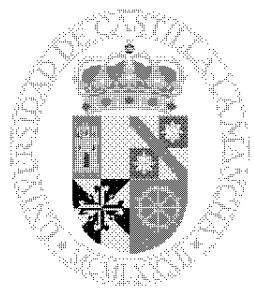


Asignatura de Lógica: Prácticas

Departamento de Tecnologías y Sistemas de Información
Universidad de Castilla-La Mancha.



Primera versión:	Otoño de 2008
Primera revisión:	Primavera de 2010
Segunda versión:	Otoño de 2011
Segunda revisión:	Otoño de 2012

Índice general

1. BOOLE: conceptos semánticos en la lógica proposicional.	5
1.1. Introducción.	5
1.2. Objetivos.	5
1.3. Descripcion.	6
2. EL MUNDO DE TARSKI: conceptos semánticos en la lógica de predicados.	17
2.1. Introducción.	17
2.2. Lenguajes de primer orden y el mundo de Tarski.	18
2.3. Objetivos.	20
2.4. Descripcion.	20
3. PROLOG: la lógica de predicados como lenguaje programación.	37
3.1. Introducción.	37
3.2. El lenguaje PROLOG.	38
3.2.1. Sintaxis.	38
3.2.2. Semántica.	39
3.2.3. Modo de operación.	42
3.3. Objetivos.	43
3.4. Descripcion.	44

Capítulo 1

BOOLE: conceptos semánticos en la lógica proposicional.

1.1. Introducción.

Esta práctica tiene como objetivo general afianzar conceptos semánticos de gran importancia relativos a la lógica proposicional así como adquirir habilidad en la representación formal de enunciados. Con tales fines se utiliza el *software* de aplicación BOOLE [1], que facilita la construcción de tablas de verdad. La construcción de tablas de verdad en esta aplicación se realiza en tres pasos: escribir la forma enunciativa (o las formas enunciativas) que se desea analizar; construir las columnas iniciales (columnas de referencia); llenar la tabla con los valores de verdad. Una vez que se ha completado una tabla de verdad se puede utilizar para realizar un análisis (*assessment*) de las propiedades semánticas de la forma enunciativa (o las formas enunciativas).

1.2. Objetivos.

Esta práctica persigue los siguientes objetivos concretos:

1. Conocer el entorno de trabajo de la aplicación BOOLE.
2. Familiarizar al alumno en la escritura de formas enunciativas. Saber distinguir cuál es la conectiva principal de una forma enunciativa.
3. Que el alumno adquiera habilidad en la construcción de tablas de verdad.
4. Que el alumno sea capaz de utilizar las tablas de verdad para clasificar enunciados o determinar si existen relaciones de equivalencia lógica o consecuencia lógica entre los mismos.

5. Saber como aplicar el método de las tablas de verdad y cuándo puede resultar útil.
6. Introducir al alumno en las técnicas de representación del conocimiento mediante el empleo del lenguaje formal de la lógica.

1.3. Descripción.

Para la correcta realización de esta práctica se aconseja la lectura previa de los apartados relativos a la semántica de la lógica de proposiciones que aparecen en alguno de los libros recomendados como bibliografía básica: [4, 5, 8]. El guión propuesto es el siguiente:

Parte I.

Ejercicio 1.1 Mediante la documentación y las explicaciones facilitadas por el profesor ejecutar y probar el entorno de trabajo de la aplicación BOOLE, familiarizándose con los efectos de las diferentes opciones de la barra de menú y la barra de herramientas. Concretando, utilizar las opciones apropiadas de la barra de menú y la barra de herramientas para construir las tablas de verdad de las formas enunciativas siguientes:

1. $\mathcal{A} \rightarrow \neg\mathcal{B}$
2. $\mathcal{A} \wedge \neg\mathcal{A}$
3. $(\mathcal{A} \wedge \mathcal{B}) \rightarrow (\mathcal{A} \vee \mathcal{B})$

analizar si son contingencia, tautología o contradicción

Ejercicio 1.2 Analice la relación de consecuencia lógica entre las premisas y la conclusión del siguiente argumento:

$$\{\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C}), \neg\mathcal{D} \vee \mathcal{A}\} \vdash \mathcal{D} \rightarrow \mathcal{C}.$$

Haga uso del método de las tablas de verdad.

Parte II.

Una vez comprendido el funcionamiento de la aplicación BOOLE, cada grupo (formado por dos alumnos) deberá realizar los ejercicios propuestos en uno de los siguientes bloques, según asignación expresa del profesor. Algunos de los ejercicios de esos bloques se han extraído de los libros [6, 7, 11].

Dado un bloque, por cada uno de los ejercicios resueltos se generará un fichero con la solución (usando la opción “save as” del menú *File*). Los nombres de los ficheros deben ajustarse al siguiente formato:

XXX_B_NNN.EXT

donde, **XXX** son las iniciales del nombre y apellidos del alumno responsable del grupo, **B** es la letra del bloque, **NNN** es el número del ejercicio y **EXT** la extensión que por defecto genera la aplicación. Estos ficheros se comprimirán formando un archivo ZIP y se enviarán para su corrección usando Moodle. El nombre del archivo ZIP debe ajustarse al siguiente formato:

GGG_APELLIDOS_P1_B.zip

donde, **GGG** es el código del grupo, **APELLIDOS**, los apellidos del alumno responsable del grupo, **P1**, es el código de la práctica y **B**, es la letra del bloque. Ejemplo:

g1_Botella_Blanco_P2_B.zip

Bloque A.

Ejercicio 1.3 *Construir la tabla de verdad de la forma enunciativa siguiente: $((p \rightarrow q) \rightarrow p) \rightarrow q$. Indique el conector principal y analice si es contingencia, tautología o contradicción*

Ejercicio 1.4 *Formalíce el siguiente enunciado de la lógica de proposiciones:*

No es el caso que 7 sea mayor que 5 y 5 mayor que 3.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.5 *Compruébe si las formas enunciativas $(A \wedge B \rightarrow C)$ y $(A \rightarrow \neg B \vee C)$ son lógicamente equivalentes, realizando sus tablas de verdad.*

Ejercicio 1.6 *Formalíce el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:*

Si no hay un control de nacimientos, entonces la población crece ilimitadamente. Pero si la población crece ilimitadamente, aumentará el índice de pobreza. Por consiguiente, si no hay control de nacimientos, aumentará el índice de pobreza.

Bloque B.

Ejercicio 1.7 *Construir la tabla de verdad de la forma enunciativa siguiente: $((p \rightarrow \neg r) \vee r)$. Indique el conector principal y analice si es contingencia, tautología o contradicción*

Ejercicio 1.8 *Formalíce el siguiente enunciado de la lógica de proposiciones:*

Cuando quiera que me lo pidas lo haré.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.9 Compruébe si las formas enunciativas $(p \rightarrow q)$ y $\neg(p \wedge \neg q)$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.10 Formalíce el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si haces deporte, entonces tendrás buen tipo y disfrutarás de buena salud. No tienes buen tipo. Luego, no haces deporte.

Bloque C.

Ejercicio 1.11 Construir la tabla de verdad de la forma enunciativa siguiente: $((p \leftrightarrow \neg q) \vee r) \rightarrow q$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.12 Formalíce el siguiente enunciado de la lógica de proposiciones:

Juan no trabaja a menos que gane dinero o suceda una catástrofe.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.13 Compruébe si las formas enunciativas $(p \rightarrow q)$ y $(\neg p \vee q)$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.14 Formalíce el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si vamos de paseo, no vamos al cine. Si vamos al cine, no vamos de paseo. Luego, no se da el caso de que vayamos de paseo y al cine.

Bloque D.

Ejercicio 1.15 Construir la tabla de verdad de la forma enunciativa siguiente: $(p \wedge (p \rightarrow r)) \rightarrow (p \rightarrow \neg q \vee r)$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.16 Formalíce el siguiente enunciado de la lógica de proposiciones:

La buena marcha de las empresas fue condición necesaria y suficiente para que aumentase su cuenta de resultados.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.17 Compruebe si las formas enunciativas $(p \rightarrow q)$ y $(\neg q \rightarrow \neg p)$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.18 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si trabajas y estudias, apruebas. Si tienes voluntad, estudias. Por tanto, si tienes buena voluntad, entonces, si trabajas, apruebas.

Bloque E.

Ejercicio 1.19 Construir la tabla de verdad de la forma enunciativa siguiente: $(p \wedge (p \rightarrow r)) \rightarrow (p \wedge (\neg q \vee r))$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.20 Formalice el siguiente enunciado de la lógica de proposiciones:

Si un número entero es primo, no es compuesto, aunque es divisible por la unidad y, consiguientemente, también es divisible por números primos.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.21 Compruebe si las formas enunciativas $\neg(p \vee q)$ y $(\neg p \wedge \neg q)$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.22 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si trabajas, ganas dinero. Por tanto, si trabajas, ganas dinero o vas de vacaciones.

Bloque F.

Ejercicio 1.23 Construir la tabla de verdad de la forma enunciativa siguiente: $(A \rightarrow B) \vee (B \rightarrow A)$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.24 Formalice el siguiente enunciado de la lógica de proposiciones:

Si una recta es paralela a otra recta contenida en un plano, entonces o está contenida en el plano o es paralela al plano.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.25 Compruébe si las formas enunciativas $\neg(p \wedge q)$ y $(\neg p \vee \neg q)$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.26 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si eres feliz, estarás alegre y si estas alegre, no lloraras. Si estás triste, entonces lloraras. Pero eres feliz o estas triste. Por tanto, estarás alegre o lloraras.

Bloque G.

Ejercicio 1.27 Construir la tabla de verdad de la forma enunciativa siguiente: $((A \rightarrow B) \rightarrow A) \rightarrow A$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.28 Formalice el siguiente enunciado de la lógica de proposiciones:

Si una recta es paralela a otra recta contenida en un plano, entonces o está contenida en el plano o es paralela al plano.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.29 Compruébe si las formas enunciativas $(p \wedge (q \vee r))$ y $((p \wedge q) \vee (p \wedge r))$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.30 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Apruebas si estudias. Estudias si tienes interés. Por tanto, apruebas si tienes interés.

Bloque H.

Ejercicio 1.31 Construir la tabla de verdad de la forma enunciativa siguiente: $\neg(A \wedge \neg B) \rightarrow (B \rightarrow A)$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.32 Formalice el siguiente enunciado de la lógica de proposiciones:

La mayoría de los peces son ovíparos, sin embargo algunos tiburones son vivíparos.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.33 Compruebe si las formas enunciativas $(p \wedge (p \vee q))$ y p son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.34 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Apruebas sólo si estudias. Estudias sólo si tienes interés. Por tanto, apruebas sólo si tienes interés.

Bloque I.

Ejercicio 1.35 Construir la tabla de verdad de la forma enunciativa siguiente: $(A \rightarrow B) \vee A$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.36 Formalice el siguiente enunciado de la lógica de proposiciones:

Al menos Juan o Pedro llegaron a tiempo, pero María se retrasó.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.37 Compruebe si las formas enunciativas $(p \vee (q \wedge r))$ y $((p \vee q) \wedge (p \vee r))$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.38 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si estás en la romería, entonces no trabajas. Si estás en la fábrica, entonces trabajas. Por tanto, no se da el caso de que estés en la romería y en la fábrica.

Bloque J.

Ejercicio 1.39 Construir la tabla de verdad de la forma enunciativa siguiente: $(A \rightarrow C) \wedge (B \rightarrow C) \wedge \neg C \rightarrow \neg A \vee \neg B$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.40 Formalice el siguiente enunciado de la lógica de proposiciones:

Trabajes o no trabajes te has de ganar el dinero necesario para subsistir.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.41 Compruebe si las formas enunciativas $(p \vee (p \wedge q))$ y p son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.42 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Un hombre que no corrige sus errores, se hace culpable de engaño.

Si corrige sus errores, será acusado de contradicción. Pero o corrige sus errores o no. Por eso, o será culpable de engaño o será acusado de contradicción.

Bloque K.

