo 9

Refinamientos de resolución

9.1 Resolución semántica

Definición 9.1.1 Una **interpretación** I es un conjunto de símbolos proposicionales.

Definición 9.1.2 La **valoración correspondiente** a la interpretación I está definida por

$$v_I(p) = \begin{cases} 1, & \text{si } p \in I; \\ 0, & \text{si } p \notin I; \end{cases}$$

Definición 9.1.3 Sea I una interpretación.

1. I es un **modelo** de una cláusula C, $I \models C$, si $v_I(C) = 1$.
2. I es un **modelo** de un conjunto de cláusulas S, $I \models S$, si $v_I(S) = 1$.

Definición 9.1.4 Sea S un conjunto de cláusulas e I una interpretación.

1. El conjunto de las cláusulas de S verdaderas en I es

$$S_T(I) = \{C \in S : I \models C\}.$$

2. El conjunto de las cláusulas de S falsas en I es

$$S_F(I) = S - S_T(I).$$

Definición 9.1.5 Sea S un conjunto de cláusulas e I una interpretación.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por I–resolución** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

   (a) $C_i \in S$.

   (b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k pertenece a $S_T(I)$ y la otra a $S_F(I)$.

2. La cláusula C es **deducible por I–resolución** a partir de S, $S \vdash_I C$, si existe una deducción por I–resolución a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Definición 9.1.6 Sea S un conjunto de cláusulas e I una interpretación.

1. El conjunto de las resolventes de S respecto de I es

$$Res_I(S) = S \cup \left(\bigcup \{ Res(C_1, C_2) : C_1 \in S_T(I), C_2 \in S_F(I) \} \right)$$

2. La sucesión $(Res_I^n(S))_{n \geq 0}$ está definida por:

$$\begin{aligned} Res_I^0(S) &= S \\ Res_I^{n+1} &= Res_I(Res_I^n(S)) \end{aligned}$$

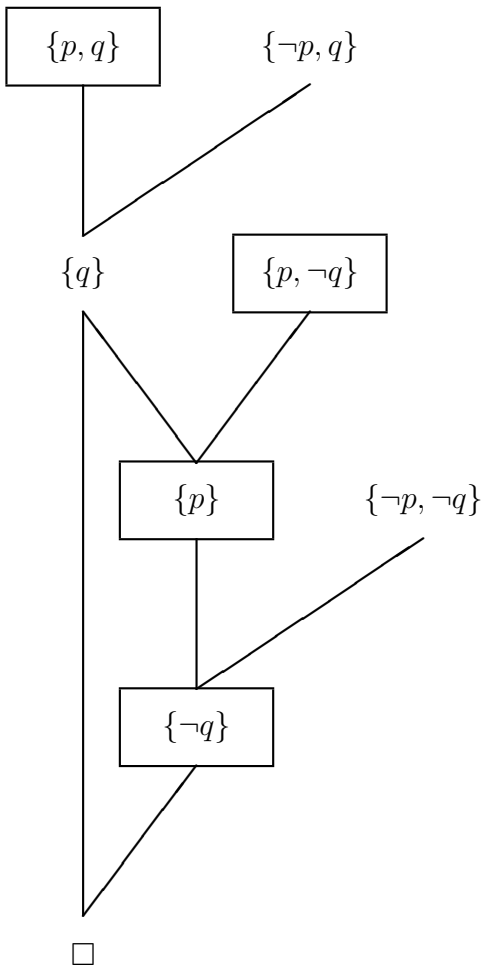
3. $Res_I^*(S) = \bigcup_{n \geq 0} Res_I^n(S)$

Lema 9.1.7 $S \vdash_I C$ syss $C \in Res_I^*(S)$.

Ejemplo 9.1.8 Sea $S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$ e $I = \{p\}$. Entonces

$$(\{p, q\}, \{q\}, \{p\}, \square)$$

es una I–resolución a partir de S. Puede ilustrarse mediante el siguiente diagrama en el que se han encuadrado los elementos de $S_T(I)$.



Teorema 9.1.9 (de corrección y completitud de la resolución semántica) Sea I una interpretación. Un conjunto de cláusulas S es inconsistente syss $S \vdash_I \square$.

9.2 Resolución $\mathbf{P_1}$ y $\mathbf{N_1}$

Definición 9.2.1 Una cláusula es **positiva** si todos sus literales son positivos.

Definición 9.2.2 Sea S un conjunto de cláusulas.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por P_1–resolución** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

(a) $C_i \in S$.

(b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k es positiva.

2. La cláusula C es **deducible por P_1–resolución** a partir de S, $S \vdash_P C$, si existe una deducción por P_1–resolución a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Definición 9.2.3 Sea S un conjunto de cláusulas.

1. $S_P = \{C : C \text{ es positiva}\}$
2. El conjunto de las P_1 resolventes de S es

$$Res_P(S) = S \cup \left(\bigcup \{Res(C_1, C_2) : C_1 \in S_P \text{ ó } C_2 \in S_P\}\right)$$

3. La sucesión $(Res_P^n(S))_{n\geq 0}$ está definida por:

$$\begin{aligned} Res_P^0(S) &= S \\ Res_P^{n+1} &= Res_P(Res_P^n(S)) \end{aligned}$$

4. $Res_P^*(S) = \bigcup_{n\geq 0} Res_P^n(S)$

Lema 9.2.4 $S \vdash_P C$ syss $C \in Res_P^*(S)$.

Teorema 9.2.5 (de corrección y completitud de la P_1–resolución) Un conjunto de cláusulas S es inconsistente syss $S \vdash_P \square$.

Definición 9.2.6 Una cláusula es **negativa** si todos sus literales son negativos.

Definición 9.2.7 Sea S un conjunto de cláusulas.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por N_1–resolución** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

(a) $C_i \in S$.

(b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k es negativa.

2. La cláusula C es **deducible por N_1–resolución** a partir de S, $S \vdash_N C$, si existe una deducción por N_1–resolución a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Definición 9.2.8 Sea S un conjunto de cláusulas.

1. $S_N = \{C : C \text{ es negativa}\}$
2. El conjunto de las N_1 resolventes de S es

$$Res_N(S) = S \cup \left(\bigcup \{Res(C_1, C_2) : C_1 \in S_N \text{ ó } C_2 \in S_N\}\right)$$

3. La sucesión $(Res_N^n(S))_{n\geq 0}$ está definida por:

$$\begin{aligned} Res_N^0(S) &= S \\ Res_N^{n+1} &= Res_N(Res_N^n(S)) \end{aligned}$$

4. $Res_N^*(S) = \bigcup_{n\geq 0} Res_N^n(S)$

Lema 9.2.9 $S \vdash_N C$ syss $C \in Res_N^*(S)$.

Teorema 9.2.10 (de corrección y completitud de la N_1–resolución) Un conjunto de cláusulas S es inconsistente syss $S \vdash_N \square$.

9.3 Resolución lineal

Definición 9.3.1 Sea S un conjunto de cláusulas.

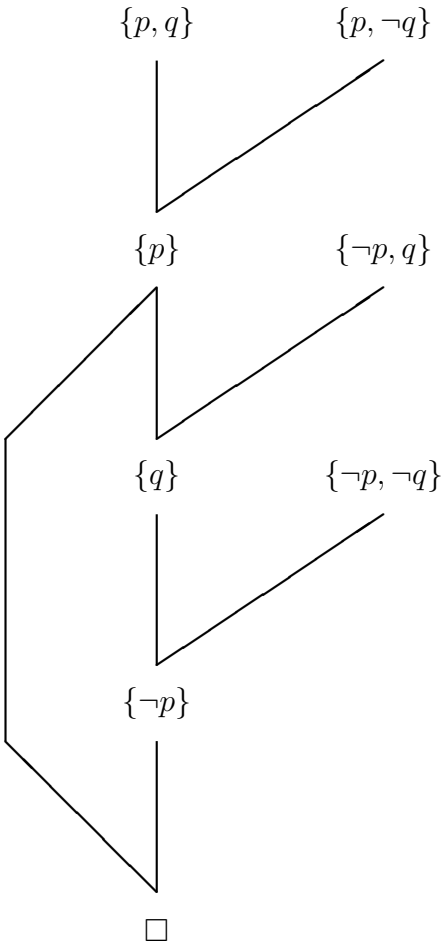
1. La sucesión $(C_0, C_1, \ldots, C_n)$ es una **resolución lineal** a partir de S si se cumplen las siguientes condiciones:
   (a) $C_0 \in S$;
   (b) para todo $i \in \{1, \ldots, n\}$, existe un $B \in S \cup \{C_0, \ldots, C_{i-1}\}$ tal que $C_i \in Res(C_{i-1}, B)$.

   La cláusula C_0 se llama **cláusula base**, las C_i se llaman **cláusulas centrales** y las B se llaman **cláusulas laterales**.
2. La cláusula C es **deducible por resolución lineal** a partir de S, $S \vdash_L C$, si existe una deducción por resolución lineal a partir de S, $(C_0, \ldots, C_n)$, tal que $C_n = C$.

Ejemplo 9.3.2 Sea $S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$. Entonces

$$(\{p, q\}, \{p\}, \{q\}, \square)$$

es una resolución lineal a partir de S. Puede ilustrarse mediante el siguiente diagrama.



Teorema 9.3.3 (de corrección y completitud de la resolución lineal)
Un conjunto de cláusulas S es inconsistente syss $S \vdash_L \square$.

9.4 Resolución con soporte

Definición 9.4.1 Sea S un conjunto de cláusulas y $T \subseteq S$ tal que $S - T$ es consistente.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por resolución con soporte** T a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:
   (a) $C_i \in S$.
   (b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k no pertenece a $S - T$.
2. La cláusula C es **deducible por resolución con soporte** T a partir de S, $S \vdash_T C$, si existe una deducción por resolución con soporte T a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Definición 9.4.2 Sea S un conjunto de cláusulas y $T \subseteq S$ tal que $S - T$ es consistente,

1. El conjunto de las resolventes de S con soporte en T es
$$Res_T(S) = S \cup \left(\bigcup \{ Res(C_1, C_2) : C_1 \notin S - T \text{ ó } C_2 \notin S - T \} \right)$$
2. La sucesión $(Res_T^n(S))_{n \geq 0}$ está definida por:
$$Res_T^0(S) = S$$
$$Res_T^{n+1} = Res_T(Res_T^n(S))$$
3. $Res_T^*(S) = \bigcup_{n \geq 0} Res_T^n(S)$

Lema 9.4.3 $S \vdash_T C$ syss $C \in Res_T^*(S)$.

Teorema 9.4.4 (de corrección y completitud de la resolución con soporte)
Sea S un conjunto de cláusulas y $T \subseteq S$ tal que $S - T$ es consistente. Entonces S es inconsistente syss $S \vdash_T \square$.

9.5 Resolución unidad y por entradas

Definición 9.5.1 Sea S un conjunto de cláusulas.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por resolución unidad** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

(a) $C_i \in S$.

(b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k es una cláusula unitaria.

2. La cláusula C es **deducible por resolución unitaria** a partir de S, $S \vdash_U C$, si existe una deducción por resolución unidad a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Lema 9.5.2 (Corrección de la resolución unidad)
Si $S \vdash_U \square$, entonces S es inconsistente.

Definición 9.5.3 Sea S un conjunto de cláusulas.

1. El conjunto de las resolventes unidad de S es

$$Res_U(S) = S \cup \Big(\bigcup \{ Res(C_1, C_2) : C_1 \text{ ó } C_2 \text{ es una cláusula unitaria} \} \Big)$$

2. La sucesión $(Res_U^n(S))_{n \geq 0}$ está definida por:

$$\begin{aligned} Res_U^0(S) &= S \\ Res_U^{n+1} &= Res_U(Res_U^n(S)) \end{aligned}$$

3. $Res_U^*(S) = \bigcup_{n \geq 0} Res_U^n(S)$

Lema 9.5.4 $S \vdash_U C$ syss $C \in Res_U^*(S)$.

Ejemplo 9.5.5 Si $S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$, entonces $Res_U^*(S) = S$.

Nota 9.5.6 (Incompletitud de la resolución unidad)
Existen conjuntos inconsistentes S para los cuales $S \nvdash_U \square$.

Definición 9.5.7 Sea S un conjunto de cláusulas.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por resolución por entradas** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

(a) $C_i \in S$.

(b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$ y una de las cláusulas C_j, C_k pertenece a S.

2. La cláusula C es **deducible por resolución por entradas** a partir de S, $S \vdash_E C$, si existe una deducción por resolución por entradas a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

Lema 9.5.8 (Corrección de la resolución por entradas)
Si $S \vdash_E \square$, entonces S es inconsistente.

Definición 9.5.9 Sea S un conjunto de cláusulas.

1. El conjunto de las resolventes por entradas de S es

$$Res_E(S) = S \cup \left(\bigcup \{ Res(C_1, C_2) : C_1 \in S \text{ ó } C_2 \in S \} \right)$$

2. La sucesión $(Res_E^n(S))_{n \geq 0}$ está definida por:

$$Res_E^0(S) = S$$
$$Res_E^{n+1} = Res_E(Res_E^n(S))$$

3. $Res_E^*(S) = \bigcup_{n \geq 0} Res_E^n(S)$

Lema 9.5.10 $S \vdash_E C$ syss $C \in Res_E^*(S)$.

Ejemplo 9.5.11 Si $S = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$, entonces $Res_E^*(S) = S \cup \{\{p\}, \{q\}, \{\neg p\}, \{\neg q\}\}$.

Nota 9.5.12 (Incompletitud de la resolución por entrada)
Existen conjuntos inconsistentes S para los cuales $S \nvdash_E \square$.

Teorema 9.5.13 $S \vdash_U \square$ syss $S \vdash_E \square$.

Capítulo 10

Cláusulas de Horn

10.1 Cláusulas de Horn

Definición 10.1.1 Una cláusula C es una **cláusula de** HORN si contiene como máximo un literal positivo.

Definición 10.1.2

1. Las cláusulas de Horn se clasifican en **negativas** (si tiene todos sus literales negativos) y **definidas** (si tiene un literal positivo).
2. Las cláusulas definidas se clasifican en **hechos** (si son unitarias) y **reglas**.

Lema 10.1.3

1. $\{\neg p_1, \ldots, \neg p_n, q\} \equiv \bigwedge_{i=1}^{n} p_i \rightarrow q$
2. $\{\neg p_1, \ldots, \neg p_n\} \equiv \neg \left(\bigwedge_{i=1}^{n} p_i \right)$

Lema 10.1.4 Sea S un conjunto de cláusulas de Horn tal que $\square \notin S$.

1. Si S no contiene cláusulas negativas, entonces S es consistente.
2. Si S no contiene hechos, entonces S es consistente.

10.2 Resolución y cláusulas de Horn

Teorema 10.2.1 (completitud de la resolución unidad para cláusulas de Horn)
Sea S un conjunto de cláusulas de Horn. Si S es inconsistente, entonces $S \vdash_U \square$.

Corolario 10.2.2 (completitud de la resolución por entradas para cláusulas de Horn)
Sea S un conjunto de cláusulas de Horn. Si S es inconsistente, entonces $S \vdash_E \square$.

Algoritmo 10.2.3

Entrada: Un conjunto finito de cláusulas de Horn, S.

Salida: `Consistente`, si S es consistente; `Inconsistente`, en caso contrario.

Procedimiento:

Hacer $S_1 := \{C \in S : C \text{ es un hecho}\}$
$S_2 := S - S_1$
mientras que $S_1 \neq \emptyset$ **y** $(\square \notin S_2)$
hacer $C_1 := car(S_1)$
$L \in C_1$
si $\overline{L} \in \bigcup S_2$
entonces $S_1 := S_1 - \{C_1\}$
e.o.c. $C_2 := car(\{C \in S_2 : \overline{L} \in C\})$
$S_2 := S_2 - \{C_2\}$
$C_3 := res(C_1, C_2)$
si C_3 es un hecho
entonces $S_1 := S_1 \cup \{C_3\}$
e.o.c. $S_2 := S_2 \cup \{C_3\}$
si $S = \emptyset$
entonces devolver `Consistente`
e.o.c devolver `Inconsistente`

Teorema 10.2.4 El algoritmo anterior es correcto y su complejidad es de orden n^2, donde $n = \sum_{C \in S} |C|$.

