Departamento de Ciencias de la Computación
e Inteligencia Artificial

Formalización en Isar de la metalógica de primer orden

Memoria presentada por
Fabián Fernando Serrano Suárez
para optar al grado de
Doctor en el Programa de
Lógica, Computación e Inteligencia Artificial
por la Universidad de Sevilla

V. B. Directores

José Antonio Alonso Jiménez Francisco Jesús Martín Mateos

Sevilla, 6 de febrero de 2012

Creo poder hacer muy clara la relación de mi conceptografía con el lenguaje común si la comparo con la que hay entre el microscopio y el ojo. Este último, por el campo de su aplicabilidad y la movilidad con que se sabe adaptar a las más diversas situaciones, posee gran superioridad frente al microscopio. Considerado como aparato óptico, muestra sin duda muchas imperfecciones, las cuales pasan desapercibidas, por lo común, sólo como consecuencia de su estrecha conexión con la vida mental. Pero tan pronto como los propósitos científicos establecen mayores exigencia en la precisión de las distinciones, el ojo resulta insuficiente. Por el contrario, el microscopio es de lo más apropiado para tales fines, aunque, por ello, no es utilizable para otros.

Gottlob Frege (1879) *Conceptografía (Un lenguaje de fórmulas, semejante al de la aritmética, para el pensamiento puro)* p. 8–9.

Agradecimientos

A la Secretaria y al Grupo de Lógica Computacional del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla por su hospitalidad durante mi permanencia en Sevilla; un reconocimiento especial a Francisco J. Martín por sus oportunas observaciones a la versión preliminar del trabajo de tesis.

Hoy que vuelvo a mis labores docentes recordé de José Antonio, mi asesor de tesis, que la sencillez es un principio esencial para que un Sistema de Razonamiento Complejo sea completo, correcto y eficiente.

Índice

1 Introducción **11**

1.1 Antecedentes . . . 12
1.2 Hipótesis y objetivos . . . 15
1.3 Metodología y plan de trabajo . . . 16
1.4 Estructura de la memoria . . . 18

2 Introducción a Isabelle/HOL/Isar **21**

2.1 Elementos básicos de Isabelle/HOL . . . 22
2.1.1 Teorías . . . 22
2.1.2 Inducción estructural . . . 25
2.1.3 Funciones recursivas . . . 26
2.1.4 Definiciones . . . 28
2.1.5 Definiciones inductivas . . . 28
2.1.6 Pruebas en Isabelle . . . 29
2.2 Pruebas estructuradas en Isar . . . 31
2.2.1 Esquemas de demostración . . . 33
2.3 Ejemplo . . . 35

3 Completitud de un sistema axiomático de la lógica proposicional **39**

3.1 Formalización del sistema axiomático . . . 39
3.1.1 La regla de debilitamiento . . . 42
3.1.2 Teoremas elementales . . . 43
3.1.3 Teorema de la deducción . . . 46

3.2 Semántica de la lógica proposicional . . . 52
3.3 Teorema de corrección del cálculo proposicional . . . 59
3.4 Teorema de completitud . . . 60
3.4.1 Prueba informal . . . 61
3.4.2 Prueba formal del teorema de completitud de la lógica proposicional 63

4 Forma normal conjuntiva 77
4.1 Sintaxis y semántica proposicional . . . 77
4.2 Notación uniforme . . . 82
4.3 Disyunciones y conjunciones generalizadas . . . 87
4.3.1 Semántica de las disyunciones y conjunciones generalizadas . . . 87
4.3.2 Equivalencia entre fórmulas . . . 89
4.4 Descripción de la formalización del algoritmo *FNC* . . . 90
4.5 Reglas de reescritura . . . 91
4.6 Algoritmo para hallar una forma normal conjuntiva . . . 94
4.7 Formalización de la terminación de la función *FNC* . . . 96
4.7.1 La funciones de medida . . . 97
4.8 Corrección del algoritmo *FNC* . . . 104

5 Método de prueba basado en tableros semánticos 107
5.1 Descripción de la formalización de la deducibilidad mediante tableros semánticos . . . 107
5.1.1 Definición de tablero semántico . . . 107
5.2 Sistema de prueba por tableros . . . 110
5.2.1 Terminación de la función *PruebaTablero* . . . 112
5.3 Semántica de los tableros semánticos . . . 114
5.3.1 Equivalencia entre tableros . . . 116
5.3.2 Corrección del algoritmo *PruebaTablero* . . . 118

6 Teorema de existencia de modelos proposicionales 121
6.1 Introducción . . . 121
6.2 Conjuntos consistentes . . . 123

6.3 Conjuntos consistentes y clausura por subconjuntos . . . 124
6.4 Propiedad de carácter finito . . . 127
6.5 Extensión a una propiedad de carácter finito . . . 128
6.6 Extensión a conjuntos consistentes maximales . . . 133
6.7 Conjuntos de Hintikka y modelos de Hintikka . . . 139
6.7.1 Conjuntos de Hintikka . . . 139
6.7.2 Satisfacibilidad de conjuntos de Hintikka . . . 139
6.8 Completitud del método de prueba basado en tableros semánticos . . . 145
6.9 Conjuntos maximales y conjuntos de Hintikka . . . 150
6.10 Enumeración de fórmulas proposicionales . . . 152
6.10.1 Enumeración de árboles binarios . . . 153
6.10.2 Enumeración de fórmulas proposicionales . . . 156
6.11 Formalización de la existencia de modelos proposicionales . . . 159

7 Aplicaciones del teorema de existencia de modelos proposicionales 167
7.1 Teorema de compacidad . . . 167
7.2 Teorema de interpolación de Craig . . . 176

8 Sintaxis y semántica de la lógica de primer orden 203
8.1 Sintaxis de la lógica de primer orden . . . 203
8.1.1 Índices de de Bruijn . . . 205
8.1.2 Símbolos de función de una fórmula . . . 206
8.1.3 Sustituciones . . . 207
8.2 Semántica de la lógica de primer orden . . . 210
8.2.1 Valor de verdad de una fórmula . . . 210
8.2.2 Satisfacibilidad y Consecuencia lógica . . . 214

9 Teorema de existencia de modelos de primer orden 217
9.1 Conjuntos consistentes . . . 217
9.1.1 Conjuntos consistentes y clausura por subconjuntos . . . 219
9.2 Propiedad de consistencia y consistencia alternativa . . . 222
9.2.1 Propiedad de consistencia alternativa . . . 222

9.3 Extensión de una propiedad de consistencia alternativa a una de carácter finito . . . 231
9.4 Extensión a conjuntos consistentes maximales . . . 235
9.5 Conjuntos de Hintikka y modelos de Herbrand . . . 243
9.5.1 Sentencias . . . 243
9.5.2 Conjuntos de Hintikka . . . 244
9.5.3 Estructuras de Herbrand . . . 245
9.5.4 Satisfacibilidad de conjuntos de Hintikka . . . 247
9.6 Conjuntos maximales y conjuntos de Hintikka . . . 254
9.7 Enumeración de fórmulas proposicionales . . . 267
9.7.1 Enumeración de árboles binarios . . . 268
9.8 Enumeración de las fórmulas de un lenguaje de primer orden . . . 270
9.8.1 Enumeración de términos . . . 271
9.8.2 Enumeración de fórmulas . . . 275
9.9 Formalización de la existencia de modelos de primer orden . . . 279
9.10 Teorema de Löwenheim-Skolem . . . 288

10 Deducción natural para la lógica de primer orden 299
10.1 Un sistema de deducción natural para la lógica de primer orden . . . 299
10.1.1 Ejemplos . . . 302
10.2 Teorema de corrección del sistema de deducción natural . . . 307
10.3 Completitud del sistema de deducción natural . . . 318

11 Conclusiones y trabajo futuro 349

Bibliografía 352

Capítulo 1

Introducción

Las pruebas correctas en matemáticas se caracterizan porque pueden ser descritas mediante una serie de pasos básicos válidos, es decir, usando razonamientos irrefutables que puedan ser verificados de manera simple. En particular, las pruebas correctas en lógica son demostraciones dentro de un sistema formal expresadas en un lenguaje formal [16], [33] (páginas 10-12).

Habitualmente las demostraciones en matemáticas de teoremas que resultan falsos, están escritas en un contexto donde se presuponen algunos conceptos y definiciones, se omiten casos o pasos lógicos, o parte de la argumentación está basada solamente en la intuición. Algunos ejemplos que ilustran este hecho aparecen en [23].

La corrección y la precisión son dos aspectos que conciernen al discurso de la formalización de las matemáticas. Para formalizar es necesario poder expresar las definiciones, los teoremas y las pruebas en un lenguaje formal, es decir, un lenguaje generado por una gramática no ambigua que permita la verificación ”mecánica” de las pruebas. En la formalización o verificación mecánica de teoremas se dispone de un lenguaje para crear documentos matemáticos, tal que el razonamiento puede ser verificado por el ordenador. En [16] y [33] se describen las características de las demostraciones mecánicas y su evolución desde la mitad del siglo pasado.

Para la formalización o verificación de pruebas de teoremas asistida por ordenador existen básicamente dos formas de hacerlo, de manera completamente automática, por el ordenador, y de manera interactiva usuario-ordenador. En los sistemas interactivos hay dos procedimientos de verificación, en el primero el usuario escribe el texto de la prueba y posteriormente el sistema verifica la corrección. En el segundo método, el sistema, llamado asistente de pruebas, genera en cada paso un estado de prueba que el usuario modífica por medio de “tácticas”, guiando de este modo la verificación de la prueba, en [3] se presenta un resumen de las propiedades de los probadores interactivos de teoremas (ITP).

Entre los asistentes de pruebas más utilizados en la formalización de las matemáticas, con base a una lista de 100 teoremas importantes seleccionados por Freek Wiedijk en la página web [59], están los sistemas HOL Ligth, Mizar, ProofPower, Isabelle/Isar, y Coq.

En este trabajo formalizamos algunos metateoremas de la lógica de primer orden: verificamos usando el asistente de pruebas Isabelle/Isar algunas demostraciones que se presentan en la metalógica acerca de propiedades de la lógica clásica.

1.1 Antecedentes

En esta sección presentamos un resumen cronológico de la *deducción automática*. Una exposición detallada aparece en [41], [30] y también en [12].

La idea general de la formalización de las matemáticas tiene su origen en la Grecia antigua. Se cree que Thales (624–547 AC) introdujo la noción de prueba y a él se le atribuyen cinco teoremas de la geometría. Platón (427–347 AC) inició un proceso de estructuración deductiva de la Matemática que se vería reflejada en el método axiomático de Aristóteles (384–332 AC) y en los Elementos de Euclides (325–265 AC).

Gottfried Leibnitz (1646–1716) concibió e intentó diseñar un lenguaje universal, en el que todo el conocimiento pudiera ser formalmente expresado, y un cálculo de razonamiento tal que el planteamiento y solución de problemas quedara reducido a computaciones numéricas [65].

George Boole (1815–1864) siguiendo las ideas de Leibnitz escribió el libro llamado "Las Leyes del Pensamiento" en donde describe un método algebraico (álgebra de Boole) para la solución de problemas prácticos a través del planteamiento de ecuaciones simples [65].

Gottlob Frege (1848–1925) inició el estudio de la lógica moderna introduciendo los cuantificadores en su teoría de la cuantificación. A diferencia de Leibnitz y Boole, que únicamente consideraban símbolos proposicionales para tratar de reducir problemas de la lógica a solo cálculos y por lo tanto sin considerar reglas de prueba, Frege trató de expresar los conceptos de las matemáticas en afirmaciones de la lógica, incluyendo el concepto de número [28], [65].

Alfred North Whitehead (1861–1947) y Bertrand Russell (1872–1970) publicaron entre 1910 y 1913 su obra *Principia mathematica* que permitió apreciar el poder de la lógica formal al intentar deducir la mayor parte de los conocimientos matemáticos de la época a partir de un conjunto de principios o axiomas y además mostró que el trabajo en lógica de Gottlob Frege sobre las leyes básicas de la aritmética contenía inconsistencias, en particular la que se conoce como la paradoja de Russell, las cuales fueron evitadas en los

Principia mediante la construcción de la teoría de tipos [28], [65].

David Hilbert (1862–1943) contribuyó al estudio de reglas para el razonamiento de que trata la lógica formal, y se mostró interesado en combinar las ideas de Leibnitz y Frege para formalizar teorías matemáticas en teorías de la lógica usando lenguajes formales, y a partir de ahí comprobar mecánicamente la validez de las pruebas. Así, con el "Programa de Hilbert" la idea era formalizar las diferentes ramas de las matemáticas y entonces probar mecánicamente que, cada una de ellas, así formalizada, estaba libre de contradicciones [65].

Kurt Gödel (1906–1978) en 1931 mostró que la formalización no puede considerarse como una técnica matemática, a través de la cual se pueda probar que las matemáticas están exentas de contradicciones: cualquier sistema formal que contenga la teoría básica de los números naturales, no puede probar su propia consistencia [28].

Adicionalmente, Hilbert estudió el problema de la decisión, para la lógica de primer orden, que consiste en determinar si existe un algoritmo general de decisión tal que para cualesquier conjunto de axiomas y conjetura, calcule en un tiempo finito una prueba de la conjetura a partir de los axiomas en caso de que tal prueba exista, o en caso contrario una respuesta de que tal prueba no existe.

Alonzo Church (1903–1995) en 1936 utizando las nociones de lambda cálculo, cálculo efectivo y funciones recursivas mostró que tal algoritmo en el cálculo de primer orden no existe: el conjunto de fórmulas lógicamente válidas del cálculo de predicados no es decidible [11].

Alan Turing (1912–1954), independientemente y en forma casi simultánea redujo el problema de decisión al problema de la parada para maquinas de Turing, para demostrar que tal algoritmo no existe.

Es importante notar que si se restringe el problema a una teoría de primer orden específica, es posible que exista un algoritmo de decisión para la teoría. Por ejemplo, la aritmética de Presburger y la lógica proposicional son teorías decidibles.

Sin embargo, la teoría general de primer orden para los números naturales conocida como la aritmética de Peano no puede ser decidida con ese tipo de algoritmo.

El primer teorema de la incompletitud de Gödel demuestra que cualquier sistema que permita definir los números naturales no es completo: contiene afirmaciones que ni se pueden demostrar ni refutar. A pesar de lo anterior, Gödel en 1930, en su tesis doctoral, demostró el teorema de completitud: en la lógica de primer orden si una formúla es lógicamente válida entonces es demostrable [28].

Una consecuencia importante de los resultados teóricos obtenidos por Turing, Church, y Gödel fue el desarrollo de los fundamentos de la teoría de la computación y del estudio de algoritmos para la demostración automática de teoremas.

Jacques Herbrand (1908–1931) en 1930 demostró uno de los teoremas más importantes en los que se basan algunos métodos de demostración automática de teoremas. El teorema de Herbrand permite reducir el problema de la validez lógica al caso de fórmulas sin cuantificadores, y al cálculo puramente proposicional [28], [65].

Gerhard Gentzen (1904–1945) en 1934 introduce la noción de sistema de deducción natural para la lógica clásica y la lógica intuicionista. Gentzen demostró que toda prueba puede escribirse de manera normalizada sin cortes [65].

R.M. Smullyan desarrolló en los años 60 el método de los tableros semánticos [50], introducido por E.W Beth a finales de la década de los años 1950 [65]. Este método de refutación está basado en una búsqueda exhaustiva de contramodelos.

P.C. Gilmore, en 1960 fue el primero en presentar una implementación en un ordenador del procedimiento de Herbrand [65].

Davis y Putnam mejoraron el método de Gilmore en el mismo año de 1960 pero sin presentar ninguna implementación [62].

J.A. Robinson, con base al teorema de Herbrand, desarrolló en 1965 su método de Resolución de Prueba de Teoremas [47], el cual se basa en una sola regla de inferencia denominada principio de resolución. La idea principal del principio de resolución es derivar una nueva sentencia de otras dos usando el concepto de unificación.

En la segunda mitad del siglo veinte con la invención de los ordenadores empiezan a aparecer los primeros programas de ordenador que implementaban algunos procedimientos de decisión y demostradores automáticos de teoremas. Por ejemplo, el demostrador de teoremas "Logic Theory Machine", desarrollado por A. Newell, H. Simon [1],[61], fue implementado por J.C. Shaw en 1955 para probar los teoremas de la lógica proposicional de los "Principia Matematica" de B. Russell.

La construcción de asistentes de pruebas formales tuvo su inicio en 1967 con el proyecto Automath de De Brujin Automath Archive que tenía como objetivo el desarrollo de un sistema para la verificación mecánica de las matemáticas, en particular un lenguaje formal para representar pruebas matemáticas en el ordenador [16].

Entre los sistemas interactivos más conocidos se tienen los siguientes:

- El demostrador automático de Boyer y Moore Nqthm [66]. Tiene su origen en 1972 y fue implementado en Lisp; la lógica de Nqthm es una lógica de primer orden libre de cuantificadores y con igualdad, básicamente aritmética recursiva primitiva. Posee un alto grado de automatización pero poca expresividad. Basado en las mismas ideas, un descendiente de este sistema es el demostrador automático de teoremas ACL2 [63].

- El sistema Coq [15] implementado en el lenguaje ML está basado en una teoría intuicionista de tipos, un cálculo de construcciones inductivas.

- El sistema Mizar [60] basado en la teoría de conjuntos e implementado en el lenguaje Pascal. Posee una librería bastante completa que contiene los textos en forma de artículos de la formalización de resultados de matemáticas MML, los cuales son publicados en la revista Formalized Mathematics. Sin embargo, Mizar tiene grado bajo de automatización y por lo tanto, en particular, su razonamiento es bastante laborioso.

- El sistema HOL–Light [27] en el que se han formalizado un gran número de teoremas, es una versión simplificada del sistema HOL que está basado en una lógica de orden superior e implementado en el lenguaje OCaml.

- El sistema Isabelle/HOL [52] implementado en ML no está basado sobre una lógica específica. Le permite al usuario elegir entre varias lógicas, como HOL, FOL y ZF, y escribir pruebas formales en forma bastante técnica como en Coq y HOL; o en forma declarativa como en Mizar, usando el lenguaje *Isar*, de tal forma que las presentaciónes de las pruebas sea legibles y muy similares a las que aparecen en los textos de matemáticas. Además soporta un alto grado de automatización.

Los siguientes ejemplos sobre la formalización de teoremas importantes en el desarrollo de las matemáticas, muestran el alcance de los asistentes de pruebas en la construcción y verificación de demostraciones complejas:

- Primer teorema de incompletitud de Gödel: por Natarajan Shankar usando el asistente de pruebas Nqthm en 1986 [48] y por Russell O'Connor usando Coq en 2003 [42].

- Teorema de la curva de Jordan: por Thomas Hales usando HOL Light en 2005 y por Artur Korniłowicz usando Mizar en 2005, [22].

- Formalización de la conjetura de Kepler en 2007 por Thomas Hales [21].

- Teorema de los números primos: por Jeremy Avigad usando Isabelle en 2004 [4], y por John Harrison usando HOL Light in 2008 [25].

- Teorema de los cuatro colores: por Georges Gonthier usando Coq in 2004 [17].

1.2 Hipótesis y objetivos

El presente trabajo se basa en la hipótesis de que el estado actual de desarrollo de los sistemas de razonamiento asistido permiten abordar la formalización de teorías complejas de forma que las demostraciones formalizadas se correspondan fielmente a las

demostraciones que aparecen en los textos matemáticos. El objetivo del trabajo es confirmar la hipótesis anterior, es decir, formalizar razonamiento complejo de forma legible por los humanos y procesable por las máquinas. Más precisamente, formalizar la metateoría de la lógica de primer orden como se expone en los primeros cinco capítulos del libro de Fitting [14] usando el sistema de razonamiento Isabelle/HOL/Isar.

Se ha seleccionado el texto de Fitting [14] puesto que en él se exponen las demostraciones de manera informalmente rigurosas, es decir, se argumentan de forma precisa los resultados principales en los que se basa una prueba determinada y cómo se estructuran para justificar el resultado principal; además, con la intención pedagógica de no oscurecer este razonamiento, algunos de los resultados usados no se demuestran y se proponen como ejercicio al lector. Esta forma tradicional de presentación de una demostración natural, permite estudiar de manera real las dificultades que se pueden presentar al intentar reflejar fielmente una prueba natural en una prueba mecanizada.

1.3 Metodología y plan de trabajo

Para la formalización de cada uno de los metateoremas objeto de estudio se seguirán los siguientes pasos:

- A partir de un análisis de las demostraciones que se presentan en los textos clásicos de lógica y haciendo énfasis en la demostración que aparece en el texto de Fitting [14], se escribe una prueba matemática del metateorema correspondiente en donde se hacen explícitas todas las hipótesis, se tratan por completo los diferentes casos y se justifican los razonamientos intuitivos.
- Guiados por la prueba establecida en el punto anterior, se formalizan en Isar las definiciones y lemas necesarios para la prueba formal.
- Por último, con base a los puntos anteriores, se desarrolla la prueba formal en Isar.
- Para hacer la presentación de la prueba formal más legible, en el documento en donde esta se incluye, se "ocultan" los lemas "técnicos", que básicamente corresponden a los resultados auxiliares o intuitivos que no se mencionan en la prueba natural.

Por otro lado, los temas cubiertos en el texto de Fitting [14] hacen referencia a la formalización de las propiedades que tienen que ver con los fundamentos teóricos de la prueba automática de teoremas (metaformalización). Así, el presente trabajo trata principalmente de los siguientes tres temas:

- Formalización de la corrección y completitud de cálculos de prueba. Se estudian, en la lógica proposicional el sistema axiomático $\mathcal{H}$ de Hilbert y un sistema deductivo basado en tableros semánticos, además de la correción del algoritmo general para hallar una forma normal conjuntiva de una fórmula proposicional, y en la lógica de primer orden un sistema de deducción natural. Estos cálculos de prueba son de características distintas. Los sistemas axiomáticos de Hilbert que se caracterizan por tener pocas reglas de inferencia y en general un gran número de esquemas de axiomas, no son apropiados para la prueba automática de teoremas debido a la gran cantidad de combinaciones posibles de poder aplicar los axiomas al intentar una prueba. El método de prueba basado en tableros semánticos, a diferencia de los sistemas axiomáticos, no genera un sucesión de conclusiones a partir de un conjunto de hipótesis, es una técnica de carácter semántico, de busqueda de modelos para una fórmula. La deducción natural es un sistema formal para representar las pruebas en lógica de primer orden de forma aproximada a las que realizan los humanos. Contrariamente a los sistemas de Hilbert basados en un conjunto de axiomas, el sistema de deducción natural amplía el conjunto de reglas de inferencia clasificándolas en reglas de introducción y eliminación.