Ejercicio 1.43 Construir la tabla de verdad de la forma enunciativa siguiente: $\mathcal{A} \wedge (\mathcal{A} \rightarrow \mathcal{B}) \wedge \neg \mathcal{B}$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.44 Formalice el siguiente enunciado de la lógica de proposiciones:

Si 2 es menor que 3 y 3 es menor que 5, 2 es menor que 5.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.45 Compruebe si las formas enunciativas $\mathcal{A} \rightarrow (\mathcal{B} \wedge \mathcal{C})$ y $\mathcal{A} \wedge \mathcal{B} \rightarrow \mathcal{C}$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.46 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Triunfa quien no desea la victoria. No es el caso que no deseé la victoria o pelee con valentía. Por eso, no triunfa.

Bloque L.

Ejercicio 1.47 Construir la tabla de verdad de la forma enunciativa siguiente: $\neg(\mathcal{A} \wedge \mathcal{B}) \wedge (\mathcal{C} \rightarrow \neg \mathcal{A})$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.48 Formalice el siguiente enunciado de la lógica de proposiciones:

O bien 17 es un número primo y par o bien 17 no es un número primo ni par.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.49 Compruebe si las formas enunciativas $\mathcal{A} \rightarrow (\mathcal{A} \wedge \mathcal{B})$ y $\neg \mathcal{B} \rightarrow \neg \mathcal{A}$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.50 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si no hay un control de nacimientos, entonces la población crece ilimitadamente. Pero si la población crece ilimitadamente, aumentará el índice de pobreza. Por consiguiente, si no hay control de nacimientos, aumentará el índice de pobreza.

Bloque M.

Ejercicio 1.51 Construir la tabla de verdad de la forma enunciativa siguiente: $(\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C}) \wedge \mathcal{A} \wedge \mathcal{B} \wedge \neg \mathcal{C})$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.52 Formalice el siguiente enunciado de la lógica de proposiciones:

Obtienes un sobresaliente en el examen parcial pero no haces todos los ejercicios del libro.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.53 Compruebe si las formas enunciativas $\mathcal{A} \rightarrow (\mathcal{B} \wedge \mathcal{A})$ y $\mathcal{B} \wedge (\neg \mathcal{B} \rightarrow \mathcal{A})$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.54 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si los jóvenes socialistas alemanes apoyan a Brandt, entonces renuncian a su programa de reivindicaciones. Y si no apoyan a Brandt, entonces favorecen a Strauss. Por consiguiente, habrán de renunciar a su programa de reivindicaciones o favorecer a Strauss.

Bloque N.

Ejercicio 1.55 Construir la tabla de verdad de la forma enunciativa siguiente: $\mathcal{A} \wedge \neg \mathcal{B} \wedge \neg \mathcal{C} \wedge \neg (\mathcal{B} \rightarrow \mathcal{C})$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.56 Formalice el siguiente enunciado de la lógica de proposiciones:

Obtienes un sobresaliente en el examen final si haces todos los ejercicios del libro y obtienes previamente un sobresaliente en el examen parcial.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.57 Compruebe si las formas enunciativas $(p \rightarrow r) \vee \neg(p \rightarrow r)$ y $(p \rightarrow r) \wedge p \rightarrow r$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.58 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Nadie distinto de Juan, Pedro o Luis está implicado. Juan no trabaja nunca sin contar por lo menos con un cómplice. Luis no es culpable. Por tanto, Pedro es culpable.

Bloque O.

Ejercicio 1.59 Construir la tabla de verdad de la forma enunciativa siguiente: $\mathcal{A} \vee \neg\mathcal{B} \vee \neg\mathcal{C} \rightarrow (\neg\mathcal{A} \wedge \mathcal{B})$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.60 Formalice el siguiente enunciado de la lógica de proposiciones:

Obtener un sobresaliente en el examen parcial y hacer todos los ejercicios del libro es condición necesaria para tener un sobresaliente en el examen final.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.61 Compruebe si las formas enunciativas $\mathcal{B} \rightarrow \mathcal{C} \vee \mathcal{A}$ y $\neg\mathcal{C} \wedge (\mathcal{A} \rightarrow \mathcal{B})$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.62 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si Antonio cometió el robo, entonces también lo cometió su amigo de la infancia Juan. Juan nunca se meten un trabajo peligroso sin llevar con él a su hermano Luis. Antonio confesó haber participado en el robo. Por consiguiente, Luis participó.

Bloque P.

Ejercicio 1.63 Construir la tabla de verdad de la forma enunciativa siguiente: $(\neg p \leftrightarrow \neg q) \rightarrow (p \leftrightarrow q)$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.64 Formalíce el siguiente enunciado de la lógica de proposiciones:

Para cursar matemáticas discretas debes haber cursado cálculo o un curso de informática teórica.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.65 Compruébe si las formas enunciativas $p \wedge q \rightarrow p$ y $\neg(p \rightarrow q) \rightarrow \neg q$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.66 Formalíce el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Hoy es el cumpleaños de Pablo o de Miguel. Si es el cumpleaños de Miguel, recibirá una máquina fotográfica. Miguel no recibirá una máquina fotográfica. Por consiguiente, hoy es el cumpleaños de Pablo.

Bloque Q.

Ejercicio 1.67 Construir la tabla de verdad de la forma enunciativa siguiente: $(\neg p \leftrightarrow \neg q) \vee (p \rightarrow q)$. Indique el conector principal y analice si es contingencia, tautología o contradicción

Ejercicio 1.68 Formalíce el siguiente enunciado de la lógica de proposiciones:

Los manzanos florecerán si llueve en primavera y el clima se mantiene templado durante más de una semana.

Hágase explícito el vocabulario empleado e indíquese el conector principal de la forma enunciativa resultante. Construya su tabla de verdad y analice si es contingencia, tautología o contradicción.

Ejercicio 1.69 Compruébe si las formas enunciativas $p \rightarrow q \vee p$ y $\neg(p \rightarrow q) \rightarrow p$ son lógicamente equivalentes, realizando sus tablas de verdad.

Ejercicio 1.70 Formalice el siguiente argumento de la lógica de proposiciones y analice la relación de consecuencia lógica entre las premisas y la conclusión haciendo uso del método de las tablas de verdad:

Si María estudia francés entonces también estudia latín o español.
María no estudia latín. María estudia francés o latín o español. Por consiguiente, María estudia español.

Capítulo 2

EL MUNDO DE TARSKI: conceptos semánticos en la lógica de predicados.

2.1. Introducción.

Esta práctica tiene como objetivo general afianzar conceptos semánticos de gran importancia relativos a la lógica predicados así como adquirir habilidad en la representación formal de enunciados en este tipo de lógica. Con tales fines se utiliza el *software* de aplicación “El mundo de Tarski” [1], que presenta mundos tridimensionales compuestos por bloques geométricos de varias clases y tamaños y comprueba la verdad o falsedad de sentencias¹ de la lógica de predicados interpretadas en esos mundos.

Este software presenta las siguientes ventanas para interactuar con el usuario:

- La ventana del mundo (*world window*): contiene un tablero de ajedrez en el que se sitúan los bloques y, a la izquierda, tres botones mostrando un tetraedro, un cubo y un dodecaedro; estos botones son útiles para añadir nuevos bloques en un nuevo mundo o en un mundo ya creado.
- La ventana de las sentencias (*sentence window*): es el lugar en el que se introducen las sentencias para su posterior evaluación en la interpretación que representa el mundo definido en la ventana del mundo.
- La ventana del teclado (*keyboard window*): se utiliza para introducir sentencias.
- La ventana de inspección (*inspector window*): es el lugar en el que se muestra el resultado de la evaluación de las sentencias y donde se puede

¹Una *sentencia* es un fórmula cerrada, esto es, sin variables libres. Las sentencias, cuando se interpretan, se convierten en enunciados declarativos relativos a un dominio de discurso.

cambiar el tamaño y la forma de las figuras.

2.2. Lenguajes de primer orden y el mundo de Tarski.

El lenguaje de primer orden de la lógica de predicados se caracteriza por poseer unos símbolos comunes a todos los formalismos (variables, conectivas y signos de puntuación) y otros símbolos peculiares (símbolos de constante, función y relación). En realidad, más que hablar de “el lenguaje de primer orden” deberíamos hablar de “lenguajes de primer orden”. Diferentes versiones del lenguaje de primer orden utilizan símbolos diferentes de constante, función y relación.

En este apartado se presenta un lenguaje de primer orden diseñado para describir bloques dispuestos sobre un tablero de ajedrez, que pueden crearse mediante el programa “el mundo de Tarski”. Este lenguaje posee variables, como **u**, **v**, **w**, **x**, **y** y **z**, constantes, como **a**, **b**, **c**, **d**, **e**, **f** y **n2**, y predicados como **Cube**, **Larger**, y **Between**. Algunos ejemplos de fórmulas atómicas en este lenguaje son: **Cube(b)**, **Larger(c, f)**, y **Between(b, c, d)**. Estas fórmulas atómicas dicen respectivamente, que **b** es un cubo, que **c** es más grande que **f**, y que **b** está entre **c** y **d**. Así pues una de las principales características que distingue a este lenguaje de primer orden de otros comúnmente estudiados es, aparte de los símbolos peculiares utilizados, su condición de lenguaje interpretado.

El mundo de Tarski asigna a cada uno de sus predicados una interpretación fija que se ajusta de forma razonable y consistente al significado empleado en la lengua inglesa para esos nombres de relación. Sin embargo pueden existir ligeras discrepancias entre el uso común del término en lenguaje natural y el significado asignado en el mundo de Tarski. Esto es debido a que a estos símbolos, contrariamente a lo que sucede en el lenguaje natural, se les asigna una interpretación muy precisa, que sugiere el significado pero no siempre contempla toda la complejidad del predicado empleado en el lenguaje natural. El caso en el que esa discrepancia resulta mayor es, probablemente, entre el significado del predicado **Between** y el significado usual de la palabra inglesa “between”. Los ejercicios que se proponen más adelante arrojarán luz sobre este punto.

Para conocer el conjunto de los símbolos de relación con los que vamos a tratar y ayudar a comprender el significado de los mismos, la Tabla 2.1 proporciona un listado de fórmulas atómicas (que usan esos símbolos de relación) junto con sus interpretaciones.

Nótese que, en la lógica de primer orden podemos tener símbolos de relación de cualquier aridad. Sin embargo, en el lenguaje de bloques utilizado en el mundo de Traski solamente existen predicados con aridades 1, 2, y 3. Tampoco en el mundo de Traski existen símbolos de función.

Por otra parte, el lenguaje del mundo de Tarski es un lenguaje con identidad. Incluye el símbolo de relación infijo **=** que se interpreta como la identidad. La identidad, posee cuatro propiedades de gran importancia que la caracterizan:

Fórmula atómica	Interpretación
Tet(a)	a es un tetraedro
Cube(a)	a es un cubo
Dodec(a)	a es un dodecaedro
Small(a)	a es pequeño
Medium(a)	a es de tamaño medio
Large(a)	a es grande
SameSize(a, b)	a es del mismo tamaño que b
SameShape(a, b)	a tiene la misma forma que b
Larger(a, b)	a es más grande que b
Smaller(a, b)	a es más pequeño que b
SameCol(a, b)	a está en la misma columna que b
SameRow(a, b)	a está en la misma fila que b
Adjoins(a, b)	a y b están colocados en casillas adyacentes (pero no en la misma diagonal)
LeftOf(a, b)	a está colocado más cerca del lado izquierdo del tablero que b
RightOf(a, b)	a está colocado más cerca del lado derecho del tablero que b
FrontOf(a, b)	a está colocado más cerca de la parte delantera del tablero que b
BackOf(a, b)	a está colocado más cerca de la parte trasera del tablero que b
Between(a, b, c)	a, b y c están en la misma fila, columna, o diagonal, y a está entre b y c
$a = b$	a es idéntico a b

Cuadro 2.1: Predicados del lenguaje de bloques.

1. Indescernibilidad de los idénticos: Si $b = c$ entonces cualquier propiedad que posea b la posee c .
2. Reflexibilidad: Siempre es cierto que $b = b$.
3. Simetría: Si $b = c$ entonces $c = b$.
4. Transitividad: Si $a = b$ y $b = c$ entonces $a = c$.

Las dos últimas propiedades se siguen de las dos primeras.

Antes de abandonar esta sección es preciso indicar que para facilitar la escritura de fórmulas bien formadas más legibles, podrán emplearse los corchetes (“[”, “]”) y las llaves (“{”, “}”), que en el mundo de Tarski se tratan de forma equivalente a los paréntesis. Por ejemplo, `[Cube(b) ∧ Larger(b, c)]` se entiende como `(Cube(b) ∧ Larger(b, c))`.