10.3 Resolución SLD

Definición 10.3.1 Sea S un conjunto de cláusulas de Horn.

1. La sucesión $(C_0, C_1, \ldots, C_n)$ es una **resolución SLD** a partir de S si se cumplen las siguientes condiciones:

   (a) $C_0 \in S$ es una cláusula negativa;

   (b) para todo $i \in \{1, \ldots, n\}$, existe una cláusula no negativa $B \in S$ tal que $C_i \in Res(C_{i-1}, B)$.

   La cláusula C_0 se llama **cláusula base**, las C_i se llaman **cláusulas centrales** y las B se llaman **cláusulas laterales**.

2. La cláusula C es **deducible por resolución SLD** a partir de S, $S \vdash_{SLD} C$, si existe una deducción por resolución SLD a partir de S, $(C_0, \ldots, C_n)$, tal que $C_n = C$.

Nota 10.3.2 El nombre de SLD viene de "Linear resolution whit Selection function for Definite clauses".

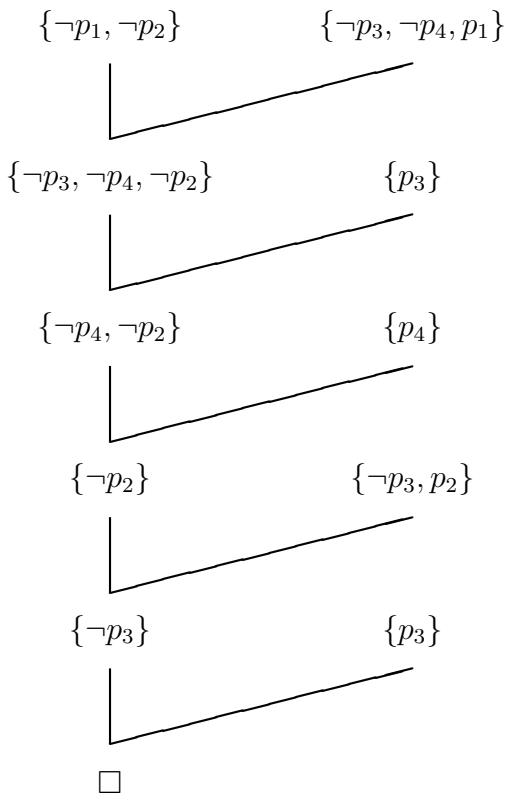
Ejemplo 10.3.3 Sea

$$S = \{\{p_3\}, \{p_4\}, \{\neg p_1, \neg p_2\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, p_2\}, \{\neg p_1, \neg p_2\}\}.$$

Entonces

$$(\{\neg p_1, \neg p_2\}, \{\neg p_3, \neg p_4, \neg p_2\}, \{\neg p_4, \neg p_2\}, \{\neg p_2\}, \{\neg p_3\}, \square\})$$

es una resolución SLD partir de S. Puede ilustrarse mediante el siguiente diagrama.



Teorema 10.3.4 (de corrección y completitud de la resolución SLD) Un conjunto de cláusulas de Horn S es inconsistente syss $S \vdash_{SLD} \square$.

Capítulo 11

Sintaxis de la lógica de primer orden

11.1 El lenguaje de la lógica de primer orden

Definición 11.1.1 El **alfabeto** de un lenguaje de primer orden **L** consta de:

1. Un conjunto numerable de **variables**: $V = \{x_0, x_1, \ldots\}$.
2. Las **conectivas lógicas**: $\neg$ (negación) y $\vee$ (disyunción).
3. El **cuantificador existencial**: $\exists$ (existe).
4. **Símbolos auxiliares**: "(" y ")".
5. Un conjunto finito o numerable (posiblemente vacío) de **símbolos de función** $SF = \{f_0, f_1, \ldots\}$ y una función **rango** ó **aridad**, r, que asigna a cada $f \in SF$ un entero no negativo, $r(f)$, llamado el rango del símbolo de función f.
6. Un conjunto finito o numerable (posiblemente vacío) de **símbolos de predicados** $SP = \{p_0, p_1, \ldots\}$ y una función **rango** ó **aridad**, r, que asigna a cada $p \in SP$ un entero no negativo, $r(p)$, llamado el rango del símbolo de predicado p.

Los conjuntos V, SF y SP son disjuntos.

Nota 11.1.2

1. Los símbolos de función de aridad 0 se llaman **constantes**.

2. El conjunto de las constantes se representa por SC.

3. Los símbolos de predicado de aridad 0 son símbolos proposicionales.

Definición 11.1.3 Sea **L** un lenguaje de primer orden y $\Gamma = V \cup SF$. Para cada $f \in SF - SC$ de rango n se define la aplicación $C_f : (\Gamma^*)^n \to \Gamma^*$ por:

$$C_f(t_1, \ldots, t_n) = ft_1 \ldots f_n$$

El conjunto $TERM_{\mathbf{L}}$ de los **términos** de **L** es el conjunto engendrado por $V \cup SC$ mediante las aplicaciones C_f.

Nota 11.1.4 Más informalmente, los términos de **L** se definen por:

1. Las variables y las constantes de **L** son términos de **L**.

2. Si $t_1, \ldots, t_n$ son términos de **L** y f es un símbolo de función de **L** de aridad n, entonces $ft_1 \ldots t_n$ es un término de **L**.

Lema 11.1.5 (Principio de inducción para términos)
Sea $S \subseteq TERM$ tal que:

1. $V \cup SC \subseteq S$;

2. si f es un símbolo de función de aridad $n > 0$ y $(t_1, \ldots, t_n) \in S^n$, entonces $ft_1 \ldots t_n \in S$.

Entonces $S = TERM$.

Definición 11.1.6 Las **fórmulas atómicas** de un lenguaje de primer orden **L** son las expresiones de la forma $pt_1 \ldots t_n$, donde p es un símbolo de predicado de aridad n y $t_1, \ldots, t_n$ son términos.

Definición 11.1.7 Sea **L** un lenguaje de primer orden y

$$\Sigma = V \cup SF \cup SP \cup \{\neg, \vee, \exists, (,)\}$$

Se definen las aplicaciones

$$C_\neg : \Sigma^* \to \Sigma^*$$
$$C_\vee : \Sigma^* \times \Sigma^* \to \Sigma^*$$
$$E_i : \Sigma^* \to \Sigma^*$$

por:

$$C_\neg(F) = \neg F$$
$$C_\vee(F, G) = (F \vee G)$$
$$E_i(F) = \exists x_i F$$

El conjunto $FORM_{\mathbf{L}}$ de las **fórmulas** de $\mathbf{L}$ es el conjunto engendrado por el conjunto de las fórmulas atómicas de $\mathbf{L}$ mediante las aplicaciones $C_\neg, C_\vee, E_i$.

Nota 11.1.8 Más informalmente, las fórmulas de $\mathbf{L}$ se definen por:

1. Las fórmulas atómicas de $\mathbf{L}$ son fórmulas de $\mathbf{L}$.
2. Si F y G son fórmulas de $\mathbf{L}$, entonces $\neg F$ y $(F \vee G)$ también lo son.
3. Si x_i es una variable y F es una fórmula de $\mathbf{L}$, entonces $\exists x_i F$ también lo es.

Lema 11.1.9 (Principio de inducción para fórmulas)
Sea $S \subseteq FORM$ tal que:

1. si F es una fórmula atómica, entonces $F \in S$;
2. si $F \in S$, entonces $\neg F \in S$;
3. si $F, G \in S$, entonces $(F \vee G) \in S$;
4. si $F \in S$, entonces $\exists x_i F \in S$.

Entonces $S = FORM$.

Nota 11.1.10

1. Los símbolos $x, y, z, \ldots$ representarán variables.
2. Los símbolos $a, b, c, \ldots$ representarán constantes.
3. Los símbolos $f, g, h, \ldots$ representarán símbolos de funciones.
4. Los símbolos $p, q, r, \ldots$ representarán símbolos de predicados.
5. Los símbolos $t_1, t_2, \ldots$ representarán términos.
6. Los símbolos $F, G, H, \ldots$ representarán fórmulas.

Nota 11.1.11 Se usarán las siguientes abreviaturas:

1. $(F \wedge G)$ en lugar de $\neg(\neg F \vee \neg G)$.
2. $(F \to G)$ en lugar de $(\neg F \vee G)$.
3. $(F \leftrightarrow G)$ en lugar de $((F \to G) \wedge (G \to F))$.
4. $\left(\bigvee_{i=1}^{n} F_i\right)$ en lugar de $(F_1 \vee (F_2 \vee \ldots \vee (F_{n-1} \vee F_n)\ldots))$.
5. $\left(\bigwedge_{i=1}^{n} F_i\right)$ en lugar de $(F_1 \wedge (F_2 \wedge \ldots \wedge (F_{n-1} \wedge F_n)\ldots))$.
6. $\forall x_i F$ en lugar de $\neg\exists x_i \neg F$.

11.2 Libre generación de términos y fórmulas

Lema 11.2.1 Sea $\Gamma = V \cup SF$ y $K : \Gamma^* \to \mathbb{N}$ definida por:

$$K(w) = \begin{cases} 1, & \text{si } w \text{ es una variable;} \\ 1-n, & \text{si } w \text{ es un símbolo de función de aridad } n; \\ K(w_1) + \ldots + K(w_n), & \text{si } w = w_1 \ldots w_n \in \Gamma^* - \Gamma \end{cases}$$

Entonces $K(t) = 1$ para todo término t.

Lema 11.2.2 Si t' es un sufijo propio no vacío de un término t, entonces t' es una concatenación de uno o más términos.

Lema 11.2.3 Ningún prefijo propio de un término es un término.

Teorema 11.2.4 El conjunto $TERM$ de los términos está libremente generado por las variables, las constantes y las aplicaciones C_f.

Lema 11.2.5 Sea **L** un lenguaje de primer orden,

$$\Sigma = V \cup SF \cup SP \cup \{\neg, \vee, \exists, (,)\}$$

y $K : \Sigma^* \to \mathbb{N}$ definida por:

$$K(w) = \begin{cases} 1, & \text{si } w \text{ es una variable;} \\ 0, & \text{si } w = \neg \\ -1, & \text{si } w = \vee \\ -1, & \text{si } w = \exists \\ -1, & \text{si } w = (\\ 1, & \text{si } w =) \\ 1 - n, & \text{si } w \text{ es un símbolo de función de aridad } n; \\ 1 - n, & \text{si } w \text{ es un símbolo de predicado de aridad } n; \\ K(w_1) + \ldots + K(w_n), & \text{si } w = w_1 \ldots w_n \in \Sigma^* - \Sigma \end{cases}$$

1. Si F es una fórmula, entonces $K(F) = 1$.
2. Si w es un prefijo propio de una fórmula, entonces $K(w) \leq 0$.

Lema 11.2.6 Ningún prefijo propio de una fórmula es una fórmula.

Teorema 11.2.7 El conjunto $FORM$ de las fórmulas está libremente generado por las fórmulas atómicas y las aplicaciones $C_\neg, C_\vee$ y E_i.

Nota 11.2.8

1. Para disminuir el número de paréntesis se usan los mismos convenios que en el caso proposicional.
2. A veces, escribiremos:

$f(t_1, \ldots, t_n)$	en lugar de	$ft_1 \ldots t_n$
$p(t_1, \ldots, t_n)$	en lugar de	$pt_1 \ldots t_n$
$(\exists x_i)[F]$	en lugar de	$\exists x_i F$
$(\forall x_i)[F]$	en lugar de	$\forall x_i F$

11.3 Variables libres y ligadas

Definición 11.3.1 El conjunto $Libre(t)$ de las **variables libres** de un término t se define recursivamente por:

$$Libre(t) = \begin{cases} \{x\}, & \text{si } t = x \text{ es una variable;} \\ Libre(t_1) \cup \ldots \cup Libre(t_n), & \text{si } t = ft_1 \ldots t_n \end{cases}$$

Definición 11.3.2 El conjunto $Libre(F)$ de las **variables libres** de una fórmula F se define recursivamente por:

$$Libre(F) = \begin{cases} Libre(t_1) \cup \ldots \cup Libre(t_n), & \text{si } t = pt_1 \ldots t_n \\ Libre(G) & \text{si } F = \neg G; \\ Libre(G) \cup Libre(H) & \text{si } F = (G \vee H); \\ Libre(G) - \{x_i\} & \text{si } F = \exists x_i G; \end{cases}$$

Definición 11.3.3

1. Un término t es **cerrado** si $Libre(t) = \emptyset$.
2. Una fórmula F es **cerrada** si $Libre(F) = \emptyset$.
3. Una **sentencia** es una fórmula cerrada.
4. Una fórmula es **abierta** si no contiene cuantificadores.

Definición 11.3.4 El conjunto $Ligada(F)$ de las **variables ligadas** de una fórmula F se define recursivamente por:

$$Ligada(F) = \begin{cases} Ligada(t_1) \cup \ldots \cup Ligada(t_n), & \text{si } t = pt_1 \ldots t_n \\ Ligada(G) & \text{si } F = \neg G; \\ Ligada(G) \cup Ligada(H) & \text{si } F = (G \vee H); \\ Ligada(G) \cup \{x_i\} & \text{si } F = \exists x_i G; \end{cases}$$

Nota 11.3.5 En general, $Libre(F) \cap Ligada(F) \neq \emptyset$.

11.4 Sustituciones

Definición 11.4.1

1. Una **sustitución** θ es una aplicación del conjunto V de las variables en el conjunto $TERM$ de los términos.
2. El **dominio** de una sustitución θ es
$$D(\theta) = \{x \in V : \theta(x) \neq x\}$$
3. Una sustitución θ es **finita** si $D(\theta)$ es finito.

Nota 11.4.2 En lo que sigue, sólo consideraremos sustituciones finitas y omitiremos el adjetivo.

Nota 11.4.3 Por $\{(x_1, t_1), \ldots, (x_n, t_n)\}$ representaremos la sustitución θ definida por

$$\theta(x) = \begin{cases} t_i, & \text{si } x = x_i; \\ x, & \text{si } x \in V - \{x_1, \ldots, x_n\} \end{cases}$$

Teorema 11.4.4 Para cada sustitución θ existe una única aplicación $\hat{\theta}$ de $TERM$ en $TERM$ definida por:

$$\hat{\theta}(t) = \begin{cases} \theta(t), & \text{si } t \text{ es una variable;} \\ f\hat{\theta}(t_1) \ldots \hat{\theta}(t_n), & \text{si } t = ft_1 \ldots t_n \end{cases}$$

En lo sucesivo, escribiremos $\theta(t)$ en lugar de $\hat{\theta}(t)$.

Definición 11.4.5 Sean s y t dos términos. El término resultante de sustituir en s la variable x por el término t es

$$s[x/t] = \theta(s),$$

donde $\theta = \{(x, t)\}$.

Definición 11.4.6 Sean F una fórmula y t un término. La fórmula resultante de sustituir en F la variable x por el término t se define recursivamente por:

$$F[x/t] = \begin{cases} pt_1[x/t] \ldots t_n[x/t], & \text{si } F = pt_1 \ldots t_n; \\ \neg F[x/t], & \text{si } F = \neg G; \\ G[x/t] \vee H[x/t], & \text{si } F = G \vee H: \\ \exists y G[x/t], & \text{si } F = \exists y G \text{ y } x \neq y; \\ \exists x G, & \text{si } F = \exists x G; \end{cases}$$

Definición 11.4.7 Una variable x de F es **sustituible** por el término t si se cumple una de las siguientes condiciones:

1. F es atómica;
2. $F = \neg G$ y x es sustituible por t en G;
3. $F = G \vee H$ y x es sustituible por t en G y en H;
4. $F = \exists x G$;

5. $F = \exists y G$, $x \neq y$, $y \notin Libre(t)$ y x es sustituible por t en G.