- Formalización del teorema de existencia de modelos en la lógica proposicional y en la lógica de primer orden. Este resultado que garantiza la existencia de un modelo para un conjunto de fórmulas proposicionales o sentencias de primer orden, relaciona la sintaxis y semántica de un lenguaje de primer orden: cualquier conjunto de fórmulas consistente es satisfactible. La formalización en Isabelle/Isar corresponde a la prueba descrita en el texto de Fitting [14], la cual está basada en un argumento abstracto de completitud que generaliza los argumentos comunes que se utilizan en la mayoria de las demostraciones indirectas de la completitud de procedimientos de prueba tales como, sistemas de Hilbert, sistemas de Gentzen, resolución, tableros semánticos, etc. Tales demostraciones de completitud hacen uso del concepto de consistencia, que es relativo a cada procedimiento de prueba particular. Un conjunto de fórmulas es consistente, con respecto a un procedimiento de prueba, si ninguna contradicción se puede derivar a partir del conjunto usando el cálculo de prueba específico. Sin embargo, Fitting usa el enfoque más general de caracterizar el concepto de consistencia, sin hacer uso de la noción de "cálculo de prueba", por medio de las propiedades básicas que todos los conjuntos consistentes deben tener.

- Aplicaciones del teorema de existencia de modelos en:

  (a) La formalización de importantes metateoremas de la lógica clásica. Específicamente, en la lógica proposicional, el teorema de compacidad y el teorema de interpolación de Craig, y en la lógica de primer orden el teorema de Löwenheim-Skolem.

(b) La formalización de la completitud de sistemas de prueba, en nuestro caso, del sistema de deducción natural para la lógica de primer orden.

De los anteriores temas, en [6] aparece la formalización aplicativa en Isabelle de: el teorema de existencia de modelos en la lógica de primer orden, el teorema de Löwenheim-Skolem y el sistema de deducción natural para la lógica de primer orden. En este trabajo se desarrollan estos temas en Isar a un nivel de detalle tal que la construcción de la formalización es análoga a la presentación declarativa de las pruebas naturales.

1.4 Estructura de la memoria

Como lo hemos señalado anteriormente, la formalización de los temas que se estudiarán en esta memoria será en Isabelle/HOL/Isar.

En el capítulo 2 se presenta una introducción de los principales conceptos de Isabelle/HOL/Isar utilizados en el desarrollo de este trabajo.

En el capítulo 3 se presenta una formalización de un sistema axiomático de la lógica proposicional y la demostración de la completitud del sistema usando el método de Kalmar de eliminación de variables. La formalización se basa en la del libro de Mendelson [36] y en el desarrollo en Isabelle que aparece en [49].

El capítulo 4 comienza con la formalización de la noción de notación uniforme para las fórmulas de la lógica proposicional, que consiste en particionar las fórmulas en cuatro clases de fórmulas *literal, nono, alfa y beta*. El uso de esta notación para las fórmulas proposicionales, permitirá describir de manera más simple los conceptos que se estudian en los capítulos 4, 5, 6 y 7 referentes a la lógica proposicional. Posteriormente, en este capítulo, se presenta una formalización de la terminación y corrección del algoritmo general para calcular una forma normal conjuntiva de una fórmula proposicional. Para la terminación del algoritmo se define una función de medida en términos del concepto de *multiconjunto* y del *rango* de una fórmula escrita en notación uniforme.

En el capítulo 5 se formaliza la corrección de un procedimiento de prueba de tautologías usando tableros semánticos. La terminación se prueba de la misma forma que en el caso del algoritmo de la forma normal conjuntiva. Los tableros semánticos son árboles finitos etiquetados por fórmulas, donde cada rama representa la conjunción de sus nodos y el árbol la disyunción de sus ramas. Los tableros utilizados en el procedimiento de prueba son *estrictos*, es decir, las "reglas" que permiten su construcción no pueden ser aplicadas dos veces sobre una misma rama. Como consecuencia, en este caso, el teorema de existencia de modelos no es apropiado para demostrar la completitud del método de prueba. por lo tanto, en el capítulo 6 se utiliza el lemma de Hintikka para tal fin. Una formalización diferente en ACL2 se presenta en [35] basada en la

noción de "eventos".

En el capítulo 6 se formaliza la demostración del teorema de existencia de modelos para un conjunto consistente S de fórmulas pertenecientes a un lenguaje proposicional L. Esta demostración está basada en extender S a un conjunto consistente maximal M que se demuestra que es consistente (por ser un conjunto de Hintikka) y, por tanto, S también es consistente.

Adicionalmente, como una aplicación del lema de Hintikka se formaliza la completitud del método de prueba basado en tableros semánticos estudiado en el capítulo 5.

En el capítulo 7 como una aplicación del teorema de existencia de modelos se formaliza el teorema de compacidad: si todos los subconjuntos finitos de un conjunto S de fórmulas proposicionales tienen modelos, entonces S tiene un modelo; también se formaliza el teorema de interpolación de Craig: si $X \longrightarrow Y$ es una tautología, entonces tiene un interpolante; es decir, existe una fórmula Z tal que cada símbolo proposicional que ocurre en Z también ocurre en X y Y, y $X \longrightarrow Z$ y $Z \longrightarrow Y$ son tautologías.

En el capítulo 8 se presenta la formalización de la sintaxis y semántica de la lógica de primer orden. Para la representación de las fórmulas se emplea la notación de de Bruijn [13]; un índice de de Bruijn es un número natural n que representa una ocurrencia de una variable en un término. Para la representación de fórmulas cuantificadas utilizando índices de de Bruijn se omite la variable que acompaña a cada cuantificador, y en este caso un índice de de Bruijn en una fórmula denota el número de cuantificadores que aparecen entre la variable respectiva y el cuantificador que liga a la variable. En [44] se explica el uso de la notación de de Bruijn, en el diseño de un demostrador automático de teoremas, para la representación de fórmulas de primer orden. Ejemplos de esta aplicacion aparecen en [2], [29], [38]. La ventaja de esta representación es que permite automatizar de manera simple y eficiente el proceso de sustitur en una fórmula las ocurrencias libres de una variable x, por un término t libre para x, [10]; además permite considerar como idénticas las fórmulas que difieren únicamente en los nombres de sus variables cuantificadas.

En el capítulo 9 se formaliza el teorema de existencia de modelos en la lógica de primer orden siguiendo el mismo orden de ideas que en el caso proposicional. Sin embargo, hay algunas diferencias, que en consecuencia hacen que las demostraciones sean más complejas:

- La demostración del teorema de existencia de modelos está basada en poder extender una propiedad de consistencia a una propiedad de consistencia que sea cerrada por subconjuntos y de carácter finito. Sin embargo, para el caso de primer orden tal extensión no es tan fácil de realizar como en el caso proposicional; para facilitar tal extensión se introduce el concepto de propiedad de consistencia alter-

nativa.

- Para demostrar, en el caso de primer orden, que el conjunto maximal M que se construye a partir del conjunto consistente S, es consistente, es necesario que el conjunto de símbolos de función (parámetros) que ocurren en las fórmulas de S sea finito.
- Para la demostración del lema de Hintikka es necesario introducir el concepto de modelos de Herbrand.

Por último, en este capítulo, como una aplicación del teorema de existencia de modelos en la lógica de primer orden, se formaliza la demostracion del teorema de Löwenheim-Skolem: si S es un conjunto de sentencias satisfactible, entonces S es satisfactible en un modelo numerable (de Herbrand).

En el capítulo 10 se define un sistema de deducción natural para lo lógica de primer orden. Se formaliza su corrección y como una consecuencia del teorema de existencia de modelos para la lógica de primer orden, se formaliza su completitud.

En el capítulo 11 se presentan las conclusiones y trabajo futuro.

Capítulo 2

Introducción a Isabelle/HOL/Isar

Isabelle ([43], [45]) es un sistema genérico para la prueba interactiva de teoremas y la implementación de formalismos lógicos; fue implementado en el lenguage de programación funcional ML, y desarrollado por Larry Paulson y Tobias Nipkow. Se encuentra disponible en Isabelle (http://isabelle.in.tum.de).

Isabelle forma parte de la familia de probadores interactivos de teoremas, basados en tácticas, característica heredada del sistema LCF (Logic for Computable Functions) [18], [32], que permite construir tácticas con el fin de simplificar la aplicación mecánica de reglas de inferencia en el proceso de deducción [43].

Isabelle soporta el razonamiento formal en varias lógicas objeto: lógicas de primer orden constructiva y clásica (FOL), lógicas de orden superior (HOL [8], [20]), teoría de conjuntos de Zermelo–Fraenkel (ZF), teoría de tipos de Martin Löf, lógicas modales, etc.

En particular Isabelle/HOL [52] es la especialización de Isabelle para la lógica de orden superior HOL (Higher-Order Logic). Es un asistente de demostraciones interactivas, basado en el sistema HOL de Gordon [19],[51], que implementa una extensión de la lógica de orden superior de Church [11]. Tiene una amplia librería de teorías definidas que incluye teoría de conjuntos, números reales y complejos, álgebra abstracta, etc. Esta librería aparece en HOL (Higher-Order Logic).

Isabelle/HOL ha sido utilizado exitosamente para el razonamiento formal en distintas áreas del conocimiento: Matemática pura [59], verificación de sistemas de computación, lenguajes de programación, etc, [34]. En Archive of Formal Proofs se encuentran disponibles una variedad de teorías que corresponden a ejemplos y desarrollos científicos que han sido formalmente verificados en Isabelle y en particular en Isabelle/HOL.

Isabelle/HOL ofrece dos facilidades para encontrar contraejemplos, a partir de instancias aleatorias de variables libres Quickcheck [7] y a partir de modelos finitos refute [53].

Por otro lado, Isabelle/Isar es una extensión de Isabelle que permite pruebas estructuradas en el lenguaje *Isar* ("Intelligible semi-automated reasoning") [37], [58]. Así, Isar es una extensión de Isabelle que permite construir, en forma semiautomática, pruebas estructuradas de teoremas en un lenguaje de alto nivel. El lenguaje Isar soporta documentos de pruebas formales legibles por los humanos y verificables por la máquina [54], [57]. Las teorías, teoremas, procedimientos de prueba, etc, se pueden usar igualmente en las pruebas aplicativas clásicas de Isabelle y en los documentos de Isabelle/Isar. Isar al ser tan genérico como Isabelle puede soportar una variedad de lógicas objeto.

En las siguientes secciones se introducen los elementos básicos de Isabelle/HOL/Isar utilizados en este trabajo para la formalización de las correspondientes teorías.

2.1 Elementos básicos de Isabelle/HOL

En esta sección se ilustran algunos aspectos importantes de Isabelle/HOL que aparecen descritos en [52].

2.1.1 Teorías

La formalización en Isabelle de la solución de un problema consiste en definir teorías. Las teorías son módulos que contienen información de una lógica, y en el caso de aplicaciones contienen definiciones, términos, fórmulas, tipos de datos, funciones, teoremas, etc, que describen la solución de un problema.

Cuando usamos Isabelle/HOL para razonar acerca de un dominio, definimos teorías como extensiones de otras teorías y en consecuencia podemos disponer de los tipos y estructuras de datos definidos en otras teorías. Isabelle/HOL contiene una teoría *Main* que incluye las teorías básicas predefinidas, tales como la aritmética de los números naturales, las listas y los conjuntos. El formato general de una teoría T es:

theory T
imports $T_1 \ldots T_n$
begin
declaraciones, definiciones y pruebas
end

donde $T_1 \ldots T_n$ son los nombres de teorías existentes con respecto a las cuales se define T. En la parte de *declaraciones, definiciones, y pruebas* se introducen los nuevos conceptos tales como tipos, funciones, teoremas, etc, y las respectivas pruebas.

Cada teoría contiene tipos, términos y fórmulas de HOL. HOL es una lógica tipada, cuyo sistema de tipos se asemeja al de lenguajes funcionales como ML o Haskell. Los

términos se construyen como en programación funcional, aplicando funciones a argumentos, y las fórmulas son términos de tipo *bool* cuya sintaxis está dada por la siguiente gramática:

$$\begin{array}{rl} form ::= & (form) \mid True \mid False \\ & \mid termino = termino \mid \neg form \mid form \wedge form \mid form \vee form \\ & \mid form \longrightarrow form \\ & \mid \forall x.form \mid \exists x.form \end{array}$$

Por ejemplo, la siguiente teoría resuelve el problema: definir una función primitiva recursiva $(pot\ x)$ que calcule x^n en los números naturales y demostrar la siguiente identidad $pot\ x\ (m * n) = pot\ (pot\ x\ m)\ n$.

```
theory Potenciacion
imports Nat
begin

primrec pot :: nat => nat => nat where
 pot x 0      = Suc 0
| pot x (Suc n) = x * pot x n

lemma pot-adic: pot x (m + n) = pot x m * pot x n
 apply (induct n)
 apply auto
 apply(rule mult-ac)
 done

theorem pot-mult: pot x (m * n) = pot (pot x m) n
 apply (induct n)
 apply auto
 apply (simp add: pot-adic)
 done

end
```

En el ejemplo anterior se observa lo siguiente.

La teoría *Nat.thy* contiene la especificación de los números naturales. En particular la definición del tipo de dato números naturales *nat*,

datatype *nat* = 0 | *Suc nat*

el cual incluye la definición de funciones tales como $+$, $*$ y la relación $\leq$. Otros tipos predefinidos en HOL son el tipo básico valores booleanos,

datatype *bool* = *True* | *False*,

el tipo constructor lista,

datatype *'a list = Nil | Cons "'a" "'a list"*

donde *'a* es un parámetro de tipo genérico, y el tipo constructor conjunto *'a set*, que en HOL es simplemente un predicado,

types *'a set = "'a ⇒ bool"*

y el cual incluye las operaciones básicas entre conjuntos.

La función *pot* es un ejemplo de definición primitiva recursiva sobre los números naturales. La funciones sobre tipos de datos son usualmente definidas por recursión, en la mayoría de los casos por recursión primitiva. En este caso la palabra reservada **primrec** es seguida de una lista de ecuaciones $f\ x_1 \ldots (C\ y_1 \ldots y_k) \ldots x_n = r$ en donde C es un constructor del tipo de dato t y todas las llamadas recursivas de f en r son de la forma $f \ldots y_i \ldots$ para algun i.

El lema *pot-adic* prueba la identidad $x^{(m+n)} = x^m \cdot x^n$. El comando **lemma** inicia la prueba y el comando **apply**(*induct n*) le indica a Isabelle que comience la prueba por inducción sobre n. En respuesta, Isabelle muestra el siguiente estado de la prueba:

1. $pot\ x\ (m + 0) = pot\ x\ m * pot\ x\ 0$
2. $\bigwedge n.\ pot\ x\ (m + n) = pot\ x\ m * pot\ x\ n \Longrightarrow pot\ x\ (m + Suc\ n) = pot\ x\ m * pot\ x\ (Suc\ n)$

Las afirmaciones (1) y (2) son conocidas como subobjetivos. El primer subobjetivo es el caso base, el segundo el paso de inducción. El prefijo $\bigwedge n.$ significa "para un número n arbitrario pero fijo". El $\Longrightarrow$ separa las hipótesis de la conclusión.

Los símbolos $\bigwedge$ y $\Longrightarrow$ representan en Isabelle cuantificación universal e implicación, la otra conectiva de la metalógica de Isabelle es la igualdad representada por $\equiv$.

El comando **apply**(*auto*) orienta a Isabelle a resolver todos los subobjetivos automáticamente, principalmente por simplificación de términos.

El caso base $pot\ x\ (m + 0) = pot\ x\ m * pot\ x\ 0$ se tiene por la definición de *pot*. Para resolver el paso de inducción, el comando **apply**(*rule mult-ac*) le indica a Isabelle que utilice el lema *mult-ac*, que hace parte de la teoría *Nat.thy*, el cual afirma que se cumplen las siguientes identidades:

*?a * ?b * ?c = ?a * (?b * ?c)*

*?a * ?b = ?b * ?a*

*?b * (?a * ?c) = ?a * (?b * ?c)*.

para cualesquiera ?a, ?b, ?c números naturales.

En el enunciado de este lema los nombres de las variables comienzan por ?, esto

significa que son variables esquemáticas. Isabelle distingue tres clases de variables: libres, ligadas, y esquemáticas. Las variables esquemáticas son variables libres que en una demostración pueden ser instanciadas por unificación con otros términos. De esta forma, Isabelle resuelve el anterior suobjetivo por unificación con la tercera identidad. Por último, el teorema *pot-mult* es demostrado en forma similar que el lema *pot-adic*. Obsérvese que para demostrar el teorema *pot-mult* fue conveniente primero demostrar el lema *pot-adic*.

2.1.2 Inducción estructural

En Isabelle puede hacerse inducción estructural sobre cualquier tipo recursivo, la inducción matemática es un caso particular de inducción estructural sobre el tipo de los números naturales. Por ejemplo el esquema de inducción estructural sobre listas es

$$\frac{P\ [] \qquad \bigwedge a\ list.\ \dfrac{P\ list}{P\ (a{\cdot}list)}}{P\ list}(list.induct)$$

De esta forma, podemos utilizar el principio de inducción estructural para demostrar propiedades de funciones definidas por recursión primitiva (sobre un tipo de dato recursivo). Por ejemplo, teniendo en cuenta que en la teoría *List.thy* está definida la concatenación de listas (@) como sigue

primrec *append :: "'a list ⇒ 'a list ⇒ 'a list"* (**infixr** *"@" 65*)
where
append-Nil: "[] @ ys = ys"
| *append-Cons: "(x # xs) @ ys = x # xs @ ys"*

La anotación (**infixr** *"@" 65*) permite utilizar una sintaxis más cercana a la usada en matemáticas. En este caso definimos @ como un operador infijo con asociatividad por derecha.

La propiedad asociativa de la operación @ se puede demostrar por inducción estructural como sigue.

lemma *conc-asociativa*: $xs\ @\ (ys\ @\ zs) = (xs\ @\ ys)\ @\ zs$

iniciamos la prueba por indución estructural:

apply(*induct xs*)

Así, hay que resolver los dos subobjetivos:

1. $[]\ @\ ys\ @\ zs = []\ @\ ys\ @\ zs$
2. $\bigwedge a\ xs.\ xs\ @\ ys\ @\ zs = xs\ @\ ys\ @\ zs \Longrightarrow a{\cdot}xs\ @\ ys\ @\ zs = a{\cdot}xs\ @\ ys\ @\ zs$

los cuales se prueban automáticamente:

apply *auto*
done

2.1.3 Funciones recursivas

En HOL todas las funciones son totales. Cuando no es posible definir una función recursiva total por recursión primitiva, se utiliza el comando **fun** o el comando **function** [31]. Las ecuaciones que definan la función deben corresponder al patrón de definición del tipo de dato sobre el cual se está definiendo la función y en algunos casos, en donde dos ecuaciones coinciden con un mismo patrón, el orden de las ecuaciones es importante. Por ejemplo, la formalización de la definición recursiva de la sucesión de Fibonacci es la siguiente.

fun *fib* :: *nat* => *nat*
where
fib 0 = *1*
| *fib* (*Suc 0*) = *1*
| *fib* (*Suc* (*Suc n*)) = *fib n* + *fib* (*Suc n*)

Para que Isabelle pruebe automáticamente la terminación de una función definida de esta forma, es necesario que en las llamadas recursivas, los argumentos de la función en la parte derecha de cada ecuación sean estrictamente menores que los argumentos correspondiente de la parte izquierda. En los casos en que esto no es posible, el usuario debe definir la *relación bien fundamentada* o la *función de medida* que permita probar la terminación de la función. Para hacer esto último y en general, para probar con la ayuda del usuario que una función recursiva está bien definida, es conveniente definir la función usando el comando **function**. Por ejemplo, la siguiente función *mezclar* que combina los elementos de dos listas de la siguiente forma:

mezclar [a1, a2, a3] [b1, b2, b3] = [a1, b1, a2, b2, a3, b3] y

mezclar [a1] [b1, b2, b3] = [a1, b1, b2, b3]

se puede definir recursivamente por,

function *mezclar* :: *'a list* ⇒ *'a list* ⇒ *'a list* **where**
mezclar [] *ys* = *ys*
| *mezclar xs* [] = *xs*
| *mezclar*(*x*#*xs*) (*y*#*ys*) = *x* # (*mezclar* (*y*#*ys*) *xs*)
apply *pat-completeness*
apply *auto*
done
termination *mezclar*

apply (*relation measures* [$\lambda(xs, ys)$. *length xs* + *length ys*])
apply *auto*
done

En la prueba anterior, la combinación de los métodos *pat-completeness* y *auto* permiten probar la completitud y la compatibilidad de las ecuaciones que definen la función con respecto al patrón que define el tipo de dato sobre el cual se está definiendo la función.

El comando **termination** indica el comienzo de la prueba de terminación de la función *mezclar*.

Al método *relation* hay que asociarle una relación R de la forma $('b \times 'b)$ *set* donde *'b* es el tipo del argumento de la función o, si la función tiene más de un argumento, una tupla correspondiente al número y al tipo de los argumentos de la función, como sucede en el ejemplo de arriba. Esta relación deber ser *bien fundamentada* y de tal forma que los argumentos de la función en las llamadas recursivas decrezcan con respecto a esta relación.

En el ejemplo, la relación R está definida a partir de la función predefinida *measure* que a cada función de la forma m: $'b \to \mathbb{N}$, llamada *función de medida* le asocia una relación bien fundamentada. En este caso la función de medida m: $('a \times 'a) \to \mathbb{N}$ está definida por, $m(l_1, l_2) =$ la suma de las longitudes de l_1 y l_2.

El comando **apply**(*relation measures* [λ(*xs*,*ys*). *length xs* + *length ys*]), indica que debemos demostrar que la relación *measures* [λ(*xs*,*ys*). *length xs* + *length ys*]) es bien fundamentada y que los argumentos de la función *mezclar* en las llamadas recursivas decrecen con respecto a esta relación:

1. *wf* (*measures* [λ(*xs*,*ys*). *length xs* + *length ys*])
2. $\bigwedge$*x xs y ys*. ((*y*#*ys*,*xs*), *x*#*xs*, *y*#*ys*) $\in$ *measures* [λ(*xs*,*ys*). *length xs* + *length ys*]

Estos objetivos, en este caso, son resueltos por **auto**.

Las funciones recusivas tienen su propio esquema de inducción que refleja el principio de inducción estructural. Así, por ejemplo, al formalizar en Isabelle la definición de la función *mezclar* se tiene la siguiente regla de inducción:

$$\frac{\bigwedge ys.\ P\ []\ ys \qquad \bigwedge xs.\ P\ xs\ [] \qquad \bigwedge x\ xs\ y\ ys.\ \dfrac{P\ (y{\cdot}ys)\ xs}{P\ (x{\cdot}xs)\ (y{\cdot}ys)}}{P\ a0\ a1}(mezclar.induct)$$

Por ejemplo, el siguiente lema se demuestra automáticamente usando la anterior regla de inducción.

lemma *length* (*mezclar x y*) = *length x* + *length y*
apply(*induct x y rule*: *mezclar.induct*)
apply *auto*
done

2.1.4 Definiciones

En ocasiones es útil introducir, por medio de definiciones, nuevos conceptos en términos de conceptos ya establecidos. En Isabelle podemos introducir definiciones con la palabra **definition**. Por ejemplo,

definition
divide :: *nat* ⇒ *nat* ⇒ *bool* (**infixr** *DVD 100*)
where
m DVD n = (∃ *k*. *n* = *m* ∗ *k*)

En este caso definimos *DVD* como un predicado infijo con asociativadad por derecha.