2.3. Objetivos.

Esta práctica persigue los siguientes objetivos concretos:

1. Conocer el entorno de trabajo de la aplicación “El mundo de Tarski”.
2. Familiarizar al alumno con la escritura de fórmulas bien formadas. Saber distinguir entre fórmulas existenciales y universales.
3. Que el alumno adquiera habilidad en la interpretación y evaluación de fórmulas.
4. Que el alumno sea capaz de distinguir cuándo una fórmula es verdadera o falsa en una interpretación.
5. Saber realizar pruebas de independencia, esto es, búsqueda de contraejemplos.
6. Que el alumno adquiera habilidad en la traducción de enunciados, representados en lenguaje natural, a fórmulas de la lógica de predicados.

2.4. Descripcion.

Para la correcta realización de esta práctica se aconseja la lectura previa de los apartados relativos a la semántica de la lógica de predicados que aparecen en alguno de los libros recomendados como bibliografía básica: [4, 8]. También se recomienda la lectura del manual de usuario que describe la aplicación el mundo de Tarski [1] (que se suministra a través de Moodle). El guión propuesto es el siguiente:

Parte I.

En esta primera parte, se trata de ejecutar y probar el entorno de trabajo de la aplicación “El mundo de Tarski”, familiarizándose con los efectos de las diferentes opciones de la barra de menú, la barra de herramientas y las diferentes ventanas. Para ello, a partir de la documentación y las explicaciones facilitadas por el profesor, deben de resolverse una serie de ejercicios, de relativa simplicidad, que también sirven para cubrir algunos de los objetivos concretos listados en el Apartado 2.3.

Ejercicio 2.1 *En este ejercicio nos familiarizaremos con la interpretación de sentencias atómicas del lenguaje de bloques. Para ello proceder con los siguientes puntos:*

1. *Ejecute el mundo de Tarski y abra los ficheros denominados “Wittgenstein’s World” y “Wittgenstein’s Sentences”. Estos ficheros se encuentran en la carpeta “TW Exercises” y contienen la descripción de un mundo de bloques y una lista de sentencias atómicas (con comentarios añadidos a algunas de las sentencias; los comentarios van precedidos de un signo de punto y coma, “;”, que ordena a la aplicación que ignore el resto de la línea.)*
2. *Desplácese a través de las sentencias, usando las flechas del teclado, evaluando mentalmente el valor de verdad de cada sentencia en el mundo de Wittgenstein. Utilice el botón “Verify” para comprobar si su asignación es correcta. Si le sorprende alguna de las evaluaciones, trate de imaginar como su interpretación del predicado difiere de la interpretación correcta.*
3. *Modifique el mundo de Wittgenstein de diferentes maneras y vea que sucede con el valor de verdad de algunas sentencias. El interés aquí es comprender como el mundo de Tarski interpreta esos predicados. Por ejemplo, que significa `BackOf(d, c)`? Piensa que dos objetos deben de estar en la misma columna para que estén uno detrás del otro?. Por otra parte, compruebe que el predicado “Between” no tiene el mismo significado que la palabra “entre” del lenguaje natural. Por simplicidad, para que un objeto esté entre otros dos, los tres deben estar alineados en la misma fila, columna o diagonal.*
4. *En el mundo original, ninguna de las sentencias relativas al predicado “Adjoins” es verdadera. Intente modificar el mundo para que algunas de ellas sea verdadera.*
5. *Una vez que se haya terminado, cierre los ficheros sin salvar los cambios realizados.*

Ejercicio 2.2 *Este ejercicio proporciona práctica con la ventana del teclado del mundo de Tarski, así como con la sintaxis y escritura de sentencias atómicas. Cree un nuevo fichero de sentencias y copie en él las sentencias que se relacionan*

más adelante. La aplicación el mundo de Tarski comprueba cada fórmula después de haber sido escrita para ver si está bien formada. Si se comete un error, corrijalo antes de seguir adelante. Asegurarse de que se emplea el comando “Add Sentence” para añadir cada sentencia, no la tecla retorno de carro. Si se hace esto correctamente, las sentencias de la lista aparecerán numeradas y separadas por líneas horizontales.

1. Tet(a)	4. Cube(c)	8. Larger(a, b)
2. Medium(a)	5. FrontOf(a, b)	9. Smaller(a, c)
3. Dodec(b)	6. Between(a, b, c)	10. LeftOf(b, c)
7. a = d		

Salve las sentencias en un fichero denominado “Sentencias_2.2” que se entregará comprimido en un archivo ZIP junto con otros ficheros.

Ejercicio 2.3 Construya un mundo en el que todas las sentencias del Ejercicio 2.2 sean simultáneamente verdaderas. Salve las sentencias en un fichero denominado “Sentencias_2.3” y el mundo construido en un fichero denominado “Mundo_2.3”, que se entregará comprimido en un archivo ZIP junto con otros ficheros.

Ejercicio 2.4 Cree un nuevo fichero de sentencias y traduzca al lenguaje de la lógica de predicados los siguientes enunciados:

1. a es un cubo.	7. e es un dodecaedro.
2. b es más pequeño que a.	8. e está a la derecha de b.
3. c está entre a y d.	9. a es más pequeño que e.
4. d es grande.	10. d está detrás de a.
5. e es más grande que a.	11. b está en la misma fila que d.
6. b es un tetraedro.	12. b es del mismo tamaño que c.

Después de traducir los enunciados, construya un mundo en el que todas las fórmulas traducidas sean ciertas. Salve las sentencias y el mundo en dos ficheros denominados “Sentencias_2.4” y “Mundo_2.4” que se entregarán comprimidos en un archivo ZIP junto con otros ficheros.

Ejercicio 2.5 Abra los ficheros “Lestrade’s Sentences” y “Lestrade’s World”. Nótese que ninguno de los objetos en ese mundo tiene nombre. La tarea consiste en asignar nombres a los objetos de forma que todas las sentencias de la lista se conviertan en verdaderas. Salve la solución en un fichero denominado “Mundo_2.5”. Asegúrese de que se usa la opción “Save World As” y no la opción “Save World”, para impedir que se altere el fichero original.

En el mundo de Tarski se puede jugar un juego, denominado “juego de Henkin-Hintikka”², que consiste en analizar las razones por las que una fórmula es verdadera o falsa, teniendo como contrincante al propio ordenador. Para concretar, supóngase que la fórmula es $\mathcal{A} \wedge \mathcal{B}$. La forma en la que el juego

²En honor de los lógicos Leon Henkin y Jaakko Hintikka.

evoluciona depende de si se afirma que es verdadera o es falsa. Si se afirma que la fórmula es verdadera es porque se supone que tanto \mathcal{A} como \mathcal{B} lo son. Así que el ordenador selecciona una de esas fórmulas más simples e interroga sobre la verdad de ella (Cuál de ellas elige? depende. Si hay una fórmula falsa, elige la falsa, de forma que él pueda ganar el juego. Si ambas son verdaderas, elige una de ellas aleatoriamente, esperando que el jugador se equivoque cuando llegue su turno.). Si se afirma que la fórmula $\mathcal{A} \wedge \mathcal{B}$ es falsa, es porque se supone que al menos \mathcal{A} o \mathcal{B} es falsa. En este caso, el ordenador pedirá que se elija una de ellas y se afirme explícitamente su falsedad. Si el jugador elige una que no es falsa, pierde el juego.

La virtud del juego de Henkin-Hintikka radica en descubrir las razones por las que una sentencia es verdadera o falsa. Emplearemos este juego como un medio alternativo para determinar el significado de las conectivas.

Ejercicio 2.6 1. Abra el mundo “Claire’s World”. Cree un fichero de sentencias nuevo e introduzca la sentencia:

$$\neg\text{Cube}(a) \wedge \neg\text{Cube}(b) \wedge \neg\text{Cube}(c)$$

Nótese que esta sentencia es falsa porque, en ese mundo, c es un cubo.

2. Juegue el juego de Henkin-Hintikka afirmando (erróneamente) que la sentencia es verdadera y vea que sucede.
3. Ahora, juegue pero afirmando (acertadamente) que la sentencia es falsa y vea que sucede. Advierta que se puede perder el juego aunque la elección inicial fuese correcta, ya que, posteriormente, se puede hacer una mala elección a lo largo del juego.
4. Cuando haya terminado, salve el fichero de sentencias como “Sentencias_2.6”.

Las reglas del juego cuando se trata con cuantificadores son más interesantes que para las funciones de verdad de las conectivas. Con las conectivas, los movimientos que se realizan al desarrollar el juego conllevan la elección de partes de la sentencia que se nos ha asignado. Cuando se trata con cuantificadores, las reglas del juego obligan a elegir objetos (de un universo de discurso) en lugar de sentencias. Por ejemplo, suponga, que está comprometido con la verdad de $(\exists x)P(x)$, entonces para mantener ese compromiso deberá buscar la existencia de un objeto, x , que satisfaga $P(x)$. Por el contrario, si está comprometido con la falsedad de $(\exists x)P(x)$, deberá comprobar que ningún objeto, x , satisface $P(x)$. Las reglas para el generalizador son justo las contrarias.

Ejercicio 2.7 Abra los ficheros “Game World” y “Game Sentences”.

1. Analice cada sentencia y vea si es verdadera o falsa. Compruebe su propia evaluación.

- Independientemente de que haya acertado o no, por cada sentencia, juegue dos veces, una afirmando su verdad y otra su falsedad. Asegúrese de entender como funciona el juego en cada paso.

En este ejercicio no se requiere salvar fichero alguno, lo único que se pretende es que se entienda el funcionamiento del juego de Henkin-Hintikka.

Ejercicio 2.8 Suponga que está evaluando la siguiente sentencia en un mundo con cuatro cubos alineados en la primera fila del tablero:

$$(\forall x)(\forall y)[(Cube(x) \wedge Cube(y)) \rightarrow (LeftOf(x, y) \vee RightOf(x, y))]$$

- Piensa que la sentencia es verdadera en ese mundo?.
- Abra los ficheros “Cantor’s Sentences” y “Cantor’s World” y evalúe la sentencia. Si está sorprendido por la respuesta, juegue el juego de Henkin-Hintikka afirmando la verdad de la sentencia.
- Si realmente queremos expresar que “todo cubo está a la izquierda o a la derecha de cualquier otro cubo” (entendiendo por otro cubo un cubo diferente de él mismo), debemos modificar la sentencia original de la forma:
 $(\forall x)(\forall y)[(Cube(x) \wedge Cube(y) \wedge x \neq y) \rightarrow (LeftOf(x, y) \vee RightOf(x, y))]$
Ahora, introduzca la nueva sentencia y evalúela en ese mundo.
- Modifique el mundo borrando todos los cubos salvo uno. Juegue el juego de Henkin-Hintikka afirmando la verdad de la sentencia, para ver que sucede.
- Salve el fichero de sentencias como “Sentencias_2.8”.

En general, para demostrar la incorrección de un argumento, con un conjunto de premisas $\Gamma = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ y conclusión \mathcal{C} , debe buscarse una interpretación (esto es, una posible circunstancia) que haga todas las premisas verdaderas y la conclusión falsa. Tal interpretación muestra que \mathcal{Q} no es consecuencia lógica de Γ o, lo que es lo mismo, \mathcal{Q} es independiente de Γ . Este tipo de demostración se denomina *prueba de independencia*. Para el mundo de los bloques, una prueba formal de que \mathcal{Q} es independiente de Γ , consiste en un fichero de sentencias, con $\mathcal{A}_1, \dots, \mathcal{A}_n$ etiquetadas como premisas y \mathcal{C} etiquetado como conclusión, y un fichero de mundo, que hace verdaderas las premisas y falsa la conclusión. El mundo descrito en el fichero de mundo se denomina el *contraejemplo* para el argumento del fichero de sentencias.

Ejercicio 2.9 Abra el fichero de sentencias “Bill’s Argument”.

- El argumento, contenido en ese fichero, dice que `Between(b,a,d)` se sigue de estas premisas: `Between(b,c,d)`, `Between(a,b,d)`, y `Left(a,c)`. Es eso cierto?
- Cree un nuevo mundo y ponga cuatro bloques, etiquetados como *a*, *b*, *c*, y *d*, sobre una fila del tablero.