Nota 11.4.8

1. En lo sucesivo, al escribir $F[x/t]$, supondremos que x es sustituible por t en F.

2. Si una fórmula F contiene una variable libre x, escribiremos F como $F(x)$ y abreviaremos $F[x/t]$ por $F(t)$.

Definición 11.4.9 El conjunto $Subf(F)$ de las **subfórmulas** de una fórmula F se define recursivamente por:

$$Subf(F) = \begin{cases} \{F\}, & \text{si } F \text{ es atómica;} \\ \{F\} \cup Subf(G), & \text{si } F = \neg G; \\ \{F\} \cup Subf(G) \cup Subf(H), & \text{si } F = G \vee H; \\ \{F\} \cup (\bigcup \{Subf(G[x/t]) : \ t \text{ es un término}\}), & \text{si } F = \exists x G; \end{cases}$$

Capítulo 12

Semántica de la lógica de primer orden

12.1 Semántica de los términos y las fórmulas

Definición 12.1.1 Sea **L** un lenguaje de primer orden. Una **L–estructura** (o, simplemente, **estructura**) es un par $\mathbf{M} = (M, I)$ donde M es un conjunto no vacío llamado el **universo** (o **dominio**) de la estructura e I es una aplicación llamada la **interpretación** de la estructura tal que:

1. Para cada constante c, $I(c) \in M$.
2. Para cada símbolo de función de aridad $n > 0$, $I(f) : M^n \to M$.
3. Para cada símbolo de predicado de aridad 0, $I(p) \in 2$.
4. Para cada símbolo de predicado de aridad $n > 0$, $I(p) : M^n \to 2$.

A veces, escribiremos $c_{\mathbf{M}}$, $f_{\mathbf{M}}$ ó $p_{\mathbf{M}}$ en lugar de $I(c)$, $I(f)$ ó $I(p)$, respectivamente.

Definición 12.1.2 Sea **L** un lenguaje de primer orden y **M** una **L**–estructura. Una **asignación** σ es una aplicación del conjunto de las variables de **L** en el universo de **M**. El conjunto de las asignaciones se representa por M^V.

Teorema 12.1.3 Sea **M** una **L**–estructura. Para cada asignación σ existe una única aplicación $\hat{\sigma}$ de $TERM$ en M definida por:

$$\hat{\sigma}(t) = \begin{cases} \sigma(t), & \text{si } t \text{ es una variable;} \\ f_{\mathbf{M}}(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n)), & \text{si } t = f t_1 \ldots t_n \end{cases}$$

En lo sucesivo, escribiremos $\sigma(t)$ ó $t_{\mathbf{M}}$ en lugar de $\hat{\sigma}(t)$.

Definición 12.1.4 Sea $\mathbf{M}$ una $\mathbf{L}$–estructura, $m \in M$ y $\sigma \in V^M$. Se representa por $\sigma[x/m]$ la asignación σ' definida por:

$$\sigma'(y) = \begin{cases} \sigma(y), & \text{si } y \neq x; \\ m, & \text{si } y = x. \end{cases}$$

Definición 12.1.5 La aplicación $H_\exists : \mathbf{P}(2) \to 2$ está definida por

$$H_\exists(X) = \begin{cases} 1, & \text{si } 1 \in X; \\ 0, & \text{en caso contrario} \end{cases}$$

Teorema 12.1.6 Sea $\mathbf{M}$ una $\mathbf{L}$–estructura. Para cada asignación σ existe una única aplicación $\hat{\sigma}$ de $FORM$ en 2 definida por:

$$\hat{\sigma}(F) = \begin{cases} p_{\mathbf{M}}(\sigma(t_1), \ldots, \sigma(t_n)), & \text{si } F = pt_1 \ldots t_n; \\ H_\neg(G), & \text{si } F = \neg G; \\ H_\vee(G, H), & \text{si } F = G \vee H; \\ H_\exists(\{\widehat{\sigma[x/m]}(G) : m \in M\}), & \text{si } F = \exists x G; \end{cases}$$

En lo sucesivo, escribiremos $\sigma(F)$ ó $F_{\mathbf{M}}$ en lugar de $\hat{\sigma}(t)$.

Lema 12.1.7 Sea $\mathbf{M}$ una $\mathbf{L}$–estructura y σ una asignación.

1. $\sigma(F \wedge G) = H_\wedge(\sigma(F), \sigma(G))$.
2. $\sigma(F \to G) = H_\to(\sigma(F), \sigma(G))$.
3. $\sigma(F \leftrightarrow G) = H_\leftrightarrow(\sigma(F), \sigma(G))$.
4. $\sigma(\forall x F) = 1$ syss $\sigma[x/m](F) = 1$ para todo $m \in M$.
5. $\sigma(\exists x F) = 1$ syss $\sigma[x/m](F) = 1$ para algún $m \in M$.

12.2 Consistencia, validez y modelos

Definición 12.2.1 Sea $\mathbf{L}$ un lenguaje de primer orden y F una fórmula.

1. F es **válida en** la $\mathbf{L}$–estructura $\mathbf{M}$ **respecto de** la asignación σ, $\mathbf{M} \models_\sigma F$, si $\sigma(F) = 1$.

2. F es **válida en** la **L**–estructura **M**, $\mathbf{M} \models F$, si $\sigma(F) = 1$ para todas las asignaciones σ. En este caso, se dice que **M** es un **modelo** de F.

3. F es **válida**, $\models F$, si es válida en todas las **L**–estructuras.

4. F es **consistente** (o **satisfacible**) si tiene algún modelo.

Definición 12.2.2 Sea **L** un lenguaje de primer orden y Γ un conjunto de fórmulas.

1. Γ es **válido en** la **L**–estructura **M** **respecto de** la asignación σ, $\mathbf{M} \models_\sigma \Gamma$, si $\sigma(F) = 1$ para toda $F \in \Gamma$.

2. Γ es **válido en** la **L**–estructura **M**, $\mathbf{M} \models \Gamma$, si $\mathbf{M} \models_\sigma \Gamma$ para todas las asignaciones σ. En este caso, se dice que **M** es un **modelo** de Γ.

3. Γ es **válido**, $\models \Gamma$, si es válido en todas las **L**–estructuras.

4. Γ es **consistente** (o **satisfacible**) si tiene algún modelo.

Definición 12.2.3 Sea **L** un lenguaje de primer orden y Γ, Γ' dos conjuntos de fórmulas.

1. Γ' es **consecuencia semántica** de Γ, $\Gamma \models \Gamma'$, si todos los modelos de Γ también son modelos de Γ'.

2. La fórmula F es **consecuencia semántica** de Γ, $\Gamma \models F$, si todos los modelos de Γ también son modelos de F.

Nota 12.2.4 Se escribirá $F_1, \ldots, F_n \models G$ en lugar de $\{F_1, \ldots, F_n\} \models G$.

Lema 12.2.5 Sea **M** una **L**–estructura, F una fórmula y σ_1, σ_2 dos asignaciones. Si $\sigma_1(x) = \sigma_2(x)$ para toda $x \in \mathit{Libre}(F)$, entonces $\sigma_1(F) = \sigma_2(F)$.

Corolario 12.2.6 Si **M** es una **L**–estructura y F es una fórmula cerrada, entonces $\mathbf{M} \models F$ ó $\mathbf{M} \models \neg F$.

Lema 12.2.7 Sea $\Gamma = \{F_1, \ldots, F_n\}$ un conjunto de fórmulas cerradas y G una fórmula cerrada.

1. $\mathbf{M} \models \Gamma$ syss $\mathbf{M} \models \bigwedge_{i=1}^{n} F_i$.

2. $\Gamma \models F$ syss $\models \bigwedge_{i=1}^{n} F_i \to G$.

Lema 12.2.8 Sea Γ un conjunto de fórmulas cerradas y F una fórmula cerrada. Entonces $\Gamma \models F$ syss $\Gamma \cup \{\neg F\}$ es inconsistente.

Corolario 12.2.9 Una fórmula cerrada F es válida syss $\neg F$ es inconsistente.

Nota 12.2.10

1. El **problema de la consistencia** (o de la satisfacibilidad) consiste en dada una fórmula determinar si es consistente.
2. El **problema de la validez** consiste en dada una fórmula determinar si es válida.

12.3 Semántica mediante extensiones de lenguajes

Definición 12.3.1 Sea $\mathbf{M}$ una $\mathbf{L}$–estructura. El **lenguaje extendido** $\mathbf{L}(\mathbf{M})$ se obtiene añadiéndole al conjunto de constantes de $\mathbf{L}$ una nueva constante, $\mathbf{m}$, por cada elemento $m \in M$. La interpretación I de la estructura $\mathbf{M}$ se extiende definiendo $I(\mathbf{m}) = m$ para las nuevas constantes.

Lema 12.3.2 Sea $\mathbf{M}$ una $\mathbf{L}$–estructura, σ una asignación, x una variable, $m \in M$ y $\sigma' = \sigma[x/m]$.

1. $\sigma'(t) = \sigma(t[x/\mathbf{m}])$ para todo término t de $\mathbf{L}$.
2. $\sigma'(F) = \sigma(F[x/\mathbf{m}])$ para todo fórmula F de $\mathbf{L}$.

12.4 Fórmulas válidas

Teorema 12.4.1 Sea F una fórmula proposicional y G una fórmula obtenida sustituyendo las variables proposicionales de F por fórmulas del lenguaje de primer orden $\mathbf{L}$. Si F es una tautología, entonces G es válida.

Definición 12.4.2 Sea F una fórmula y $x_1, \ldots, x_n$ las variable libres de F.

1. El **cierre universal** de F, $\forall(F)$, es la fórmula $\forall x_1 \ldots \forall x_n F$.
2. El **cierre existencial** de F, $\exists(F)$, es la fórmula $\exists x_1 \ldots \exists x_n F$.

Lema 12.4.3

1. F es válida syss $\forall(F)$ es válida.
2. F es consistente syss $\exists(F)$ es consistente .

Definición 12.4.4 La fórmula F' es una **instancia** de la fórmula F si existe una sustitución σ tal que $\sigma(F) = F'$.

Lema 12.4.5 Sea F' una instancia de la fórmula F.

1. Si $\models \forall(F)$, entonces $\models F'$.
2. Si $\models F'$, entonces $\models \exists(F)$.

Definición 12.4.6 Dos fórmulas F y G son **equivalentes**, y se representa por $F \equiv G$, si $\models F \leftrightarrow G$.

Lema 12.4.7 La relación $\equiv$ es de equivalencia en $FORM$.

Lema 12.4.8

1. Si $F \equiv F'$, entonces $\neg F \equiv \neg F'$.
2. Si $F \equiv F'$, $G \equiv G'$ y $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, entonces $F * G \equiv F' * G'$.
3. Si $F \equiv F'$ y $Q \in \{\forall, \exists\}$, entonces $QxF \equiv QxF'$.

Teorema 12.4.9 (de sustitución)
Sea G una subfórmula de F y F' la fórmula obtenida sustituyendo una ocurrencia de G en F por G'. Si $G \equiv G'$, entonces $F \equiv F'$.

Lema 12.4.10 Las siguientes equivalencias se verifican para todas las fórmulas:

1. Idempotencia:
$$F \vee F \equiv F \qquad F \wedge F \equiv F$$

2. Conmutatividad:

$$F \vee G \equiv G \vee F \qquad F \wedge G \equiv G \wedge F$$

3. Asociatividad:

$$F \vee (G \vee H) \equiv (F \vee G) \vee H \qquad F \wedge (G \wedge H) \equiv (F \wedge G) \wedge H$$

4. Absorción:

$$F \wedge (G \vee H) \equiv F \qquad F \vee (G \wedge H) \equiv F$$

5. Distributividad:

$$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H) \qquad F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$$

6. Doble negación:

$$\neg\neg F \equiv F$$

7. Leyes de de Morgan:

$$\neg(F \wedge G) \equiv \neg F \vee \neg G \qquad \neg(F \vee G) \equiv \neg F \wedge \neg G$$

Lema 12.4.11

1. Leyes de de Morgan:

$$\neg\left(\bigwedge_{i=1}^{n} F_i\right) \equiv \bigvee_{i=1}^{n} \neg F_i \qquad \neg\left(\bigvee_{i=1}^{n} F_i\right) \equiv \bigwedge_{i=1}^{n} \neg F_i$$

2. Distributividad:

$$\left(\bigvee_{i=1}^{m} F_i\right) \wedge \left(\bigvee_{j=1}^{n} G_j\right) \equiv \bigvee_{i=1}^{m}\left(\bigvee_{j=1}^{n}(F_i \wedge G_j)\right) \quad \left(\bigwedge_{i=1}^{m} F_i\right) \vee \left(\bigwedge_{j=1}^{n} G_j\right) \equiv \bigwedge_{i=1}^{m}\left(\bigwedge_{j=1}^{n}(F_i \vee G_j)\right)$$

Lema 12.4.12 (propiedades de los cuantificadores)

1. Reglas de dualidad:

$$\neg\forall x F \equiv \exists x \neg F \qquad \neg\exists x F \equiv \forall x \neg F$$

2. Reglas de distribución condicional:

$$\forall x(F \wedge G) \equiv \forall xF \wedge \forall xG \qquad \exists x(F \vee G) \equiv \exists xF \vee \exists xG$$

3. Reglas de distribución: Si x no ocurre libre en G, entonces

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

4. Reglas de intercambio de cuantificadores:

$$\forall x \forall y F \equiv \forall y \forall x F \qquad \exists x \exists y F \equiv \exists y \exists x F$$

Nota 12.4.13 En general,

$$\forall x(F \vee G) \not\equiv \forall xF \vee \forall xG, \qquad \exists x(F \wedge G) \not\equiv \exists xF \wedge \forall xG$$

Lema 12.4.14 (cambio de variables ligadas)
Si y es una variable sustituible por x en F y no ocurre libre en F, entonces

$$\forall xF \equiv \forall yF[x/y] \qquad \exists xF \equiv \exists yF[x/y]$$

Definición 12.4.15 Una fórmula F es **rectificada** si $Libre(F) \cap Ligada(F) = \emptyset$ y todos los cuantificadores se aplican a variables distintas.

Lema 12.4.16 Para cada fórmula F existe una fórmula rectificada F' tal que $F \equiv F'$.

Capítulo 13

Formas normales y cláusulas

13.1 Formas prenexas

Definición 13.1.1 Una fórmula F está en **forma prenexa** si es de la forma

$$Q_1 x_1 \dots Q_n x_n G$$

donde $Q_i \in \{\forall, \exists\}$, x_i son variables distintas y G es una fórmula abierta.

Se dice que $Q_1 x_1 \dots Q_n$ es el **prefijo** de F y que G es la **matriz** de F.

Teorema 13.1.2 Para cada fórmula F existe una fórmula equivalente F' en forma prenexa.

Nota 13.1.3 Existen algoritmos que para cada fórmula F devuelven una fórmula equivalente F' en forma prenexa.

Corolario 13.1.4 Para cada fórmula F existe una fórmula equivalente F' en forma prenexa con su matriz en forma normal conjuntiva (resp. disyuntiva).

13.2 Formas de Skolem

Definición 13.2.1 Para cada fórmula F, definimos su **forma de Skolem**, $Skolem(F)$, como el resultado del siguiente "algoritmo":

- Sea F_1 una fórmula en forma normal equivalente a $\forall(F)$.

- Si F_1 no contiene cuantificadores existenciales, entonces $Skolem(F)$ es F_1.
- Si F_1 es de la forma $\forall x_1 \ldots \forall x_n \exists y G$, entonces

  $$Skolem(F) = Skolem(\forall x_1 \ldots \forall x_n G[y/f x_1 \ldots x_n]$$

  donde f es un símbolo de función de aridad n que no ocurre en F. En este caso, se dice que f es una **función de Skolem**.

Ejemplo 13.2.2

1. La forma de Skolem de

   $$(\exists x_1)(\forall x_2)(\exists x_3)(\exists x_4)[p(x_1, x_2) \rightarrow q(x_3, x_4)]$$

   es

   $$(\forall x_2)[p(a, x_2) \rightarrow q(f_1(x_2), f_2(x_2))]$$

2. La forma de Skolem de

   $$(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)(\exists x_5)[p(x_1, x_2, x_3, x_4, x_5)]$$

   es

   $$(\forall x_2)(\forall x_4)[p(a, x_2, f_1(x_2), x_4, f_2(x_2, x_4))]$$

Lema 13.2.3 $\models Skolem(F) \rightarrow F$.