2.1.5 Definiciones inductivas

Las definiciones recursivas comienzan con la palabra *inductive* y un nombre de *predicado*. En general tienen la siguiente forma:

inductive I :: "τ ⇒ *bool*" **where**

seguida por una sucesión (posiblemente vacía) de reglas compuestas de premisas y una conclusión, separadas por |. Una regla con n premisas tiene la forma,

$$[\![I_1; I_2; \ldots; I_n]\!] \Longrightarrow I'.$$

el caso $n = 0$ corresponde a una regla sin premisas.

Por ejemplo, el conjunto de los números pares está definido inductivamente por el siguiente predicado.

inductive *par* :: *nat* ⇒ *bool* **where**
par1: *par 0*
| *par2*: *par n* ⟹ *par* (*Suc* (*Suc n*))

Al igual que las funciones definidas por recursión, cualquier predicado definido inductivamente tiene su propia *regla de inducción* que permite demostrar propiedades acerca del predicado. Por ejemplo, la regla de inducción para el predicado *par* es la siguiente.

$$\frac{par\ x \qquad P\ 0 \qquad \bigwedge n.\ \dfrac{par\ n \qquad P\ n}{P\ (Suc\ (Suc\ n))}}{P\ x}\,(par.induct)$$

El siguiente lema se demuestra automáticamente usando la anterior regla de inducción.

lemma *par n* $\Longrightarrow \neg(\exists k.\ n = 2{*}k{+}1)$

iniciamos la prueba por la regla de inducción:

apply(*induct rule*: *par.induct*)

Así, hay que resolver los dos subobjetivos:

1. $\nexists k.\ 0 = 2 * k + 1$
2. $\bigwedge n.\ [\![par\ n; \nexists k.\ n = 2 * k + 1]\!] \Longrightarrow \nexists k.\ Suc\ (Suc\ n) = 2 * k + 1$

los cuales se prueban automáticamente:

apply *auto*
apply *arith*
done

El segundo subobjetivo se probó usando el método *arith* que simplifica expresiones *aritméticas lineales*.

2.1.6 Pruebas en Isabelle

El entorno básico de razonamiento en Isabelle para la demostración de teoremas es *deducción natural* [64] junto con una metalógica de orden superior. La metalógica de Isabelle tiene tres conectivas: implicación ($\Longrightarrow$), cuantificación universal ($\bigwedge$) e igualdad ($\equiv$). Los teoremas son escritos en la forma de reglas de inferencia (con n premisas y una conclusión). La regla de inferencia

$$\frac{\phi_1\ \phi_2\ \ldots \phi_n}{\psi}(R)$$

se escribe,

$$[\![\phi_1; \phi_2; \ldots; \phi_n]\!] \Longrightarrow \psi$$

y representa formalmente en Isabelle la sucesión de implicaciones

$$\phi_1 \Longrightarrow (\ldots \phi_n \Longrightarrow \psi).$$

Operacionalmente, esta expresión también puede ser vista como el estado de una prueba con subobjetivos $\phi_1, \phi_2, \ldots, \phi_n$ y objetivo principal ψ.

Por ejemplo, a las reglas de inferencia

$$\frac{P \wedge Q}{P} \qquad \frac{P \wedge Q}{Q}$$

les corresponde en Isabelle la regla

$$\dfrac{P \wedge Q \qquad \dfrac{P \qquad Q}{R}}{R} \qquad [\![P \wedge Q; [\![P; Q]\!] \Longrightarrow R]\!] \Longrightarrow R \quad (conjE)$$

Para la demostración de teoremas, Isabelle soporta *razonamiento progresivo* (de las premisas a la conclusión) y *razonamiento regresivo* (de la conclusión a las premisas). Una de las características importantes de Isar es que facilita la combinación de estas dos formas de razonar para la demostración de teoremas.

Para demostrar en Isabelle, por razonamiento regresivo, un objetivo usando una regla de inferencia existen varios métodos: *rule, erule, drule y frule*. En este trabajo solo usaremos el método *rule*.

Para resolver el objetivo O usando la regla de inferencia R, definida anteriormente, el método *rule* R **unifica** ψ con O y reemplaza O por los n nuevos subobjetivos $\phi'_1, \ldots, \phi'_n$ correspondientes a las **instancias** de $\phi_1, \ldots, \phi_n$.

Algunas reglas de inferencia de deducción natural se escriben en Isabelle de las siguiente forma.

Notación usual	Notación en Isabelle		
$\dfrac{P \qquad Q}{P \wedge Q}$	$\dfrac{P \qquad Q}{P \wedge Q}$	$[\![P; Q]\!] \Longrightarrow P \wedge Q$	(*conjI*)
$\dfrac{P}{P \vee Q}$	$\dfrac{P}{P \vee Q}$	$P \Longrightarrow P \vee Q$	(*disjI1*)
$\dfrac{Q}{P \vee Q}$	$\dfrac{Q}{P \vee Q}$	$Q \Longrightarrow P \vee Q$	(*disjI2*)
$\dfrac{P \vee Q \quad \begin{array}{c}[P]\\ \vdots\\ R\end{array} \quad \begin{array}{c}[Q]\\ \vdots\\ R\end{array}}{R}$	$\dfrac{P \vee Q \qquad \dfrac{P}{R} \qquad \dfrac{Q}{R}}{R}$	$[\![P \vee Q; P \Longrightarrow R; Q \Longrightarrow R]\!] \Longrightarrow R$	(*disjE*)
$\dfrac{\begin{array}{c}[P]\\ \vdots\\ Q\end{array}}{P \longrightarrow Q}$	$\dfrac{\dfrac{P}{Q}}{P \longrightarrow Q}$	$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$	(*impI*)
$\dfrac{P \longrightarrow Q \quad P}{Q}$	$\dfrac{P \longrightarrow Q \qquad P \qquad \dfrac{Q}{R}}{R}$	$[\![P \longrightarrow Q; P; Q \Longrightarrow R]\!] \Longrightarrow R$	(*impE*)

$$\frac{\begin{array}{c}[x]\\ \vdots\\ P(x)\end{array}}{\forall x.P(x)} \qquad \frac{\bigwedge x.\ P\ x}{\forall\, x.\ P\ x} \qquad (\bigwedge x.\ P\ x) \Longrightarrow \forall\, x.\ P\ x \qquad (allI)$$

$$\frac{\forall x.P(x)}{P(t)} \qquad \frac{\forall\, x.\ P\ x \qquad \dfrac{P\ x}{R}}{R} \qquad [\![\forall\, x.\ P\ x; P\ x \Longrightarrow R]\!] \Longrightarrow R \qquad (allE)$$

$$\frac{P(t)}{\exists x.P(x)} \qquad \frac{P\ x}{\exists\, x.\ P\ x} \qquad P\ x \Longrightarrow \exists\, x.\ P\ x \qquad (exI)$$

$$\frac{\exists x.P(x) \qquad \begin{array}{c}[x, P(x)]\\ \vdots\\ Q\end{array}}{Q} \qquad \frac{\exists\, x.\ P\ x \qquad \bigwedge x.\ \dfrac{P\ x}{Q}}{Q} \qquad [\![\exists\, x.\ P\ x; \bigwedge x.\ P\ x \Longrightarrow Q]\!] \Longrightarrow Q \qquad (exE)$$

Nótese que no se acostumbra a usar variables esquemáticas en la descripción de una regla (teorema), sin embargo Isabelle reescribe las reglas (teoremas) de tal forma que las variables libres se convierten en esquemáticas para poder realizar la unificación correspondiente.

2.2 Pruebas estructuradas en Isar

En esta sección introducimos las características del lenguaje Isar por medio de ejemplos. Esta parte está basada en [39]; más ejemplos se encuentran en [56]. Un estudio formal de la sintaxis de Isar aparece en [55]. El núcleo de una prueba en Isar corresponde a la siguiente gramática:

demostración	::=	**proof** [método] declaración* **qed**
	|	**by** método
declaración	::=	**fix** variable+
	|	**assume** proposición
	|	(**from** hecho+)? **have** proposición demostración
	|	(**from** hecho+)? **show** proposición+ demostración
proposición	::=	(etiqueta**:**)? "fórmula"
hecho	::=	etiqueta
método	::=	-
	|	**this**
	|	**rule** hecho
	|	**simp**
	|	**blast**
	|	**auto**
	|	**induct** variable
	|	...

De esta forma, una demostración en Isar se compone de un solo método precedido de la palabra **by** o de un bloque *proof-qed* de cero o más declaraciones. Un bloque puede opcionalmente comenzar con la declaración de un método con el fin de indicar como iniciar la prueba, por ejemplo, *(induct n)*.

Una declaración consta de una de dos clases de proposiciones, una hipótesis o una sentencia junto con su demostración.

La palabra opcional **from** indica cuáles hechos son usados en la demostración. Las proposiciones intermedias comienzan con la palabra **have** y la proposición principal con **show**. Una declaración puede introducir nuevas variables locales con **fix**.

Desde el punto de vista de la metalógica de Isabelle, $\bigwedge$ introduce variables cuantificadas, **assume** introduce la hipótesis de una implicación ($\longrightarrow$) y **have/show** la conclusión.

Las proposiciones son fórmulas precedidas, opcionalmente, de un nombre (etiqueta) que permite posteriormente hacer referencia a la fórmula correspondiente en una afirmación antecedida de la palabra **from**.

Las reglas de inferencia de deducción natural, enunciadas anteriormente, se escriben en Isar de la siguiente manera.

conjI: **assumes** P **and** Q **shows** $P \wedge Q$

conjE: **assumes** $P \wedge Q$ **obtains** P **and** Q

disjI1: **assumes** P **shows** $P \vee Q$

disjI2: **assumes** Q **shows** $P \vee Q$

disjE: **assumes** $P \vee Q$ **obtains** P **or** Q

impI: **assumes** $P \Longrightarrow Q$ **shows** $P \longrightarrow Q$

impE: **assumes** $P \longrightarrow Q$ **and** P **obtains** Q

allI: **assumes** $\bigwedge x.P(x)$ **shows** $\forall x.P(x)$

allE: **assumes** $\forall x.P(x)$ **obtains** $P(t)$

exI: **assumes** $P(t)$ **shows** $\exists x.P(x)$

exE: **assumes** $\exists x.P(x)$ **obtains** x **where** $P(x)$

2.2.1 Esquemas de demostración

En lo que sigue mostramos algunos esquemas básicos de demostración en Isar [39]. En su mayoría, estos patrones de prueba están basados en las reglas de inferencia de deducción natural que, en particular, son aplicadas indirectamente al comando **proof** en el caso de que no se especifique explícitamente otro método de prueba al comienzo de una demostración.

Demostración por distinción de casos:

show "R"
proof *cases*
  assume "P"
  ⋮
  show "R" ...
next
  assume "$\neg P$"
  ⋮
  show "R"...
qed

have "$P \vee Q$" ...
then show "R"
proof
  assume "P"
  ⋮
  show "R"...
next
  assume "Q"
  ⋮
  show "R"...
qed

Demostración de equivalencias lógicas:

show "$P \longleftrightarrow Q$"
proof
  assume "P"
  ⋮
  show "Q" ...
next
  assume "Q"
  ⋮
  show "P"...
qed

Demostración por contradicción:

show "$\neg P$"
proof
  assume "P"
  ⋮
  show "*False*" ...
qed

show "P"
proof*(rule ccontr)*
  assume "$\neg P$"
  ⋮
  show "*False*" ...
qed

Demostración de inclusión e igualdad de conjuntos:

show "$A \subseteq B$"
proof
  fix "x"
  assume "$x \in A$"
  ⋮
  show "$x \in B$" ...
qed

show "$A = B$"
proof
  show "$A \subseteq B$" ...
next
  show "$B \subseteq A$" ...
qed

Demostración de fórmulas cuantificadas:

show "$\forall x.\, P(x)$"
proof
  fix "x"
  ⋮
  show "$P(x)$" ...
qed

show "$\exists x.\, P(x)$"
proof
  ⋮
  show "$P(t)$" ...
qed

En los dos esquemas anteriores, en la prueba de $\forall\, x.\ P(x)$ el comando **fix** x introduce una variable local. En la prueba de $\exists\, x.\ P(x)$, t es algún término arbitrario para el cual demostramos que se satisface P. Otra forma forma de demostrar $\exists\, x.\ P(x)$ por razonamiento progresivo es:

have "$\exists x.\ P(x)$" ...
then obtain x **where** p : "$P(x)$" **by** *blast*.

En el comando **obtain** x, x es una variable local fija y p es el nombre del hecho $P(x)$. Este mismo patrón sirve para más de una variable cuantificada existencialmente.

El método *blast* [46] es un procedimiento de prueba apropiado para pruebas automatizadas de propiedades de lógica, conjuntos y relaciones. En el siguiente ejemplo se demuestra por el método *blast* que si P y Q son dos relaciones binarias tal que P es total, Q antisimétrica y P es subconjunto de Q entonces Q es subconjunto de P.

lemma
 assumes $\forall\ x\ y.\ P\ x\ y\ \vee P\ y\ x$
 and $\forall\ x\ y.\ Q\ x\ y \wedge Q\ y\ x \longrightarrow x = y$
 and $\forall\ x\ y.\ P\ x\ y \longrightarrow Q\ x\ y$
 shows $\forall\ x\ y.\ Q\ x\ y \longrightarrow\ P\ x\ y$
using *assms*
by *blast*

La palabra *assms* hace referencia a las hipótesis y la palabra **using** al igual que **from** sirve para introducir los hechos (en este caso *assms*), que son usados en una demostración,

(**have**| **show**) *prop* **using** *hechos* = **from** *hechos* (**have** | **show**) *prop*

Por último, se tienen las siguientes abreviaciones en *Isar*.

then = **from** *this*

thus = **then show**

hence = **then** *have*

2.3 Ejemplo

El siguiente ejemplo, sobre la formalización del teorema de Cantor, ilustra los conceptos introducidos anteriormente y muestra la metodología que se seguirá en el desarrollo del trabajo con el fin de obtener la formalización de los teoremas estudiados, a partir de su pruebas naturales.

El teorema de Cantor afirma que la cardinalidad de cualquier conjunto es menor que la cardinalidad de su conjunto potencia; es decir, que no existe una función sobreyectiva de un conjunto en su conjunto potencia.

Una formulación usual del teorema de Cantor es la siguiente:

Teorema 2.3.1 (Teorema de Cantor) *Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existe una biyección entre X y $\wp(X)$. Más aún, la cardinalidad de $\wp(X)$ es estrictamente mayor que la cardinalidad de X; es decir, $|X| < |\wp(X)|$.*

Demostración: La función $F{:}n\ X \to \wp(X)$, definida por $F(x)=\{x\}$ para cada $x \in X$, es una función inyectiva de X a $\wp(X)$. Esto prueba que $|X| \leq |\wp(X)|$. Falta demostrar que X no es equivalente a $\wp(X)$.

Supongamos que existe una función biyectiva f: $X \to \wp(X)$; nuestro objetivo es mostrar que esta hipótesis conduce a una contradicción.

Consideremos el conjunto $A \subseteq X$ definido de la siguiente forma:

$$A = \{x \in X \mid x \notin f(x)\}$$

Puesto que $A \in \wp(X)$ y f es sobreyectiva, existe un $a \in X$ tal que $f(a) = A$. El elemento a pertenece o no pertenece al conjunto A.

Si $a \in A$, entonces, por la definición de A, debemos tener que $a \notin f(a)$, y puesto que $f(a) = A$ esto es imposible.

Si $a \notin A$, entonces, de nuevo por la definición de A, debemos tener que $a \in f(a)$, y esto también es imposible.

De esta forma, hemos llegado a una contradicción; por lo tanto, $|X| < |\wp(X)|$.

□

Sea *'α set* la colección de los conjuntos de tipo *'α*. La formulación del teorema de Cantor en lógicas de orden superior afirma que para cada función f: *'α* $\to$ *'α set*, existe un conjunto en *'α* que no pertenece al rango de f. Formalmente,

theorem *Cantor*:
fixes *f* :: *'α* ⇒ *'α set*
shows ∃ *S*. *S* ∉ *range f*

La primera demostración que presentamos es una demostración estructurada que se aproxima a la demostración usual y muestra respectivamente los resultados de la teoría de conjuntos utilizados.

theorem *Cantor-p1*:
fixes *f* :: *'α* ⇒ *'α set*

```
 shows ∃ A. A ∉ range f
proof (rule exI)
 let ?A = {x. x ∉ f x}
 show ?A ∉ range f
 proof (rule notI)
  assume  ?A ∈ range f
  hence ?A ∈ {u. ∃ x. u = f x} by (simp only: full-SetCompr-eq)
  hence ∃ x. ?A = f x by (simp only: mem-Collect-eq)
  then obtain a where ?A = f a by (rule exE)
  show False
  proof cases
   assume hip1: a ∈ ?A
   hence a ∉ f a by (simp add: mem-Collect-eq)
   hence a ∉ ?A using ‹?A = f a› by simp
   thus False using hip1 by contradiction
  next
   assume hip2: a ∉ ?A
   hence a ∈ f a by (simp add: mem-Collect-eq)
   hence a ∈ ?A using ‹?A = f a› by simp
   thus False using hip2 by contradiction
  qed
 qed
qed
```

En la demostración anterior se tiene lo siguiente:

Se utilizó el método de prueba *rule* con las siguientes reglas de inferencia como argumento

- $$\frac{P\ x}{\exists\, x.\ P\ x}$$ (*exI*)
- $$\frac{\dfrac{P}{False}}{\neg\ P}$$ (*notI*)

También se utilizaron los métodos de prueba

- $$\frac{\dfrac{P}{Q} \qquad \dfrac{\neg\ P}{Q}}{Q}$$ (*cases*)
- $$\frac{\neg\ P \qquad P}{R}$$ (*contradiction*)

Los lemas de la teoría de conjuntos que hemos usado son

- $\{u \mid \exists x.\ u = f\,x\} = \mathit{range}\,f$ (*full-SetCompr-eq*)
- $(a \in \{x \mid P\,x\}) = P\,a$ (*mem-Collect-eq*)

Además, hemos usado la regla de eliminación del cuantificador existencial

- $$\frac{\exists x.\ P\,x \qquad \bigwedge x.\ \dfrac{P\,x}{Q}}{Q}$$ (*exE*)

El uso de las reglas de inferencia y lemas no es necesario explicitarlo, también se pueden eliminar los detalles de la obtención del elemento del rango y además las demostraciones de las contradicciones en cada uno de los casos se pueden hacer en forma automática usando el método *blast*:

```
theorem Cantor-p2:
 fixes f :: 'α ⇒ 'α set
 shows ∃ A. A ∉ range f
proof
 let ?A = {x. x ∉ f x}
 show ?A ∉ range f
 proof
  assume ?A ∈ range f
  then obtain a where ?A = f a by auto
  show False
  proof cases
    assume a ∈ ?A
    show False using ⟨?A = f a⟩ by blast
   next
    assume a ∉ ?A
    show False using ⟨?A = f a⟩ by blast
  qed
 qed
qed
```

Capítulo 3

Completitud de un sistema axiomático de la lógica proposicional

En este capítulo se presenta una formalización de un sistema axiomático para la lógica proposicional y la demostración de la completitud de dicho sistema usando el método de Kalmar de eliminación de variables. La formalización se basa en la presentada en el libro de Mendelson [36]. El objetivo es explorar la posibilidad de formalizar razonamiento complejo de forma legible por los humanos y procesable por las máquinas.

3.1 Formalización del sistema axiomático

En esta sección se formaliza en Isabelle la sintaxis de la lógica proposicional y su cálculo axiomático presentado en el libro de Mendelson [36]. También se formalizan las demostraciones de algunos resultados metalógicos como la regla de debilitamiento, el teorema de deducción y la regla de corte.

Definición 3.1.1 *El **alfabeto** de la lógica proposicional se compone de los siguientes símbolos:*

1. *Los **símbolos proposicionales**: $P_0, P_1, \ldots$*
2. *Las **conectivas**: $\neg$ y $\rightarrow$*
3. *Los **símbolos auxiliares**: "(" y ")"*

Definición 3.1.2 *Las **fórmulas** de la lógica proposicional se definen recursivamente con las siguientes reglas:*

1. *Los símbolos proposicionales son fórmulas.*

2. *Si F es una fórmula, entonces ¬F también lo es.*

3. *Si F y G son fórmulas, entonces $(F \to G)$ también lo es.*

Su formalización es:

datatype *'b formula* =
  Atom 'b
| *Neg 'b formula* (¬.(-) [*110*] *110*)
| *Implica 'b formula 'b formula* (**infixl** →. *101*)

Nota 3.1.3 *En la representación de las fórmulas se observa que:*

1. *El tipo de dato genérico 'b representa los símbolos proposicionales del lenguaje.*

2. *para distinguir el lenguaje objeto del metalenguaje, se ha puesto un punto al final de las conectivas del lenguaje objeto.*

3. *La negación tiene precedencia sobre el condicional.*

4. *El condicional asocia por la izquierda.*

Con estos convenios, se tiene por ejemplo, el siguiente lema.

lemma (¬.¬. *Atom P* →. *Atom Q* →. *Atom R*) =
    (((¬. (¬. *Atom P*)) →. *Atom Q*) →. *Atom R*)
by *simp*

La axiomática de la lógica proposicional usada en el libro de Mendelson es la siguiente.

Definición 3.1.4 ([36] p. 35) *Los **axiomas** de la lógica proposicional son las fórmulas que tienen las siguientes formas.*

- $F_1 \to (F_2 \to F_1)$ *(A1)*
- $(F_1 \to (F_2 \to F_3)) \to ((F_1 \to F_2) \to (F_1 \to F_3))$ *(A2)*
- $(\neg F_1 \to \neg F_2) \to ((\neg F_1 \to F_2) \to F_1)$ *(A3)*

Su formalización es:

definition
A1 :: *'b formula* ⇒ *'b formula* ⇒*'b formula* **where**

```
A1 F1 F2 ≡ (F1 →. (F2 →. F1))
definition
A2 :: 'b formula ⇒ 'b formula ⇒'b formula ⇒'b formula where
A2 F1 F2 F3 ≡ ((F1 →. (F2 →. F3)) →. ((F1 →. F2) →. (F1 →. F3)))
definition
A3 :: 'b formula ⇒ 'b formula ⇒ 'b formula where
A3 F1 F2 ≡ ((¬. F1 →. ¬. F2) →. ((¬. F1 →. F2) →. F1))
definition
Axiomas :: 'b formula ⇒ 'b formula ⇒ 'b formula ⇒ 'b formula ⇒
        'b formula ⇒ 'b formula ⇒ 'b formula ⇒ 'b formula set where
Axiomas F1 F2 F3 F4 F5 F6 F7 ≡ {(A1 F1 F2), (A2 F3 F4 F5), (A3 F6 F7)}
```

Las siguientes declaraciones permiten usar las anteriores definiciones como reglas de simplificación.

```
declare
A1-def [simp]
A2-def [simp]
A3-def [simp]
Axiomas-def [simp]
```

Ahora, estamos en condiciones de introducir el concepto de deducibilidad.