3. Reordene los bloques de forma que la conclusión sea falsa. Comprobar las premisas. Si alguna es falsa, reordenar los bloques hasta que todos sean ciertos. La conclusión es todavía falsa? si no, siga intentándolo.
4. Si tiene problemas, inténtelo poniéndolos en el orden: d, a, b, c. Ahora puede comprobar que todas las premisas son verdaderas y la conclusión falsa. Este mundo es un contraejemplo al argumento. Así, se ha demostrado que la conclusión es independiente de las premisas; es decir, la conclusión no se sigue de las premisas.
5. Salve el contraejemplo en un fichero denominado “contraejemplo_2.9”.

Parte II.

Una vez comprendido el funcionamiento de la aplicación “El mundo de Tarski”, cada grupo (formado por dos alumnos) deberá realizar los ejercicios propuestos en uno de los siguientes bloques, según asignación expresa del profesor. La mayoría de los ejercicios de esos bloques se han extraído del libro [4].

Bloque A.

Ejercicio 2.10 Abra el fichero “Abelard’s Sentences” y evalúelas en el mundo “Wittgenstein’s World”.

1. Si comete un error, juegue el juego de Henkin-Hintikka para ver la razón por la que está equivocado.
2. Una vez analizadas todas las sentencias, seleccione las falsas y cambie los nombres (constantes) utilizados en las sentencias, tratando que se conviertan en verdaderas.

Salve las sentencias como “Sentencias_2.24”

Dado el significado de los predicados en el lenguaje de los bloques, razonar sobre la validez³ de los siguientes argumentos (es posible suponer cualquier hecho general acerca de los mundos que puedan construirse con la aplicación el mundo de Tarski). Si el argumento se piensa que es válido, dar una prueba informal e incluirla en el fichero “resp_adicionales.txt”. Si la conclusión no es una consecuencia de las premisas, enviar un contraejemplo en el fichero denominado “contraejemplo_XXX”, donde XXX es el número del ejercicio.

Ejercicio 2.11 $\{Large(a), Larger(a, c)\} \vdash Small(c)$

Ejercicio 2.12 $\{Adjoins(a, b) \wedge Adjoins(b, c), SameRow(a, c)\} \vdash a \neq c.$

Ejercicio 2.13 $\vdash (\forall x)SameSize(x, x)$

³Observe que aquí “validez” puede entenderse en el sentido menos restrictivo de “verdadero en todo mundo de Tarski”.

Ejercicio 2.14 $\vdash \neg(\exists z)Small(z) \leftrightarrow (\exists z)\neg Small(z)$

Ejercicio 2.15 $\{\neg(\exists x)Tet(x)\} \vdash (\forall x)(Cube(x) \leftrightarrow \neg Dodec(x))$

Ejercicio 2.16 Traduzca los siguientes enunciados al lenguaje formal de la lógica de predicados (utilizando el lenguaje de bloques):

1. Si a es un tetraedro entonces está en frente de d .
2. a está a la izquierda o a la derecha de d solamente si es un cubo.
3. c está entre a y e o a y d .
4. c está a la derecha de a , siempre que él sea pequeño.
5. c está a la derecha de d solamente si b está a la derecha de c y a la izquierda de e .

Antes de salvar el fichero de sentencias como “Sentencias_2.30”, realice algunas comprobaciones:

- Abra el mundo “Bolzano’s World”. Todos los enunciados en ese mundo son verdaderos, así que si la formalización no es errónea y es fiel al sentido de esos enunciados, las sentencias formalizadas deberán evaluarse a verdadero. Si se ha cometido algún error de formalización corríjase.

(Observación: Aunque las traducciones se evalúen a verdadero en el mundo de Bolzano, eso no garantiza que sea una traducción adecuada. Si la traducción es correcta, tiene que tener el mismo significado (valor) que los enunciados castellanos en cualquier otro mundo. Por tanto, siempre conviene comprobar como se comportan las sentencias formalizadas en otros mundos.)

- Abra el mundo “Wittgenstein’s World”. Aquí, los enunciados 3 y 5 son falsos, mientras que el resto son verdaderos. Compruebe que lo mismo sucede con sus traducciones. Si no, corrija sus traducciones (asegurándose de que todavía son verdaderas en el mundo de Bolzano).
- Ahora abra el mundo “Leibniz’s World”. Aquí, los enunciados verdaderos son el 1, 2, y el 4, y los falsos son el 3 y el 5. Actúe como en el caso anterior.
- Finalmente, abra el mundo “Venn’s World”. En ese mundo, todos los enunciados son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, actúe en consecuencia.

Ejercicio 2.17 Abra el mundo “Montague’s Sentences”. Este fichero contiene expresiones que están a medio camino de ser formalizadas. Lea los enunciados de más abajo, entendiendo como se ha llegado a ese estado de formalización, intente completar su formalización sustituyendo las partes incompletas por fórmulas de la lógica de predicados.

1. Todo cubo está a la izquierda de cualquier tetraedro.
2. Todo cubo pequeño está detrás de un cubo grande.
3. Algún cubo está en frente de cualquier tetraedro.
4. Un cubo grande está en frente de un cubo pequeño.
5. Nada es más grande que cualquier cosa.

Antes de salvar el fichero de sentencias como “Sentencias_2.31”, realice algunas comprobaciones:

- Abra el mundo “Peirce’s World”. Todos los enunciados en ese mundo son verdaderos. Compruebe que las formalizaciones son también verdaderas cuando se interpretan en ese mundo. Si no lo son, trate de averiguar dónde se ha equivocado.
- Abra el mundo “Leibniz’s World”. Ahora el enunciado 5 es verdadero, mientras que el resto son falsos. Verifique que sus traducciones tienen el mismo valor de verdad cuando se evalúan. Si no es así corrija sus traducciones para ajustarse al sentido de los enunciados.
- Abra el mundo “Ron’s World”. Aquí los enunciados verdaderas son el 2, 3, 4 y 5. Proceda de forma similar a los casos anteriores.

Bloque B.

Ejercicio 2.18 Abra el fichero de sentencias “Carnap’s Sentences” y el mundo “Bolzano’s World”.

1. Interprete cada sentencia, produciendo enunciados (en castellano), y verifique si es verdadera en el mundo de Bolzano.
2. Utilice el juego de Henkin-Hintikka para evaluar las sentencias sobre las que albergue dudas.
3. Para cada sentencia, decida si es una fórmula lógicamente verdadera, verdadera en todo mundo de Tarski o falsa en algún mundo de Tarski. Para aquellas sentencias que sean falsificables, construya un mundo en el que la sentencia se interprete como falsa y salvarlo como “contraejemplo_2.32-X”, donde X el número de la sentencia. (**Ayuda:** Hay tres sentencias que son lógicamente verdaderas y tres falsificables.)

Describa las razones por las que se toman esas decisiones en el fichero de respuestas adicionales, “resp_adicionales.txt”.

Dado el significado de los predicados en el lenguaje de los bloques, razonar sobre la validez⁴ de los siguientes argumentos (es posible suponer cualquier

⁴Observe que aquí “validez” puede entenderse en el sentido menos restrictivo de “verdadero en todo mundo de Tarski”.

hecho general acerca de los mundos que puedan construirse con la aplicación el mundo de Tarski). Si el argumento se piensa que es válido, dar una prueba informal e incluirla en el fichero “resp_adicionales.txt”. Si la conclusión no es una consecuencia de las premisas, enviar un contraejemplo en el fichero denominado “contraejemplo_XXX”, donde XXX es el número del ejercicio.

Ejercicio 2.19

$$\{ \text{SameRow}(b, c), \text{SameRow}(a, d), \text{SameRow}(d, f), \text{LeftOf}(a, b) \} \vdash \text{LeftOf}(f, c)$$

Ejercicio 2.20 $\vdash \neg(\text{Cube}(b) \wedge b = c) \vee \text{Cube}(c).$

Ejercicio 2.21

$$\{ \neg(\exists x) \text{Larger}(x, a), \neg(\exists x) \text{Larger}(b, x), \text{Larger}(c, d) \} \vdash \text{Larger}(a, b)$$

Ejercicio 2.22 $\vdash ((\forall x) \text{Cube}(x) \vee (\forall x) \text{Dodec}(x)) \leftrightarrow (\forall x) (\text{Cube}(x) \vee \text{Dodec}(x))$

Ejercicio 2.23 $\{(\forall x) (\text{Cube}(x) \leftrightarrow \text{SameShape}(x, c))\} \vdash \text{Cube}(c)$

Ejercicio 2.24 Traduzca los siguientes enunciados al lenguaje formal de la lógica de predicados (utilizando el lenguaje de bloques):

1. Si **e** es un tetraedro, entonces está a la derecha de **b** si y solamente si está también en frente de **b**.
2. Si **b** es un dodecaedro, entonces si está en frente de **d** entonces no está detrás de **d**.
3. **c** está detrás de **a** pero en frente de **e**.
4. **e** está en frente de **d** a menos que él sea un tetraedro de gran tamaño.
5. Al menos uno de entre **a**, **c**, y **e** es un cubo.

Antes de salvar el fichero de sentencias como “Sentencias_2.38”, realice algunas comprobaciones:

- Abra el mundo “Bolzano’s World”. Todos los enunciados en ese mundo son verdaderos, así que si la formalización no es errónea y es fiel al sentido de esos enunciados, las sentencias formalizadas deberán evaluarse a verdadero. Si se ha cometido algún error de formalización corríjase.

(**Observación:** Aunque las traducciones se evalúen a verdadero en el mundo de Bolzano, eso no garantiza que sea una traducción adecuada. Si la traducción es correcta, tiene que tener el mismo significado (valor) que los enunciados castellanos en cualquier otro mundo. Por tanto, siempre conviene comprobar como se comportan las sentencias formalizadas en otros mundos.)

- Abra el mundo “Wittgenstein’s World”. Aquí, el enunciado 4 es falso, mientras que el resto son verdaderos. Compruebe que lo mismo sucede con sus traducciones. Si no, corrija sus traducciones (asegurándose de que todavía son verdaderas en el mundo de Bolzano).
- Ahora abra el mundo “Leibniz’s World”. Aquí, los enunciados verdaderos son el 1, 2, y el 5, mientras que los otros son falsos. Actúe como en el caso anterior.
- Finalmente, abra el mundo “Venn’s World”. En ese mundo, todos los enunciados son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, actúe en consecuencia.

Ejercicio 2.25 Abra el mundo “Montague’s Sentences”. Este fichero contiene expresiones que están a medio camino de ser formalizadas. Lea los enunciados de más abajo, entendiendo como se ha llegado a ese estado de formalización, intente completar su formalización sustituyendo las partes incompletas por fórmulas de la lógica de predicados.

1. Todo cubo en frente de cualquier tetraedro es grande.
2. Todo a la derecha de un cubo grande es pequeño.
3. Nada detrás de un cubo y en frente de un cubo es grande.
4. Ninguna cosa que no tenga ningún objeto a su espalda es un cubo.
5. Todo dodecaedro es más pequeño que algún tetraedro.

Antes de salvar el fichero de sentencias como “Sentencias_2.39”, realice algunas comprobaciones:

- Abra el mundo “Peirce’s World”. Todos los enunciados en ese mundo son verdaderos. Compruebe que las formalizaciones son también verdaderas cuando se interpretan en ese mundo. Si no lo son, trate de averiguar dónde se ha equivocado.
- Abra el mundo “Leibniz’s World”. Ahora los enunciados 1, 3, y 5 son verdaderos, mientras que el resto son falsos. Verifique que sus traducciones tienen el mismo valor de verdad cuando se evalúan. Si no es así corrija sus traducciones para ajustarse al sentido de los enunciados.
- Abra el mundo “Ron’s World”. Aquí el único enunciado verdadero es el 3. Proceda de forma similar a los casos anteriores.

Bloque C.

Ejercicio 2.26 Construya un mundo que contenga un único cubo de tamaño grande y ningún objeto más.

1. Escriba la siguiente sentencia en la ventana de sentencias:

$$(\exists x)(Cube(x) \rightarrow Large(x))$$

Compruebe que es verdadera en ese mundo.

2. Ahora cambie el cubo grande por un tetraedro pequeño y compruebe si la sentencia es verdadera o falsa. Entiende por qué es verdadera? Juegue el juego de Henkin-Hintikka suponiendo que es verdadera y luego que es falsa.
3. Añada una segunda sentencia que exprese correctamente que hay un cubo grande. Asegúrese de que es falsa en el mundo actual pero verdadera cuando se añade un cubo grande.