Teorema 13.2.4 Una fórmula F es inconsistente syss su forma de Skolem es inconsistente.

Nota 13.2.5

1. Una forma de Skolem es una forma prenexa cuyo prefijo no contiene cuantificadores existenciales.
2. A veces, se suprime el prefijo de las formas de Skolem (sobrentendiéndose que todas las variables que aparecen en la matriz están universalmente cuantificadas).

13.3 Formas clausales

Definición 13.3.1 Una **forma clausal** es una forma de Skolem cuya matriz está en forma normal conjuntiva.

Lema 13.3.2 Para cada forma de Skolem existe una forma clausal equivalente.

Nota 13.3.3 Existe un algoritmo que para cada fórmula F devuelve una fórmula F' en forma clausal tal que F es consistente syss F' es consistente.

Capítulo 14

Cláusulas

14.1 Cláusulas

Definición 14.1.1 Sea **L** un lenguaje de primer orden.

1. Un **átomo** es una fórmula atómica de **L**.
2. Un **literal** es un átomo o su negación.
3. Un literal es **positivo** si es un átomo y **negativo**, en caso contrario.
4. El **complementario** de un literal L es

$$\overline{L} = \begin{cases} \neg pt_1 \dots t_n & \text{si } L = pt_1 \dots t_n; \\ pt_1 \dots t_n & \text{si } L = \neg pt_1 \dots t_n; \end{cases}$$

5. Los literales L y L' son **complementarios** si $L' = \overline{L}$.
6. Una **cláusula** es un conjunto de literales.
7. La **cláusula vacía** es el conjunto vacío y se representa por $\square$.

Nota 14.1.2 Usaremos las siguientes variables sintácticas:

1. A, B, C para átomos.
2. L para literales.
3. C, D para cláusulas.
4. S, P para conjuntos de cláusulas.

Definición 14.1.3 Sea **M** una **L**–estructura y θ una sustitución

1. Para cada literal L,

$$\theta(L) = \begin{cases} p\theta(t_1)\dots\theta(t_n), & \text{si } L = pt_1\dots t_n; \\ \neg p\theta(t_1)\dots\theta(t_n), & \text{si } L = \neg pt_1\dots t_n; \end{cases}$$

2. Para cada cláusula C,

$$\theta(C) = \{\theta(L) : L \in C\}$$

3. Para cada conjunto de cláusulas S,

$$\theta(S) = \{\theta(C) : C \in S\}$$

Definición 14.1.4 Sea **M** una **L**–estructura y σ una asignación.

1. Para cada literal L,

$$\sigma'(L) = \begin{cases} p_{\mathbf{M}}(\sigma(t_1),\dots,\sigma(t_n)), & \text{si } L = pt_1\dots t_n; \\ 1 - p_{\mathbf{M}}(\sigma(t_1),\dots,\sigma(t_n)), & \text{si } L = \neg pt_1\dots t_n; \end{cases}$$

2. Para cada cláusula C,

$$\sigma''(C) = 1 \text{ syss existe un } L \in C \text{ tal que } \sigma'(L) = 1$$

3. Para cada conjunto de cláusulas S,

$$\sigma'''(S) = 1 \text{ syss para todo } C \in S,\ \sigma''(C) = 1$$

Lema 14.1.5 Sea **M** una **L**–estructura y σ una asignación.

1. $\sigma''(\square) = 0$.
2. $\sigma'''(\emptyset) = 1$.

Nota 14.1.6 En lo que sigue, escribiremos $\sigma(L), \sigma(C)$ ó $\sigma(S)$ en lugar de $\sigma'(L), \sigma''(C)$ ó $\sigma'''(S)$

Definición 14.1.7 Sea **L** un lenguaje de primer orden y C una cláusula.

1. C es **válida en** la **L**–estructura **M respecto de** la asignación σ, $\mathbf{M} \models_\sigma C$, si $\sigma(C) = 1$.

2. C es **válida en** la **L**–estructura **M**, $\mathbf{M} \models C$, si $\sigma(C) = 1$ para todas las asignaciones σ. En este caso, se dice que **M** es un **modelo** de C.

3. C es **válida**, $\models C$, si es válida en todas las **L**–estructuras.

4. C es **consistente** (o **satisfacible**) si tiene algún modelo.

Definición 14.1.8 Sea **L** un lenguaje de primer orden y S un conjunto de cláusulas.

1. S es **válido en** la **L**–estructura **M** **respecto de** la asignación σ, $\mathbf{M} \models_\sigma S$, si $\sigma(F) = 1$ para toda $F \in S$.

2. S es **válido en** la **L**–estructura **M**, $\mathbf{M} \models S$, si $\mathbf{M} \models_\sigma S$ para todas las asignaciones σ. En este caso, se dice que **M** es un **modelo** de S.

3. S es **válido**, $\models S$, si es válido en todas las **L**–estructuras.

4. S es **consistente** (o **satisfacible**) si tiene algún modelo.

Definición 14.1.9 Sea **L** un lenguaje de primer orden y S, S' dos conjuntos de cláusulas.

1. S' es **consecuencia semántica** de S, $S \models S'$, si todos los modelos de S también son modelos de S'.

2. La cláusula C es **consecuencia semántica** de S, $S \models C$, si todos los modelos de S también son modelos de C.

Nota 14.1.10

1. Si $S \subseteq S'$ y S' es consistente, entonces S es consistente.

2. Si $\square \in S$, entonces S es inconsistente.

14.2 Cláusulas y fórmulas

Definición 14.2.1

1. El conjunto de fórmulas correspondiente a la cláusula $C \neq \square$ es

$$Form(C) = \left\{ \forall \left(\bigvee_{i=1}^{n} L_i \right) : C = \{L_1, \ldots, L_n\} \right\}$$

2. El conjunto de fórmulas correspondiente al conjunto de cláusulas $C \neq \emptyset$ es

$$Form(S) = \left\{ \bigwedge_{i=1}^{n} F_i : S = \{C_1, \ldots, C_n\}, F_i \in Form(C_i) \right\}$$

Lema 14.2.2

1. Si $F, G \in Form(C)$, entonces $F \equiv G$.
2. Si $F, G \in Form(S)$, entonces $F \equiv G$.

Definición 14.2.3 Un conjunto de cláusulas S es una **forma clausal** de la fórmula F si son equiconsistentes (es decir, F es consistente syss S es consistente).

Lema 14.2.4 Una forma clausal de la fórmula cerrada

$$\forall x_1 \ldots \forall x_k \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right)$$

es

$$\{\{L_{i,j} : 1 \leq j \leq m_i\} : 1 \leq i \leq n\}$$

Algoritmo 14.2.5

Entrada: Un conjunto finito de fórmulas cerradas, Γ, y una fórmula cerrada, G.

Salida: Un conjunto de cláusulas, S, tal que S es consistente syss $\Gamma \models G$.

Procedimiento:

- Sea $\Gamma_1 = \Gamma \cup \{\neg G\}$.
- Sea Γ_2 el conjunto de las formas prenexas de las fórmulas de Γ_1.
- Sea Γ_3 el conjunto de las formas de Skolem de las fórmulas de Γ_2.
- Sea Γ_4 el conjunto de las formas normales de las fórmulas de Γ_3.
- Sea S_1 el conjunto de las formas clausales de las fórmulas de Γ_4.
- Devolver $S = \bigcup S_1$.

Ejemplo 14.2.6 Sea

$$\Gamma = \{(\forall x)[p(x) \to (\exists y)[r(y) \wedge q(x,y)]], \quad (\exists x)p(x)\}$$

y $G = (\exists x)(\exists y)q(x,y)$. Entonces,

$$\begin{aligned}
\Gamma_1 &= \{ \quad (\forall x)[p(x) \to (\exists y)[r(y) \wedge q(x,y)]], \\
&\qquad (\exists x)p(x), \\
&\qquad \neg(\exists x)(\exists y)q(x,y)\} \\
\Gamma_2 &= \{ \quad (\forall x)(\exists y)[\neg p(x) \vee (r(y) \wedge q(x,y))], \\
&\qquad (\exists x)p(x), \\
&\qquad (\forall x)(\forall y)\neg q(x,y)\} \\
\Gamma_3 &= \{ \quad (\forall x)[\neg p(x) \vee (r(f(x)) \wedge q(x,f(x)))], \\
&\qquad p(a), \\
&\qquad \neg q(x,y)\} \\
\Gamma_4 &= \{ \quad (\forall x)[(\neg p(x) \vee r(f(x))) \wedge (\neg p(x) \vee q(x,f(x)))], \\
&\qquad p(a), \\
&\qquad \neg q(x,y)\} \\
S_1 &= \{ \quad \{\{\neg p(x), r(f(x))\}, \{\neg p(x), q(x,f(x))\}\}, \\
&\qquad \{\{p(a)\}\}, \\
&\qquad \{\{\neg q(x,y)\}\}\} \\
S &= \{ \quad \{\neg p(x), r(f(x))\}, \\
&\qquad \{\neg p(x), q(x,f(x))\}, \\
&\qquad \{p(a)\}, \\
&\qquad \{\neg q(x,y)\}\}
\end{aligned}$$

Capítulo 15

Teorema de Herbrand

15.1 Modelos de Herbrand

Nota 15.1.1 En lo que sigue, **L** es un lenguaje de primer orden cuyo conjunto de constantes es no vacío (en caso contrario, se le añade una constante, a).

Definición 15.1.2

1. El **universo de Herbrand** $U(\mathbf{L})$ de **L** es el conjunto de los términos cerrados de **L**.
2. La **base de Herbrand** de **L**, $B(\mathbf{L})$, es el conjunto de los átomos cerrados de **L**.

Definición 15.1.3 Una **L** estructura **M** es una **estructura de Herbrand** de **L** si:

1. El universo de **M** es el universo de Herbrand de **L**; es decir,
$$M = U(\mathbf{L})$$
2. Para toda constante c de **L**,
$$c_{\mathbf{M}} = c$$
3. Para todo símbolo de función f de aridad $n > 0$, y todo $t_1, \ldots, t_n \in U(\mathbf{L})$,
$$f_{\mathbf{M}}(t_1, \ldots, t_n) = ft_1 \ldots t_n$$

Definición 15.1.4 Una **interpretación de Herbrand** de **L** es un subconjunto de la base de Herbrand de **L**.

Nota 15.1.5 Usaremos los símbolos $I, I_1, I_2, \ldots$ para representar interpretaciones de Herbrand.

Lema 15.1.6 Sea $\mathbb{M}$ el conjunto de las estructuras de Herbrand de **L** e $\mathbb{I}$ el conjunto de las interpretaciones de Herbrand de **L**. Las aplicaciones

$$\Phi : \mathbf{M} \in \mathbb{M} \mapsto \Phi(\mathbf{M}) = \{A \in B_{\mathbf{L}} : \mathbf{M} \models A\} \in \mathbb{I}$$
$$\Psi : I \in \mathbb{I} \mapsto \Psi(I) = \mathbf{M} \in \mathbb{M},$$

donde **M** es la **L**–estructura de Herbrand tal que para todo símbolo de predicado de aridad n y para todo $t_1, \ldots, t_n \in U(\mathbf{L})$,

$$p_{\mathbf{M}}(t_1, \ldots, t_n) = 1 \text{ syss } pt_1 \ldots t_n \in I,$$

son biyectivas. Además, $\Phi^{-1} = \Psi$.

Nota 15.1.7 En lo sucesivo, identificaremos las estructuras de Herbrand de **L** y sus interpretaciones de Herbrand.

Definición 15.1.8 Un **modelo de Herbrand** de un conjunto Γ de fórmulas de **L** es una estructura de Herbrand de **L** que es un modelo de Γ.

Nota 15.1.9 Los conceptos de universo, base, estructuras, interpretaciones y modelos de Herbrand definidos anteriormente para el lenguaje **L** se aplican a fórmulas, conjuntos de fórmulas, cláusulas y conjuntos de cláusulas, considerando en cada caso el lenguaje formado a partir de sus símbolos de funciones y predicados.

Lema 15.1.10 Sea I una interpretación de Herbrand y

$$C = \{A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m\}$$

una cláusula. Son equivalentes:

1. $I \models C$
2. Para toda sustitución $\theta : V \to U(C)$,

$$\{\theta(B_1), \ldots, \theta(B_m)\} \subseteq I \quad \Longrightarrow \quad \{\theta(A_1), \ldots, \theta(A_m)\} \cap I \neq \emptyset$$

Lema 15.1.11 Sea **M** una **L**–estructura de Herbrand y θ una asignación. Entonces,

$$\theta(F[x/t]) = \theta[x/t](F)$$

Teorema 15.1.12 Si un conjunto de cláusulas S es consistente, entonces tiene un modelo de Herbrand.

Corolario 15.1.13 Un conjunto de cláusulas S es inconsistente syss no tiene modelos de Herbrand.

Corolario 15.1.14 (Lwenheim–Skolem)
Si Γ es un conjunto de fórmulas consistente, entonces tiene un modelo numerable (i.e. su universo es un conjunto numerable).

15.2 Teorema de Herbrand

Definición 15.2.1 Sea S un conjunto finito de cláusulas.

1. La sustitución θ es **básica** si $rang(\theta) \subseteq U(S)$.
2. La cláusula C' es una **instancia básica** de la cláusula $C \in S$ si existe una sustitución básica, θ, tal que $C' = \theta(C)$.
3. La **extensión de Herbrand** de S; $E(S)$, es el conjunto de las instancias básicas de las cláusulas de S.

Nota 15.2.2 Mediante la expansión de Herbrand asociamos a un conjunto finito de cláusulas de primer orden un conjunto (posiblemente infinito) de cláusulas del cálculo proposicional.

Teorema 15.2.3 (de Skolem–Herbrand–Gdel)
Sea S un conjunto finito de cláusulas. Entonces S es consistente syss $E(S)$ es consistente (en el sentido de la lógica proposicional).

Teorema 15.2.4 (de Herbrand)
Sea S un conjunto finito de cláusulas. Entonces S es inconsistente syss $E(S)$ contiene un subconjunto finito inconsistente (en el sentido de la lógica proposicional).

15.3 Métodos de deducción basados directamente en el teorema de Herbrand

Definición 15.3.1 Sea S un conjunto de cláusulas.

1. La sucesión $(U_i(S))_{i\geq 0}$ está definida por:

$$\begin{array}{lcl} U_0(S) & = & \left\{ \begin{array}{ll} \text{el conjunto de constantes de } S, & \text{si tiene alguna;} \\ \{a\}, & \text{en caso contrario.} \end{array} \right. \\ U_{i+1}(S) & = & U_i(S) \cup \{ft_1 \ldots t_n : f \in SF; r(f) = n; t_1, \ldots, t_n \in U_i(S)\} \end{array}$$

2. Para cada $i \geq 0$,

$$S_i = \{\theta(C) : C \in S \text{ y } \theta \text{ es una sustitución tal que } rang(\theta) \subseteq U_i(S)\}$$

Lema 15.3.2 Sea S un conjunto de cláusulas.

1. $U(L) = \bigcup_{i\geq 0} U_i(S)$.
2. S es inconsistente syss existe un $i \geq 0$ tal que S_i es inconsistente.

Algoritmo 15.3.3

Entrada: Un conjunto finito de cláusulas, S.

Salida: `Inconsistente`, si S es inconsistente.

Procedimiento:

Hacer $i := 0$
mientras que S_i es consistente (en el sentido proposicional)
hacer $i := i + 1$
devolver `Inconsistente`

Nota 15.3.4 Para determinar la consistencia de los conjuntos S_i se utilizan los algoritmos estudiados para la lógica proposicional; en particular, el algoritmo de Davis–Putnam.