Definición 3.1.5 ([36] p. 34-35) *Sea S un conjunto de fórmulas. Una fórmula F es **deducible** a partir de S (y se representa por $S \vdash F$) si se obtiene aplicando las siguientes reglas:*

- *(Hipótesis) Si $F \in S$, entonces $S \vdash F$.*
- *(Axiomas) Los axiomas son deducibles a partir de S.*
- *(Modus ponens) Si $F \to G$ y F son deducibles a partir de S, entonces G también lo es.*

Su formalización usando conjuntos inductivos es:

```
inductive
esDeducible :: 'b formula set ⇒ 'b formula ⇒ bool (- ⊢ - [50,50] 50)
for S :: 'b formula set
where
 hip: ⟦ F ∈ S ⟧ ⟹ S ⊢ F
| ax: (F ∈ Axiomas F1 F2 F3 F4 F5 F6 F7) ⟹ S ⊢ F
| mp: ⟦ S ⊢ (F →. G); S ⊢ F ⟧ ⟹ S ⊢ G
```

Utilizaremos la notación $\emptyset \vdash F$, cuando la fórmula F sea deducible a partir del conjunto vacío de fórmulas.

3.1.1 La regla de debilitamiento

En esta subsección comenzamos la formalización de demostraciones de metateoremas. El primero es la regla de debilitamiento.

Lema 3.1.6 (Debilitamiento) *Si $S \vdash F$ y $S \subseteq T$, entonces $T \vdash F$.*

Demostración: Por inducción sobre la deducibilidad de F. Se consideran tres casos:

Caso 1: Supongamos que $F \in S$. Entonces se prueba
$$\forall\, T.\ S \subseteq T \longrightarrow T \vdash F$$
usando la regla de las hipótesis.

Caso 2: Supongamos que F es un axioma. Entonces se prueba
$$\forall\, T.\ S \subseteq T \longrightarrow T \vdash F$$
usando la regla de de los axiomas.

Caso 3: Supongamos que F se obtiene por modus ponens a partir de $G \to F$ y G. Por hipótesis de inducción se tiene que
$$\forall\, T.\ S \subseteq T \longrightarrow T \vdash G \to F \tag{1}$$
$$\forall\, T.\ S \subseteq T \longrightarrow T \vdash G \tag{2}$$
y hay que demostrar que
$$\forall\, T.\ S \subseteq T \longrightarrow T \vdash F$$

Sea T tal que $S \subseteq T$. Por (1) y (2), se tiene
$$T \vdash G \to F \tag{3}$$
$$T \vdash G \tag{4}$$
Aplicando modus ponens a (3) y (4) se tiene
$$T \vdash F.$$

□

Su formalización es:

```
lemma debilitamiento [rule-format]:
 fixes S:: 'b formula set
 shows  S ⊢ F ⟹ (∀ T. S ⊆ T ⟶ T ⊢ F)
proof (induct rule: esDeducible.induct)
 — Caso 1:
 {fix F
  assume F ∈ S
  thus ∀ T. S ⊆ T ⟶ T ⊢ F using hip by auto }
  — Caso 2:
  next
 {fix F  F1 F2 F3 F4 F5 F6 F7
  assume (F::'b formula) ∈ (Axiomas F1 F2 F3 F4 F5 F6 F7)
```

```
  thus ∀ T. S ⊆ T ⟶ T ⊢ F using ax by blast }
  — Caso 3:
  next
 {fix G F
  assume
    S ⊢ G →. F and
  1: ∀T. S ⊆ T ⟶ T ⊢ G →. F and
    S ⊢ G and
  2: ∀T. S ⊆ T ⟶ T ⊢ G
  show ∀T. S ⊆ T ⟶ T ⊢ F
  proof
   fix T
   show S ⊆ T ⟶ T ⊢ F
   proof
    assume S ⊆ T
    hence T ⊢ G →. F and T ⊢ G using 1 2 by auto
    thus T ⊢ F by (rule mp)
   qed
  qed }
qed
```

Como consecuencia inmediata de la regla de debilitamiento se tiene el siguiente corolario.

Corolario 3.1.7 (Segunda regla de debilitamiento) *Si $S \vdash F$, entonces $S \cup \{G\} \vdash F$.*

Demostración: Basta observar que $S \subseteq S \cup \{G\}$ y aplicar la regla de debilitamiento. □

Su formalización es:

```
corollary debilitamiento2:
 assumes S ⊢ F
 shows S ∪ {G} ⊢ F
proof −
 have S ⊆ S ∪ {G} by auto
 with assms show S ∪ {G} ⊢ F by (rule debilitamiento)
qed
```

3.1.2 Teoremas elementales

En esta sección presentamos la formalización de lemas necesarios para la prueba del teorema de deducción que veremos en la siguiente sección. Los primeros son las generalizaciones de los axiomas.

Lema 3.1.8 (Axioma 1) $S \vdash F \to (G \to F)$.

Demostración: Por el axioma 1 y la regla de los axiomas.

□

Su formalización es:

lemma *Axioma1*: $S \vdash F \to. (G \to. F)$
proof −
have $F \to. (G \to. F) \in$ *Axiomas F G F3 F4 F5 F6 F7* **by** *simp*
thus *?thesis* **by** (*rule ax*)
qed

Lema 3.1.9 (Axioma 2) $S \vdash (F \to (G \to H)) \to ((F \to G) \to (F \to H))$.

Demostración: Por el axioma 2 y la regla de los axiomas.

□

Su formalización es:

lemma *Axioma2*: $S \vdash (F \to. (G \to. H)) \to. ((F \to. G) \to. (F \to. H))$
proof −
have $(F \to. (G \to. H)) \to. ((F \to. G) \to. (F \to. H)) \in$ *Axiomas F1 F2 F G H F6 F7* **by** *simp*
thus *?thesis* **by** (*rule ax*)
qed

Lema 3.1.10 (Axioma 3) $S \vdash (\neg G \to \neg F) \to ((\neg G \to F) \to G)$.

Demostración: Por el axioma 3 y la regla de los axiomas.

□

Su formalización es:

lemma *Axioma3*: $S \vdash (\neg. G \to. \neg. F) \to. ((\neg. G \to. F) \to. G)$
proof −
have $(\neg. G \to. \neg. F) \to. ((\neg. G \to. F) \to. G) \in$ *Axiomas F1 F2 F3 F4 F5 G F*
by *simp*
thus *?thesis* **by** (*rule ax*)
qed

Finalmente, formalizamos 3 lemas que usaremos en el teorema de deducción. Sus demostraciones son análogas a las anteriores.

Lema 3.1.11 *Si $F \in S$, entonces $S \vdash G \to F$.*

Su formalización es:

lemma *por-hip*:
assumes $F \in S$
shows $S \vdash G \to. F$

Lema 3.1.12 *Si $F \to G \in S$ y $F \in S$, entonces $S \vdash G$.*

Su formalización es:

lemma *mp-con-hip*:
assumes $F \to. G \in S$ **and** $F \in S$
shows $S \vdash G$

Lema 3.1.13 (Identidad ([36] p. 36)) $S \vdash F \to F$.

Demostración:

1.	$S \vdash (F \to ((F \to F) \to F)) \to ((F \to (F \to F)) \to (F \to F))$	[Axioma 2]
2.	$S \vdash F \to ((F \to F) \to F)$	[Axioma 1]
3.	$S \vdash (F \to (F \to F)) \to (F \to F)$	[Modus ponens, 1 y 2]
4.	$S \vdash F \to (F \to F)$	[Axioma 1]
5.	$S \vdash F \to F$	[Modus ponens, 3 y 4]

□

Su formalización es:

lemma *identidad*: $S \vdash F \to. F$
proof −
have *1*: $S \vdash (F \to. ((F \to. F) \to. F)) \to. ((F \to. (F \to. F)) \to. (F \to. F))$
by (*rule Axioma2*)
have *2*: $S \vdash F \to. ((F \to. F) \to. F)$ **by** (*rule Axioma1*)
have *3*: $S \vdash (F \to. (F \to. F)) \to. (F \to. F)$ **using** *1 2* **by** (*rule mp*)
have *4*: $S \vdash F \to. (F \to. F)$ **by** (*rule Axioma1*)
show *5*: $S \vdash F \to. F$ **using** *3 4* **by** (*rule mp*)
qed

Nota 3.1.14 *Nótese que la formalización de las demostraciones de los lemas anteriores se corresponden exactamente con las naturales.*

3.1.3 Teorema de la deducción

En esta sección presentamos la formalización del segundo metateorema: el teorema de deducción y, como consecuencias, las reglas de encadenamiento y de corte.

Lema 3.1.15 (Lema de deducción ([36] p. 37)) *Si $S \vdash F$, entonces para toda G se tiene que si $S = \{G\} \cup S'$, entonces $S' \vdash G \to F$.*

Demostración: Por inducción sobre la deducibilidad de F. Se consideran tres casos:

Caso 1: Supongamos que $F \in S$. Hay que probar que
$$\forall\, G\, S'.\, S = \{G\} \cup S' \longrightarrow S' \vdash G \to F$$

Sean G y S' tales que $S = \{G\} \cup S'$. Entonces $F \in \{G\} \cup S'$ y pueden darse dos casos:

Caso 1a: Supongamos que $F \in \{G\}$. Entonces, $F = G$ y se tiene que $S' \vdash G \to F$ por el lema 3.1.13.

Caso 1b: Supongamos que $F \in S'$. Entonces se tiene que $S' \vdash G \to F$ por el lema 3.1.11.

Caso 2: Supongamos que F es un axioma. Hay que probar que
$$\forall\, G\, S'.\, S = \{G\} \cup S' \longrightarrow S' \vdash G \to F$$

Sean G y S' tales que $S = \{G\} \cup S'$. Entonces

(1)	$S' \vdash F \to (G \to F)$	[Axioma 1]
(2)	$S' \vdash F$	[Regla de los axiomas, y F es axioma]
(3)	$S' \vdash G \to F$	[Modus ponens, 1 y 2]

Caso 3: Supongamos que F se obtiene por modus ponens a partir de $G \to F$ y G. Por hipótesis de inducción se tiene que
$$\forall\, H\, S'.\, S = \{H\} \cup S' \longrightarrow S' \vdash H \to (G \to F) \tag{5}$$
$$\forall\, H\, S'.\, S = \{H\} \cup S' \longrightarrow S' \vdash H \to G \tag{6}$$
y hay que demostrar que
$$\forall\, H\, S'.\, S = \{H\} \cup S' \longrightarrow S' \vdash H \to F$$

Sean H y S' tales que
$$S = \{H\} \cup S' \tag{7}$$

Entonces,

(8)	$S' \vdash H \to (G \to F)$	[5 y 7]
(9)	$S' \vdash H \to G$	[6 y 7]
(10)	$S' \vdash (H \to (G \to F)) \to ((H \to G) \to (H \to F))$	[Axioma 2]
(11)	$S' \vdash (H \to G) \to (H \to F)$	[Modus p., 10 y 8]
(12)	$S' \vdash H \to F$	[Modus p., 11 y 9]

□

Su formalización es:

lemma *lema-deduccion*:
fixes S::*'b formula set*
shows $S \vdash F \Longrightarrow (\forall G\ S'.\ S = \{G\} \cup S' \longrightarrow S' \vdash (G \rightarrow .\ F))$
proof(*induct rule*: *esDeducible.induct*)
— Caso 1:
{ **fix** F
assume $F \in S$
show $\forall G\ S'.\ S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$
proof
fix G
show $\forall S'.\ S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$
proof
fix S'
show $S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$
proof
assume $S = \{G\} \cup S'$
show $S' \vdash G \rightarrow .\ F$
proof (*rule UnE*)
show $F \in \{G\} \cup S'$ **using** ‹$F \in S$› **and** ‹$S = \{G\} \cup S'$› **by** *simp*
next
— Caso 1a:
{ **assume** $F \in \{G\}$
thus $S' \vdash G \rightarrow .\ F$ **by** (*simp add*:*identidad*) }
next
— Caso 1b:
{ **assume** $F \in S'$
thus $S' \vdash G \rightarrow .\ F$ **by** (*simp add*:*por-hip*) }
qed
qed
qed
qed }
next
— Caso 2:
{ **fix** *F F1 F2 F3 F4 F5 F6 F7*
assume (*F*:: *'b formula*) ∈ *Axiomas F1 F2 F3 F4 F5 F6 F7*
show $\forall G\ S'.\ S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$
proof
fix G
show $\forall S'.\ S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$
proof
fix S'
show $S = \{G\} \cup S' \longrightarrow S' \vdash G \rightarrow .\ F$

```
    proof
     assume S = {G} ∪ S'
     show S' ⊢ G →. F
     proof −
      have 1: S' ⊢ F →. (G →. F) by (rule Axioma1)
      have 2: S' ⊢ F using ⟨F ∈ Axiomas F1 F2 F3 F4 F5 F6 F7⟩
       by (rule ax)
      show S' ⊢ G →. F using 1 2 by (rule mp)
     qed
    qed
   qed
  qed }
  next
  — Caso 3:
{ fix G F
  assume
    S ⊢ G →. F and
  5: ∀ H S'. S = {H} ∪ S' ⟶ S' ⊢ H →. (G →. F) and
    S ⊢ G and
  6: ∀ H S'. S = {H} ∪ S' ⟶ S' ⊢ H →. G
  show ∀ H S'. S = {H} ∪ S' ⟶ S' ⊢ H →. F
  proof
   fix H
   show ∀ S'. S = {H} ∪ S' ⟶ S' ⊢ H →. F
   proof
    fix S'
    show S = {H} ∪ S' ⟶ S' ⊢ H →. F
    proof
     assume 7: S = {H} ∪ S'
     show S' ⊢ H →. F
     proof −
      have 8: S' ⊢ H →. (G →. F) using 5 7 by simp
      have 9: S' ⊢ H →. G using 6 7 by simp
      have S' ⊢ (H →. (G →. F)) →. ((H →. G) →. (H →. F))
       by (rule Axioma2)
      hence S' ⊢ (H →. G) →. (H →. F) using 8 by (rule mp)
      thus S' ⊢ H →. F using 9 by (rule mp)
     qed
    qed
   qed
  qed }
qed
```

Como consecuencia inmediata del lema anterior se obtiene el teorema de deducción.

Teorema 3.1.16 (Teorema de deducción ([36] p. 37))
Si $\{F\} \cup S \vdash G$, *entonces* $S \vdash (F \to G)$.

Su formalización es:

theorem *deduccion*:
assumes $(\{F\} \cup S) \vdash G$
shows $S \vdash (F \to . G)$
using *assms* **by** (*simp add*:*lema-deduccion*)

A partir del teorema de deducción se obtiene la regla de corte.

Lema 3.1.17 (Regla de corte) *Si* $S \vdash F$ *y* $(\{F\} \cup S) \vdash G$, *entonces* $S \vdash G$.

Demostración: De la segunda hipótesis y el teorema de la deducción, tenemos que $S \vdash (F \to G)$. De esto último y la primera hipótesis, usando Modus ponens, tenemos que $S \vdash G$.

□

Su formalización es:

lemma *regla-de-corte*:
assumes $S \vdash F$ **and** $(\{F\} \cup S) \vdash G$
shows $S \vdash G$
proof −
have $S \vdash (F \to . G)$ **using** *assms*(2) **by** (*rule deduccion*)
thus $S \vdash G$ **using** *assms*(1) **by** (*rule mp*)
qed

Otra consecuencia del teorema de deducción es la regla de encadenamiento.

Lema 3.1.18 (Encadenamiento ([36] p. 38)) $\{F \to G, G \to H\} \vdash F \to H$.

Demostración:

1.	$\{F \to G, G \to H, F\} \vdash F \to G$	[Regla de las hipótesis]
2.	$\{F \to G, G \to H, F\} \vdash G \to H$	[Regla de las hipótesis]
3.	$\{F \to G, G \to H, F\} \vdash F$	[Regla de las hipótesis]
4.	$\{F \to G, G \to H, F\} \vdash G$	[Modus ponens, 1 y 3]
5.	$\{F \to G, G \to H, F\} \vdash H$	[Modus ponens, 2 y 4]
6.	$\{F \to G, G \to H\} \vdash (F \to H)$	[Teorema deducción, 5]

□

Su formalización es:

lemma *encadenamiento*: $\{F \rightarrow. G, G \rightarrow. H\} \vdash F \rightarrow. H$
proof $-$
have *1*: $\{F \rightarrow. G, G \rightarrow. H, F\} \vdash F \rightarrow. G$ **using** *hip*[*of* $F \rightarrow. G$] **by** *simp*
have *2*: $\{F \rightarrow. G, G \rightarrow. H, F\} \vdash G \rightarrow. H$ **using** *hip*[*of* $G \rightarrow. H$] **by** *simp*
have *3*: $\{F \rightarrow. G, G \rightarrow. H, F\} \vdash F$ **using** *hip*[*of* F] **by** *simp*
have *4*: $\{F \rightarrow. G, G \rightarrow. H, F\} \vdash G$ **using** *1 3* **by** (*rule mp*)
have *5*: $\{F \rightarrow. G, G \rightarrow. H, F\} \vdash H$ **using** *2 4* **by** (*rule mp*)
show *6*: $\{F \rightarrow. G, G \rightarrow. H\} \vdash (F \rightarrow. H)$ **using** *5* **by** (*simp add*:*deduccion*)
qed

Finalizamos esta subsección presentando algunos lemas necesarios en la prueba del teorema de completitud y la formalización de sus enunciados. Sus demostraciones son análogas a las anteriores.

Lema 3.1.19 (Segunda regla de encadenamiento)
Si $S \vdash F \rightarrow G$ *y* $S \vdash G \rightarrow H$, *entonces* $S \vdash F \rightarrow H$.

Su formalización es:

lemma *encadenamiento2*:
assumes $S \vdash F \rightarrow. G$ **and** $S \vdash G \rightarrow. H$
shows $S \vdash F \rightarrow. H$

Lema 3.1.20 (MP2 ([36] p. 38)) $\{F \rightarrow (G \rightarrow H), G\} \vdash (F \rightarrow H)$.

Su formalización es:

lemma *mp2*: $\{F \rightarrow. (G \rightarrow. H), G\} \vdash (F \rightarrow. H)$

Lema 3.1.21 (MP2a) *Si* $S \vdash F \rightarrow (G \rightarrow H)$ *y* $S \vdash G$, *entonces* $S \vdash F \rightarrow H$.

Su formalización es:

lemma *mp2a*:
assumes $S \vdash F \rightarrow. (G \rightarrow. H)$ **and** $S \vdash G$
shows $S \vdash F \rightarrow. H$

Lema 3.1.22 (Segunda regla de corte) *Si* $S \vdash F$, $S \vdash G$ *y* $\{F, G\} \vdash H$, *entonces* $S \vdash H$.

Su formalización es:

lemma *corte2*:
assumes $S \vdash F$ **and** $S \vdash G$ **and** $\{F, G\} \vdash H$
shows $S \vdash H$

Lema 3.1.23 (Eliminación de la doble negación ([36] p. 38)) $S \vdash \neg\neg F \to F$.

Su formalización es:

lemma *eliminaDN*: $S \vdash \neg.\ \neg.\ F \to.\ F$

Lema 3.1.24 (Introducción de la doble negación ([36] p. 36)) $S \vdash F \to \neg\neg F$.

Su formalización es:

lemma *introDN*: $S \vdash F \to.\ \neg.\ \neg.\ F$

Lema 3.1.25 (Elimación de la negación ([36] p. 36)) $S \vdash \neg F \to (F \to G)$.

Su formalización es:

lemma *elimNeg*: $S \vdash \neg.\ F \to.\ (F \to.\ G)$

Lema 3.1.26 (Contrarecíproco-1 ([36] p. 36)) $S \vdash (\neg G \to \neg F) \to (F \to G)$.

Su formalización es:

lemma *contrareciproco-1*: $S \vdash (\neg.\ G \to.\ \neg.\ F) \to.\ (F \to.\ G)$

Lema 3.1.27 (Contrarecíproco-2 ([36] p. 36)) $S \vdash (F \to G) \to (\neg G \to \neg F)$.

Su formalización es:

lemma *contrareciproco-2*: $S \vdash (F \to.\ G) \to.\ (\neg.\ G \to.\ \neg.\ F)$

Lema 3.1.28 (Negación del condicional ([36] p. 36)) $S \vdash F \to (\neg G \to \neg(F \to G))$.

Su formalización es:

lemma *negacion-del-condicional*: $S \vdash F \rightarrow. (\neg. G \rightarrow. \neg. (F \rightarrow. G))$

Lema 3.1.29 (Uso del tercio excluso ([36] p. 36)) *$S \vdash (F \rightarrow G) \rightarrow ((\neg F \rightarrow G) \rightarrow G)$.*

Su formalización es:

lemma *tercio-excluso-1*: $S \vdash (F \rightarrow. G) \rightarrow. ((\neg. F \rightarrow. G) \rightarrow. G)$

Lema 3.1.30 (Tercio excluso) *Si $(\{F\} \cup S) \vdash G$ y $(\{\neg F\} \cup S) \vdash G$, entonces $S \vdash G$.*

Su formalización es:

lemma *tercio-excluso*:
assumes $(\{F\} \cup S) \vdash G$ **and** $(\{\neg. F\} \cup S) \vdash G$
shows $S \vdash G$

Corolario 3.1.31 *Sea S un conjunto finito. Si $(\{F\} \cup S) \vdash G$ y $(\{\neg F\} \cup S) \vdash G$, entonces $S \vdash G$.*

Su formalización es:

corollary *tercio-excluso-2*:
assumes *insert* F $S \vdash G$ **and** *insert* $(\neg. F)$ $S \vdash G$
shows $S \vdash G$
using *tercio-excluso*[*of* F S G] *assms* **by** *simp*

3.2 Semántica de la lógica proposicional

En esta sección presentamos la formalización de los conceptos semánticos de la lógica proposicional. Comenzamos formalizando el concepto del valor de una fórmula en una interpretación (3.2.3), para lo que se necesitan los conceptos de valores de verdad (3.2.1) y de interpretaciones (3.2.2).

Definición 3.2.1 *Los **valores de verdad** de la lógica proposicional clásica son* V *(que se interpreta como **verdadero**) y* F *(que se interpreta como **falso**).*

En la formalización se identifica el conjunto de los valores de verdad con el tipo *v-verdad*, el valor V con la constante *Verdad* y el valor F con la constante *Falso* respectivamente.

datatype *v-verdad* = *Verdad* | *Falso*

Definición 3.2.2 *Una **interpretación** es una aplicación del conjunto de los símbolos proposicionales en el conjunto de los valores de verdad.*

La representación en Isabelle del concepto de interpretación es una función de la forma $I :: 'b \Rightarrow$ *v-verdad*.