Salve las sentencias como “Sentencias_2.40”.

Dado el significado de los predicados en el lenguaje de los bloques, razonar sobre la validez⁵ de los siguientes argumentos (es posible suponer cualquier hecho general acerca de los mundos que puedan construirse con la aplicación el mundo de Tarski). Si el argumento se piensa que es válido, dar una prueba informal e incluirla en el fichero “resp_adicionales.txt”. Si la conclusión no es una consecuencia de las premisas, enviar un contraejemplo en el fichero denominado “contraejemplo_XXX”, donde XXX es el número del ejercicio.

Ejercicio 2.27 $\vdash \neg(SameRow(a, b) \wedge SameRow(b, c) \wedge FrontOf(c, a))$

Ejercicio 2.28

$$\begin{aligned} & \{Small(a) \rightarrow Small(b), Small(b) \rightarrow (SameSize(b, c) \rightarrow Small(c)), \\ & \neg Small(a) \rightarrow (Large(a) \wedge Large(c))\} \vdash SameSize(b, c) \rightarrow (Large(c) \vee Small(c)). \end{aligned}$$

Ejercicio 2.29 $\vdash (Small(b) \wedge SameSize(b, c)) \rightarrow Small(c)$

Ejercicio 2.30 $\vdash (\forall x)Tet(b) \leftrightarrow (\exists w)Tet(b)$

Ejercicio 2.31 $\vdash (\exists w)(Dodec(w) \wedge Large(w)) \leftrightarrow ((\exists w)Dodec(w) \wedge (\exists w)Large(w))$

Ejercicio 2.32 Traduzca los siguientes enunciados al lenguaje formal de la lógica de predicados (utilizando el lenguaje de bloques):

1. a es un tetraedro solamente si está en frente de b.
2. b es más grande que a y e.
3. a y e son, ambos, más grandes que c, pero ninguno es un bloque grande.
4. d tiene la misma forma que b solamente si tienen el mismo tamaño.

⁵Observe que aquí “validez” puede entenderse en el sentido menos restrictivo de “verdadero en todo mundo de Tarski”.

5. a es grande si y solamente si es un cubo.

Antes de salvar el fichero de sentencias como “Sentencias_2.46”, realice algunas comprobaciones:

- Abra el mundo “Bolzano’s World”. Todos los enunciados en ese mundo son verdaderos, así que si la formalización no es errónea y es fiel al sentido de esos enunciados, las sentencias formalizadas deberán evaluarse a verdadero. Si se ha cometido algún error de formalización corrijase.

(Observación: Aunque las traducciones se evalúen a verdadero en el mundo de Bolzano, eso no garantiza que sea una traducción adecuada. Si la traducción es correcta, tiene que tener el mismo significado (valor) que los enunciados castellanos en cualquier otro mundo. Por tanto, siempre conviene comprobar como se comportan las sentencias formalizadas en otros mundos.)

- Abra el mundo “Wittgenstein’s World”. Aquí, el único enunciado verdadero es el 5, mientras que el resto son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, corrija sus traducciones (asegurándose de que todavía son verdaderas en el mundo de Bolzano).
- Ahora abra el mundo “Leibniz’s World”. Aquí, los enunciados 1 y 4 son verdadero y los enunciados 2, 3, y 5 son falsos. Actúe como en el caso anterior.
- Finalmente, abra el mundo “Venn’s World”. En ese mundo, todos los enunciados son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, actúe en consecuencia.

Ejercicio 2.33 Abra el fichero “Finsler’s World” e inicie un nuevo fichero de sentencias. Las sentencias serán el resultado de formalizar los siguientes enunciados que describen ese mundo:

1. Todos los bloques pequeños están en frente de los bloques grandes.
2. Existe un cubo que es más grande que un tetraedro.
3. Todos los cubos están en la misma columna.
4. No todos los tetraedros están en la misma columna.
5. Cada uno de los cubos está en una fila diferente que otro cubo.
6. No todos los tetraedros están en una fila diferente que otro tetraedro.
7. Existen diferentes tetraedros que tienen el mismo tamaño.
8. No existen diferentes cubos del mismo tamaño.

Son todas las traducciones verdaderas en el mundo de Finsler? Si no lo son, trate de comprender la razón. Compruebe que las sentencias de la lógica de primer orden tienen el mismo significado (los mismos valores de verdad) que los enunciados que representan. Por otra parte, los enunciados enumerados más arriba son todos falsos en el mundo de König. Abra el mundo “König’s World” y compruebe que, ahora, las sentencias son todas falsas. Si no sucede así, modifique la traducción y vuelva a comprobar el valor de verdad que toma en el mundo de Finsler. Cuando haya terminado, salve el fichero de sentencias como “Sentencias_2.47”

Bloque D.

Ejercicio 2.34 Construya un mundo que contenga un único cubo de tamaño pequeño y ningún objeto más.

1. Escriba la siguiente sentencia en la ventana de sentencias:

$$(\exists x)(Cube(x) \rightarrow \neg Large(x))$$

Compruebe que es verdadera en ese mundo.

2. Ahora cambie el cubo pequeño por un tetraedro grande y compruebe si la sentencia es verdadera o falsa. Entiende por qué es verdadera? Juegue el juego de Henkin-Hintikka suponiendo que es verdadera y luego que es falsa.
3. Añada una segunda sentencia que exprese correctamente que hay un cubo pequeño. Asegúrese de que es falsa en el mundo actual pero verdadera cuando se añade un cubo pequeño.

Salve las sentencias como “Sentencias_2.10”.

Dado el significado de los predicados en el lenguaje de los bloques, razonar sobre la validez⁶ de los siguientes argumentos (es posible suponer cualquier hecho general acerca de los mundos que puedan construirse con la aplicación el mundo de Tarski). Si el argumento se piensa que es válido, dar una prueba informal e incluirla en el fichero “resp_adicionales.txt”. Si la conclusión no es una consecuencia de las premisas, enviar un contraejemplo en el fichero denominado “contraejemplo_XXX”, donde XXX es el número del ejercicio.

Ejercicio 2.35

$$\{Cube(a) \vee Tet(a) \vee Large(a), \neg Cube(a) \vee a = b \vee Large(a), \neg Large(a) \vee a = c, \\ \neg(c = c \wedge Tet(a))\} \vdash \neg(Large(a) \vee Tet(a))$$

Ejercicio 2.36 $\vdash \neg(a = b \wedge Dodec(a) \wedge Cube(b))$

⁶Observe que aquí “validez” puede entenderse en el sentido menos restrictivo de “verdadero en todo mundo de Tarski”.

Ejercicio 2.37

$\{Small(a) \wedge (Medium(b) \vee Large(c)), Medium(b) \rightarrow FrontOf(a, b),$
 $Large(c) \rightarrow Tet(c)\} \vdash \neg Tet(c) \rightarrow FrontOf(a, b).$

Ejercicio 2.38 $\vdash (\forall x)Cube(x) \rightarrow Cube(b)$

Ejercicio 2.39 $\vdash \neg(\forall x)(Cube(x) \rightarrow (Small(x) \vee Large(x)))$

Ejercicio 2.40 Traduzca los siguientes enunciados al lenguaje formal de la lógica de predicados (utilizando el lenguaje de bloques):

1. *b* es un cubo a menos que *c* sea un tetraedro.
2. Si *e* no es un cubo, *b* o *d* son de tamaño grande.
3. *b* o *d* es un cubo si *a* o *c* es un tetraedro.
4. *a* es grande en el caso de que *d* sea pequeño.
5. *a* es grande en el caso de que *e* lo sea.

Antes de salvar el fichero de sentencias como “Sentencias_2.16”, realice algunas comprobaciones:

- Abra el mundo “Bolzano’s World”. Todos los enunciados en ese mundo son verdaderos, así que si la formalización no es errónea y es fiel al sentido de esos enunciados, las sentencias formalizadas deberán evaluarse a verdadero. Si se ha cometido algún error de formalización corríjase.

(Observación: Aunque las traducciones se evalúen a verdadero en el mundo de Bolzano, eso no garantiza que sea una traducción adecuada. Si la traducción es correcta, tiene que tener el mismo significado (valor) que los enunciados castellanos en cualquier otro mundo. Por tanto, siempre conviene comprobar como se comportan las sentencias formalizadas en otros mundos.)

- Abra el mundo “Wittgenstein’s World”. Aquí, el enunciado 5 es falso, mientras que el resto son verdaderos. Compruebe que lo mismo sucede con sus traducciones. Si no, corrija sus traducciones (asegurándose de que todavía son verdaderas en el mundo de Bolzano).
- Ahora abra el mundo “Leibniz’s World”. Aquí, aproximadamente la mitad de los enunciados es verdadera (3, y 5) y la otra mitad falsa (1, 2, y 4). Actúe como en el caso anterior.
- Finalmente, abra el mundo “Venn’s World”. En ese mundo, todos los enunciados son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, actúe en consecuencia.

Ejercicio 2.41 Abra los ficheros “Hilbert’s Sentences” y “Peano’s World”. Evalúe (mentalmente) las sentencias una a una, y verifiquelas, jugando el juego de Henkin-Hintikka si alguna de las evaluaciones le sorprende. Modifique las sentencias falsas introduciendo la conectiva de negación hasta que la sentencia se convierta en verdadera. El problema es que no se permite introducir simplemente la negación al principio de la sentencia. Asegúrese de que entiende porque la sentencia original era falsa y la modificada de esa forma es falsa. Cuando haya acabado, salve el fichero de sentencias como “Sentencias_2.17”.

Bloque E.

Dado el significado de los predicados en el lenguaje de los bloques, razonar sobre la validez⁷ de los siguientes argumentos (es posible suponer cualquier hecho general acerca de los mundos que puedan construirse con la aplicación el mundo de Tarski). Si el argumento se piensa que es válido, dar una prueba informal e incluirla en el fichero “resp_adicionales.txt”. Si la conclusión no es una consecuencia de las premisas, enviar un contraejemplo en el fichero denominado “contraejemplo_XXX”, donde XXX es el número del ejercicio.

Ejercicio 2.42

$$\{Larger(a, b) \vee Larger(a, c), Smaller(b, a) \vee \neg Larger(a, c)\} \vdash Larger(a, b)$$

Ejercicio 2.43

$$\{Cube(b) \leftrightarrow (Cube(a) \leftrightarrow Cube(c))\} \vdash Dodec(b) \rightarrow a = c.$$

Ejercicio 2.44

$$\vdash (Cube(b) \wedge b = c) \rightarrow Cube(c)$$

Ejercicio 2.45 Abra el fichero “Ramsey’s Sentences”. Construya un mundo en el que las sentencias 1-10 (ignórense las sentencias 11-20 por ahora) sean todas verdaderas. Esas sentencias realizan afirmaciones particulares (esto es, no contienen cuantificadores) o existenciales (esto es, aseveran que cosas de una cierta clase existen). Consecuentemente, se pueden hacer verdaderas añadiendo, sucesivamente, objetos al mundo que se está creando. Lo que se pide en este ejercicio es que se hagan verdaderas incluyendo el menor número de objetos posibles. Así que en lugar de añadir un objeto por sentencia, solamente añada nuevos objetos cuando sea necesario. Cuando termine (de añadir objetos), revise que ninguna de las sentencias 1-10 es falsa y salve la solución en un fichero denominado “Mundo_2.21”.

(Ayuda: Es posible completar el ejercicio añadiendo solamente seis objetos, si se considera la posibilidad de que alguno de ellos posea dos nombres.)

Ejercicio 2.46 Las sentencias 11-20 del fichero “Ramsey’s Sentences” todas hacen afirmaciones universales. Esto es, dicen que todo objeto en ese mundo tiene una u otra propiedad. Compruebe si el mundo construido en el Ejercicio 2.21 satisface las afirmaciones universales expresadas por dichas sentencias. Si no es así, modifique el mundo de modo que las 20 sentencias sean verdaderas a un tiempo. Salve la solución en un fichero denominado “Mundo_2.22”.

⁷Observe que aquí “validez” puede entenderse en el sentido menos restrictivo de “verdadero en todo mundo de Tarski”.

Ejercicio 2.47 Traduzca los siguientes enunciados al lenguaje formal de la lógica de predicados (utilizando el lenguaje de bloques):

1. Cada tetraedro está en frente de cada dodecaedro.
2. Nigún dodecaedro tiene nada detrás de él.
3. Nigún tetraedro es del mismo tamaño que cualquier cubo.
4. Todo dodecaedro es del mismo tamaño que algún cubo.
5. Ningún objeto entre dos dodecaedros es un cubo.