Ejemplo 15.3.5 Para el conjunto

$$S = \{\{q(x), \neg p(x)\}, \{p(f(x))\}, \{\neg q(f(x))\}\}$$

se obtiene

$S_0 = \{\{q(a), \neg p(a)\}, \{p(f(a))\}, \{\neg q(f(a))\}\}$ es consistente,
$S_1 = S_0 \cup \{\{q(f(a)), \neg p(f(a))\}, \{p(f(f(a)))\}, \{\neg q(f(f(a)))\}\}$ es inconsistente;

por tanto, S es inconsistente.

Teorema 15.3.6 Dado un conjunto finito de cláusulas, S, el algoritmo anterior termina tras un número finito de pasos syss S es inconsistente.

Corolario 15.3.7 Los problemas de consistencia y validez para la lógica de primer orden son semidecidibles.

15.4 Resolución básica

Algoritmo 15.4.1 (de resolución básica)

Entrada: Un conjunto finito de cláusulas, S.

Salida: `Inconsistente`, si S es inconsistente.

Procedimiento:

Hacer $i := 0$
mientras que $\square \notin Res^*(S_i)$
hacer $i := i + 1$
devolver `Inconsistente`

Teorema 15.4.2 Dado un conjunto finito de cláusulas, S, el algoritmo anterior termina tras un número finito de pasos syss S es inconsistente.

Nota 15.4.3 Los algoritmos anteriores requieren la generación de los conjuntos S_i que pueden crecer exponencialmente. Por ejemplo, si

$$S = \{\{p(x, g(x), y, h(x, y), z, k(x, y, z))\}, \{\neg p(u, v, e(v), w, f(v, w), x)\}\},$$

entonces $|U_0(S)| = 1$, $|U_1(S)| = 6$, ..., y $|S_0| = 2$, $|S_1| = 4825$,.... El primer conjunto S_i inconsistente es S_5 que tiene del orden de 10^{256} elementos.

Capítulo 16

Sustitución y unificación

16.1 Comparación de términos

Nota 16.1.1 En lo que sigue, **L** es un lenguaje de primer orden y T es el conjunto de sus términos.

Definición 16.1.2 En T se define la relación $\leq$ por

$$t_1 \leq t_2 \text{ syss existe una sustitución } \theta \text{ tal que } \theta(t_1) = t_2.$$

Si $t_1 \leq t_2$ se dice que t_1 es **menos particular** que t_2; o bien, que t_2 es una **instancia** de t_1.

Ejemplo 16.1.3

$$x \leq f(x,y) \leq f(g(y),y) \leq f(g(x),x) \leq f(g(a),a).$$

Lema 16.1.4 La relación $\leq$ es un preorden en T (i.e. es reflexiva y transitiva en T).

Definición 16.1.5 En T se define la relación $\equiv$ por

$$t_1 \equiv t_2 \text{ syss } t_1 \leq t_2 \wedge t_2 \leq t_1$$

Si $t_1 \equiv t_2$ se dice que t_1 y t_2 son **equivalentes**.

Ejemplo 16.1.6 $f(g(x,a),y) \equiv f(g(x,a),z)$.

Definición 16.1.7 Una **permutación** es una aplicación $\xi : V \to V$ biyectiva.

Lema 16.1.8 Para cada permutación ξ existe una única aplicación $\hat{\xi} : T \to T$ definida por:

$$\hat{\xi}(t) = \begin{cases} \xi(t), & \text{si } t \text{ es una variable;} \\ f\hat{\xi}(t_1), \ldots, \hat{\xi}(t_n), & \text{si } t = ft_1 \ldots t_n \end{cases}$$

En lo sucesivo, escribiremos $\xi(t)$ en lugar de $\hat{\xi}(t)$.

Lema 16.1.9 $t_1 \equiv t_2$ syss existe una permutación ξ tal que $\xi(t_1) = t_2$.

Nota 16.1.10 Si $t_1 \equiv t_2$ se dice que t_2 es una **variante** de t_1.

Lema 16.1.11 La relación $\equiv$ es de equivalencia en T.

Definición 16.1.12

1. Para cada témino t, representaremos por $[t]$ su clase de equivalencia; i.e.
$$[t] = \{t' \in T : t' \equiv t\}$$
2. Representaremos por $\mathbf{T}$ el conjunto cociente de T respecto de $\equiv$; i.e.
$$\mathbf{T} = \{[t] : t \in T\}$$

Definición 16.1.13 En $\mathbf{T}$ se define la relación

$$[t] \leq [t'] \text{ syss } t \leq t'$$

Lema 16.1.14 La relación $\leq$ es un orden parcial en $\mathbf{T}$.

Nota 16.1.15 El menor elemento de $\mathbf{T}$ es el conjunto de las variables.

Definición 16.1.16

1. El conjunto de variables de un término t se representa por $var(t)$.
2. El número de variables distintas de un término t se representa por $\nu(t)$.

3. El **tamaño** del término t se define por:

$$|t| = \begin{cases} 0, & \text{si } t \text{ es una variable;} \\ 1 + \sum_{i=1}^{n} |t_i|, & \text{si } t = t_1 \dots t_n \end{cases}$$

4. La aplicación $\mu : T \to T$ está definida por:

$$\mu(t) = |t| - \nu(t)$$

Definición 16.1.17

1. En T se define la relación

$$t_1 < t_2 \iff (t_1 \leq t_2) \wedge \neg(t_2 \leq t_1)$$

2. En $\mathbf{T}$ se define la relación

$$[t_1] < [t_2] \iff t_1 < t_2$$

Lema 16.1.18 Si $t_1 < t_2$, entonces $\mu(t_1) < \mu(t_2)$.

Lema 16.1.19

1. La relación $<$ está bien fundamentada en T; i.e. no existen sucesiones infinitas decrecientes.

2. La relación $<$ es un buen orden (estricto) en $\mathbf{T}$.

Definición 16.1.20 Sea φ una biyección entre $T \times T$ y V. Definimos la operación binaria $\cap$ en T por

$$t \cap t' = \begin{cases} f(t_1 \cap t'_1, \dots t_n \cap t'_n), & \text{si } t = f(t_1, \dots t_n) \text{ y } t' = f(t'_1, \dots t'_n) \\ \varphi(t, t'), & \text{en caso contrario} \end{cases}$$

Lema 16.1.21

1. $t \cap t' \leq t$ y $t \cap t' \leq t'$

2. $t'' \leq t \wedge t'' \leq t' \implies t'' \cap t \leq t'$

Definición 16.1.22 En **T** se define la operación

$$[t] \cap [t'] = [t \cap t']$$

Ejemplo 16.1.23 $[f(g(x,a),h(b))] \cap [f(g(y,c),d)] = [f(g(u,v),w)]$

Lema 16.1.24 En $(\mathbf{T}, <)$, $[t] \cap [t']$ es el ínfimo de t y t'.

Nota 16.1.25 Representaremos por $\mathcal{T}$ el conjunto obtenido añadiéndole a **T** un mayor elemento $\top$.

Teorema 16.1.26 $(\mathcal{T}, <)$ es un retículo completo.

Corolario 16.1.27 Si dos términos t y t' tiene una cota superior (i.e. una instancia común $\theta(t) = \theta'(t')$), entonces tienen un supremo, que se representa por $t \cup t'$ y es único módulo $\equiv$.

Ejemplo 16.1.28 $f(a,x_1,x_2) \cup f(x_3,x_3,x_4) \equiv f(a,a,x_5)$

16.2 Comparación de sustituciones

Definición 16.2.1 En el conjunto de las sustituciones se define la relación

$$\theta \leq \theta' \text{ syss existe una sustitución } \theta'' \text{ tal que } \theta''\theta = \theta'.$$

Si $\theta \leq \theta'$ se dice que θ es **menos particular** que θ'.

Lema 16.2.2 La relación $\leq$ es reflexiva y transitiva en el conjunto de las sustituciones.

Lema 16.2.3 Si **L** tiene símbolos de funciones, entonces

$$\theta \leq \theta' \text{ syss para todo término } t,\ \theta(t) \leq \theta'(t).$$

Definición 16.2.4 En el conjunto de las sustituciones se define la relación

$$\theta \equiv \theta' \iff \theta \leq \theta' \wedge \theta' \leq \theta$$

Si $\theta \equiv \theta'$ se dice que θ es **equivalente** a θ'.

Lema 16.2.5 $\theta \equiv \theta'$ syss existe una permutación ξ tal que $\xi\theta = \theta'$.

Lema 16.2.6 La relación $\equiv$ es de equivalencia en el conjunto de las sustituciones.

Lema 16.2.7 Si **L** tiene símbolos de funciones, entonces

$$\theta \equiv \theta' \text{ syss para todo término } t,\ \theta(t) \equiv \theta'(t).$$

16.3 Unificación

Definición 16.3.1 Sean t_1, t_2 dos términos.

1. Una sustitución θ es un **unificador** de t_1 y t_2 si $\theta(t_1) = \theta(t_1)$.
2. Representaremos por $u(t_1, t_2)$ el conjunto de los unificadores de t_1 y t_2.
3. t_1 y t_2 son **unificables** si $u(t_1, t_2) \neq \emptyset$.
4. θ es un **unificador de máxima generalidad** de t_1 y t_2 si $\theta \in u(t_1, t_2)$ y para todo $\theta' \in u(t_1, t_2)$, $\theta' \leq \theta$.
5. Representaremos por $umg(t_1, t_2)$ el conjunto de los unificadores de t_1 y t_2 de máxima generalidad.

Ejemplo 16.3.2

1. Los términos $t_1 = f(g(a), h(x))$ y $t_2 = f(y, y)$ no son unificables.
2. Los términos $t_1 = f(x, x)$ y $t_2 = f(y, g(y))$ no son unificables.
3. $t_1 = f(x, g(y))$ y $t_2 = f(a, z)$, entonces
   (a) $\theta_1 = \{(x, a), (z, g(b))\} \in u(t_1, t_2) - umg(t_1, t_2)$.
   (b) $\theta_2 = \{(x, a), (z, g(y))\} \in umg(t_1, t_2)$.
   (c) $\theta_3 = \{(x, a), (y, u), (z, g(u))\} \in umg(t_1, t_2) - \{\theta_2\}$

Lema 16.3.3

1. Si $\theta \in u(t_1, t_2)$, entonces $t_1 \cup t_2 \leq \theta(t_1)$.

2. Si $\theta \in umg(t_1, t_2)$, entonces $t_1 \cup t_2 \equiv \theta(t_1)$.

Lema 16.3.4

1. Si $\theta_1 \in u(t_1, t_2)$ y $\theta_1 \leq \theta_2$, entonces $\theta_2 \in u(t_1, t_2)$.
2. Si $\theta, \theta' \in umg(t_1, t_2)$, entonces $\theta \equiv \theta'$.

Definición 16.3.5 Sea $N = \{t_1, \ldots, t_n\}$ un conjunto finito de términos.

1. Una sustitución θ es un **unificador** de N si $\theta(t_1) = \ldots = \theta(t_n)$.
2. Representaremos por $u(N)$ el conjunto de los unificadores de N.
3. N es **unificable** si $u(N) \neq \emptyset$.
4. θ es un **unificador de máxima generalidad** de N si $\theta \in u(N)$ y para todo $\theta' \in u(N)$, $\theta' \leq \theta$.
5. Representaremos por $umg(N)$ el conjunto de los unificadores de N de máxima generalidad.

Definición 16.3.6

1. Una **ecuación** en T es un par ordenado de términos.
2. Utilizaremos los símbolos $E, E_1, E_2, \ldots$ para representar conjuntos finitos de ecuaciones.

Definición 16.3.7 Sea $E = \{(t_i, t_i') : 1 \leq i \leq n\}$ un conjunto finito de ecuaciones.

1. La sustitución θ es una **solución** de E si $\theta(t_i) = \theta(t_i')$ para $1 \leq i \leq n$.
2. El conjunto de las soluciones de E se representa por $s(E)$.
3. La sustitución θ es una **solución de máxima generalidad** de E si es una solución de E y para cualquier solución θ' de E, $\theta' \leq \theta$.
4. El conjunto de las soluciones de E de máxima generalidad se representa por $smg(E)$.

Lema 16.3.8

1. $u(t_1, t_2) = s(\{(t_1, t_2)\})$
2. $umg(t_1, t_2) = smg(\{(t_1, t_2)\})$
3. $u(\{t_1, \ldots, t_n\}) = s(\{(t_1, t_2), \ldots, (t_1, t_n)\})$
4. $umg(\{t_1, \ldots, t_n\}) = smg(\{(t_1, t_2), \ldots, (t_1, t_n)\})$

Definición 16.3.9 Sea E un conjunto finito de ecuaciones y θ una sustitución. Entonces,

$$\theta(E) = \{(\theta(t), \theta(t')) : (t, t') \in E\}$$

Algoritmo 16.3.10 (de simplificación)

Entrada: Un conjunto finito de ecuaciones, E, y una sustitución, θ.

Salida: `Sin solucion`, si $\theta(E)$ no tiene solución; una solución de $\theta(E)$ de máxima generalidad, en caso contrario.

Procedimiento: $Simpl(E, \theta)$

si $E = \emptyset$ **entonces devolver** θ y **parar**
si $E \neq \emptyset$ **entonces**
  hacer $(t, t') = car(E)$
    $E := E - \{(t, t')\}$
  si $t = ft_1 \ldots t_n$ y $t' = ft'_1 \ldots t'_n$,
    entonces $Simpl(E \cup \{(t_1, t'_1), \ldots, (t_n, t'_n)\}, \theta)$
  si $t = ft_1 \ldots t_n$, $t' = gt'_1 \ldots t'_m$ y $f \neq g$
    entonces devolver `Sin solucion` y **parar**
  si $t = x$ y $t' = x$,
    entonces $Simpl(E, \theta)$
  si t no es una variable y t' es una variable,
    entonces $Simpl(E \cup \{(t', t)\}, \theta)$
  si $t = x$, $x \in var(t')$ y $t' \neq x$
    entonces devolver `Sin solucion` y **parar**
  si $t = x$ y $x \notin var(t')$
    entonces $Simpl(\{(x, t)\}(E), \{(x, t)\}\theta)$

Algoritmo 16.3.11 (de unificación)

Entrada: Un conjunto finito de términos, $N = \{t_1, \ldots, t_n\}$.

Salida: `No unificable`, si N no es unificable; $\theta \in umg(N)$, en caso contrario.

Procedimiento:

Hacer $E := \{(t_1, t_2) \ldots (t_1, t_n)\}$
$\theta := Simpl(E, \emptyset)$
si $\theta =$ `Sin solucion` **devolver** `No unificable`
e.o.c devolver θ

Teorema 16.3.12 El algoritmo de unificación siempre termina y es correcto.

16.4 Unificación para fórmulas atómicas

Definición 16.4.1 Sean A_1, A_2 dos fórmulas atómicas.

1. Una sustitución θ es un **unificador** de A_1 y A_2 si $\theta(A_1) = \theta(A_1)$.
2. Representaremos por $u(A_1, A_2)$ el conjunto de los unificadores de A_1 y A_2.
3. A_1 y A_2 son **unificables** si $u(A_1, A_2) \neq \emptyset$.
4. θ es un **unificador de máxima generalidad** de A_1 y A_2 si $\theta \in u(A_1, A_2)$ y para todo $\theta' \in u(A_1, A_2)$, $\theta' \leq \theta$.
5. Representaremos por $umg(A_1, A_2)$ el conjunto de los unificadores de A_1 y A_2 de máxima generalidad.

Lema 16.4.2 Sean $A_1 = pt_1 \ldots t_n$ y $A_2 = qt'_1 \ldots t'_m$ dos fórmulas atómicas. Entonces,

1. $u(A_1, A_2) = \begin{cases} \emptyset, & \text{si } p \neq q; \\ u(\{(t_1, t'_1), \ldots, (t_n, t'_n)\}), & \text{en otro caso} \end{cases}$
2. $umg(A_1, A_2) = \begin{cases} \emptyset, & \text{si } p \neq q; \\ umg(\{(t_1, t'_1), \ldots, (t_n, t'_n)\}), & \text{en otro caso} \end{cases}$

Definición 16.4.3 Sea $N = \{A_1, \ldots, A_n\}$ un conjunto finito de fórmulas atómicas.