Ahora se puede definir el valor de una fórmula respecto de una interpretación por recursión.

Definición 3.2.3 *El **valor** de una fórmula F en una interpretación I (representado por $I'(F)$) se define por recursión como sigue:*

- $I'(P) = I(P)$ *si P es un símbolo proposicional*
- $I'(\neg F) = \begin{cases} \mathsf{V}, & \text{si } I'(F) = \mathsf{F} \\ \mathsf{F}, & \text{si } I'(F) = \mathsf{V} \end{cases}$
- $I'(F \to G) = \begin{cases} \mathsf{F}, & \text{si } I'(F) = \mathsf{V} \text{ y } I'(G) = \mathsf{F} \\ \mathsf{V}, & \text{en caso contrario.} \end{cases}$

Para formalizarlo, definimos las funciones de verdad de las conectivas.

definition *v-negacion* :: *v-verdad* ⇒ *v-verdad* **where**
v-negacion x ≡ (*if x* = *Verdad then Falso else Verdad*)

definition *v-implicacion* :: *v-verdad* ⇒ *v-verdad* ⇒ *v-verdad* **where**
v-implicacion x y ≡ (*if x* = *Falso then Verdad else y*)

La formalización del valor de una fórmula es:

primrec *valor* :: ('*b* ⇒ *v-verdad*) ⇒ '*b formula* ⇒ *v-verdad* **where**
valor I (*Atom P*) = *I P*
| *valor I* (¬. *F*) = (*v-negacion* (*valor I F*))
| *valor I* (*F* →. *G*) = (*v-implicacion* (*valor I F*) (*valor I G*))

Desde el punto de vista semántico, la primera clasificación de las fórmulas las divide en tautologías y las que no lo son.

Definición 3.2.4 *Una fórmula F es una* ***tautología*** *si* $I'(F) = \mathsf{V}$ *para toda interpretación I.*

Su formalización es:

definition *tautologia* :: *'b formula* ⇒ *bool* **where**
tautologia F ≡ (∀ *I*. ((*valor I F*) = *Verdad*))

Para la demostración del teorema de corrección necesitamos probar que los axiomas son tautologías. Lo probaremos en los siguientes lemas.

Lema 3.2.5 (tautología-A1) *La fórmula a1:* $F \to (G \to F)$ *es una tautología.*

Demostración: La demostración se realiza fácilmente por casos en los posibles valores de la fórmula F en la interpretación I, utilizando la definición del valor de una implicación. Tal y como se muestra en la siguiente tabla:

	F	$G \to F$	$F \to (G \to F)$
caso 1	V	V	V
caso 2	F		V

□

Su formalización es:

lemma *tautologia-A1*: *tautologia* (*F* →. (*G* →. *F*))
proof −
have ∀ *I*. *valor I* (*F* →. (*G* →. *F*)) = *Verdad*
proof
fix *I*
show *valor I* (*F* →. (*G* →. *F*)) = *Verdad*
proof (*cases valor I F*)
— Caso 1:
{ **assume** *valor I F* = *Verdad*
thus *?thesis* **by** (*simp add*: *v-implicacion-def*) }
next
— Caso 2:
{ **assume** *valor I F* = *Falso*
thus *?thesis* **by**(*simp add*: *v-implicacion-def*) }
qed
qed
thus *?thesis* **by** (*simp add*: *tautologia-def*)
qed

Lema 3.2.6 (tautología-A2) *La fórmula a2:* $(F \to (G \to H)) \to ((F \to G) \to (F \to H))$ *es una tautología.*

Demostración: La demostración se realiza fácilmente por casos en los posibles valores de las fórmulas F, H y G en la interpretación I, utilizando la definición del valor de una implicación. Tal y como se muestra en la siguiente tabla:

	F	H	G	GH	$F \to (G \to H)$	$F \to G$	$F \to H$	$(F \to G) \to (F \to H)$	$a2$
caso $1a$	V	V					V	V	V
caso $1b.1$	V	F	V	F	F				V
caso $1b.2$	V	F	F			F		V	V
caso 2	F						V	V	V

□

Su formalización es:

```
lemma tautologia-A2:
 tautologia ((F →. (G →. H)) →. ((F →. G) →. (F →. H)))
proof −
 have ∀ I. valor I ((F →. (G →. H)) →. ((F →. G) →. (F →. H))) = Verdad
 proof
  fix I
  show valor I ((F →. (G →. H)) →. ((F →. G) →. (F →. H))) = Verdad
  proof (cases valor I F)
   — Caso 1:
  { assume valor I F = Verdad
   show ?thesis
   proof (cases valor I H)
    — Caso 1a:
   { assume valor I H = Verdad
    thus ?thesis by (simp add: v-implicacion-def) }
    next
    — Caso 1b
   { assume Vr: valor I H = Falso
    show ?thesis
    proof (cases valor I G)
     — Caso 1b.1:
    { assume valor I G = Verdad
     thus ?thesis using Vr by (simp add: v-implicacion-def) }
     next
     — Caso 1b.2:
    { assume valor I G = Falso
     thus ?thesis by (simp add: v-implicacion-def) }
    qed }
   qed  }
   next
```

```
    — Caso 2:
   { assume valor I F = Falso
     thus ?thesis by (simp add: v-implicacion-def) }
   qed
  qed
  thus ?thesis  by (simp add: tautologia-def)
qed
```

Lema 3.2.7 (tautología-A3) *La fórmula a3*: $(\neg\ G \to \neg\ F) \to ((\neg\ G \to F) \to G)$ *es una tautología.*

Demostración: La demostración se realiza por casos en los posibles valores de las fórmulas G y F en la interpretación I, utilizando las definiciones del valor de una implicación y de una negación. Tal y como se muestra en la siguiente tabla:

	G	F	$\neg G \to \neg F$	$\neg G \to F$	$(\neg G \to F) \to G$	$a3$
caso 1	V				V	V
caso 2*a*	F	V	F			V
caso 2*b*	F	F		F	V	V

□

Su formalización es:

```
lemma tautologia-A3: tautologia ((¬. G →. ¬. F) →. ((¬. G →. F) →. G))
proof −
 have ∀ I. valor I ((¬. G →. ¬. F) →. ((¬. G →. F) →. G)) = Verdad
 proof
  fix I
  show valor I ((¬. G →. ¬. F) →. ((¬. G →. F) →. G)) = Verdad
  proof (cases valor I G)
   — Caso 1:
  { assume valor I G = Verdad
   thus ?thesis by (simp add: v-implicacion-def) }
   next
   — Caso 2:
  { assume valor I G = Falso
   show ?thesis
   proof (cases valor I F)
    — Caso 2a:
   { assume valor I F = Verdad
    thus ?thesis by (simp add: v-negacion-def v-implicacion-def) }
    next
```

```
    — Caso 2b:
    { assume valor I F = Falso
    thus ?thesis by (simp add: v-negacion-def v-implicacion-def) }
   qed }
  qed
 qed
 thus ?thesis by (simp add: tautologia-def)
qed
```

Dado un conjunto de fórmulas S, una clasificación semántica de las interpretaciones las divide entre las que son modelo de S y las que no lo son.

Definición 3.2.8 *Una interpretación I es **modelo** de un conjunto de fórmulas S si, para toda fórmula F de S, $I'(F) = \mathsf{V}$.*

Su formalización es:

```
definition modelo :: ('b ⇒ v-verdad) ⇒ 'b formula set ⇒ bool
  (- mod - [80,80] 80) where
  I mod S ≡ (∀ F ∈ S. valor I F = Verdad)
```

Una clasificación semántica de los conjuntos de fórmulas los divide entre los que tienen modelos y los que no tienen.

Definición 3.2.9 *Un conjunto de fórmulas es **satisfacible** si tiene algún modelo. En caso contrario se dice que es **insatisfacible**.*

Su formalización es:

```
definition satisfacible :: 'b formula set ⇒ bool where
  satisfacible S ≡ (∃ v. v mod S)
```

El último concepto semántico que formalizamos es el de consecuencia lógica.

Definición 3.2.10 *Una fórmula F es una **consecuencia lógica** de un conjunto de fórmulas S si para todo modelo I de S se tiene que $I'(F) = \mathsf{V}$. Se representa por $S \models F$ y por $\models F$ en el caso particular en que el conjunto S sea vacío.*

Su formalización es:

```
definition consecuencia :: 'b formula set ⇒ 'b formula ⇒ bool
  (- ⊨ - [80,80] 80) where
  S ⊨ F ≡ (∀ I. I mod S ⟶ valor I F = Verdad)
```

Terminamos esta sección formalizando metateoremas que relacionan los conceptos anteriores. El primero establece que la fórmula F es una tautología si y solo si $\models F$. Para demostrarlo, probamos en primer lugar el siguiente lema.

Lema 3.2.11 *Todas las interpretaciones son modelo del conjunto vacío.*

Demostración: Por la definición de modelo de un conjunto.

□

Su formalización es:

```
lemma modelo-de-vacio:
 ∀ I. I mod {}
proof −
 have ∀ F ∈ {}. valor I F = Verdad by simp
 thus ∀ I. I mod {} by (simp add: modelo-def)
qed
```

Lema 3.2.12 *Una fórmula es tautología si y solo si es consecuencia del conjunto vacío.*

Demostración: Es consecuencia del lema anterior y de las definiciones de consecuencia lógica y de tautología.

□

Su formalización es:

```
lemma CNS-tautologia:
 tautologia F = ({} ⊨ F)
by (simp add: tautologia-def consecuencia-def modelo-de-vacio)
```

El segundo metateorema es el siguiente.

Lema 3.2.13 (CNS-tautología1) *Si una fórmula es tautología entonces es consecuencia lógica de cualquier conjunto.*

Demostración: Basta con aplicar las definiciones de tautología y consecuencia lógica.

□

Su formalización es:

```
lemma CNS-tautologia1:
 assumes tautologia F
 shows S ⊨ F
proof −
 show ?thesis using assms  by (simp add: tautologia-def consecuencia-def)
qed
```

3.3 Teorema de corrección del cálculo proposicional

Utilizando los resultados anteriores se puede demostrar el primer metateorema que relaciona los conceptos de deducibilidad y consecuencia: el teorema de corrección.

Teorema 3.3.1 (de corrección ([36] p. 40)) *Las fórmulas deducibles a partir de un conjunto de fórmulas S son consecuencias de dicho conjunto; es decir, si* $S \vdash F$*, entonces* $S \models F$*.*

Demostración: Por inducción sobre la deducibilidad de F. Se consideran tres casos:

Caso 1: Supongamos que $F \in S$. Sea I un modelo de S, entonces, por definición de modelo, $I'(G) = \mathsf{V}$ para toda fórmula $G \in S$. Luego, por hipótesis, $I'(F) = \mathsf{V}$. De esta forma, $S \models F$.

Caso 2: Supongamos que F es un axioma. Entonces F es de la forma $(A1\ F1\ F2)$, $(A2\ F3\ F4\ F5)$ o $(A3\ F6\ F7)$, y por los lemas 3.2.5, 3.2.6, 3.2.7 se tiene que F es una tautología. Luego, por 3.2.13, $S \models F$.

Caso 3: Supongamos que F se obtiene por modus ponens a partir de $G \to F$ y G. Por hipótesis de inducción se tiene que $S \models F \to G$ y $S \models F$. Luego, por definicion de consecuencia lógica, para todo modelo I de S se tiene que $I'(F \to G) = \mathsf{V}$ y $I'(F) = \mathsf{V}$. Por lo tanto, por la definición del valor de una implicación, necesariamente se tiene que $I'(G) = \mathsf{V}$. Así, queda demostrado que $S \models G$.

□

Su formalización es:

```
theorem validez:
 fixes S :: 'b formula set
 shows S ⊢ F  ⟹ S ⊨ F
proof (induct rule: esDeducible.induct)
 — Caso 1:
{fix F
 assume hip1: F ∈ S
 have ∀ I. I mod S ⟶ valor I F = Verdad
 proof
  fix I
  show I mod S ⟶ valor I F = Verdad
  proof
   assume I mod S
   thus valor I F = Verdad using hip1 by (simp add: modelo-def)
  qed
 qed
 thus S ⊨ F by (simp add: consecuencia-def)
 }
```

```
 next
  — Caso 2:
 {fix F
 fix F1 :: 'b formula
 fix F2 :: 'b formula
 fix F3 :: 'b formula
 fix F4 :: 'b formula
 fix F5 :: 'b formula
 fix F6 :: 'b formula
 fix F7 :: 'b formula
 assume hip2: F ∈ Axiomas F1 F2 F3 F4 F5 F6 F7
 have S ⊨ A1 F1 F2 using tautologia-A1 and CNS-tautologia1 by auto
 moreover
 have S ⊨ A2 F3 F4 F5
   using tautologia-A2 and CNS-tautologia1 by (unfold A2-def,blast)
 moreover
 have S ⊨ A3 F6 F7
   using tautologia-A3 and CNS-tautologia1 by (unfold A3-def,blast)
 ultimately
 show S ⊨ F using hip2 by auto
 }
 next
 — Caso 3:
{ fix F G
 assume S ⊢ F →. G and
 1: S ⊨ F →. G and
    S ⊢ F and
 2: S ⊨ F
 show S ⊨ G using 1 and 2
   by (simp add: consecuencia-def v-implicacion-def)
 }
qed
```

3.4 Teorema de completitud

En esta sección se formaliza la demostración del teorema de completitud, el cual afirma que en nuestro sistema deductivo se pueden deducir *todas* las tautologías. Un sistema con esta propiedad se dice que es *deductivamente completo*.

3.4.1 Prueba informal

En esta parte presentamos una prueba del teorema de completitud, similar a las que aparecen en los textos clásicos de lógica, por ejemplo en ([36] p. 42). Para ello, comenzamos introduciendo el concepto de relativización.

Definición 3.4.1 *Sea I una interpretación. La **relativización** de una fórmula F respecto de I se define por*

$$F^I = \begin{cases} F, & si\ I'(F) = \mathsf{V} \\ \neg F, & si\ I'(F) = \mathsf{F} \end{cases}$$

Esta definición nos permite enunciar el siguiente lema, a partir del cual se demuestra el teorema de completitud.

Lema 3.4.2 (Kalmar ([36] p. 41)) *Sean $p_1, p_2, \ldots, p_m$ los símbolos proposicionales de F y sea I una interpretación, entonces $\{p_1^I, \ldots, p_m^I\} \vdash F^I$.*

Demostración: La prueba es por inducción en el número n de ocurrencias de conectivas en F.

Caso base. Supongamos que $n = 0$. Entonces F es un símbolo proposicional p_1, luego para cada uno de los casos $I(p_1) = \mathsf{V}$ e $I(p_1) = \mathsf{F}$ el lema $\{p_1^I\} \vdash p_1^I$ se reduce respectivamente a las afirmaciones $\{p_1\} \vdash p_1$ y $\{\neg p_1\} \vdash \neg p_1$, que son verdaderas por la definición de deducibilidad (3.1.5).

Paso de inducción. Supongamos que el lema se tiene para $j < n$. Distinguimos los siguientes casos.

Caso 1. Supongamos que F es $\neg G$. Entonces G tiene menos de n ocurrencias de conectivas, y tiene los mismos símbolos proposicionales que F. Luego, por hipótesis de inducción, se tiene que

$$\{p_1^I, \ldots, p_m^I\} \vdash G^I. \tag{1}$$

Distinguimos los siguientes dos subcasos.

Caso 1a. Supongamos que $I'(G) = \mathsf{V}$, entonces $I'(F) = \mathsf{F}$. Por lo tanto, $G^I = G$ y $F^I = \neg F$. Por consiguiente,

1. $\{p_1^I, \ldots, p_m^I\} \vdash G$ [(1) y $G^I = G$]
2. $\{p_1^I, \ldots, p_m^I\} \vdash G \rightarrow \neg\neg G$ [lema 3.1.24]
3. $\{p_1^I, \ldots, p_m^I\} \vdash \neg\neg G$ [Modus ponens, 2 y 1]
4. $\{p_1^I, \ldots, p_m^I\} \vdash \neg F$ [F es $\neg G$]
5. $\{p_1^I, \ldots, p_m^I\} \vdash F^I$ [$F^I = \neg F$]

Caso 1b. Supongamos que $I'(G) = \mathsf{F}$, entonces $I'(F) = \mathsf{V}$. Por lo tanto, $G^I = \neg G$ y $F^I = F$. Por la hipótesis de inducción tenemos que, $\{p_1^I, \ldots, p_m^I\} \vdash \neg G$. De esto último, y teniendo en cuenta que $\neg G$ es F^I, se tiene el lema.

Caso 2. Supongamos que F es $G_1 \to G_2$. Entonces, G_1 y G_2 tienen menos ocurrencias de conectivas que F. Luego, por hipótesis de inducción $S \vdash G_1^I$ y $T \vdash G_2^I$, en donde S y T son los conjuntos de símbolos proposicionales que ocurren en G_1 y G_2 respectivamente. Luego,

$$\{p_1^I, \ldots, p_m^I\} \vdash G_2^I \tag{2a}$$
$$\{p_1^I, \ldots, p_m^I\} \vdash G_1^I \tag{2b}$$

Distinguimos los siguientes tres subcasos.

Caso 2a. Supongamos que $I'(G_2) = \mathsf{V}$, entonces $I'(F) = \mathsf{V}$. Luego, $G_2^I = G_2$ y $F^I = F$. Por lo tanto,

1. $\{p_1^I, \ldots, p_m^I\} \vdash G_2$ [(2a)]
2. $\{p_1^I, \ldots, p_m^I\} \vdash G2 \to (G1 \to G2)$ [Axioma A1]
3. $\{p_1^I, \ldots, p_m^I\} \vdash G_1 \to G_2$ [Modus ponens, 2 y 1]
4. $\{p_1^I, \ldots, p_m^I\} \vdash F$ [F es $G_1 \to G_2$]
5. $\{p_1^I, \ldots, p_m^I\} \vdash F^I$ [$F^I = F$]

Caso 2b. Supongamos que $I'(G_1) = \mathsf{V}$ y $I'(G_2) = \mathsf{V}$. Entonces $I'(F) = \mathsf{F}$. Luego, $G_1^I = G_1$, $G_2^I = \neg G_2$, y $F^I = \neg F$. Por lo tanto,

1. $\{p_1^I, \ldots, p_m^I\} \vdash G_1$ [(2b) y $G_1^I = G_1$]
2. $\{p_1^I, \ldots, p_m^I\} \vdash \neg G_2$ [(2a) y $G_2^I = \neg G_2$]
3. $\{p_1^I, \ldots, p_m^I\} \vdash G1 \to (\neg G2 \to \neg(G_1 \to G_2))$ [Lema 3.1.28]
4. $\{p_1^I, \ldots, p_m^I\} \vdash \neg(G_1 \to G_2)$ [Modus ponens, 3 y 1]
5. $\{p_1^I, \ldots, p_m^I\} \vdash \neg F$ [4 y F es $G_1 \to G_2$]
6. $\{p_1^I, \ldots, p_m^I\} \vdash F^I$ [5 y $F^I = \neg F$]

Caso 2c. Supongamos que $I'(G_1) = \mathsf{F}$. Entonces $I'(F) = \mathsf{V}$. Luego, $G_1^I = \neg G_1$ y $F^I = F$. Por lo tanto,

1. $\{p_1^I, \ldots, p_m^I\} \vdash \neg G_1$ [(2a) y $G_1^I = \neg G_1$]
2. $\{p_1^I, \ldots, p_m^I\} \vdash \neg G_1 \to (G_1 \to G_2)$ [Lema 3.1.25]
3. $\{p_1^I, \ldots, p_m^I\} \vdash G_1 \to G_2$ [Modus ponens, 2 y 1]
4. $\{p_1^I, \ldots, p_m^I\} \vdash F$ [3 y F es $G_1 \to G_2$]
5. $\{p_1^I, \ldots, p_m^I\} \vdash F^I$ [4 y $F^I = F$]

□

A partir del lema se puede demostrar el teorema de completitud como se muestra a continuación.

Teorema 3.4.3 (Teorema de completitud) *Si* $\vDash F$ *entonces* $\vdash F$.

Demostración: Sea F una tautología. Entonces, para toda interpretación I, se tiene que $I'(F) = \mathsf{V}$ y, por lo tanto, $F^I = F$. Por el lema 3.4.2 tenemos que

$$\{p_1^I, \ldots, p_n^I\} \vdash F \tag{3.1}$$

Sea I una interpretación de $p_2, \ldots, p_n$. Entonces I se puede extender a $p_1, p_2, \ldots, p_n$ de dos maneras: $I(p_1) = \mathsf{V}$ en cuyo caso $p_1^I = p_1$ y tenemos por (3.1)

$$\{p_1, p_2^I, \ldots, p_n^I\} \vdash F,$$

o $I(p_1) = \mathsf{F}$ en cuyo caso $p_1^I = \neg p_1$ y por lo tanto

$$\{\neg p_1, p_2^I, \ldots, p_n^I\} \vdash F.$$

Así, por el teorema del medio excluso, tenemos que

$$\{p_2^I, \ldots, p_n^I\} \vdash F.$$

Si repetimos este proceso n veces podemos eliminar todas las premisas hasta obtener $\{\} \vdash F$.

□

Obsérvese que en la última argumentación de la prueba anterior aparece la parte informal de la demostración. Para su formalización se debe tener un procedimiento que permita la eliminación de hipótesis (premisas). Para tal fin, en lo que sigue hacemos una nueva presentación del lema de Kalmar con base a los conceptos de "hipótesis" y "conclusión" de una fórmula.

3.4.2 Prueba formal del teorema de completitud de la lógica proposicional

En esta sección se formaliza la prueba del teorema de completitud mostrada en la sección anterior.

Comenzamos introduciendo algunos conceptos auxiliares. El primero consiste en asociar a cada conjunto de símbolos proposicionales una interpretación.

Definición 3.4.4 *Sea T un conjunto de símbolos proposicionales.*

*La **interpretación asociada** a T es la interpretación definida por*

$$I_T(P) = \begin{cases} \mathsf{V}, & \text{si } P \in T \\ \mathsf{F}, & \text{de lo contrario.} \end{cases}$$

Su formalización es:

definition *interpretacion-asociada* :: *'b set* ⇒ (*'b* ⇒ *v-verdad*) **where**
interpretacion-asociada T ≡ λ *x*. (*if x* ∈ *T then Verdad else Falso*)

El segundo concepto auxiliar es el conjunto de hipótesis de una fórmula respecto de un conjunto de símbolos proposicionales.