Antes de salvar el fichero de sentencias como “Sentencias_2.23”, realice algunas comprobaciones:

- Abra el mundo “Bolzano’s World”. Todos los enunciados en ese mundo son verdaderos, así que si la formalización no es errónea y es fiel al sentido de esos enunciados, las sentencias formalizadas deberán evaluarse a verdadero. Si se ha cometido algún error de formalización corríjase.

(Observación: Aunque las traducciones se evalúen a verdadero en el mundo de Bolzano, eso no garantiza que sea una traducción adecuada. Si la traducción es correcta, tiene que tener el mismo significado (valor) que los enunciados castellanos en cualquier otro mundo. Por tanto, siempre conviene comprobar como se comportan las sentencias formalizadas en otros mundos.)

- Abra el mundo “Ron’s World”. Aquí, los enunciados 4, y 5 son verdaderos, mientras que el resto son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, corrija sus traducciones (asegurándose de que todavía son verdaderas en el mundo de Bolzano).
- Ahora abra el mundo “Claire’s World”. Aquí, los enunciados 1, 3, y 5 son verdaderos, mientras que el resto son falsos. Actúe como en el caso anterior.
- Finalmente, abra el mundo “Peano’s World”. En ese mundo, todos los enunciados son falsos. Compruebe que lo mismo sucede con sus traducciones. Si no, actúe en consecuencia.

Envío de los trabajos de la Práctica 2.

Los tipos de ficheros generados al realizar un ejercicio podrán ser: ficheros de sentencias, ficheros de mundos, o contraejemplos. Los nombres de los ficheros generados se ajustarán a lo indicado en cada ejercicio.

En ocasiones, la resolución de un ejercicio necesitará de explicaciones adicionales (por ejemplo, una prueba informal de que un argumento es válido) que deberán escribirse y detallarse en un fichero de texto que se denominará “**resp_adicionales.txt**”.

Cada grupo deberá enviar los ficheros generados al realizar tanto los ejercicios de la primera parte, como los del bloque que se le asigne en la segunda parte de esta práctica.

Estos ficheros se comprimirán formando un archivo ZIP y se enviarán para su corrección usando Moodle. El nombre del archivo ZIP debe ajustarse al siguiente formato:

GGG_APELLIDOS_P2_B.zip

donde, **GGG** es el código del grupo, **APELLIDOS**, los apellidos del alumno responsable del grupo, **P2**, es el código de la práctica y **B**, es la letra del bloque.
Ejemplo:

g1_Botella_Blanco_P2_B.zip

Capítulo 3

PROLOG: la lógica de predicados como lenguaje programación.

3.1. Introducción.

Esta práctica tiene por objetivo general mostrar que fragmentos de la lógica de predicados pueden emplearse como un lenguaje de programación y, de forma más concreta, presentar una de las realizaciones prácticas del denominado *paradigma de programación lógica*: el lenguaje Prolog.

En el paradigma de programación lógica, un *programa* es un conjunto de fórmulas lógicas y la *computación* se concibe como un proceso deductivo por el que se extraen consecuencias a partir de las fórmulas del programa. Esta noción de computabilidad basada en derivaciones fue estudiada por primera vez en los años treinta del pasado siglo por Kurt Gödel y Jacques Herbrand [3]. Investigaciones adicionales de Herbrand sirvieron de fundamento teórico para los primeros trabajos en el área de la deducción automática. In 1965 Alan Robinson publicó su artículo fundamental [12] que sentó las bases del campo de la deducción automática. En ese artículo introdujo el principio de resolución, la noción de unificación y un algoritmo de unificación. Empleando el método de resolución se pueden demostrar teoremas de la lógica de primer orden, pero se necesitaba otro paso adicional para saber cómo uno podía computar dentro de ese marco.

Fue Robert Kowalski quien dió ese paso, cuando publicó su artículo [10], en el que se introdujeron los programas lógicos con una forma restingida de resolución. La diferencia entre esta forma de resolución y la propuesta por Robinson es que su sintaxis es más restrictiva, pero ahora el proceso de prueba tiene un efecto colateral en forma de una sustitución. Esta sustitución puede verse como el resultado de una computación y consecuentemente un cierto conjunto de fórmulas

lógicas puede verse como un programa. En paralelo, Alain Colmerauer con sus colegas trabajaban en un lenguaje de programación para el procesamiento del lenguaje natural basado en técnicas provenientes del área de la demostración automática de teoremas. Finalmente, el encuentro de estas dos líneas de trabajo condujo a la creación del lenguaje Prolog en 1973. Kowalski y Colmerauer, junto con su equipo, a menudo interactuaron durante el periodo 1971–1973. Esto influyó sus puntos de vista y les ayudó a cristalizar sus ideas [3].

3.2. El lenguaje PROLOG.

El lenguaje de programación Prolog (PROgramación en LOGica Colmenauer et. al. 1973) es la realización más conocida de las ideas introducidas en el campo de la programación lógica. En este apartado introducimos un subconjunto puro de Prolog, que no hace uso de las características extralógicas del lenguaje.

3.2.1. Sintaxis.

Un programa Prolog es una secuencia de *reglas* (un tipo especial de fórmulas condicionales denominadas *cláusulas*) que representan nuestro conocimiento en un dominio determinado: propiedades de los elementos y relaciones entre ellos. En general una regla es una fórmula de la forma:

A :- Q

donde A es átomo¹ denominado *cabeza* y Q una fórmula formada por átomos o negaciones de átomos unidos por conjunciones o disyunciones, que denominamos *cuerpo*. Una regla que no posee cuerpo (esto es, una simple fórmula atómica) se denomina *hecho* y una regla que no posee cabeza, se denomina *objetivo*. Como veremos, las cláusulas objetivo juegan un papel especial y servirán para plantear preguntas relativas a la teoría que define un determinado programa. Por último, decir que las variables que aparecen en las reglas se suponen cuantificadas universalmente.

El siguiente es un ejemplo de programa Prolog en el que se muestran hechos y reglas que especifican un conjunto de propiedades y relaciones familiares sobre un dominio finito formado por personajes bíblicos.

```
%% PROGRAMA: biblia.pl
%% HECHOS
padre(abraham,isaac).
padre(haran,lot).
padre(haran,milcah).
padre(haran,yiscah).
hombre(abraham).
hombre(haran).
hombre(isaac).
```

¹ Átomo es otro nombre con el que se conocen las fórmula atómicas.

```

hombre(lot).
mujer(yiscah).
mujer(milcah).
%% REGLAS
hijo(X,Y) :- hombre(X), padre(Y,X).
hija(X,Y) :- mujer(X), padre(Y,X).
abuelo(X,Z) :- hombre(X), padre(X,Y), (madre(Y,Z);padre(Y,Z)).

```

Es de destacar que un programa Prolog genera un lenguaje interpretado, en el que los símbolos poseen un significado esperado acorde con su denotación. También adviértase que dentro de ese lenguaje se relaja la rigidez de la notación de la lógica de predicados, al permitir el uso de identificadores para denotar los símbolos de relación, de función y de constante. Un identificador es cualquier cadena resultante de la combinación de caracteres alfanuméricos (*a*, ..., *z*, *A*, ..., *Z*, 0, ..., 9) y el subrayado “_”, con la única restricción de que los identificadores para las variables² deben comenzar por una letra mayúscula o subrayado, y los de los símbolos de constante, función, y relación por una letra minúscula.

Prolog emplea una notación especial para las conectivas lógicas. La Tabla 3.1 se muestra la correspondencia entre la notación empleada en programación lógica y en Prolog para las conectivas lógicas.

Cuadro 3.1: Conectivas lógicas y sintaxis de Prolog.

	Conectiva lógica	Símbolo	Sintaxis
Nombre			Prolog
Conjunción		\wedge	,
Disjunción		\vee	;
Implícación inversa (en una regla)		\leftarrow	$:-$
Negación		\neg	not
Implícación inversa (en una cláusula objetivo)		\leftarrow	$?-$

3.2.2. Semántica.

En el lenguaje Prolog, los hechos y las reglas constituyen las instrucciones de un programa. Las reglas de un programa lógico admiten dos tipos de lectura: la declarativa y la procedimental. Por ejemplo, supongamos la regla

P :- Q, R, S.

son posibles estas dos lecturas:

- Lectura declarativa: “P es cierto si Q, R y S son ciertos”

²Cuando se utiliza únicamente el símbolo “_” como identificador de una variable, decimos que la variable es *anónima*. Las variables anónimas reciben un tratamiento especial de los sistemas Prolog, ya que, los valores enlazados a estas variables no se muestran como resultado de un cálculo. Por otra parte, las distintas ocurrencias de variables anónimas en una expresión, internamente, se tratan como variables diferentes sin conexión alguna.

- **Lectura procedimental:** “Para satisfacer el objetivo P, es necesario satisfacer (ejecutar) primero los objetivos Q, R y S”

De forma semejante, un programa admite dos interpretaciones. La interpretación procedural explica *cómo* se efectúa la computación, mientras que la interpretación declarativa tiene que ver con la pregunta de *qué* se quiere computar. En otras palabras, la primera concierne sobre el método mientras que la segunda sobre el significado. En la interpretación procedural un programa se ve como la descripción de un algoritmo que puede ejecutarse. En la interpretación declarativa un programa se ve como un conjunto de fórmulas de las que se puede razonar sobre su corrección sin ninguna referencia al mecanismo operacional subyacente. Esto hace que los programas lógicos (y en general los declarativos) sean más fáciles de entender y de desarrollar.

Por otra parte, la visión procedural de las cláusulas y los programas nos conduce a una reflexión sobre lo qué significa computar en el marco de la Programación Lógica y en Prolog. En este marco “computar” es “deducir”. El programa es una teoría (conjunto de premisas) y a partir de él se intentan establecer conclusiones mediante un procedimiento de prueba por refutación que emplea una adaptación del principio de resolución de Robinson como única regla de inferencia.

Normalmente, las fórmulas que se desean deducir son de la forma:

$$(\exists x_1) \dots (\exists x_n)(B_1, B_2, \dots, B_k)$$

Estas fórmulas, una vez negadas, toman la forma de cláusulas objetivo, “?- B_1, B_2, \dots, B_k ” que se suministran al sistema Prolog como preguntas. Entonces, el sistema realiza una secuencia de pasos de deducción intentando encontrar una contradicción (la *cláusula vacía*). Si se encuentra, el proceso tiene *éxito*, si no *falla*. Como consecuencia del proceso deductivo se “asigna” valor a las variables, x_1, \dots, x_k . Estas asignaciones se agrupan en una estructura que denominamos *sustitución* y que constituye el resultado de la computación.

Las sustituciones son la contrapartida de la familiar noción de estado que se emplea en programación imperativa. Denotamos una sustitución mediante el conjunto $\{x_1/t_1, \dots, x_n/t_n\}$. Esta notación implica que cada una de las x_1, \dots, x_n son variables diferentes, t_1, \dots, t_n son términos y que ningún término t_i es igual a la variable x_i (a la que sustituye). Las variables $\{x_1, \dots, x_n\}$ se denominan el *dominio* de la sustitución y los términos $\{t_1, \dots, t_n\}$, el *rango* de la sustitución. Decimos que la sustitución $\{x_1/t_1, \dots, x_n/t_n\}$ *enlaza* cada variable x_i a un término t_i .

Utilizando una sustitución se puede evaluar un término en una forma similar a como se usa un estado para evaluar una expresión en un lenguaje imperativo. Este proceso de evaluación se denomina *aplicación* de una sustitución σ a un término t y lo denotamos como $\sigma(t)$. El resultado es un reemplazo simultáneo de todas las variables x_i del término t , que aparecen en el dominio de la sustitución, por los correspondientes términos t_i del rango de la sustitución. Por ejemplo, dada la sustitución $\{x/f(z), y/g(z)\}$, su aplicación al término $h(x, y)$ conduce al término $h(f(z), g(z))$. De la misma manera se puede definir la aplicación de

una sustitución a un átomo o una cláusula. Por último, nótese que la evaluación de un término empleando una sustitución conduce de nuevo a un término. Esto contrasta con el marco de programación imperativa, donde la evaluación de una expresión utilizando un estado conduce a un valor que pertenece al tipo de la expresión.