1. Una sustitución θ es un **unificador** de N si $\theta(A_1) = \ldots = \theta(A_n)$.
2. Representaremos por $u(N)$ el conjunto de los unificadores de N.
3. N es **unificable** si $u(N) \neq \emptyset$.

4. θ es un **unificador de máxima generalidad** de N si $\theta \in u(N)$ y para todo $\theta' \in u(N)$, $\theta' \leq \theta$.

5. Representaremos por $umg(N)$ el conjunto de los unificadores de N de máxima generalidad.

Algoritmo 16.4.4 (de unificación para dos átomos)

Entrada: Dos átomos $A_1 = pt_1 \dots t_n$ y $A_2 = qt_1 \dots t_m$.

Salida: `No unificables`, si A_1 y A_2 no son unificables; $\theta \in umg(A_1, A_2)$, en caso contrario.

Procedimiento: $Unif(A_1, A_2)$

si $p \neq q$ **entonces devolver** `No unificables`
e.o.c. hacer $\theta = Simpl(\{(t_1, t'_1), \dots, (t_n, t'_n)\}, \emptyset)$
si $\theta =$ `Sin solucion` **entonces devolver** `No unificables`
e.o.c. devolver θ

Algoritmo 16.4.5 (de unificación para conjuntos de átomos)

Entrada: Un conjunto finito de átomos, $N = \{A_1, \dots, A_n\}$)

Salida: `No unificable`, si N no es unificable; $\theta \in umg(N)$, en caso contrario.

Procedimiento: $Unif(N)$

si $n \leq 1$ **entonces devolver** $\emptyset$
e.o.c. hacer $\theta_1 = Unif(A_1, A_2)$
si $\theta_1 =$ `No unificables` **entonces devolver** `No unificable`
e.o.c. hacer $\theta_2 = Unif(\{\theta_1(A_3), \dots, \theta_1(A_n)\})$
si $\theta_2 =$ `No unificable` **entonces devolver** `No unificable`
e.o.c. devolver $\theta_2\theta_1$

Capítulo 17

Resolución en lógica de primer orden

17.1 Sistema de resolución

Definición 17.1.1 Sea C una cláusula.

1. La **complementaria** de C es
$$\overline{C} = \{\overline{L} : L \in C\}$$
2. La **parte positiva** de C es
$$C_+ = \{L \in A : L \text{ es positivo}\}$$
3. La **parte negativa** de C es
$$C_- = \{L \in A : L \text{ es negativo}\}$$
4. La **forma positiva** de C es
$$\|C\| = C_+ \cup \overline{C_-}$$

Definición 17.1.2

1. Representaremos por $var(C)$ el conjunto de las variables de la cláusula C; es decir,
$$var(C) = \{var(A) : A \in \|C\|\}$$

2. Las cláusulas C_1 y C_2 están **separadas** si $var(C_1) \cap var(C_2) = \emptyset$.

3. Las permutaciones ξ_1 y ξ_2 **separan** las cláusulas C_1 y C_2 si $\xi_1(C_1)$ y $\xi_2(C_2)$ están separadas.

Definición 17.1.3 La cláusula C es una **resolvente** de las cláusulas C_1 y C_2 si se verifican las siguientes condiciones:

1. Existen dos permutaciones ξ_1, ξ_2 que separan a las cláusulas C_1 y C_2.

2. Existen $D_1 \subseteq \xi_1(C_1)$ y $D_2 \subseteq \xi_2(C_2)$ no vacíos tales que $\|D_1 \cup \overline{D_2}\|$ es unificable. Sea $\theta \in umg(\|D_1 \cup \overline{D_2}\|)$

3. C es de la forma:

$$C = \theta((\xi_1(C_1) - D_1) \cup (\xi_2(C_2) - D_2))$$

Ejemplo 17.1.4 La cláusula $\{p(a, f(a))\}$ es una resolvente de $\{p(z, f(z)), p(z, a)\}$ y $\{\neg p(z, z), \neg p(z, x), \neg p(x, z)\}$.

Definición 17.1.5 Representaremos por $Res(C_1, C_2)$ el conjunto de las resolventes de C_1 y C_2.

Definición 17.1.6 Sea S un conjunto de cláusulas.

1. La sucesión $(C_1, \ldots, C_n)$ es una **deducción por resolución** a partir de S si para todo $i \in \{1, \ldots, n\}$ se verifica una de las siguientes condiciones:

   (a) $C_i \in S$.

   (b) existen $j, k < i$ tales que $C_i \in Res(C_j, C_k)$.

2. La cláusula C es **deducible por resolución** a partir de S, $S \vdash C$, si existe una deducción por resolución a partir de S, $(C_1, \ldots, C_n)$, tal que $C_n = C$.

3. La sucesión $(C_1, \ldots, C_n)$ es una **refutación por resolución** de S si es una deducción por resolución a partir de S y $C_n = \square$.

4. S es **refutable** si $S \vdash \square$.

Definición 17.1.7 Sea S un conjunto de cláusulas.

1. $Res(S) = S \cup (\bigcup\{Res(C_1, C_2) : C_1, C_2 \in S\})$.

2. La sucesión $(Res^n(S))_{n\geq 0}$ está definida por:

$$Res^0(S) = S$$
$$Res^{n+1} = Res(Res^n(S))$$

3. $Res^*(S) = \bigcup_{n\geq 0} Res^n(S)$

Lema 17.1.8 $S \vdash C$ syss $C \in Res^*(S)$.

17.2 Corrección y completitud de la resolución

Lema 17.2.1 Si $C \in Res(C_1, C_2)$, entonces $\{C_1, C_2\} \models C$.

Teorema 17.2.2 (de corrección)
Si $S \vdash \square$, entonces S es inconsistente.

Lema 17.2.3 (del ascenso)
Si C_1' y C_2' son instancias básicas de C_1 y C_2, respectivamente, y C' una resolvente de C_1' y C_2' (en el sentido proposicional). Entonces, existe una $C \in Res(C_1, C_2)$ tal que C' es una instancia de C.

Teorema 17.2.4 (de completitud)
Si S es inconsistente, entonces $S \vdash \square$.

Corolario 17.2.5 S es inconsistente syss $S \vdash \square$.

17.3 Reglas de simplificación

Definición 17.3.1 Una clásula C es una **tautología** si existe una instancia C' de C tal que $C'_+ \cap C'_- \neq \emptyset$.

Teorema 17.3.2 (Regla de tautología)
Si $C \in S$ es una tautología, entoces S es consistente syss $S - \{C\}$ es consistente.

Definición 17.3.3 L es un **literal puro** de S si $L \in \bigcup S$ y para toda sustitución θ, $\overline{L} \notin \theta(\bigcup S)$.

Teorema 17.3.4 (Regla de los literales puros)
Si L es un literal puro de S, entonces S es consistente syss $S - \{C \in S : L \in C\}$ es consistente.

Definición 17.3.5 La cláusula C **subsume** a la cláusula D si existe una sustitución θ tal que $\theta(C) \subset D$.

Teorema 17.3.6 (Regla de subsunción)
Sean $C, D \in S$. Si C subsume a D, entonces S es consistente syss $S - \{D\}$ es consistente.

17.4 Refinamientos de resolución

Nota 17.4.1 Los refinamientos de resolución estudiados para el caso proposicional son aplicables al caso de primer orden.

Capítulo 18

Programas lógicos: semántica declarativa

18.1 Programas lógicos

Definición 18.1.1 Una **cláusula definida** es una cláusula C que tiene exactamente un literal positivo; es decir, $|C_+| = 1$.

Definición 18.1.2 Sea $C = \{A, \neg B_1, \ldots, \neg B_n\}$ una cláusula definida.

1. Si $n = 0$, se dice que C es un **hecho** y se representa por

$$A \leftarrow$$

2. Si $n > 0$, se dice que C es un **regla** y se representa por

$$A \leftarrow B_1, \ldots, B_n$$

3. La **cabeza** de C es A y su **cuerpo** es $B_1, \ldots, B_n$.

Definición 18.1.3 Sea C una cláusula y F una fórmula cerrada. Se dice que C y F son **equivalentes**, $C \equiv F$, si C y F tienen los mismos modelos.

Lema 18.1.4 $A \leftarrow B_1, \ldots, B_n \equiv \forall(B_1 \wedge \ldots \wedge B_n \rightarrow A)$

Definición 18.1.5 Un **programa lógico** (o, simplemente, **programa**) es un conjunto finito de cláusulas definidas.

Ejemplo 18.1.6 El siguiente conjunto es un programa lógico (definiendo la suma):

$$suma(x, 0, x) \leftarrow$$
$$suma(x, s(y), s(z)) \leftarrow suma(x, y, z)$$

Definición 18.1.7 Un **objetivo** G es una cláusula negativa; es decir,

$$G = \{\neg B_1, \ldots, \neg B_n\}$$

Representaremos el objetivo G por

$$\leftarrow B_1, \ldots, B_n$$

Lema 18.1.8 $\leftarrow B_1, \ldots, B_n \equiv \neg\exists(B_1 \wedge \ldots \wedge B_n)$

Definición 18.1.9 Una **cláusula de Horn** es una cláusula definida o un objetivo.

Nota 18.1.10

1. Los símbolos $P, P_1, P_2, \ldots$ representarán programas.
2. Los símbolos $G, G_1, G_2, \ldots$ representarán objetivos.

Definición 18.1.11 Sea G el objetivo $\leftarrow A_1, \ldots, A_n$ y θ una sustitución.

1. $\tilde{G} = A_1 \wedge \ldots \wedge A_n$
2. $\theta(\tilde{G}) = \theta(A_1) \wedge \ldots \wedge \theta(A_n)$

Definición 18.1.12 Sea S un conjunto de cláusulas y F una fórmula cerrada.

1. F es **consecuencia semántica** de S, $S \models F$, si todos los modelos de S son modelos de F.
2. $S \cup \{F\}$ es **inconsistente** si ningún modelo de S es modelo de F.

Lema 18.1.13 Sea S un conjunto de cláusulas y F una fórmula cerrada. Entonces, $S \models F$ syss $S \cup \{\neg F\}$ es inconsistente.

Definición 18.1.14 Sea P un programa y G el objetivo $\leftarrow A_1, \ldots, A_n$.

1. La sustitución θ es una **respuesta** para $P \cup \{G\}$ si $D(\theta) \subseteq var(P \cup \{G\})$.

2. La sustitución θ es una **respuesta correcta** para $P \cup \{G\}$ si es una respuesta para $P \cup \{G\}$ y

$$P \models \forall(\theta(\tilde{G}))$$

Ejemplo 18.1.15 Si P es el programa del ejemplo anterior y G es el objetivo

$$\leftarrow suma(x, y, s(0))$$

entonces las sustituciones $\theta_1 = \{(x, 0), (y, s(0))\}$ y $\theta_2 = \{(x, s(0)), (y, 0)\}$ son respuestas correctas para $P \cup \{G\}$.

Lema 18.1.16 Sea P un programa, G un objetivo y θ una respuesta para $P \cup \{G\}$. Entonces, θ es una respuesta correcta para $P \cup \{G\}$ syss $P \cup \{\neg\forall(\theta(\tilde{G}))\}$ es inconsistente.

Definición 18.1.17 Sea P un programa y G un objetivo. Se dice que "no" es la respuesta correcta para $P \cup \{G\}$ si $P \cup \{G\}$ es consistente.

18.2 Modelos de Herbrand

Lema 18.2.1 Sea P un programa y G un objetivo. Entonces, $P \cup \{G\}$ es consistente syss tiene un modelo de Herbrand.

Lema 18.2.2 Si P es un programa y $B(P)$ es su base de Herbrand, entonces $B(P)$ es un modelo de Herbrand de P

Lema 18.2.3 Si P es un programa y $\{M_i : i \in I\}$ un conjunto no vacío de modelos de Herbrand de P, entonces $\bigcap_{i \in I} M_i$ es un modelo de Herbrand de P.

Definición 18.2.4 El **menor modelo de Herbrand** de un programa P es

$$M_P = \bigcap \{M : M \text{ es un modelo de Herbrand de } P\}$$

Teorema 18.2.5 $M_P = \{A \in B(P) : P \models A\}$.

Teorema 18.2.6 Sea P un programa, G un objetivo y θ una respuesta para $P \cup \{G\}$ tal que $\theta(G)$ es básica. Son equivalentes:

1. θ es correcta.
2. Si M es un modelo de Herbrand de P, entonces $M \models \theta(\tilde{G})$.
3. $M_P \models \theta(\tilde{G})$.

Capítulo 19

Programas lógicos: semántica de puntos fijos

19.1 Operadores y sus puntos fijos

Definición 19.1.1 Un conjunto parcialmente ordenado, $(L, \leq)$, es un **retículo completo** si para todo $X \subseteq L$, existe el supremo, $sup(X)$, y el ínfimo, $inf(X)$.

Ejemplo 19.1.2 Si S es un conjunto, entonces $(\mathbf{P}(S), \subseteq)$ es un retículo completo.

Lema 19.1.3 Si $(L, \leq)$ es un retículo completo, entonces tiene un mayor elemento, $\top$, y un menor elemento, $\bot$.

Definición 19.1.4 Sea $(L, \leq)$ un retículo completo.

1. La aplicación $T : L \to L$ es un **homomorfismo** si

$$(\forall x, y \in L)[x \leq y \to T(x) \leq T(y)]$$

2. El conjunto $X \subseteq L$ es **dirigido** si todos sus subconjuntos finitos están acotados superiormente por elementos de X.

3. La aplicación $T : L \to L$ es **continua** si

$$T(sup(X)) = sup(T(X))$$

para todo subconjunto dirigido X de L.

Lema 19.1.5 Sea $(L, \leq)$ un retículo completo y $T : L \to L$. Si T es continua, entonces es un homomorfismo.

Definición 19.1.6 Sea $(L, \leq)$ un retículo completo y $T : L \to L$.

1. $x \in L$ es un **punto fijo** de T si $T(x) = x$. Representaremos por $PF(T)$ el conjunto de los puntos fijos de T.
2. El **menor punto fijo** de T se representa por $min(PF(T))$.
3. El **mayor punto fijo** de T se representa por $max(PF(T))$.

Teorema 19.1.7 Sea $(L, \leq)$ un retículo completo y $T : L \to L$ un homomorfismo.

1. Existe un menor punto fijo de T y viene dado por

$$min(PF(T)) = inf\{x \in L : T(x) \leq x\} = inf\{x \in L : T(x) = x\}$$

2. Existe un mayor punto fijo de T y viene dado por

$$max(PF(T)) = sup\{x \in L : x \leq T(x)\} = sup\{x \in L : T(x) = x\}$$

Corolario 19.1.8 Sea $(L, \leq)$ un retículo completo y $T : L \to L$ un homomorfismo.

1. Si $a \in L$ y $T(a) \leq a$, entonces existe un punto fijo a' de T tal que $a' \leq a$.
2. Si $b \in L$ y $b \leq T(b)$, entonces existe un punto fijo b' de T tal que $b \leq b'$.

Definición 19.1.9 Sea $(L, \leq)$ un retículo completo y $T : L \to L$ un homomorfismo.

1. La **potencia ascendente** de T es la aplicación $T \uparrow: Ord \to L$ definida recursivamente por

$$T \uparrow \alpha = \begin{cases} \bot, & \text{si } \alpha = 0; \\ T(T \uparrow \beta), & \text{si } \alpha = \beta + 1; \\ sup\{T \uparrow \beta : \beta < \alpha\}, & \text{si } \alpha \text{ es límite} \end{cases}$$

2. La **potencia descendente** de T es la aplicación $T\downarrow$: $Ord \to L$ definida recursivamente por

$$T\downarrow\alpha = \begin{cases} \top, & \text{si } \alpha = 0; \\ T(T\downarrow\beta), & \text{si } \alpha = \beta + 1; \\ inf\{T\downarrow\beta : \beta < \alpha\}, & \text{si } \alpha \text{ es límite} \end{cases}$$

Lema 19.1.10 Sea $(L, \leq)$ un retículo completo y $T : L \to L$ un homomorfismo.