Definición 3.4.5 *Sea T un conjunto de símbolos proposicionales. El conjunto de* ***las hipótesis de una fórmula*** *F* ***con respecto de*** *T se define de la manera siguiente.*

1. $hip_T(P) = \begin{cases} \{P\}, & si\ P \in T \\ \{\neg P\}, & en\ caso\ contrario \end{cases}$
2. $hip_T(\neg G) = hip_T(G)$
3. $hip_T(G \to H) = hip_T(G) \cup hip_T(H)$

Su formalización es:

primrec *hip* :: *'b set* ⇒ *'b formula* ⇒ *'b formula set* **where**
hip T (Atom P) = (if P ∈ T then {Atom P} else {¬. (Atom P)})
| hip T (¬. F) = hip T F
| hip T (F →. G) = hip T F ∪ hip T G

Algunas propiedades de los conjuntos de hipótesis son las siguientes.

Lema 3.4.6 *El conjunto $hip_T(F)$ es finito.*

Su formalización es:

lemma *hip-finito*: *finite (hip T F)*

Lema 3.4.7 *Todo elemento de $hip_T(F)$ es un símbolo proposicional que está en T o es la negación de un símbolo proposicional que no pertenece a T.*

Su formalización es:

lemma *pertenece-hip* [*rule-format*]:
G ∈ hip T F ⟶ (∃ P. (G = (Atom P) ∧ P ∈ T) ∨ (G = (¬.Atom P) ∧ P ∉ T))

Lema 3.4.8 *El conjunto de las hipótesis de una fórmula F respecto del conjunto de símbolos proposicionales $T - \{P\}$, está contenido en la unión de $\{\neg P\}$ con el conjunto de hipótesis de F respecto de T, distintas de P: $hip_{T-\{P\}}(F) \subseteq \{\neg P\} \cup (hip_T(F) - \{P\})$.*

Su formalización es:

lemma *hip-diferencia*: *hip (T−{P}) F ⊆ {¬.(Atom P)} ∪ ((hip T F) − {Atom P})*

Lema 3.4.9 *El conjunto de las hipótesis de una fórmula F respecto del conjunto de símbolos proposicionales $\{P\} \cup T$, está contenido en la unión de $\{P\}$ con el conjunto de hipótesis de F respecto de T, distintas de $\neg P$; es decir, $hip_{\{P\}\cup T}(F) \subseteq \{P\} \cup (hips_T(F) - \{\neg P\})$.*

Su formalización es:

lemma *hip-cons*:
hip $(\{P\} \cup T)$ $F \subseteq \{Atom\ P\} \cup ((hip\ T\ F) - \{\neg.(Atom\ P)\})$

El tercer concepto auxiliar es el de la conclusión de una fórmula respecto de un conjunto de símbolos proposicionales.

Definición 3.4.10 *La **conclusión de la fórmula** F **con respecto de** T es*

$$con_T(F) = \begin{cases} F, & si\ I'_T(F) = \mathsf{V} \\ \neg F, & si\ I'_T(F) = \mathsf{F} \end{cases}$$

Su formalización es:

definition *con* :: *'b set* $\Rightarrow$ *'b formula* $\Rightarrow$ *'b formula* **where**
con T F $\equiv$ (*if valor* (*interpretacion-asociada T*) *F* = *Verdad then F else* $\neg$. *F*)

Los siguientes lemas son consecuencia directa de los conceptos anteriores.

Lema 3.4.11 *Sea T un conjunto de símbolos proposicionales. El valor de la conclusión de F respecto de T en la interpretación asociada a T es verdadero.*

Su formalización es:

lemma *valor-con-interpretacion-asociada*:
valor (*interpretacion-asociada T*)(*con T F*) = *Verdad*

Lema 3.4.12 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \in T$. Entonces $I'_T(P) = \mathsf{V}$.*

Su formalización es:

lemma *valor-atomica-pertenece*:
assumes $P \in T$
shows *valor* (*interpretacion-asociada T*) (*Atom P*) = *Verdad*

Lema 3.4.13 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \notin T$. Entonces $I'_T(P) = \mathsf{F}$.*

Su formalización es:

lemma *valor-atomica-no-pertenece*:
assumes $P \notin T$
shows *valor* (*interpretacion-asociada* T) (*Atom* P) = *Falso*

Lema 3.4.14 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \in T$, entonces $hip_T(P) = \{P\}$.*

Su formalización es:

lemma *hip-atomica-pertenece*:
assumes $P \in T$
shows *hip* T (*Atom* P) = {*Atom* P}

Lema 3.4.15 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \notin T$, entonces $hip_T(P) = \{\neg P\}$.*

Su formalización es:

lemma *hip-atomica-no-pertenece*:
assumes $P \notin T$
shows *hip* T (*Atom* P) = {$\neg$.(*Atom* P)}

Lema 3.4.16 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \in T$, entonces $con_T(P) = P$.*

Su formalización es

lemma *con-atomica-pertenece*:
assumes F = (*Atom* P) **and** $P \in T$
shows *con* T F = (*Atom* P)

Lema 3.4.17 *Sea T un conjunto de símbolos proposicionales. Supongamos que $P \notin T$, entonces $con_T(P) = \neg P$.*

Su formalización es:

lemma *con-atomica-no-pertenece*:
assumes $P \notin T$
shows *con* T $(Atom\ P) = \neg.(Atom\ P)$

Con las definiciones anteriores, el lema de Kalmar afirma que si F es una fórmula y T es un conjunto de símbolos proposicionales, entonces $hip_T(F) \vdash con_T(F)$. En la demostración usamos los siguientes 5 lemas (3.4.18-3.4.22).

Lema 3.4.18 (casoa) *Si* $S \vdash G$ *entonces* $S \vdash F \rightarrow G$.

Su formalización es:

lemma *casoa*:
assumes $S \vdash G$
shows $S \vdash F \rightarrow. G$

Lema 3.4.19 (casob) *Si* $S \vdash F$ *y* $S \vdash \neg G$, *entonces* $S \vdash \neg(F \rightarrow G)$.

Su formalización es:

lemma *casob*:
assumes $S \vdash F$
and $S \vdash \neg. G$
shows $S \vdash \neg. (F \rightarrow. G)$

Lema 3.4.20 (casoc) *Si* $S \vdash \neg F$, *entonces* $S \vdash F \rightarrow G$.

Su formalización es:

lemma *casoc*:
assumes $S \vdash \neg. F$
shows $S \vdash F \rightarrow. G$

Los siguientes dos lemas servirán para simplificar la longitud de la prueba del lema de Kalmar.

Lema 3.4.21 *Si F es* $\neg G$, *entonces, para todo conjunto de fórmulas S y todo conjunto de símbolos proposicionales T se tiene que* $S \vdash con_T(G) \rightarrow con_T(F)$.

Su formalización es:

```
lemma con-negacion:
  assumes F = ¬. G
  shows S ⊢ con T G →. con T F
proof (cases valor (interpretacion-asociada T) G)
```

Lema 3.4.22 *Si F es $G \to H$, $S \vdash con_T(G)$ y $S \vdash con_T(H)$. Entonces, $S \vdash con_T(F)$.*

Su formalización es:

```
lemma con-implicacion:
  assumes F = G →. H and S ⊢ con T G and S ⊢ con T H
  shows S ⊢ con T F
```

La demostración del lema de Kalmar es por inducción en el *tamaño* de una fórmula. El tamaño de una fórmula se define de la siguiente manera.

Definición 3.4.23 *Sean F y G fórmulas, entonces*

- *tamaño(F) = 0 si F es una fórmula atómica, $\bot$ o $\top$*
- *tamaño$(\neg F)$ = 1 + tamaño(F)*
- *tamaño$(F \to G)$ = 1+tamaño(F)+ tamaño(G)*

Lema 3.4.24 (Kalmar) *Sea F una fórmula y T un conjunto de símbolos proposicionales, entonces $hip_T(F) \vdash con_T(F)$.*

Demostración: Por inducción en el tamaño de la fórmula. Sea n un número natural. La hipótesis de inducción (HI) afirma que para toda fórmula F de tamaño menor que n se tiene $hip_T(F) \vdash con_T(F)$. Tenemos que demostrar que para toda fórmula F de tamaño n se tiene $hip_T(F) \vdash con_T(F)$.

Sea F una fórmula. Supongamos (HI1) que el tamaño de F es n. Tenemos que demostrar que $hip_T(F) \vdash con_T(F)$. Lo haremos por distinción de casos según la forma de F.

Caso 1: Supongamos que F es la fórmula atómica P. Distinguimos dos casos:

Caso 1a: Supongamos que $P \in T$. Entonces
1. $hip_T(F) = \{P\}$ [Lema 3.4.14]
2. $con_T(F) = P$ [Lema 3.4.16]
3. $hip_T(F) \vdash con_T(F)$ [1, 2 y lema 3.1.5]

Caso 1b: Supongamos que $P \notin T$. Entonces

1. $hip_T(F) = \{\neg P\}$ [Lema 3.4.15]
2. $con_T(F) = \neg P$ [Lema 3.4.17]
3. $hip_T(F) \vdash con_T(F)$ [1, 2 y lema 3.1.5]

Caso 2: Supongamos que F es $\neg G$. Entonces,

1. $hip_T(G) \vdash con_T(G)$ [Por Hipótesis de inducción]
2. $hip_T(F) = hip_T(G)$ [Lema 3.4.5]
3. $hip_T(F) \vdash con_T(G)$ [1 y 2]
4. $hip_T(F) \vdash con_T(G) \to con_T(F)$ [Lema 3.4.21]
5. $hip_T(F) \vdash con_T(F)$ [Modus ponens, 4 y 3]

Caso 3: Supongamos que F es $G \to H$. Entonces, se tiene que

$$hip_T(F) \vdash con_T(G) \tag{1}$$

En efecto, la longitud de G es menor que la longitud de F y, por (HI1) es menor que n. Luego, por (HI), se tiene $hip_T(G) \vdash con_T(G)$. Además, $hip_T(G) \subseteq hip_T(F)$. Por lo tanto, usando la regla de debilitamiento, se tiene que $hip_T(F) \vdash con_T(G)$.

De la misma forma, se tiene que,

$$hip_T(F) \vdash con_T(H) \tag{2}$$

Por el lema 3.4.22, junto con (1) y (2), se tiene que $hip_T(F) \vdash con_T(F)$.

□

Con los anteriores resultados podemos formalizar la prueba del lema de Kalmar como sigue; la función *size* predefinida en Isabelle, formaliza la función *tamaño*.

```
lemma kalmar[rule-format]:
 ∀ (F:: 'b formula). size F = n ⟶ hip T F ⊢ con T F
proof (induct rule: nat-less-induct)
 fix n
 assume HI: ∀ m < n. ∀ F. size F = m ⟶ hip T F ⊢ con T F
 show  ∀ F. size F = n ⟶ hip T F ⊢ con T F
 proof
  fix  F
  show size F = n ⟶ hip T F ⊢ con T F
  proof
   assume HI1: size F = n
   show hip T F ⊢ con T F
   proof (cases F)
   — Caso 1:
   { fix P:: 'b
    assume Caso1: F = Atom P
    show hip T F ⊢ con T F
    proof (cases P ∈ T)
     — Caso 1a
```

```
{ assume Caso1a: P ∈ T
  hence hip T F = {Atom P}
  using Caso1 hip-atomica-pertenece by simp
  moreover
  hence con T F = Atom P
    using Caso1 Caso1a con-atomica-pertenece[of F] by auto
  ultimately show hip T F ⊢ con T F using hip by auto }
 — Caso 1b
 { assume Caso1a: P ∉ T
  hence hip T F = {¬. F}
    using Caso1 hip-atomica-no-pertenece by simp
  moreover hence con T F = ¬. F
    using Caso1 Caso1a con-atomica-no-pertenece by simp
  ultimately show hip T F ⊢ con T F using hip by auto }
 qed }
 next
 — Caso 2:
{ fix G
 assume Caso2: F = ¬. G
 hence size G < size F by simp
 hence b1: hip T G ⊢ con T G using HI and HI1 by simp
 have b2: hip T F = hip T G using Caso2 by simp
 hence b3: hip T F ⊢ con T G using b1 by simp
 have b4: hip T F ⊢ con T G →. con T F
  using Caso2 con-negacion by auto
 show hip T F ⊢ con T F
  using b4 b3 by (rule mp) }
 next
 — Caso 3:
{ fix G H
 assume Caso3: F = G →. H
 have 1: hip T F ⊢ con T G
 proof −
  have size G < size F using Caso3 by simp
  hence hip T G ⊢ con T G using HI and HI1 by simp
  moreover
  have hip T G ⊆ hip T F using Caso3 by auto
  ultimately
  show hip T F ⊢ con T G by (rule debilitamiento)
 qed
 have 2: hip T F ⊢ con T H
 proof −
  have size H < size F using Caso3 by simp
```

```
      hence hip T H ⊢ con T H using HI and HI1 by simp
      moreover
      have hip T H ⊆ hip T F using Caso3 by auto
      ultimately
      show hip T F ⊢ con T H by (rule debilitamiento)
     qed
     show hip T F ⊢ con T F
      using con-implicacion 1 2 Caso3 by blast}
    qed
   qed
  qed
qed
```

El siguiente lema de eliminación de variables permitirá la demostración del teorema de completitud

Lema 3.4.25 (Lema de eliminación de variables) *Sea F una tautología. Si H es un conjunto finito de símbolos proposicionales, entonces para todo conjunto de símbolos proposicionales T se tiene que, $hip_T(F) - H \vdash F$.*

Demostración: Por inducción finita sobre H.

Base: Supongamos que $H = \emptyset$. Sea T un conjunto de símbolos proposicionales. Se tiene

$$\begin{aligned} hip_T(F) - H &= hip_T(F) - \emptyset \\ &= hip_T(F) \\ &\vdash con_T(F) && \text{[Lema 3.4.24]} \\ &= F && [F \text{ es una tautología}] \end{aligned}$$

Paso de inducción Sea H un conjunto finito de símbolos proposicionales y $x \notin H$. La hipótesis de inducción es

$$\forall T.(hip_T(F) - H \vdash F \qquad \text{(HI)}$$

Hay que probar que

$$\forall T.(hip_T(F) - (\{x\} \cup H) \vdash F$$

Sea T un conjunto de símbolos proposicionales. Demostraremos que

$$hip_T(F) - (\{x\} \cup H) \vdash F \qquad (1)$$

distinguiendo dos casos.

Caso 1: Supongamos que $x \in hip_T(F)$. Entonces, por el lema 3.4.7, existe un símbolo proposicional P tal que x es P con $P \in T$ ó x es $\neg P$ con $P \notin T$. Demostraremos (1) distinguiendo dos subcasos:

Caso 1a: Supongamos que x es P con $P \in T$. Por el lema 3.1.31, basta probar que

$$\{P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F \qquad \text{(1a1)}$$
$$\{\neg P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F \qquad \text{(1a2)}$$

La demostración de (1a1) es la siguiente:

1. $hip_T(F) - H \subseteq \{P\} \cup (hip_T(F) - (\{x\} \cup H))$ [x es P]
2. $hip_T(F) - H \vdash F$ [Hipótesis de inducción]
3. $\{P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ [1, 2 y lema 3.1.6]

La demostración de (1a2) es la siguiente:

1. $hip_{T-\{P\}}(F) \subseteq (\neg P) \cup (hip_T(F) - \{P\})$ [Lema 3.4.8]
2. $hip_{T-\{P\}}(F) - H \subseteq (\neg P) \cup (hip_T(F) - (\{x\} \cup H))$ [1 y x es P]
3. $hip_{T-\{P\}}(F) - H \vdash F$ [Hipótesis de inducción]
4. $\{\neg P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ [2, 3 y lema 3.1.6]

Caso 1b: Supongamos que x es $\neg P$ con $P \notin T$. Por el lema 3.1.31, basta probar que

$\{P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ (1b1)
$\{\neg P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ (1b2)

La demostración de (1b1) es la siguiente:

1. $hip_{\{P\}\cup T}(F) \subseteq \{P\} \cup (hip_T(F) - \{\neg P\}$ [Lema 3.4.9]
2. $hip_{\{P\}\cup T}(F) - H \subseteq \{P\} \cup (hip_T(F) - (\{x\} \cup H))$ [1 y x es $\neg P$]
3. $hip_{\{P\}\cup T}(F) - H \vdash F$ [Hipótesis de inducción]
4. $\{P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ [2, 3 y lema 3.1.6]

La demostración de (1b2) es la siguiente:

1. $hip_T(F) - H \subseteq \{\neg P\} \cup (hip_T(F) - (\{x\} \cup H))$ [x es $\neg P$]
2. $hip_T(F) - H \vdash F$ [Hipótesis de inducción]
3. $\{\neg P\} \cup (hip_T(F) - (\{x\} \cup H)) \vdash F$ [1, 2 y lema 3.1.6]

Caso 2: Supongamos que $x \notin hip_T(F)$. Entonces,

1. $hip_T(F) - (\{x\} \cup H) = hip_T(F) - H$
2. $hip_T(F) - H \vdash F$ [Hipótesis de inducción]
3. $hip_T(F) - (\{x\} \cup H) \vdash F$ [1 y 2]

□

Su formalización es:

```
lemma eliminacion-variables:
  assumes tautologia F
  shows finite H ⟹ ∀ T. ((hip T F) - H) ⊢ F
proof (induct rule: finite-induct)
  — Base:
{ fix T
  have (hip T F) - {} = hip T F by simp
  moreover have hip T F ⊢ con T F using kalmar by auto
  moreover have  con T F = F
  using assms by(simp add: con-def tautologia-def)
  ultimately have (hip T F) - {} ⊢ F by simp }
  thus ∀ T. ((hip T F) - {}) ⊢ F by simp — Fin de la base
```

```
 next
 — Paso:
{ fix x H
 assume
finite H and  x ∉ H and HI: ∀ T. ((hip T F) − H) ⊢ F
 show ∀ T. ((hip T F) − (insert x H)) ⊢ F
 proof
{ fix T
 show (hip T F) − (insert x H) ⊢ F
 proof (cases x ∈ (hip T F))
  — Caso 1:
 { assume 1: x ∈ (hip T F)
  obtain P where 2: (x = Atom P ∧ P ∈ T) ∨ (x = ¬.Atom P ∧ P ∉ T)
    using 1  pertenece-hip by blast
  show (hip T F) − (insert x H) ⊢ F
  proof (cases x = Atom P ∧ P ∈ T)
  — Caso 1a:
  { assume x = Atom P ∧ P ∈ T
   show (hip T F) − (insert x H) ⊢ F
   proof −
  — Caso 1a1:
     have {Atom P} ∪ ((hip T F) − (insert x H)) ⊢ F
     proof −
      have (hip T F) − H ⊆
           insert x ((hip T F) − (insert x  H)) by auto
      moreover
      from HI have (hip T F) − H ⊢ F by simp
      ultimately
      have insert x ((hip T F) − (insert x H)) ⊢ F
       using debilitamiento by auto
      thus ?thesis
       using ‹x = Atom P ∧ P ∈ T› by auto
      qed — Fin del Caso 1a1
      moreover
    — Caso 1a2:
      have {¬.(Atom P)} ∪ ((hip T F) − (insert x H)) ⊢ F
      proof −
       have hip (T−{P}) F ⊆
            insert (¬. (Atom P)) ((hip T F) − {Atom P})
         using hip-diferencia by(induct F) auto
       hence (hip (T−{P}) F) − H ⊆
             {¬. (Atom P)} ∪ ((hip T F) − (insert x H))
          using ‹x = Atom P ∧ P ∈ T› by auto
```

```
      moreover
      from HI have (hip (T−{P}) F) − H ⊢ F by simp
      ultimately
      show {¬. (Atom P)} ∪ ((hip T F) − (insert x H)) ⊢ F
       using debilitamiento by simp
     qed — Fin del Caso 1a2
     ultimately show (hip T F) − (insert x H) ⊢ F
       using tercio-excluso-2[of Atom P (hip T F)−(insert x H) F]
       by auto
   qed } — Fin del Caso 1a
   next
— Caso 1b:
  { assume ¬(x = Atom P ∧ P ∈ T)
   hence x = ¬. (Atom P) ∧ P ∉ T
   using 2 by blast
   show (hip T F) − (insert x H) ⊢ F
   proof −
— Caso 1b1:
   have {Atom P} ∪ ((hip T F) − (insert x H)) ⊢ F
   proof −
    have hip ({P}∪T) F ⊆ {Atom P}∪((hip T F)−{¬. (Atom P)})
     by (rule hip-cons)
    hence (hip ({P} ∪ T) F) − H ⊆
        {Atom P}∪((hip T F) − ({x}∪H))
     using ⟨x = ¬. (Atom P) ∧ P ∉ T⟩ by auto
    moreover
    from HI have (hip ({P}∪T) F) − H ⊢ F by simp
    ultimately
    show {Atom P} ∪ ((hip T F) − (insert x H)) ⊢ F
     using debilitamiento by simp
   qed — Fin del Caso 1b1
   moreover
— Caso 1b2:
   have {¬. (Atom P)} ∪ ((hip T F) − (insert x H)) ⊢ F
   proof −
    have (hip T F) − H ⊆
        {¬. (Atom P)} ∪ ((hip T F) − (insert x H))
     using ⟨x = ¬. (Atom P) ∧ P ∉ T⟩ by auto
    moreover
    from HI have (hip T F) − H ⊢ F by simp
    ultimately
    show {¬. (Atom P)} ∪ ((hip T F) − ((insert x H))) ⊢ F
     using debilitamiento by simp
```

```
      qed — Fin del Caso 1b2
      ultimately
      show (hip T F) − (insert x H) ⊢ F
        using tercio-excluso-2[of Atom P (hip T F)−(insert x H)] by simp
    qed } — Fin del Caso 1b
  qed } — Fin del Caso 1
    next
 — Caso 2
  { assume x ∉ (hip T F)
    show (hip T F) − (insert x H) ⊢ F
    proof −
      have (hip T F) − (insert x H) = (hip T F) − H
        using ‹x ∉ (hip T F)› by simp
      moreover
      have (hip T F) − H ⊢ F using HI by simp
      ultimately show (hip T F) − (insert x H) ⊢ F by simp
    qed } — Fin del Caso 2
  qed }
 qed } — Fin del paso
qed
```

Finalmente, demostramos el teorema de completitud de la lógica proposicional.

Teorema 3.4.26 (de completitud ([36] p. 42)) *Todas las tautologías son deducibles, es decir, para toda fórmula F, si $\models F$, entonces $\vdash F$.*

Demostración: Sea F una tautología. Por el lema 3.4.6, $hip_T(F)$ es un conjunto finito y, por el lema 3.4.25, $hip_T(F) - hip_T(F) \vdash F$. Por tanto, $\emptyset \vdash F$.

□

Su formalización es:

```
theorem completitud:
 assumes tautologia F
 shows {} ⊢ F
proof −
 have finite (hip T F) by (rule hip-finito)
 hence (hip T F) − (hip T F) ⊢ F
   using assms eliminacion-variables by blast
 thus {} ⊢ F by simp
qed
```

Capítulo 4

Forma normal conjuntiva

4.1 Sintaxis y semántica proposicional

En la lógica proposicional presentada en el capítulo anterior, las fórmulas se construyeron a partir de las conectivas $\neg$, $\longrightarrow$. Para el desarrollo de los próximos capítulos acerca de la forma normal conjuntiva de una fórmula proposicional, un método de prueba para la lógica proposicional basado en tableros semánticos y el teorema de existencia de modelos en la lógica proposicional, ampliaremos el conjunto de conectivas a $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\longrightarrow$. Más precisamente, la sintaxis y semántica de los lenguajes proposicionales que consideraremos y su formalización es la siguiente.