En la base de cualquier estrategia de resolución se encuentra la unificación de expresiones (átomos y términos). Intuitivamente, *unificar* es el proceso de resolver una ecuación entre términos (esto se hace buscando una sustitución, que aplicada a esos dos términos los convierta en sintácticamente iguales) de la manera menos restrictiva posible. La sustitución resultante se denomina *unificador más general* (*mgu*). Por ejemplo, la ecuación $x = f(y)$ puede resolverse (i.e., los términos x y $f(y)$ unifican) en un número muy diferente de maneras, mediante cualquiera de las sustituciones: $\{x/f(y)\}$, $\{x/f(a), y/a\}$, $\{x/f(a), y/a, z/g(b)\}$. Se dice que éstas son sustituciones unificadoras. Claramente, la primera sustitución es la menos “restrictiva” de las tres y recibe el nombre de *unificador más general*³ de los términos x y $f(y)$. Dados dos términos t_1 y t_2 , el unificador más general de t_1 y t_2 es único (salvo renombramiento de variables) y lo denotamos como $mgu(t_1, t_2)$.

En esencia, Prolog utiliza una variante del principio de resolución de Robinson denominada *resolución SLD*. Las siglas “SLD” hacen mención a las iniciales inglesas de “resolución Lineal con función de Selección para programas Definidos”. Un *programa definido* es aquél que está compuesto por un tipo especial de reglas que se llaman *cláusulas de Horn* o *definidas*. Una *cláusula de Horn* es una fórmula conjuntiva $A :- Q$, como la descrita en el Apartado 3.2.1, con un solo átomo A en cabeza y un cuerpo Q conjuntivo (que no contiene negación ni disyunción). Un *objetivo definido* es una cláusula de Horn sin cabeza, i.e. una fórmula $?- Q$ con un cuerpo conjuntivo.

Sean A y A' átomos y Q y Q' conjunciones de átomos. Dados una cláusula definida $C \equiv (A' : -Q')$ y un objetivo $G \equiv (? - A, Q)$, que no comparten variables en común, podemos especificar un *paso de resolución SLD* mediante la siguiente regla de inferencia:

$$\frac{?- A, Q \quad A' :- Q'}{?- \theta(Q', Q)} \quad \text{si } \theta = mgu(A, A') \neq \text{fallo}$$

Decimos que el objetivo $?- \theta(Q, Q')$ es un *resolvente SLD* de C y G .

De manera informal, el resolvente de un objetivo y una regla puede obtenerse dando los siguientes pasos:

- **Selección:** Seleccionar el átomo más a la izquierda⁴ del objetivo.
- **Renombramiento:** Si es necesario, renombrar las variables de la regla (para que no haya colisiones con las del objetivo).

³La noción de sustitución menos restrictiva puede hacerse precisa mediante la definición de un orden sobre el conjunto de las sustituciones. Por ejemplo, en ese orden la sustitución $\{x/f(y)\}$ es la menor de todas las sustituciones unificadoras.

⁴Observe que empleamos una regla de selección fija. Estamos usando la regla de selección consistente en “seleccionar el átomo más a la izquierda dentro de un objetivo”, que es la que usa Prolog.

- **Instanciación:** Hallar el mgu del átomo seleccionado y la cabeza de la cláusula y aplicarlo sobre el objetivo y la cláusula originales, obteniendo las correspondientes instancias.
- **Reemplazo:** Reemplazar la instancia del átomo seleccionado por la instancia del cuerpo de la cláusula.

Nótese que si no es posible computar el mgu del átomo seleccionado y la cabeza de la cláusula, entonces no se puede dar el paso de resolución SLD.

Encadenando pasos de resolución SLD obtenemos una *derivación SLD*. Cuando una derivación SLD termina computando la cláusula vacía (i.e., una contradicción), se dice que es una *refutación SLD*. En cada paso se halla una sustitución unificadora más general que, compuesta con las otras, proporciona la respuesta de la computación.

Para terminar este apartado, hacemos mención de que cuando las reglas de los programas contienen negación y/o disyunción, el mecanismo operacional de la resolución SLD debe extenderse para tratar correctamente con este tipo de reglas.

3.2.3. Modo de operación.

Los sistemas Prolog suelen proporcionar un interprete de comandos como interfaz con el usuario. Este es un interfaz muy rudimentario en el que el modo de operación habitual es el siguiente:

1. **Editar el programa:** Editar el programa Prolog utilizando el editor de texto (no un procesador de texto) que prefiera el programador, y almacenarlo en un fichero asignándole un nombre. Por ejemplo: **biblia.pl** (la extensión depende del sistema Prolog que estemos empleando, en SWI-Prolog la extensión es “.pl”).
2. **Cargar el programa:** Para que un programa Prolog pueda ejecutarse en un interprete de Prolog, sus hechos y reglas deben de cargarse previamente en la *base de datos interna* del sistema Prolog (o *espacio de trabajo — workspace*). La carga se efectúa mediante el comando **consult/1**, que toma como argumento el nombre de un fichero contenido un programa Prolog:

```
?- consult(biblia).
% biblia compiled 0,00 sec, 2,080 bytes
true.
```

Si el fichero que se pretende cargar está en un directorio distinto que el directorio de trabajo, se tendrá que suministrar un camino absoluto o relativo (entre comillas simples):

```
?- consult('./practica3/biblia').
% biblia compiled 0,00 sec, 2,080 bytes
true.
```

El comando `consult/1` permite cargar una lista de ficheros usando la siguiente sintaxis: `[pathname1, pathname2, ...]`. Por ejemplo:

```
?- consult([pru, './practica3/biblia']).  
% pru compiled 0,00 sec, 120 bytes  
% biblia compiled 0,00 sec, 2,080 bytes  
true.
```

carga el programa `pru.pl` almacenado en el directorio de trabajo y después el programa `biblia.pl` almacenado en el directorio `./practica3`. Por razones históricas el comando `consult/1` suele abreviarse escribiendo únicamente la lista de nombres. Esto es:

```
?- [biblia].  
% biblia compiled 0,00 sec, 2,080 bytes  
true.
```

3. Una vez efectuada la carga, el usuario puede plantear preguntas al sistema (objetivos), que éste resolverá haciendo uso del motor de inferencias del sistema Prolog. Por ejemplo, puede preguntar quienes son las hijas de `haran`:

```
?- hija(X,haran).  
X = milcah ;  
X = yiscah.  
true.
```

3.3. Objetivos.

Esta práctica persigue los siguientes objetivos concretos:

1. Que el alumno adquiera habilidad para representar conocimiento y especificar problemas mediante el empleo del lenguaje de la lógica de predicados y, en especial, Prolog.
2. Conocer el entorno de trabajo del intérprete SWI-Prolog.
3. Proporcionar una introducción a la programación en un subconjunto puro de Prolog, que no hace uso de las características extralógicas de ese lenguaje.
4. Familiarizar al alumno con la escritura de pequeños programas Prolog que resuelvan problemas extraídos de diferentes campos de aplicación.

3.4. Descripción.

Para la correcta realización de esta práctica se aconseja la lectura previa de parte de los capítulos dedicados al lenguaje Prolog que aparecen en [9], especialmente el capítulo 6. También se recomienda la lectura del manual de usuario del lenguaje Prolog que se suministra a través de Moodle. El guión propuesto es el siguiente:

Parte I.

En esta primera parte, se trata de conocer y manejar el entorno de trabajo del intérprete SWI-Prolog, familiarizándose con las diferentes opciones de la barra de menú, y la dinámica de operación que conlleva la creación, edición y ejecución de programas PROLOG. Para ello, a partir de las explicaciones facilitadas por el profesor, deben de resolverse una serie de ejercicios, de relativa simplicidad, que también sirven para cubrir algunos de los objetivos concretos listados en el Apartado 3.3.

Dominios finitos.

Siguiendo la estela de K.R. Apt [2] organizaremos las exposición en términos de los dominios sobre los que tiene lugar el cómputo. Comenzamos con un par de ejemplos sobre *dominios finitos*, esto es, dominios en los que no se contemplan funciones y por lo tanto cada elemento del dominio se representa por una constante.

Ejercicio 3.1 *Un grafo es una estructura compuesta de un conjunto de nodos (o vértices) y un conjunto de arcos⁵ (o aristas) que conectan unos nodos con otros. Hay diversos tipos de grafos dependiendo de la naturaleza de los arcos y del número de ellos que conectan un par de nodos. La Figura 3.1 proporciona una representación de un grafo, en la que los nodos se muestran enmarcados por círculos y los arcos por flechas (segmentos orientados que imponen un orden determinado entre el par de elementos que relacionan).*

Prolog es un lenguaje que permite muy fácilmente la representación de estructuras matemáticas abstractas, como el grafo de la Figura 3.1.

```
%% PROGRAMA: grafo.pl
%% REPRESENTACION DE UN GRAFO
%% Conexiones directas
conexion(a,b).           conexion(b,d).           conexion(a,f).
conexion(b,c).           conexion(d,e).           conexion(f,e).
conexion(c,i).           conexion(e,h).           conexion(e,i).

%% CONEXIONES INDIRECTAS: Caminos dentro del grafo
%% camino(X,Y): X esta conectado por un camino con Y
```

⁵Algunos autores reservan la palabra “arco” para las aristas de los grafos dirigidos.

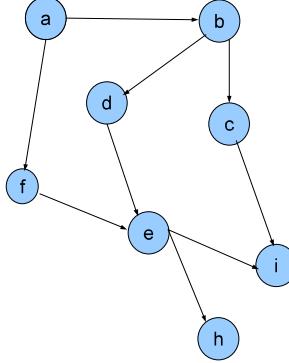


Figura 3.1: Representación de un grafo dirigido.

```

camino(X,Y) :- conexion(X,Y).
camino(X,Y) :- conexion(X,Z), camino(Z,Y).

```

Completar el programa permitiendo que las conexiones entre los vértices sean bidireccionales y generar un fichero denominado “grafo2.pl”. Contestar a preguntas tales como, ¿Existe conexión entre **h** e **i**? Comparar los resultados obtenidos con uno y otro programa cuando se les somente a las mismas preguntas.

Naturales.

El conjunto de los números naturales, $\mathbb{N} = \{0, 1, 2, \dots\}$, constituye un dominio infinito que como puso de relieve el matemático italiano Peano, puede construirse a partir del primero de los elementos, 0, y una función unaria, s , la *función sucesor*, que aplicada sobre un natural devuelve su sucesor. Más formalmente, los naturales pueden definirse inductivamente como sigue:

1. $0 \in \mathbb{N}$ (i.e., el 0 es un número natural).
2. Para todo número n , si $n \in \mathbb{N}$ entonces $s(n) \in \mathbb{N}$. El número natural $s(n)$ se denomina el *sucesor* de n .

Ejercicio 3.2 La definición inductiva de un número natural puede traducirse de forma directa a un programa Prolog:

```

%% PROGRAMA: natural.pl
% natural(N), N es un natural
natural(cero).
natural(suc(N)) :- natural(N).

```

donde denotamos el número 0 mediante una constante **cero** y la función sucesor s mediante el símbolo de función unario **succ**. El programa **natural.pl**

sirve para comprobar si un término es un natural. Lance los objetivos “?- natural(suc(suc(cero)))”, “?- natural(suc(a))” y “?- natural(suc(X))” y observe lo que sucede.

En un argumento de la lógica el orden de las premisas no importa, sin embargo, en un programa Prolog el orden de las reglas sí es importante.

Ejercicio 3.3 Invierta el orden de las reglas en el programa `natural.pl` del Ejercicio 3.2 y lance el objetivo “?- natural(suc(X))”. ¿Qué es lo que sucede ahora?

En la aritmética de Peano la operación de adición de números naturales se define mediante los siguientes dos axiomas:

- Para todo número natural n , $0 + n = n$.
- Para todo número natural n y m , $s(n) + m = s(n + m)$.

Las anteriores ecuaciones están definiendo la suma como una función recursiva que devuelve un valor, la suma de dos números naturales. Debido a que una relación puede entenderse como una función booleana, que sólo devuelve verdadero o falso, para devolver un resultado debemos hacerlo a través de uno de los parámetros del predicado que se está definiendo. Esto explica la formalización de los anteriores axiomas mediante el siguiente programa Prolog, en el que la suma se define como una relación ternaria⁶.