1. $T\uparrow\alpha \leq min(PF(T)) \leq max(PF(T)) \leq T\downarrow\beta$
2. $\alpha \leq \beta \quad \Longrightarrow \quad T\uparrow\alpha \leq T\uparrow\beta \leq T\downarrow\beta \leq T\downarrow\alpha$
3. Si existe un $\beta > \alpha$ tal que $T\uparrow\beta = T\uparrow\alpha$, entonces $T\uparrow\alpha$ es el menor punto fijo de T.
4. Si existe un $\beta > \alpha$ tal que $T\downarrow\beta = T\downarrow\alpha$, entonces $T\downarrow\alpha$ es el mayor punto fijo de T.
5. Existe un ordinal β_1 tal que para todo $\ggg \geq \beta_1$, $T\uparrow\ggg = min(PF(T))$.
6. Existe un ordinal β_2 tal que para todo $\ggg \geq \beta_2$, $T\downarrow\ggg = max(PF(T))$.

Definición 19.1.11 Sea $(L, \leq)$ un retículo completo y $T : L \to L$ un homomorfismo.

1. La **clausura ordinal ascendente** de T es

$$c.o.\uparrow(T) = min\{\alpha : T\uparrow\alpha = min(PF(T))\}$$

2. La **clausura ordinal descendente** de T es

$$c.o.\downarrow(T) = min\{\alpha : T\downarrow\alpha = max(PF(T))\}$$

Teorema 19.1.12 Si $(L, \leq)$ es un retículo completo y $T : L \to L$ es continua, entonces

$$min(PF(T)) = T\uparrow\omega$$

Corolario 19.1.13 Si $(L, \leq)$ es un retículo completo y $T : L \to L$ es continua, entonces

$$c.o.\uparrow(T) \leq \omega$$

19.2 El operador de consecuencia inmediata

Lema 19.2.1 Si P es un programa, entonces $(\mathbf{P}(B(P), \subseteq)$ es un retículo completo. Su menor elemento es $\emptyset$ y su mayor elemento es $B(P)$.

Definición 19.2.2 Para cada programa P, representaremos por $[P]$ el conjunto de sus instancias básicas.

Definición 19.2.3 Sea P un programa. Para cada interpretación de Herbrand, I, se define la interpretación $T_P(I)$ por

$$A \in T_P(I) \iff \text{existen átomos } B_1, \ldots, B_n \text{ tales que } \begin{cases} A \leftarrow B_1, \ldots, B_n \in [P] \\ \{B_1, \ldots, B_n\} \subseteq I \end{cases}$$

La aplicación $T_P : \mathbf{P}(B(P)) \to \mathbf{P}(B(P))$ se llama el **operador de consecuencia inmediata**.

Lema 19.2.4 Sea P un programa, I una interpretación de Herbrand y A un átomo. Son equivalentes:

1. $A \in T_P(I)$;
2. existe una sustitución θ y una cláusula $B \leftarrow B_1, \ldots, B_n$ de P tales que $A = \theta(B)$ y $\{\theta(B_1), \ldots, \theta(B_n)\} \subseteq I$.

Ejemplo 19.2.5 Sea P el programa

$$p(f(x)) \leftarrow p(x)$$
$$q(a) \leftarrow p(x)$$

1. Si $I_1 = B(P)$, entonces $T_P(I_1) = \{q(a)\} \cup \{p(f(t)) : t \in U(P)\}$
2. Si $I_2 = T_P(I_1)$, entonces $T_P(I_2) = \{q(a)\} \cup \{p(f(f(t))) : t \in U(P)\}$
3. Si $I_3 = \emptyset$, entonces $T_P(I_3) = \emptyset$.

Lema 19.2.6 La aplicación T_P es continua.

Lema 19.2.7 Sea P un programa e I una interpretación de Herbrand. Entonces I es un modelo del programa P syss $T_P(I) \subseteq I$.

Teorema 19.2.8 Sea P un programa. Entonces,

$$M_P = min(PF(T_P)) = T_P \uparrow \omega = \bigcup_{n \geq 0} T_P \uparrow n$$

Ejemplo 19.2.9 Sea P el programa

$$\begin{array}{l} q(b) \leftarrow \\ q(f(x)) \leftarrow q(x) \\ p(f(x)) \leftarrow p(x) \\ p(a) \leftarrow p(x) \\ r(c) \leftarrow r(x), q(x) \\ r(f(x)) \leftarrow r(x) \end{array}$$

1. $M_P = min(PF(T_P)) = \{q(f^n(b)) : n \in \omega\}$ y $c.o. \uparrow (T_P) = \omega$.
2. $max(PF(T_P)) = \{q(f^n(b)) : n \in \omega\} \cup \{p(f^n(a)) : n \in \omega\}$ y $c.o. \downarrow (T_P) = \omega 2$.

Capítulo 20

Programas lógicos: semántica procedural

20.1 Proceso de computación: La resolución SLD

Nota 20.1.1 Sea θ una sustitución y $\leftarrow A_1, \ldots, A_n$ un objetivo. Escribiremos

$$\leftarrow \theta(A_1, \ldots, A_n)$$

en lugar de

$$\leftarrow \theta(A_1), \ldots, \theta(A_n)$$

Definición 20.1.2 Sea P un programa lógico, $C = A \leftarrow B_1, \ldots, B_k$ una cláusula de P y $G = \leftarrow A_1, \ldots, A_n$ un objetivo. Se dice que G' es una **resolvente de** G **y** C **con umg** θ si se verifican las siguientes condiciones:

1. existe un $i \in \{1, \ldots, n\}$ tal que θ es un unificador de máxima generalidad de A_i y A;
2. $G' = \leftarrow \theta(A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_n)$.

En este caso, se dice que A_i es el **átomo seleccionado**.

Definición 20.1.3 Sea P un programa y G un objetivo. Se dice que

$$(G_0, G_1, G_2 \ldots; C_1, C_2, \ldots; \theta_1, \theta_2, \ldots)$$

es una **derivación SLD** (o, simplemente, **derivación**) de $P \cup \{G\}$ si

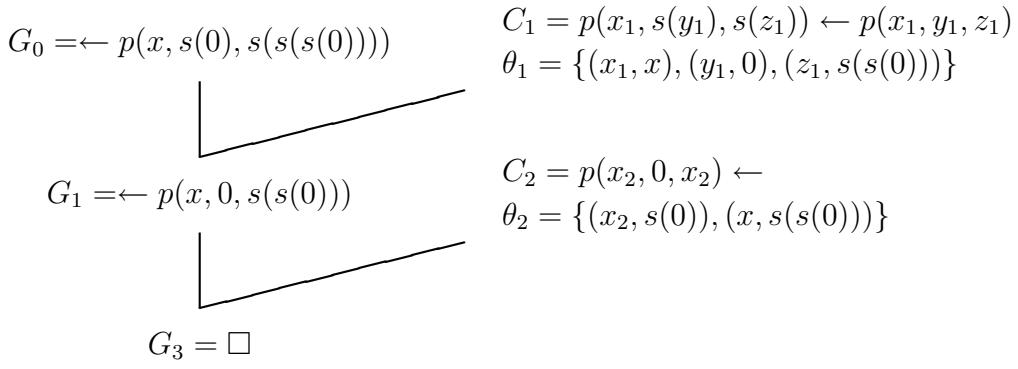
1. $G_0 = G$;
2. C_1 es una variante de una cláusula de P separada de G_0;
3. G_1 es una resolvente de G_0 y C_1 com umg θ_1;
4. para todo $i \geq 2$,
   (a) C_i es una variante de una cláusula de P;
   (b) C_i no tiene variables comunes con $G_0, \ldots, G_{i-1}, C_1, \ldots, C_{i-1}$.
   (c) G_i es una resolvente de G_{i-1} y C_i com umg θ_1;

Las cláusulas C_i se llaman **cláusulas de entrada**.

Ejemplo 20.1.4 Sea P el programa

$$p(x, 0, x) \leftarrow$$
$$p(x, s(y), s(z)) \leftarrow p(x, y, z)$$

y G es objetivo $\leftarrow p(x, s(0), s(s(s(0))))$. En la siguiente figura se muestra una derivación para $P \cup \{G\}$.



Nota 20.1.5 Las derivaciones pueden ser finitas o infinitas. Las finitas pueden terminar con éxito (si su último objetivo es $\square$) o con fallo (en caso contrario).

Definición 20.1.6 Una **refutación** de $P \cup \{G\}$ es una derivación finita de $P \cup \{G\}$

$$(G_0, G_1, G_2 \ldots, G_n; C_1, C_2, \ldots, C_n; \theta_1, \theta_2, \ldots, \theta_n)$$

tal que $G_n = \square$. Se dice que n es la **longitud** de la refutación.

Definición 20.1.7 Sea θ una sustitución y $V' \subseteq V$ un conjunto de variables. La **restricción** de θ a V' es la sustitución θ' definida por

$$\theta'(x) = \begin{cases} \theta(x), & \text{si } x \in V'; \\ x, & \text{en otro caso} \end{cases}$$

La restricción de θ a V' se representa por $\theta' \restriction V'$.

Definición 20.1.8 Sea P un programa y G un objetivo. La sustitución θ es una **respuesta computada** para $P \cup \{G\}$ si existe una refutación

$$(G_0, G_1, G_2 \ldots, G_n; C_1, C_2, \ldots, C_n; \theta_1, \theta_2, \ldots, \theta_n)$$

de $P \cup \{G\}$ y $\theta = (\theta_n \ldots \theta_1) \restriction var(G)$.

Ejemplo 20.1.9 La sustitución $\theta = \{(x, s(s(0)))\}$ es una respuesta computada para el ejemplo anterior.

Definición 20.1.10 El **conjunto de éxitos** de un programa P, E_P, es el conjunto de los $A \in B(P)$ tales que $P \cup \{\leftarrow A\}$ tiene una refutación.

20.2 Correción de la resolución SLD

Teorema 20.2.1 (Corrección computacional de la resolución SLD) Toda respuesta computada para $P \cup \{G\}$ es una respuesta correcta para $P \cup \{G\}$.

Corolario 20.2.2 (Corrección de la resolución SLD)
Si $P \cup \{G\}$ tiene una refutación, entonces $P \cup \{G\}$ es inconsistente.

Corolario 20.2.3 El conjunto de éxitos de un programa P está contenido en el menor modelo de Herbrand de P.

20.3 Completitud de la resolución SLD

Definición 20.3.1

1. Si en la definición de resolvente se sustituye la condición de "unificador de máxima generalidad" por la de "unificador", la cláusula que se obtiene se llama **resolvente no principal**.

2. Si en la definición de derivación se sustituye "resolvente" por "resolvente no principal", la sucesión obtenida se llama **derivación no principal**.

3. Si en la definición de refutación se sustituye "derivación" por "derivación no principal", la sucesión obtenida se llama **refutación no principal**.

Lema 20.3.2 Sea P un programa y G un objetivo. Si

$$(G_0, G_1, G_2 \ldots, G_n; C_1, C_2, \ldots, C_n; \theta_1, \theta_2, \ldots, \theta_n)$$

es una refutación no principal de $P \cup \{G\}$, entonces existe una refutación

$$(G_0, G'_1, G'_2 \ldots, G'_n; C'_1, C'_2, \ldots, C'_n; \theta'_1, \theta'_2, \ldots, \theta'_n)$$

de $P \cup \{G\}$ tal que $\theta'_n \ldots \theta'_1 \leq \theta_n \ldots \theta_1$.

Lema 20.3.3 (del ascenso)
Sea P un programa, G un objetivo y θ una sustitución. Si

$$(G_0, G_1, G_2 \ldots, G_n; C_1, C_2, \ldots, C_n; \theta_1, \theta_2, \ldots, \theta_n)$$

es una refutación de $P \cup \{\theta(G)\}$, entonces existe una refutación

$$(G_0, G'_1, G'_2 \ldots, G'_n; C'_1, C'_2, \ldots, C'_n; \theta'_1, \theta'_2, \ldots, \theta'_n)$$

de $P \cup \{G\}$ tal que $\theta'_n \ldots \theta'_1 \leq \theta_n \ldots \theta_1 \theta$.

Lema 20.3.4 El menor modelo de Herbrand de un programa P está contenido en el conjunto de éxitos de P.

Teorema 20.3.5 (de completitud de la resolución SLD)
Si $P \cup \{G\}$ es inconsistente, entonces existe una refutación de $P \cup \{G\}$.

Lema 20.3.6 Sea A un átomo. Si $P \models \forall(A)$, entonces la sustitución identidad es una respuesta computada para $P \cup \{\leftarrow A\}$.

Lema 20.3.7 Si θ es una respuesta correcta para $P \cup \{G\}$, entonces la sustitución identidad es una respuesta computada para $P \cup \{\theta(G)\}$.

Teorema 20.3.8 (de completitud computacional de la resolución SLD)
Si θ es una respuesta correcta para $P \cup \{G\}$, entonces existe una respuesta computada, θ', para $P \cup \{G\}$ tal que $\theta' \leq \theta$.

20.4 Reglas de computación

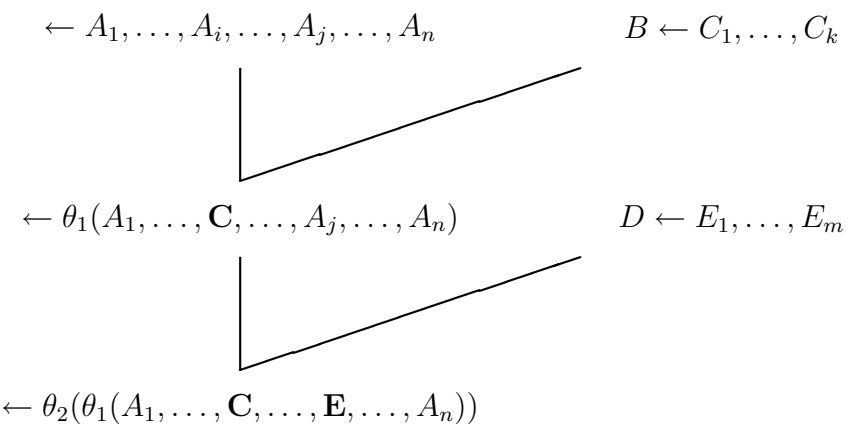
Definición 20.4.1 Sea P un programa lógico, $C = A \leftarrow B_1, \ldots, B_k$ una cláusula de P, $G = \leftarrow A_1, \ldots, A_n$ un objetivo y R una regla de computación. Se dice que G' es la **resolvente de G y C vía R con umg θ** si se verifican las siguientes condiciones:

1. existe un $i \in \{1, \ldots, n\}$ tal que $R(G) = A_i$ y θ es un unificador de máxima generalidad de A_i y A;
2. $G' = \leftarrow \theta(A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_n)$.

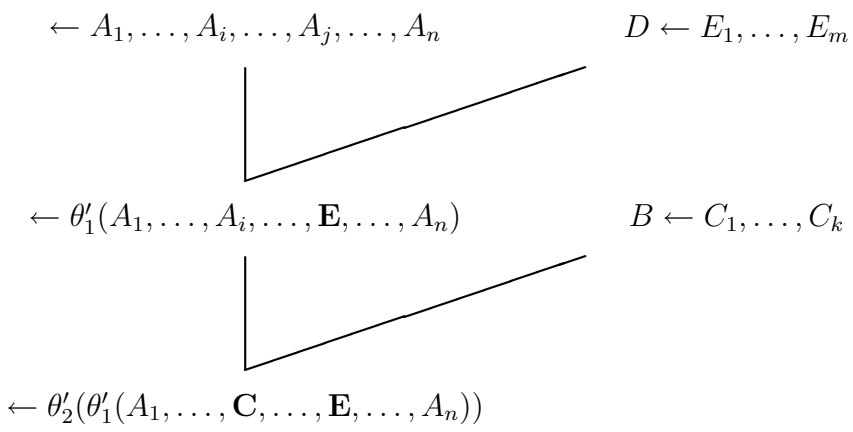
Definición 20.4.2 Sea P un programa, G un objetivo y R una regla de computación.