Definición 4.1.1 *El **alfabeto** de un lenguaje proposicional se compone de los siguientes símbolos.*

1. ***Símbolos lógicos:***
   - *conectivas:* $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\longrightarrow$
   - *símbolos de puntuación:* $'('$, $')'$, $','$
2. ***Símbolos no lógicos:***
   - *símbolos proposicionales* $P_0, P_1, \ldots$

Las *fórmulas* de un lenguaje proposicional se definen de la siguiente manera.

Definición 4.1.2 *El conjunto de* **fórmulas** *es el conjunto más pequeño que satisface las siguientes condiciones:*

1. *Los símbolos* $\bot$ *y* $\top$ *son fórmulas.*
2. *Cualquier símbolo proposicional es una fórmula.*

3. *Si F es una fórmula, entonces* $\neg F$ *es una fórmula.*

4. *Si F y G son fórmulas, entonces* $(F \wedge G), (F \vee G), (F \rightarrow G)$ *son fórmulas.*

Los símbolos proposicionales son llamados **fórmulas atómicas**.

El siguiente tipo de datos define en Isabelle las fórmulas de un lenguaje proposicional.

```
datatype 'b formula1 =
  FF
| TT
| Atomo 'b
| Nega 'b formula1                (¬.(-) [110] 110)
| Conju 'b formula1 'b formula1   (infixl ∧. 109)
| Disya 'b formula1 'b formula1   (infixl ∨. 108)
| Implica 'b formula1 'b formula1 (infixl →. 100)
```

Consideramos de nuevo la semántica de la lógica proposicional estudiada en la sección 3.2 con el fin de incluir la semántica de las constantes $\bot$, $\top$ y las conectivas $\wedge$ y $\vee$. En la semántica, ampliamos la definición del valor de verdad de una fórmula, reformulando previamente los conceptos de valor de verdad (4.1.3) e interpretación (4.1.4).

Definición 4.1.3 *Los* ***valores de verdad*** *son* V *(que se interpreta como* ***verdadero****) y* F *(que se interpreta como* ***falso****).*

En la siguiente formalización se identifica el conjunto de los valores de verdad con el tipo *v-verdad1*, el valor V con la constante *Verdad* y el valor F con la constante *Falso* respectivamente.

```
datatype v-verdad1 = Verdad | Falso
```

Definición 4.1.4 *Una* ***interpretación*** *es una aplicación del conjunto de los símbolos proposicionales en el conjunto de los valores de verdad.*

La representación en Isabelle del concepto de interpretación es una función de la forma $I :: 'b \Rightarrow$ *v-verdad1*.

Definición 4.1.5 *El* ***valor*** *de una fórmula proposicional F en una interpretación I (representado por* $I'(F)$*) se define por recursión como sigue:*

- $I'(P) = I(P)$ *si P es un símbolo proposicional*

- $I'(\neg F) = \begin{cases} \mathsf{V}, & si\ I'(F) = \mathsf{F} \\ \mathsf{F}, & si\ I'(F) = \mathsf{V} \end{cases}$
- $I'(F \wedge G) = \begin{cases} \mathsf{V}, & si\ I'(F) = \mathsf{V}\ y\ I'(G) = \mathsf{V} \\ \mathsf{F}, & en\ caso\ contrario. \end{cases}$
- $I'(F \vee G) = \begin{cases} \mathsf{F}, & si\ I'(F) = \mathsf{F}\ y\ I'(G) = \mathsf{F} \\ \mathsf{V}, & en\ caso\ contrario. \end{cases}$
- $I'(F \rightarrow G) = \begin{cases} \mathsf{F}, & si\ I'(F) = \mathsf{V}\ y\ I'(G) = \mathsf{F} \\ \mathsf{V}, & en\ caso\ contrario. \end{cases}$

El valor de una fórmula proposicional se formaliza en Isabelle de la siguiente manera.

definition *v-negacion1* :: *v-verdad1* ⇒ *v-verdad1* **where**
v-negacion1 x ≡ (*if x* = *Verdad then Falso else Verdad*)

definition *v-conjuncion1* :: *v-verdad1* ⇒ *v-verdad1* ⇒ *v-verdad1* **where**
v-conjuncion1 x y ≡ (*if x* = *Falso then Falso else y*)

definition *v-disyuncion1* :: *v-verdad1* ⇒ *v-verdad1* ⇒ *v-verdad1* **where**
v-disyuncion1 x y ≡ (*if x* = *Verdad then Verdad else y*)

definition *v-implicacion1* :: *v-verdad1* ⇒ *v-verdad1* ⇒ *v-verdad1* **where**
v-implicacion1 x y ≡ (*if x* = *Falso then Verdad else y*)

primrec *valor1* :: ('*b* ⇒ *v-verdad1*) ⇒ '*b formula1* ⇒ *v-verdad1*
where
valor1 I FF = *Falso*
| *valor1 I TT* = *Verdad*
| *valor1 I* (*Atomo p*) = *I p*
| *valor1 I* (¬. *F*) = (*v-negacion1* (*valor1 I F*))
| *valor1 I* (*F* ∧. *G*) = (*v-conjuncion1* (*valor1 I F*) (*valor1 I G*))
| *valor1 I* (*F* ∨. *G*) = (*v-disyuncion1* (*valor1 I F*) (*valor1 I G*))
| *valor1 I* (*F* →. *G*) = (*v-implicacion1* (*valor1 I F*) (*valor1 I G*))

En algunas demostraciones en las que se utiliza la noción de valor de verdad de una fórmula, en lugar de la definición se suele usar algunas de la propiedades que presentamos a continuación.

Lema 4.1.6 *Sea F un fórmula e I una interpretación. Entonces,* $I'(F) = \mathsf{V}$ *ó* $I'(F) = \mathsf{F}$.

Su formalización es:

lemma *CasosValor*:
shows *valor1 I F = Verdad ∨ valor1 I F = Falso*

Lema 4.1.7 *Sea F una fórmula e I una interpretación. Si* $I'(\neg F) = \mathsf{F}$, *entonces* $I'(F) = \mathsf{V}$.

Su formalización es:

lemma *ValoresNegacion1*:
assumes *valor1 I* $(\neg.F)$ *= Falso*
shows *valor1 I F = Verdad*

Lema 4.1.8 *Sea F un fórmula e I una interpretación. Si* $I'(\neg F) = \mathsf{V}$, *entonces* $I'(F) = \mathsf{F}$.

Su formalización es:

lemma *ValoresNegacion2*:
assumes *valor1 I* $(\neg.F)$ *= Verdad*
shows *valor1 I F = Falso*

Lema 4.1.9 *Sean F y G fórmulas e I una interpretación. Si* $I'(F \wedge G) = \mathsf{V}$, *entonces* $I'(F) = \mathsf{V}$ *e* $I'(G) = \mathsf{V}$

Su formalización es:

lemma *ValoresConjuncion*:
assumes *valor1 I* $(F \wedge. G)$ *= Verdad*
shows *valor1 I F = Verdad ∧ valor1 I G = Verdad*

Lema 4.1.10 *Sean F y G fórmulas e I una interpretación. Si* $I'(F \vee G) = \mathsf{V}$, *entonces* $I'(F) = \mathsf{V}$ *ó* $I'(G) = \mathsf{V}$

Su formalización es:

lemma *ValoresDisyuncion*:
assumes *valor1 I* $(F \vee. G)$ *= Verdad*
shows *valor1 I F = Verdad ∨ valor1 I G = Verdad*

Lema 4.1.11 *Sea F y G fórmulas e I una interpretación tales que* $I'(F \to G) = \mathsf{V}$. *Si* $I'(F) = \mathsf{V}$, *entonces* $I'(G) = \mathsf{V}$.

Su formalización es:

lemma *ValoresImplicacion*:
assumes *valor1 I* $(F \rightarrow. G) = Verdad$
shows *valor1 I F* $= Verdad \longrightarrow$ *valor1 I G* $= Verdad$

Formalizamos de nuevo las nociones de satisfactibilidad, consecuencia lógica y tautología en términos del concepto de *modelo*. Las demostraciones de las nuevas propiedades que se enuncian acerca de estos conceptos se obtienen directamente a partir de las definiciones.

Definición 4.1.12 *Una interpretación I es* ***modelo*** *de un conjunto de fórmulas S si, para toda fórmula F de S,* $I'(F) = \mathsf{V}$.

Su formalización es:

definition *modelo* :: $('b \Rightarrow v\text{-}verdad1) \Rightarrow$ *'b formula1 set* $\Rightarrow$ *bool* (- *modelo* - [*80,80*] *80*) **where**
I modelo S $\equiv (\forall F \in S.$ *valor1 I F* $= Verdad)$

Definición 4.1.13 *Un conjunto de fórmulas es* ***satisfacible*** *si tiene algún modelo. En caso contrario se dice que es* ***insatisfacible****.*

Su formalizacion es

definition *satisfacible* :: *'b formula1 set* $\Rightarrow$ *bool* **where**
satisfacible S $\equiv (\exists v.$ *v modelo S*$)$

Definición 4.1.14 *Una fórmula F es una* ***consecuencia lógica*** *de un conjunto de fórmulas S si para todo modelo I de S se tiene que* $I'(F) = \mathsf{V}$. *Se representa por* $S \models F$.

Su formalización es:

definition *consecuencia1* :: *'b formula1 set* $\Rightarrow$ *'b formula1* $\Rightarrow$ *bool* (- $\models$*1* - [*80,80*] *80*) **where**
$S \models 1\ F \equiv (\forall I.$ *I modelo S* $\longrightarrow$ *valor1 I F* $= Verdad)$

El siguiente resultado establece la relación entre los conceptos de consecuencia lógica y satisfactibilidad.

Teorema 4.1.15 $S \models F$ *syss* $S \cup \{\neg F\}$ *no es satisfacible.*

Su formalización es:

theorem *EquiConsSat*:
shows $S \models 1\ F = (\neg$ *satisfacible* $(S \cup \{\neg. F\}))$

Por último reformulamos el concepto de tautología.

Definición 4.1.16 *Una fórmula F es una* ***tautología*** *si* $I'(F) = \mathsf{V}$ *para toda interpretación I.*

Su formalización es:

definition *tautologia* :: *'b formula1* ⇒ *bool* **where**
tautologia F ≡ (∀ *I*. ((*valor1 I F*) = *Verdad*))

El concepto de tautología se puede formular en términos del concepto de consecuencia lógica:

Lema 4.1.17 *Una fórmula es tautología syss es consecuencia del conjunto vacío.*

Su formalizacion es

lemma *CNS-tautologia*: *tautologia F* = ({} |=*1 F*)

Teorema 4.1.18 *La implicación* $F \to G$ *es una tautología syss el conjunto* $\{F, \neg G\}$ *no es satisfacible.*

Su formalización es:

theorem *TautSatis*:
shows *tautologia* (*F* →. *G*) = (¬ *satisfacible*{*F*, ¬.*G*})

4.2 Notación uniforme

En el texto de Fitting [14] se considera la noción de *notación uniforme*, introducida por R. M. Smullyan [50], que permite clasificar las fórmulas de tal forma que algunos teoremas de la lógica de primer orden y, como consecuencia, algunos procedimientos utilizados en la prueba automática de teoremas, pueden describirse de manera mucho más simple. En esta sección formalizamos este concepto dividiendo las fórmulas proposicionales en cuatro tipos de fórmulas: *literales* (4.2.1), *dobles negaciones* (4.2.2), *fórmulas alfa* (4.2.3) y *fórmulas beta* (4.2.4); y definiendo el tipo *tipoNotaciónUniforme* (4.2.12) con estos cuatro valores. Los siguientes lemas y funciones caracterizan esta clasificación.

Definición 4.2.1 *Una fórmula es un* ***literal*** *si es un símbolo proposicional, la negación de un símbolo proposicional,* $\bot$, $\top$, $\neg\bot$ *o* $\neg\top$.

Su formalización es:

```
fun FormulaLiteral :: 'b formula1 ⇒ bool where
  FormulaLiteral FF = True
| FormulaLiteral (¬. FF) = True
| FormulaLiteral TT = True
| FormulaLiteral (¬. TT) = True
| FormulaLiteral (Atomo P) = True
| FormulaLiteral (¬.(Atomo P)) = True
| FormulaLiteral F = False
```

Definición 4.2.2 *Una **doble negación** es una fórmula de la forma $\neg\neg$ F, donde F es una fórmula proposicional.*

Su formalización es:

```
fun FormulaNoNo :: 'b formula1 ⇒ bool where
  FormulaNoNo (¬. (¬. F)) = True
| FormulaNoNo F = False
```

Definición 4.2.3 *Una **fórmula alfa** es una fórmula de la forma $F \wedge G$, $\neg(F \vee G)$, o $\neg(F \rightarrow G)$, donde F y G son fórmulas proposicionales.*

Su formalización es:

```
fun FormulaAlfa :: 'b formula1 ⇒ bool where
  FormulaAlfa (F ∧. G) = True
| FormulaAlfa (¬. (F ∨. G)) = True
| FormulaAlfa (¬. (F →. G)) =  True
| FormulaAlfa F = False
```

Definición 4.2.4 *Una **fórmula beta** es una fórmula de la forma $F \vee G$, $\neg(F \wedge G)$, o $F \rightarrow G$, donde F y G son fórmulas proposicionales.*

Su formalización es:

```
fun FormulaBeta :: 'b formula1 ⇒ bool where
  FormulaBeta (F ∨. G) = True
| FormulaBeta (¬. (F ∧. G)) = True
| FormulaBeta (F →. G) =  True
| FormulaBeta F = False
```

Los siguientes lemas garantizan que cada fórmula proposicional pertenence exactamente a una de las cuatro categorías anteriores.

Lema 4.2.5 *Los literales no son dobles negaciones.*

Su formalización es:

lemma *noLiteralNoNo*:
assumes *FormulaLiteral formula*
shows ¬(*FormulaNoNo formula*)
using *assms Literal NoNo*
by (*induct formula rule*: *FormulaLiteral.induct*, *auto*)

Lema 4.2.6 *Los literales no son fórmulas alfa.*

Su formalización es:

lemma *noLiteralAlfa*:
assumes *FormulaLiteral formula*
shows ¬(*FormulaAlfa formula*)
using *assms Literal Alfa*
by (*induct formula rule*: *FormulaLiteral.induct*, *auto*)

Lema 4.2.7 *Los literales no son fórmulas beta.*

Su formalización es:

lemma *noLiteralBeta*:
assumes *FormulaLiteral formula*
shows ¬(*FormulaBeta formula*)
using *assms Literal Beta*
by (*induct formula rule*: *FormulaLiteral.induct*, *auto*)

Lema 4.2.8 *Los fórmulas alfa no son dobles negaciones.*

Su formalización es:

lemma *noAlfaNoNo*:
assumes *FormulaAlfa formula*
shows ¬(*FormulaNoNo formula*)
using *assms Alfa NoNo*
by (*induct formula rule*: *FormulaAlfa.induct*, *auto*)

Lema 4.2.9 *Los fórmulas beta no son dobles negaciones.*

Su formalización es:

lemma *noBetaNoNo*:
assumes *FormulaBeta formula*
shows ¬(*FormulaNoNo formula*)
using *assms Beta NoNo*
by (*induct formula rule*: *FormulaBeta.induct*, *auto*)

Lema 4.2.10 *Los fórmulas alfa no son fórmulas betas.*

Su formalización es:

lemma *noAlfaBeta*:
assumes *FormulaAlfa formula*
shows ¬(*FormulaBeta formula*)
using *assms Alfa Beta*
by (*induct formula rule*: *FormulaAlfa.induct*, *auto*)

Lema 4.2.11 *Toda fórmula es de una de las clases literal, doble negación, alfa o beta.*

Su formalización es:

lemma *NotacionUniforme*:
FormulaLiteral F ∨ *FormulaNoNo F* ∨ *FormulaAlfa F* ∨ *FormulaBeta F*

Definición 4.2.12 *Los tipos de fórmulas son literal, alfa, beta y doble negación.*

Su formalización es:

datatype *tipoNotacionUniforme* = *Literal* | *NoNo* | *Alfa*| *Beta*

Definición 4.2.13 *El tipo de una fórmula F es la clase del tipo de F.*

Su formalización es:

fun *tipoFormula* :: *'b formula1* ⇒ *tipoNotacionUniforme* **where**
tipoFormula F =
(*if FormulaBeta F then Beta*
else if FormulaNoNo F then NoNo
else if FormulaAlfa F then Alfa
else Literal)

Definición 4.2.14 *El conjunto de las componentes de la formula F es*

- $\{G\}$ *si F es* $\neg\neg G$
- $\{G, H\}$ *si F es* $G \wedge H$
- $\{\neg G, \neg H\}$ *si F es* $\neg(G \vee H)$
- $\{G, \neg H\}$ *si F es* $\neg(G \rightarrow H)$
- $\{G, H\}$ *si F es* $G \vee H$
- $\{\neg G, \neg H\}$ *si F es* $\neg(G \wedge H)$
- $\{\neg G, H\}$ *si F es* $G \rightarrow H$

Su formalización es:

fun *componentes* :: *'b formula1* $\Rightarrow$ *'b formula1 list* **where**
componentes $(\neg.\ (\neg.\ G)) = [G]$
| *componentes* $(G \wedge.\ H) = [G, H]$
| *componentes* $(\neg.\ (G \vee.\ H)) = [\neg.\ G, \neg.\ H]$
| *componentes* $(\neg.\ (G \rightarrow.\ H)) = [G, \neg.\ H]$
| *componentes* $(G \vee.\ H) = [G, H]$
| *componentes* $(\neg.\ (G \wedge.\ H)) = [\neg.\ G, \neg.\ H]$
| *componentes* $(G \rightarrow.\ H) = [\neg.G,\ H]$

Para acceder a las componentes de una fórmula definimos las funciones $Comp_1$ y $Comp_2$.

Definición 4.2.15 *La primera componente de la fórmula F es* $Comp_1(F)$.

Su formalización es:

definition *Comp1* :: *'b formula1* $\Rightarrow$ *'b formula1* **where**
Comp1 F = *hd* (*componentes F*)

Definición 4.2.16 *La segunda componente de la fórmula F es* $Comp_2(F)$.

Su formalización es:

definition *Comp2* :: *'b formula1* $\Rightarrow$ *'b formula1* **where**
Comp2 F = *hd* (*tl* (*componentes F*))

Nota 4.2.17 *Las componentes de una fórmula de tipo alfa se denotan* α_1 *y* α_2 *respectivamente; de la misma forma, las componentes de una fórmula de tipo beta se denotan* β_1 *y* β_2 *respectivamente.*

4.3 Disyunciones y conjunciones generalizadas

Para facilitar el desarrollo de las formas normales, introducimos las disyunciones y conjunciones generalizadas.

Definición 4.3.1 *La **disyunción generalizada** de las fórmulas $X_1, X_2, \ldots, X_n$ es $X_1 \vee X_2 \vee \cdots \vee X_n$ y se representa por $[X_1, X_2, \ldots, X_n]$.*

En Isabelle representaremos una disyunción generalizada por medio de una lista de fórmulas: *'b formula1 list*.

Definición 4.3.2 *La **conjunción generalizada** de las fórmulas $X_1, X_2, \ldots, X_n$ es $X_1 \wedge X_2 \wedge \cdots \wedge X_n$ y se representa por $\langle X_1, X_2, \ldots, X_n \rangle$.*

En el estudio que sigue estamos interesados en considerar únicamente conjunciones generalizadas de disyunciones generalizadas.

En Isabelle representaremos una conjunción generalizada de disyunciones generalizadas por medio de una lista de elementos del tipo disyunción generalizada: (*'b formula1 list*) *list*.

4.3.1 Semántica de las disyunciones y conjunciones generalizadas

En esta sección extendemos la semántica para incluir las disyunciones y conjunciones generalizadas.

Definición 4.3.3 *El **valor** de una disyunción generalizada $[X_1, X_2, \ldots, X_n]$ en una interpretación I se define como sigue:*

$$I'[X_1, X_2, \ldots, X_n] = \begin{cases} \mathsf{V}, & \text{si } I'(X_i) = \mathsf{V} \text{ para algún } i \\ \mathsf{F}, & \text{en caso contrario.} \end{cases}$$

Su formalización es:

primrec *valorDisyuncionG* :: (*'b* ⇒ *v-verdad1*) ⇒ (*'b formula1 list*) ⇒ *v-verdad1* **where**
valorDisyuncionG I [] = *Falso*
| *valorDisyuncionG I* (*F*#*Fs*) = (*if valor1 I F* = *Verdad then Verdad else valorDisyuncionG I Fs*)

Definición 4.3.4 *El **valor** de una conjunción generalizada de disyunciones generalizadas $\langle D_1, D_2, \ldots, D_n \rangle$, en una interpretación I, se define como sigue:*

$$I'\langle D_1, D_2, \ldots, D_n \rangle = \begin{cases} \mathsf{V}, & \text{si } I'(D_i) = \mathsf{V} \text{ para todo } i \\ \mathsf{F}, & \text{en caso contrario.} \end{cases}$$

Su formalización es:

primrec *valorConjuncionG* :: (*'b* ⇒ *v-verdad1*) ⇒ (*'b formula1 list*) *list* ⇒ *v-verdad1* **where**
valorConjuncionG I [] = *Verdad*
| *valorConjuncionG I* (*D*#*Ds*) =
(*if valorDisyuncionG I D* = *Falso then Falso else valorConjuncionG I Ds*)

Concluimos esta sección definiendo el concepto de equivalencia ente conjunciones generalizadas y probando las equivalencias entre las fórmulas y conjunciones obtenidas a partir de sus componentes.

Definición 4.3.5 *Las conjunciones generalizadas C_1 y C_2 son **equivalentes** si para toda interpretación I el valor de C_1 en I es igual al valor de C_2 en I, y se representa por $C_1 \equiv C_2$.*

Su formalización es:

definition *equivalentesG* :: (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* ⇒ *bool* **where**
equivalentesG C1 C2 ≡ (∀ *I*. ((*valorConjuncionG I C1*) = (*valorConjuncionG I C2*)))

Se tienen las siguientes equivalencias con relación a las fórmulas alfa, beta y dobles negaciones, y sus respectivas componentes.