Ejercicio 3.4 En Prolog la suma sobre naturales puede definirse mediante el siguiente programa:

```
%% PROGRAMA: suma.pl
% suma(N, M, S), la suma de N y M es S
suma(cero, N, N).
suma(suc(N), M, suc(S)) :- suma(N, M, S).
```

Este programa puede utilizarse de múltiples maneras. Lance los objetivos “?- suma(suc(cero), suc(cero), suc(suc(cero)))”, “?- suma(suc(cero), suc(cero), Z)”, “?- suma(X, suc(cero), suc(suc(cero)))” y “?- suma(X, Y, suc(suc(cero)))” y observe lo que sucede.

Observación importante: Los predicados definidos en este apartado son una muestra de *definición inductiva* (recursiva) basada en la estructura de los elementos del dominio de discurso, por lo que también recibe el nombre de definición por *inducción estructural*. En ella distinguimos un caso base, en el que se define la propiedad para el constructor `cero`, y un caso general, en el que se define la propiedad para un natural `suc(N)`. Este estilo de definición de propiedades recibe el nombre de *definición por ajuste de patrones* en el contexto de la programación funcional. Los términos `cero` y `suc(N)` se denominan patrones. En general, un *patrón* es un término formado por símbolos constructores y variables.

⁶En general, cualquier función n -aria puede expresarse como una relación $(n + 1)$ -aria, donde el argumento “extra” se usa para retornar el resultado devuelto por la función.

Listas.

Intuitivamente una lista es una secuencia de cero o más elementos. Una lista puede representarse en Prolog usando cualquier constante, digamos `nil`, y símbolo de función binario, digamos `f` como constructores. Entonces, la secuencia `a, b, c` puede representarse mediante el término `f(a, f(b, f(c, nil)))`. Sin embargo, las listas son tan fundamentales en Prolog que éste utiliza una notación especial para los constructores de listas y para las propias listas. En particular, emplea la constante `[]` (la lista vacía) y el operador binario `[. | ..]` (que añade un elemento al principio de una lista). Con esta notación, la secuencia `a, b, c` se representa como `[a | [b | [c | []]]]` y se abrevia como `[a, b, c]`.

Ejercicio 3.5 Prolog posee un operador “`=`” que representa la relación de unificación. Cuando se plantea el objetivo “`?- Expresion1 = Expresion2.`”, éste tiene éxito si las expresiones `Expresion1` y `Expresion2` unifican. En caso de éxito, el intérprete responde dando el unificador más general de `Expresion1` y `Expresion2`. Lanzar los siguientes objetivos y ver lo que sucede: “`?- X = [a | [b | [c | []]]]`”, “`?- X = [a, b | [c | []]]`”, “`?- X = [a, b, c | []]`” “`?- [a, b, c] = [a | [b | [c | []]]]`”, “`?- [a, b, c] = [a, b, c | []]`”, “`?- [a, b | Z] = [a | [b | [c | []]]]`”, “`?- [a, b, Z] = [a | [b | [c | []]]]`”.

Formalmente, el dominio de las listas se define inductivamente como sigue:

1. `[]` es una lista;
2. `[X|L]` es una lista, si `X` es un elemento y `L` una lista; `X` se denomina la *cabeza* y `L` es la *cola* de la lista.

Ejercicio 3.6 Al igual que la definición inductiva de un número natural, la anterior definición inductiva de lista puede traducirse de forma directa a un programa Prolog:

```
%% PROGRAMA: esLista.pl
% esLista(L), L es una lista
esLista([]).
esLista([X|L]) :- esLista(L).
```

El programa `esLista.pl` sirve para comprobar si un término es una lista o no lo es. Lance los objetivos “`?- esLista([a,b,c])`”, “`?- esLista([a|L])`” y “`?- esLista(X)`” y observe lo que sucede.

La mayoría de los sistemas Prolog proporcionan una buena colección de predicados predefinidos para la manipulación de listas. Dos de los más usuales son `member` y `append`:

- `member(E, L)`, esencialmente, sirve para comprobar si un elemento `E` está contenido en la lista `L`. También puede emplearse para extraer los elementos de una lista, cuando se suministra como primer argumento una variable (lance el objetivo “`?- member(X, [a,b,c,d])`” y vea qué pasa).

- `append(L1, L2, L)`, concatena las listas `L1` y `L2` para formar la lista `L`. Al igual que el predicado `member`, este predicado es inversible y admite múltiples usos.

Ejercicio 3.7 Una definición habitual del predicado `append/3` es:

```
%% PROGRAMA: append.pl
%% app(L1, L2, L), la concateacion de L1 y L2 es L
app([~], L, L).
app([E|L1], L2, [E|L]) :- app(L1, L2, L).
```

Este programa sirve para responder tanto preguntas como “El resultado de concatenar las listas `[2, 4, 6]` y `[1, 3, 5, 7]` ¿es la lista `[2, 4, 6, 1, 3, 5, 7]`?”, o “¿Cuál es el resultado de concatenar las listas `[2, 4, 6]` y `[1, 3, 5, 7]`?”, como “¿Qué listas dan como resultado de su concatenación la lista `[2, 4, 6, 1, 3, 5, 7]`?”. Expresar estas preguntas como objetivos y lanzarlos para que los responda el intérprete de Prolog.

Observe que nuevamente hemos utilizado la técnica de definición por inducción estructural (con un caso base para el constructor `[]` y un caso general para el constructor `[. | ...]`). En la mayoría de las ocasiones, esta técnica se ajusta bien a la definición de propiedades sobre listas ya que éstas, como hemos visto, son estructuras inherentemente recursivas.

Parte II.

Una vez familiarizados con el entorno de trabajo del intérprete SWI-Prolog con algunas de las técnicas de programación con Prolog, cada grupo (formado por dos alumnos) deberá realizar los ejercicios propuestos a continuación. Algunos de los ejercicios propuestos se han extraído del libro [9].

Ejercicio 3.8 (Hongos, trufas, setas) Dado el conjunto de enunciados:

*Los hongos se reproducen por medio de esporas.
 Las trufas o las setas son hongos.
 Todas las trufas son comestibles.
 Tuber melanosporum y Tuber magnatum son trufas.
 La seta amanita muscaria es alucinógena.
 Si es alucinógena es venenosa.
 La seta boletus satanás es venenosa.
 El champiñón, el níscolo, el rebollón, boletus blanco y la amanita cesarea son setas comestibles.*

Tradúzcalo al lenguaje formal de la lógica de predicados y, después, convierta las fórmulas obtenidas en un programa Prolog. Utilice el programa para responder a las siguientes preguntas: ¿qué hongos son venenosos? ¿qué hongos comestibles se reproducen por esporas? ¿cuales son las trufas venenosas?

Ayuda. Para formalizar los enunciados, utilice como símbolos peculiares del alfabeto:

$\mathcal{C} = \{ \text{melanosporum}, \text{magnatum}, \text{amanita_muscaria}, \text{boletus_satanas}, \text{champi\~on}, \text{niscalo}, \text{rebollon}, \text{boletus_blanco}, \text{amanita_cesarea} \};$
 $\mathcal{P} = \{ \text{reproduccion_esporas}/1, \text{hongo}/1, \text{trufa}/1, \text{seta}/1, \text{comestible}/1, \text{alucinogena}, \text{venenosa}/1 \}.$

Use como interpretación de partida la interpretación $\mathcal{I} = \langle \mathcal{D}, \mathcal{J} \rangle$ tal que \mathcal{D} sea el dominio universal y \mathcal{J} la función de interpretación esperada en la que, por ejemplo, $\mathcal{J}(\text{melanosporum}) = \text{"tuber melanosporum"}$, etc. y los símbolos de relación se interpretan como, $\mathcal{J}(\text{reproduccion_esporas}) = \text{"se reproducen por medio de esporas"}$, etc.

Ejercicio 3.9 La Unión Europea está formada por 27 países europeos soberanos independientes que se conocen como los estados miembros. Los miembros de la Unión han crecido desde los seis estados fundadores (Bélgica, Francia, Alemania, Italia, Luxemburgo y Holanda) a los 27 que hoy conforman la Unión Europea. Sin embargo, para realizar este ejercicio nos centraremos en los países de la Unión a 15. Esto es, los países fundadores más: Austria, Dinamarca, España, Finlandia, Grecia, Irlanda, Portugal, Reino Unido, y Suecia. Partiendo de la anterior información defina los siguientes predicados:

- **fundador(X)** que especifique los países que fundaron la Unión Europea.
- **vecino(X,Y)** que ponga en relación aquellos países que comparten frontera.

Partiendo de estas relaciones básicas, definir los siguientes predicados:

- **vecino_alemania(X)**, que se interpreta como “el país X es vecino de alemania”.
- **un_cruce(X,Y)**, que halle los países Y a los que se puede llegar desde X cruzando otro país.
- **ruta(X,Y,L)** que proporcione en L la lista de países que es necesario atravesar para ir de X a Y. La lista L debe incluir tanto el país de origen X como el de destino Y.

Ayuda: Los países pueden representarse mediante constantes. Naturalmente la sintaxis deberá ajustarse a la sintaxis de Prolog. Para uniformar criterios proponemos que los nombres simples se obtengan eliminando acentos y haciendo que la primera letra sea minúscula (ejemplo: **belgica**). En cuanto a los nombres compuestos, seguiremos el mismo criterio pero cada una de las partes estarán unidas por un guion bajo (ejemplo: **reino_unido**).

Ejercicio 3.10 En la aritmética de Peano la operación producto de dos números naturales se define mediante los siguientes axiomas:

- Para todo número natural n, $0 * n = 0$.
- Para todo número natural n y m, $s(n) * m = (n * m) + m$.

Traslade al lenguaje Prolog estas ecuaciones para definir un predicado **mult(N,M,P)** que devuelva en el argumento P el producto de los números naturales N y M.

Envío de los trabajos de la Práctica 3.

Los programas Prolog generados al realizar cada ejercicio de la segunda parte de esta práctica se almacenarán en un fichero con extensión “.pl”, denominado:

gN_Practica3.pl

donde “N” es el número del grupo (equipo de trabajo) al que pertenecen los alumnos. Además del código Prolog de todos los programas solicitados, se generará un fichero, denominado:

gN_Practica3.txt

con los datos personales de las personas que forman el grupo y, si fuera necesario el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

Estos ficheros se comprimirán formando un archivo ZIP y se enviarán para su corrección usando Moodle. El nombre del archivo ZIP debe ajustarse al siguiente formato:

gNN_APELLIDOS_P3.zip

donde, **NN** es el código del grupo, **APELLIDOS**, los apellidos del alumno responsable del grupo, y **P3**, es el código de la práctica. Ejemplo:

g1_Botella_Blanco_P3_B.zip

A efectos de la corrección automática de las prácticas **es importante que el código del grupo comience con una “g” minúscula**. También deben respetarse los nombres propuestos para los predicados definidos a lo largo de esta práctica, ya que si se usa un nombre diferente al propuesto el ejercicio se evaluará negativamente a pesar de que pueda haberse realizado correctamente.

Bibliografía

- [1] G. Allwein, D. Barker-Plummer, J. Barwise, J. Etchemendy, and A. Liu. *LPL Software Manual*. CSLI Publications, 2000.
- [2] K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, 1997.
- [3] K. R. Apt. The logic programming paradigm and prolog. *CoRR*, cs.PL/0107013, 2001.
- [4] J. Barwise and J. Etchemendy. *Language, Proof and Logic*. CSLI Publications, 2000.
- [5] A. Deaño. *Introducción a la lógica formal*. Alianza Universidad, Madrid, 1996.
- [6] M. Garrido. *Lógica Simbólica*. Tecnos, Madrid, 1997.
- [7] D. Hilbert and W. Ackermann. *Elementos de Lógica Teórica (3^a edición)*. Tecnos, Madrid, 1993.
- [8] P. Julián-Iranzo. *Lógica Simbólica para Informáticos*. Editorial RA-MA, 2004.
- [9] P. Julián-Iranzo and M. Alpuente-Frasnedo. *Programación Lógica: Teoría y Práctica*. Pearson Prentice Hall, 2007.
- [10] R.A. Kowalski. Predicate Logic as a Programming Language. In *Information Processing, IFIP'74*, pages 569–574. North-Holland, 1974.
- [11] B. Mates. *Lógica Matemática Elemental*. Tecnos, Madrid, 1974.
- [12] J.A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, January 1965.