1. Una **derivación SLD** de $P \cup \{G\}$ **vía** R es una derivación de $P \cup \{G\}$ que usa resolventes vía R.
2. Una **refutación** de $P \cup \{G\}$ **vía** R es una derivación de $P \cup \{G\}$ que usa resolventes vía R.
3. Una **respuesta computada** de $P \cup \{G\}$ **vía** R es una respuesta computada para $P \cup \{G\}$ obtenida de una refutación de $P \cup \{G\}$ vía R.
4. El **conjunto de éxitos** de P **vía** R es el conjunto de los $A \in B(P)$ tales que $P \cup \{\leftarrow A\}$ tiene una refutación vía R.

Lema 20.4.3 (del intercambio)
Consideremos dos pasos sucesivos de una derivación



donde **C** representa a $C_1, \ldots, C_k$ y **E** representa a $E_1, \ldots, E_m$. Entonces existen dos sustituciones θ'_1, θ'_2 tales que



son dos pasos sucesivos de una derivación y $\theta'_2\theta'_1 \equiv \theta_2\theta_1$.

Teorema 20.4.4 (indenpendencia de la regla de computación) Si θ es una respuesta computada para $P \cup \{G\}$, entonces para cada regla de computación R existe una respuesta computada para $P \cup \{G\}$ vía R, θ', tal que $\theta'(G) \equiv \theta(G)$ (i.e. existe una permutación ξ de forma que $\theta'(G) = \xi(\theta(G))$.

Lema 20.4.5 El conjunto de los éxitos de P vía R es igual al menor modelo de Herbrand de P.

Teorema 20.4.6 (de completitud de la resolución SLD con reglas) Si $P \cup \{G\}$ es inconsistente, entonces existe una refutación de $P \cup \{G\}$ vía R.

Teorema 20.4.7 (de completitud fuerte de la resolución SLD) Si θ es una respuesta correcta para $P \cup \{G\}$, entonces existe una computada para $P \cup \{G\}$ vía R, θ', tal que $\theta' \leq \theta$.

20.5 Árboles SLD

Definición 20.5.1 Sean P un programa, G un objetivo y R una regla de computación. Un **árbol SLD** de $P \cup \{G\}$ vía R es un árbol verificando las siguientes condiciones:

1. Cada nodo del árbol es un objetivo.

2. El nodo raiz es G.

3. Los nodos que son la cláusula vacía no tienen hijos. Dichos nodos se llaman **nodos de éxito**. Las ramas de la raiz a los nodos de éxito se llaman **ramas de éxito**.

4. Sea $G' =\leftarrow A_1, \ldots, A_n$ $(n \geq 1)$ un nodo del árbol y $A_i = R(G')$. Entonces, para cada cláusula de P, $C = A \leftarrow B_1, \ldots, B_k$, tal que A y A_m son unificables, el nodo tiene un hijo

$$\leftarrow \theta(A_1, \ldots, A_{i-1}, B'_1, \ldots, B'_k, A_{i+1}, \ldots, A_n)$$

donde $A' \leftarrow B'_1, \ldots, B'_k$ es una variante de C separada de los antecesores de G' y θ es un unificador de máxima generalidad de A' y A_i. Si el nodo G' no tiene descendientes, se llama un **nodo de fallo** y las ramas de la raiz a G' se llaman **ramas de fallo**.

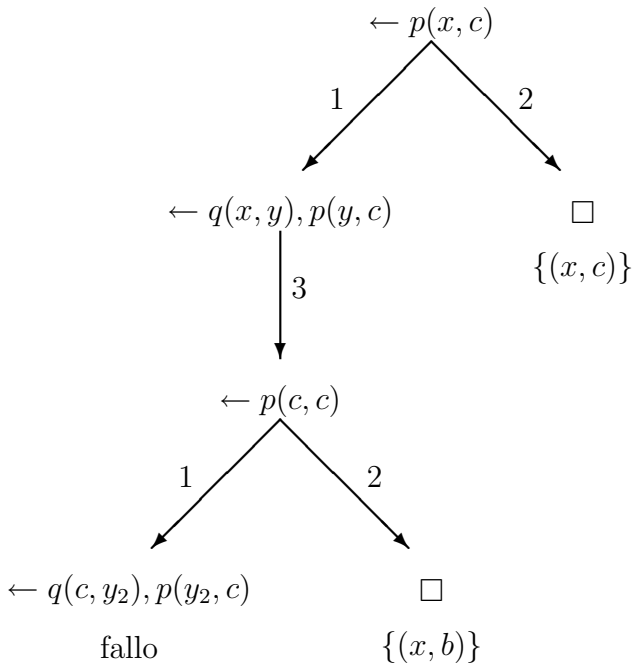
Nota 20.5.2 Cada rama del árbol representa una derivación de $P \cup \{G\}$ vía R.

Ejemplo 20.5.3 Sea P el programa

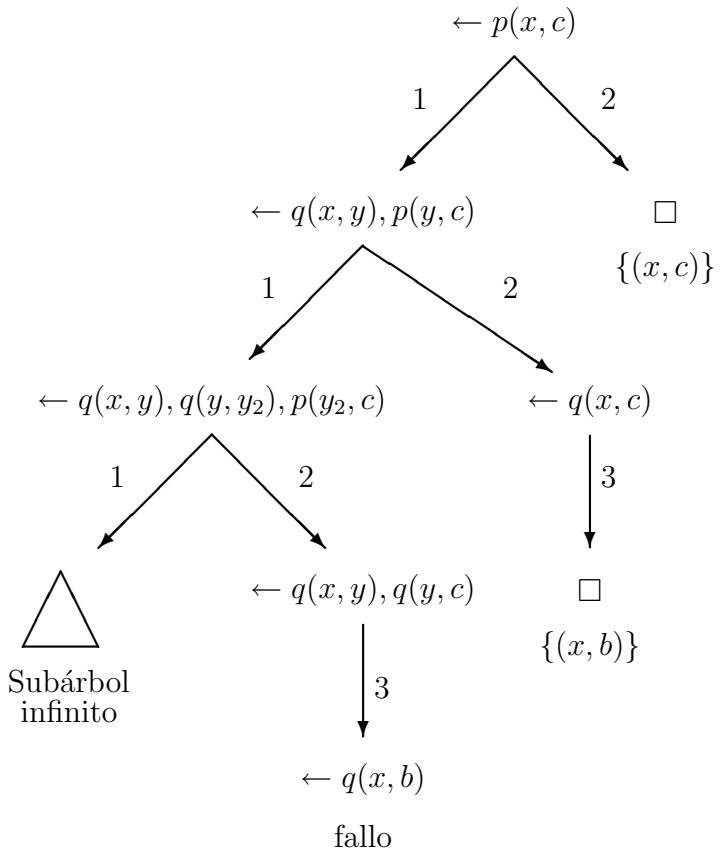
$$\begin{array}{l} p(x,z) \leftarrow q(x,y), p(y,z) \\ p(x,x) \leftarrow \\ p(b,c) \leftarrow \end{array}$$

y G el objetivo $\leftarrow p(x,c)$.

Si R la regla de computación por la izquierda (i.e., $R(\leftarrow A_1, \ldots, A_n) = A_1$), entonces el árbol SLD para $P \cup \{G\}$ vía R es:



Si R la regla de computación por la derecha (i.e., $R(\leftarrow A_1, \ldots, A_n) = A_n$), entonces el árbol SLD para $P \cup \{G\}$ vía R es:



Nótese que aunque el primer árbol es finito y el segundo es infinito, en ambos se obtienen las mismas respuestas.

Teorema 20.5.4 Si $P \cup \{G\}$ es inconsistente, entonces el árbol SLD de $P \cup \{G\}$ vía R tiene una rama de éxito.

Teorema 20.5.5 Si θ es una respuesta correcta de $P \cup \{G\}$, entonces en el árbol SLD de $P \cup \{G\}$ vía R existe una rama de éxito tal que la respuesta computada por la refutación que representa dicha rama, θ', es más general que θ (i.e. $\theta' \leq \theta$).

Teorema 20.5.6 Si P es un programa y A es un átomo básico, son equivalentes:

1. $P \models A$
2. $A \in M_P$

3. $A \in T_P \uparrow \omega$
4. $A \in E_P$
5. Todo árbol SLD para $P \cup \{\leftarrow A\}$ tiene una rama de éxito.

20.6 Procedimientos de refutación. La evaluación de Prolog

Definición 20.6.1 Una **regla de búsqueda** es una estrategia para encontrar ramas de éxitos en los árboles SLD.

Definición 20.6.2 Un **procedimiento de refutación** viene dado por una regla de computación y una regla de búsqueda.

Definición 20.6.3

1. La **regla de computación de Prolog** selecciona el primer átomo por la izquierda.
2. La **regla de búsqueda de Prolog** es una búsqueda en profundidad.

Algoritmo 20.6.4 (de evaluación de Prolog)

Entrada: Un programa $P = \{C_1, \ldots, C_n\}$ (escribiremos $C_i = B_i \leftarrow D_{i,1}, \ldots, D_{i,n_i}$), un objetivo $G = \leftarrow A_1, \ldots, A_k$ y una sustitución θ

Salida: Una respuesta computada para $P \cup \{\theta(G)\}$

Procedimiento: $Evaluar(P, G, \theta)$

si $G = \square$ **entonces devolver** θ
e.o.c. hacer $i := 0$
mientras que $(i < n)$
hacer $i := i + 1$
B'_i una variante de B_i
$\theta' := Unif(E_1, B'_i)$
si $\theta' \neq$ `No unificables`
entonces $Evaluar(P, \leftarrow \theta'(D_{i,1}, \ldots, D_{i,n_i}, A_2, \ldots, A_k), \theta'\theta)$

Nota 20.6.5 Dado un programa P y un objetivo G,

$$Evaluar(P, G, \emptyset)$$

devuelva las respuestas computadas por Prolog para $P \cup \{G\}$

Bibliografía

[1] Andrews, P.B., *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.

[2] Apt, K.R., Introduction to Logic Programming. En *Handbook of Theoretical Computer Science* (Ed. por J.V. Leeuwen). North–Holand, 1990.

[3] Bibel, W., *Automated Theorem Proving* (2nd ed.) Vieweg, 1982.

[4] Bledsoe, W.W.; Loveland, D.W. (eds.), *Automated Theorem Proving: After 25 Years*. Americam Mathematical Society, 1984.

[5] Boizumault, P., *Prolog: l'implantation*. Masson, 1988.

[6] Bundy, A., *The Computer Modelling of Mathematical Reasoning*. Academic Press, 1983.

[7] Chang, C–L; Lee, R. C–T., *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[8] Cuena, J., *Lógica informática*. Alianza, 1985.

[9] Dalen, D.V., *Logic and Structure*. (2nd ed.) Springer–Verlag, 1983.

[10] Davis, M.; Weyuker, E., *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, 1983.

[11] Delahaye, J–P., *Outils logiques pour l'intelligence artificielle*. Eyrolles, 1986.

[12] Deville, Y., *Logic Programming: Systematic Program Development*. Addison–Wesley, 1990.

[13] Ebbinghaus, H.D.; Flum, J.; Thomas, W. *Mathematical Logic*. Springer–Verlag, 1984.

[14] Enderton, H.B., *A Mathematical Introduction to Logic*. Academic Press, 1972.

[15] Farreny, H., *Ejercicios programados de inteligencia artificial*. Masson, 1988.

[16] Farreny, H.; Ghallab, M., *Éléments d'intelligence artificielle*. Hermes, 1987.

[17] Frost, R., *Bases de datos y sistemas expertos*. Díaz de Santos, 1989.

[18] Gallier, J.H., *Logic for Computer Science (Foundations of Automatic Theorem Proving)*. Harper & Row, 1986.

[19] Genesereth, M.R.; Nilsson, N.J., *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[20] Gibbins, P., *Logic with Prolog*. Clarendon Press, 1988.

[21] Hermes, H., *Introduction to Mathematical Logic*. Springer–Verlag, 1973.

[22] Huet, G., *Résolution d'equations dans des langages d'ordre* $1, 2, \ldots, \omega$. Thèse d'Etat, Univ. Paris VII, 1976.

[23] Huet, G., Deduction and Computation. En *Fundamentals of Artificial Intelligence: An Advanced Course)* (Ed. por W. Bibel y Ph. Jorrand), pp.39–74. LNCS 232. Springer–Verlag, 1987.

[24] Jorrand, Ph., Fundamentals Mechanisms for Artificial Intelligence Programming Languages. En *Advanced Topics in Artificial Intelligence* (Ed. por R.T. Nossum), pp. 1–40. LNAI, 345. Springer–Verlag, 1988.

[25] Kowalski, R., *Lógica, programación e inteligencia artificial*. Díaz de Santos, 1986.

[26] Lewis, H.R.; Papadimitriou, C.H., *Elements of the Theory of Computation*. Prentice–Hall, 1981.

[27] Lloyd, J.W., *Foundations of Logic Programming* (2nd ed.) Springer–Verlag, 1987.

[28] Loveland, D.W., *Automated Theorem Proving: A Logical Basis*. North–Holand, 1978.

[29] Maier, D.; Warren, D:S:, *Computing with Logic (Logic Programming with Prolog)*. Benjamin Cummings, 1988.

[30] Manna, Z., *Mathematical Theory of Computation*. McGraw–Hill, 1974.

[31] Manna, Z.; Waldinger, R., *The Logical Basis for Computer Programming (Vol. 1: Deductive Reasoning)*. Addison–Wesley, 1985.

[32] Manna, Z.; Waldinger, R., *The Logical Basis for Computer Programming (Vol. 2: Deductive Systems)*. Addison–Wesley, 1990.

[33] Mendelson, E., *Introduction to Mathematical Logic* (3 ed.) Wadsworth & Brooks, 1987.

[34] Nilsson, N.J., *Problem–Solving Methods in Artificial Intelligence*. McGraw–Hill, 1971.

[35] Nilsson, N.J., *Principles of Artificial Intelligence*. Springer–Verlag, 1982.

[36] Pabion, J.–F., *Logique mathématique*. Hermann, 1766.

[37] Plaisted, D.A., Mechanical Theorem Proving. En *Tormal Techniques in Artificial Intelligence: A Sourcebook*. (Ed. por R.B. Banerji), pp. 269–320. North–Holand, 1990.

[38] Quine, W.V., *Los métodos de la lógica*. Ariel, 1981.

[39] Ramsay, A., *Formal Methods in Artificial Intelligence*. Cambridge University Press, 1988.

[40] Rich, E., *Artificial Intelligence*. McGraw–Hill, 1983.

[41] Richards, T., *Clausal Form Logic: An Introduction to the Logic of Computer Reasoning*. Addison–Wesley, 1989.

[42] Robinson, J.A., *Logic: Form and Function (The Mechanization of Deductive Reasoning)*. Edinburgh University Press, 1979.

[43] Sancho, J., *Lógica, Matemática y Computabilidad*. Díaz de Santos, 1990.

[44] Schagrin, M.L.; Rapaport, W.J.; Dipert, R.R. *Logic: A Computer Approach*. McGraw–Hill, 1985.

[45] Schöning, U., *Logic for Computer Scientists*. Birkhuser, 1989.

[46] Shoenfield, J.R., *Mathematical Logic*. Addison–Wesley, 1967.

[47] Siekmann, J.; Wrightson, G. (edts.) *Automatisation of Reasoning.* Springer–Verlag, 1983.

[48] Thayse, A. y otros. *Aproche logique de l'Intelligence Artificielle. (Vol 1: de la logique classique à la programmation logique).* Dunod, 1988.

[49] Thayse, A. y otros. *Aproche logique de l'Intelligence Artificielle. (Vol 2: de la logique modale à la logique des bases de données).* Dunod, 1989.

[50] Winston, P.H., *Artificial Intelligence (2nd ed.)* Addison–Wesley, 1984.

[51] Wos, L., *Automated Reasoning: 33 Basic Research Problems.* Prentice Hall, 1988.

[52] Wos, L.; Overbeek, R.; Lusk, E.; Boyle, J., *Automated Reasoning: Introduction and Applications.* Prentice Hall, 1984.