Lema 4.3.6 *Si F es una doble negación y G es su componente, entonces $\langle [F] \rangle \equiv \langle [G] \rangle$.*

Su formalizacion es

lemma *EquiNoNo*:
assumes *tipoFormula F* = *NoNo*
shows *equivalentesG* [[*F*]] [[*Comp1 F*]]

Lema 4.3.7 *Si F es una fórmula alfa y sus componentes son F_1 y F_2, $\langle [F] \rangle \equiv \langle [F_1], [F_2] \rangle$.*

Su formalización es:

lemma *EquiAlfa*:
assumes *tipoFormula F* = *Alfa*
shows *equivalentesG* [[*F*]] [[*Comp1 F*],[*Comp2 F*]]

Lema 4.3.8 *Si F es una fórmula beta y sus componentes son F_1 y F_2, $\langle [F] \rangle \equiv \langle [F_1, F_2] \rangle$.*

Su formalización es:

lemma *EquiBeta*:
assumes *tipoFormula F* = *Beta*
shows *equivalentesG* [[*F*]] [[*Comp1 F*, *Comp2 F*]]

4.3.2 Equivalencia entre fórmulas

El concepto de equivalencia también se aplica a fórmulas.

Definición 4.3.9 *Dos fórmulas F y G son equivalentes si $I'(F) = I'(G)$ para toda interpretación I, y se representa por $F \equiv G$.*

Su formalización es:

definition *equivalentes*:: *'b formula1* $\Rightarrow$ *'b formula1* $\Rightarrow$ *bool* **where**
equivalentes F G $\equiv$ $(\forall\ I.\ (valor1\ I\ F) = (valor1\ I\ G))$

Se tienen las siguientes equivalencias entre las fórmulas alfa, beta y las dobles negaciones, y sus respectivas componentes.

Lema 4.3.10 *Si F es una doble negación y G es su componente, entonces $F \equiv G$.*

Su formalización es:

lemma *EquivNoNoComp*:
assumes *tipoFormula F* $=$ *NoNo*
shows *equivalentes F* $(Comp1\ F)$

Lema 4.3.11 *Si F es una fórmula alfa y sus componentes son F_1 y F_2, entonces $F \equiv F_1 \wedge F_2$.*

Su formalización es:

lemma *EquivAlfaComp*:
assumes *tipoFormula F* $=$ *Alfa*
shows *equivalentes F* $(Comp1\ F \wedge .\ Comp2\ F)$

Lema 4.3.12 *Si F es una fórmula beta y sus componentes son F_1 y F_2, entonces $F \equiv F_1 \vee F_2$.*

Su formalización es:

lemma *EquivBetaComp*:
assumes *tipoFormula F* $=$ *Beta*
shows *equivalentes F* $(Comp1\ F \vee .\ Comp2\ F)$

4.4 Descripción de la formalización del algoritmo *FNC*

Algunos algoritmos utilizados en la prueba automática de teoremas requieren como entrada fórmulas proposicionales expresadas en forma normal conjuntiva.

En esta sección presentamos una formalización de la terminación y corrección del algoritmo que se describe en el texto de Fitting ([14], página 30), para determinar una forma normal conjuntiva de una fórmula proposicional. Dicho algoritmo está descrito en términos de los conceptos de *fórmulas α* y *fórmulas β* presentados en la sección anterior.

En lo que sigue definimos la función *FNC* que formaliza en Isabelle el algoritmo para hallar una forma normal conjuntiva de una fórmula proposicional. El algoritmo está descrito en términos de los conceptos de *disyunción generalizada* y *conjunción generalizada*. En este sentido el algoritmo permite hallar de manera más general una "forma normal conjuntiva de una conjunción generalizada".

Comenzamos formalizando los conceptos de cláusula (4.4.1), fórmula en forma normal conjuntiva (4.4.2), forma normal conjuntiva de una conjunción generalizada (4.4.4) y forma normal conjuntiva de una fórmula (4.4.5).

Definición 4.4.1 *Una **cláusula** es una disyunción generalizada* $[X_1, X2, \ldots, X_n]$ *tal que cada* X_i *es un literal.*

Su formalización es

primrec *esClausula* :: $('b\ formula1\ list) \Rightarrow bool$ **where**
esClausula $[] = True$
| *esClausula* $(F\#Fs) = (FormulaLiteral\ F \wedge esClausula\ Fs)$

Definición 4.4.2 *Una fórmula proposicional está en **forma normal conjuntiva** si es una conjunción generalizada* $\langle D_1, D_2, \ldots, D_n \rangle$ *tal que cada* D_i *es una cláusula.*

Su formalización es

primrec *esta-en-FNC* :: $('b\ formula1\ list)\ list \Rightarrow bool$ **where**
esta-en-FNC $[] = True$
| *esta-en-FNC* $(D\#Ds) = (esClausula\ D \wedge esta\text{-}en\text{-}FNC\ Ds)$

Nota 4.4.3 *En lo que sigue llamaremos fórmulas proposicionales a las fórmulas proposicionales distintas de conjunciones y disyunciones generalizadas; para referirnos a estas últimas lo haremos explícitamente.*

Definición 4.4.4 *La conjunción generalizada C_1 es una **forma normal conjuntiva** de la conjunción generalizada C_2 si C_1 está en forma normal conjuntiva y es equivalente a C_2.*

Su formalización es

definition *es-FNCG* :: (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* ⇒ *bool* **where**
es-FNCG C1 C2 ≡ *esta-en-FNC C1* ∧ *equivalentesG C2 C1*

Definición 4.4.5 *La conjunción generalizada C es una forma normal conjuntiva de la fórmula proposicional F si C es una forma normal conjuntiva de* $\langle [F] \rangle$.

Su formalización es

definition *es-FNC* :: (*'b formula1 list*) *list* ⇒ *'b formula1* ⇒ *bool* **where**
es-FNC C F ≡ *es-FNCG C* [[*F*]]

4.5 Reglas de reescritura

En esta sección formalizamos las transformaciones utilizadas en el algoritmo para hallar una forma normal conjuntiva de una conjunción generalizada cualquiera, descritas en [14]. Estas transformaciones se llaman **reglas de reescritura** o **de reducción**.

Sea S una conjunción generalizada, D una disyunción generalizada y N una fórmula, tales que D es un elemento de S y N es un elemento de D. Las reglas de reducción son las siguientes:

- Regla de la doble negación: si N es de la forma $\neg\neg Z$, se puede reemplazar N por Z en D.
- Regla beta: si N es una fórmula β, se puede reemplazar N por β_1 y β_2 en D.
- Regla alfa: si N es una fórmula α, entonces D puede ser reemplazada en S por las dos disyunciones D_1 y D_2, en donde D_1 resulta de D reemplazando α por α_1, y D_2 resulta de D reemplazando α por α_2.

Para la formalización de estas reglas introducimos las siguientes funciones auxiliares.

Definición 4.5.1 (*elemento x ls*) *es la primera lista de la lista de listas ls a la que pertenece x y es* [] *si x no pertenece a ningún elemento de ls. Por ejemplo, elemento 1* [[2],[3,1],[1,5]] = [3,1].

Su formalización es

primrec *elemento* :: *'a ⇒ 'a list list ⇒ 'a list* **where**
elemento x [] = []
| *elemento x* (*l#ls*) = (*if* (*x ∈ set l*) *then l else* (*elemento x ls*))

Definición 4.5.2 (*borrar x ls*) *es la lista obtenida borrando la primera ocurrencia de x en ls. Por ejemplo, borrar 3* [2,3,5,3] = [2,5,3] *y borrar 7* [2,3,5,3] = [2,3,5,3].

Su formalización es

primrec *borrar*:: *'a ⇒ 'a list ⇒ 'a list* **where**
borrar x [] = []
| *borrar x* (*y#xs*) = (*if x=y then xs else y # borrar x xs*)

Definición 4.5.3 (*sustituir1 x y zs*) *es la lista obtenida al sustituir la primera ocurrencia de x en zs por y. Por ejemplo, sustituir1 2 3* [5,2,7,2] = [5,3,7,2] *y sustituir1 4 3* [5,2,7,2] = [5,2,7,2].

Su formalización es

primrec *sustituir1*:: *'a ⇒ 'a ⇒ 'a list ⇒ 'a list* **where**
sustituir1 x y [] = []
| *sustituir1 x y* (*z#xs*) = (*if x=z then* (*y#xs*) *else z #* (*sustituir1 x y xs*))

Definición 4.5.4 (*reemplazarDN x y l*) *es la lista obtenida sustituyendo la primera ocurrencia de x por y, en el primer elemento de l que contenga a x. Por ejemplo, reemplazarDN 2 3* [[5],[4,2,7,2],[2,6]] = [[4,3,7,2],[5],[2,6]].

Su formalización es

fun *reemplazarDN* :: *'a ⇒ 'a ⇒ 'a list list ⇒ 'a list list* **where**
reemplazarDN x y l = sustituir1 x y (*elemento x l*) # (*borrar* (*elemento x l*) *l*)

Definición 4.5.5 (*reemplazarAlfa x y z l*) *es la lista obtenida duplicando el primer elemento de l que contenga a x, reemplazando x por y en la primera copia y x por z en la segunda copia. Por ejemplo, reemplazarAlfa 2 3 9* [[5],[4,2,7,2],[2,6]] = [[4,3,7,2],[4,9,7,2],[5],[2,6]].

Su formalización es

fun *reemplazarAlfa* :: *'a ⇒ 'a ⇒ 'a ⇒ 'a list list ⇒ 'a list list* **where**
reemplazarAlfa x y z l =
sustituir1 x y (*elemento x l*) #
sustituir1 x z (*elemento x l*) #
(*borrar* (*elemento x l*) *l*)

Definición 4.5.6 (*sustituir2 x y w zs*) *es la lista obtenida al sustituir la primera ocurrencia de x en zs por los elementos y y w. por y seguido de w. Por ejemplo, sustituir2 2 3 6* [5,2,7,2] = [5,3,6,7,2].

Su formalización es

```
primrec sustituir2 :: 'a ⇒ 'a ⇒ 'a ⇒ 'a list ⇒ 'a list where
 sustituir2 x y w [] = []
| sustituir2 x y w (z#xs) = (if x=z then (y#w#xs) else z # (sustituir2 x y w xs))
```

Definición 4.5.7 (*reemplazarBeta x y z l*) *es la lista obtenida sustituyendo x por los elementos y y z en el primer elemento de l que contenga a x. Por ejemplo,*
reemplazarBeta 2 3 9 [[5],[4,2,7,2],[2,6]] = [[4,3,9,7,2],[5],[2,6]].

Su formalización es

```
fun reemplazarBeta :: 'a ⇒ 'a ⇒ 'a ⇒ 'a list list ⇒ 'a list list where
 reemplazarBeta x y z l =
  sustituir2 x y z (elemento x l) # (borrar (elemento x l) l)
```

Con estas funciones auxiliares podemos formalizar las reglas como se muestra a continuación.

Definición 4.5.8 (*reglaNoNo F S*) *es el resultado de aplicar la regla de la doble negación a la fórmula F y a la conjunción generalizada S; es decir, si F es de la forma ¬¬Z, se reemplaza F por Z en el primer elemento de S que contenga a F.*

Su formalización es

```
fun reglaNoNo :: 'b formula1 ⇒ ('b formula1 list) list ⇒ ('b formula1 list) list where
 reglaNoNo F S = reemplazarDN F (Comp1 F) S
```

Definición 4.5.9 (*reglaAlfa F S*) *es el resultado de aplicar la regla alfa a la fórmula F y a la conjunción generalizada S; es decir, si F es una fórmula α y D es el primer elemento de S que contiene a F, se reemplaza D por las dos disyunciones D_1 y D_2, en donde D_1 resulta de D reemplazando α por α_1, y D_2 resulta de D reemplazando α por α_2.*

Su formalización es

```
fun reglaAlfa:: 'b formula1 ⇒ ('b formula1 list) list ⇒ ('b formula1 list) list where
 reglaAlfa F S = reemplazarAlfa F (Comp1 F) (Comp2 F) S
```

Definición 4.5.10 (*reglaBeta F S*) *es el resultado de aplicar la regla beta a la fórmula F y a la conjunción generalizada S; es decir, si F es una fórmula β y D es el primer elemento de S que contiene a F, se reemplaza F por* β_1 *y* β_2 *en D.*

Su formalización es

fun *reglaBeta*:: *'b formula1* ⇒ (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* **where**
reglaBeta F S = *reemplazarBeta F* (*Comp1 F*) (*Comp2 F*) *S*

4.6 Algoritmo para hallar una forma normal conjuntiva

En esta sección formalizamos el algoritmo de cálculo de la forma normal conjuntiva, cuya especificación es la siguiente:

Entrada: Una conjuncion generalizada S.

Salida: Una forma normal conjuntiva Y de S.

Procedimiento:

Mientras que algún elemento de S contenga un no literal hacer

- Seleccionar un elemento D de S que contenga un no literal.
- Seleccionar un no literal N de D.
- Aplicar a N en D la regla de reducción apropiada.

Devolver $Y = S$

Obsérvese que para el caso particular $S{=}\langle[X]\rangle$, el algoritmo anterior permite hallar una forma normal conjuntiva de la fórmula proposicional X.

Para su formalización introducimos las siguientes definiciones.

Definición 4.6.1 (*tieneNoLiteralDisyuncion D*) *es Some F si la disyunción generalizada D contiene una fórmula F que no es literal y es None en caso contrario.*

En la definición anterior, si la disyunción D contiene no literales, el valor *(tieneNo-LiteralDisyuncion D)* es *Some F* que representa una fórmula F de la disyuncion D que es un no literal; en caso contrario es la constante *None* que significa que D no contiene literales.

Para su formalización se utiliza el tipo de dato

```
datatype 'b formula1 option = None | Some 'b formula1

primrec tieneNoLiteralDisyuncion :: ('b formula1 list) ⇒ 'b formula1 option where
 tieneNoLiteralDisyuncion [] = None
| tieneNoLiteralDisyuncion (F#D) =
  (if ¬(FormulaLiteral F) then (Some F)
  else tieneNoLiteralDisyuncion D)
```

Definición 4.6.2 *(tieneDisyun-con-NoLiteral S) se verifica si la conjunción generalizada S contiene una diyunción generalizada D que contiene una fórmula que no es literal y es False en caso contrario.*

Su formalización es

```
primrec tieneDisyun-con-NoLiteral:: ('b formula1 list) list ⇒ bool where
 tieneDisyun-con-NoLiteral [] = False
| tieneDisyun-con-NoLiteral (D#S) =
  (case tieneNoLiteralDisyuncion D of
  None ⇒ tieneDisyun-con-NoLiteral S
|  Some F ⇒ True)
```

Definición 4.6.3 *(selecDisyun-con-NoLiteral S) es la primera diyunción generalizada de S que contiene una fórmula que no es literal y es [] en caso contrario.*

Su formalización es

```
primrec selecDisyun-con-NoLiteral:: ('b formula1 list) list ⇒ ('b formula1 list) where
 selecDisyun-con-NoLiteral [] = []
| selecDisyun-con-NoLiteral(D#S) =
  (case tieneNoLiteralDisyuncion D of
  None ⇒ selecDisyun-con-NoLiteral S
|  Some F ⇒ D)
```

Definición 4.6.4 *(selecNoLiteralDisyuncion D) es la primera fórmula no literal de la disyunción generalizada D.*

Su formalización es

```
definition selecNoLiteralDisyuncion:: ('b formula1 list) ⇒ 'b formula1 where
 selecNoLiteralDisyuncion D = (case tieneNoLiteralDisyuncion D of Some F ⇒ F)
```

Definición 4.6.5 *La función FNC formaliza el algoritmo de la forma normal conjuntiva descrito anteriormente.*

Su formalización es

```
function FNC :: ('b formula1 list) list ⇒ ('b formula1 list) list  where
FNC S =
 (if ¬(tieneDisyun-con-NoLiteral S)
  then S
  else ((let D = (selecDisyun-con-NoLiteral S) in
     (let F = (selecNoLiteralDisyuncion D)
      in case tipoFormula F of
          NoNo ⇒ FNC (reglaNoNo F S)
        | Beta ⇒ FNC (reglaBeta F S)
        | Alfa ⇒ FNC (reglaAlfa F S)))))
by pat-completeness auto
```

Así, el caso particular $FNC\ \langle[X]\rangle$, es una forma normal conjuntiva de la fórmula proposicional X.

4.7 Formalización de la terminación de la función FNC

En esta sección formalizamos la terminación de la función FNC. Para demostrar la terminación de una función recursiva usando como medida una relación de orden bien fundamentada, debemos definir una relación R tal que: (a) R sea bien fundamentada y (b) los argumentos de la función en las llamadas recursivas decrezcan con respecto a R.

Una forma usual de definir una relación R binaria sobre un conjunto A, a partir de una relación R' sobre un conjunto B y una función $f\colon A \to B$, es la siguiente:

$$(x, y) \in R \Longleftrightarrow (f(x), f(y)) \in R'$$

R es llamada *la imagen inversa* de R' con respecto a f. En Isabelle está definido este concepto mediante *inv-image* como se muestra a continuación:

```
definition inv-image :: ('b × 'b) set ⇒ ('a ⇒ 'b) ⇒ ('a × 'a) set where
inv-image r f = {(x, y)| (f x, f y) ∈ r}
```

Si la relación R' es bien fundamentada, la función f se llama una *función de medida*. En este caso, la imagen inversa R también es bien fundamentada. Esta propiedad está formalizada en Isabelle como un teorema.

```
wf r ⟹ wf (inv-image r f)                                   (wf-inv-image)
```

Por ejemplo, teniendo en cuenta que la relación "*menor-que*" $<$ sobre el conjunto $\mathbb{N}$ de los números naturales es bien fundamentada, la manera más usual de definir una relación R bien fundamentada sobre un conjunto A consiste en especificar una función de medida de la forma $f\colon A \to \mathbb{N}$, así $(x, y) \in R \Longleftrightarrow f(x) < f(y)$. En Isabelle esta relación particular se denomina *measure f*.

La formalización de que la relación $<$ sobre $\mathbb{N}$ es bien fundamentada utiliza las siguientes definiciones.

definition *pred-nat* :: $(nat \times nat)$ *set* **where**
$pred\text{-}nat = \{(m, n) |\ n = Suc\ m\}$

definition*less-than* :: $(nat \times nat)$ *set* **where**
$less\text{-}than = pred\text{-}nat+$

Se tienen los siguientes resultados acerca de la relación *less-than*.

$((x, y) \in less\text{-}than) = (x < y)$ (*less-than-iff*)

$wf\ less\text{-}than$ (*wf-less-than*)

Usando los anteriores dos teoremas se prueba que $<$ es bien fundamentada.

$wf\ \{(x, y).\ x < y\}$ (*wf-less*)

En algunos casos la prueba de terminación de funciones recursivas se realiza automáticamente por el sistema. Sin embargo, existen funciones recursivas complejas para las que el sistema no puede probar su terminación sin ayuda del usuario, quien debe indicar la relación bien fundamentada y/o la función de medida con la que se ha de hacer la prueba.

Ilustraremos los procedimientos anteriores en la formalización de la terminación de la función *FNC*; para esto definiremos y probaremos la terminación de las siguientes funciones.

- *rango* :: *'b formula1* $\Rightarrow$ *nat*
- *tamano* :: *'b formula1* $\Rightarrow$ *nat*
- *medidaLista* :: *DisyuncionG* $\Rightarrow$ *nat*
- *Mcmedidas* :: *ConjuncionG* $\Rightarrow$ *nat multiset*

4.7.1 La funciones de medida

Definición 4.7.1 *El rango de una fórmula F es*

$$rango(F) = \begin{cases} 0, & \text{si } F \text{ es un literal} \\ 1 + rango(G), & \text{si } F \text{ es } \neg\neg G \\ 1 + rango(F_1) + rango(F_2), & \text{si } F \text{ es una fórmula alfa o beta y sus componentes son } F_1 \text{ y } F_2 \end{cases}$$

Su formalización es

```
function rango :: 'b formula1 ⇒ nat where
rango F =
 (case (tipoFormula F) of
        Literal ⇒ 0
      | NoNo ⇒ rango (Comp1 F) + 1
      | Alfa ⇒ rango (Comp1 F) + rango (Comp2 F) + 1
      | Beta ⇒ rango (Comp1 F) + rango (Comp2 F) + 1)
by auto
```

Para la demostración de la terminación de la funcion *rango* usamos el *tamaño* de las fórmulas, definido por

Definición 4.7.2 *Sean F y G fórmulas, entonces*

- *tamaño(F) = 1 si F es una fórmula atómica,* $\bot$ *o* $\top$
- *tamaño*$(\neg F)$ *= 1 + tamaño(F)*
- *tamaño*$(F \wedge G)$ *= tamaño*$(F \vee G)$ *= tamaño*$(F \rightarrow G)$ *= 1+tamaño(F)+ tamaño(G)*

Su formalización es

```
primrec tamano :: 'b formula1 ⇒ nat where
 tamano FF = 1
| tamano TT = 1
| tamano (Atomo P) = 1
| tamano (¬. F) = (tamano F) + 1
| tamano (F ∧. G) = (tamano F) + (tamano G) + 1
| tamano (F ∨. G) = (tamano F) + (tamano G) + 1
| tamano (F →. G) = (tamano F) + (tamano G) + 1
```

Nótese que el Isabelle prueba automáticamente la terminación de la función *tamaño*.

La terminación de la función *rango* se formaliza usando como función de medida la función *tamaño*.

```
termination rango
proof −
 show ?thesis
 by (relation measure (λformula. tamano formula))
```

Definición 4.7.3 *La función medidaLista aplicada a una lista de fórmulas devuelve la suma de sus rangos.*

Esta función sirve para demostrar la terminación de la función *FNC*.

Su formalización es

primrec *medidaLista* :: (*'b formula1 list*) ⇒ *nat* **where**
medidaLista [] = *0*
| *medidaLista* (*F#Fs*) = *rango F* + *medidaLista Fs*

Para construir la relación bien fundamentada que permita demostrar la terminación de la función *FNC* especificaremos una función de medida que involucra el concepto de *multiconjunto*.

Definición 4.7.4 *Un **multiconjunto** sobre un conjunto U es una función* $m\colon U \to \mathbb{N}$, *donde* $\mathbb{N}$ *es el conjunto de los números naturales.*

Decimos que un multiconjunto m es *finito* si hay un número finito de elementos de U tales que $m(x) > 0$. Notaremos por $M(U)$ al conjunto de todos los multiconjuntos finitos sobre U, $M(U) = \{m\colon U \to \mathbb{N} \mid \{x \mid m(x) > 0\}$ es finito$\}$.

Por ejemplo, el multiconjunto m sobre $\mathbb{N}$ definido por

$$m(n) = \begin{cases} n, & \text{si } n < 4 \\ 0, & \text{en otro caso.} \end{cases}$$

es finito y se representa por $\{\#\ 1,\ 2\ ,2\ ,3\ ,3\ \#\}$. En general, utilizaremos esta última notación para representar multiconjuntos finitos teniendo en cuenta que el orden en que se escriban los elementos no es importante.

Definición 4.7.5 *Sean* $m_1, m_2 \in M(U)$ *y* $x \in U$.

- x **pertenece** *a* m_1, *y lo notaremos* $x \in\# m_1$, *si* $m_1(x) > 0$.
- *El multiconjunto* **vacío**, *que notaremos por* $\{\#\}$, *es la función que a cada* $x \in U$ *le asigna 0.*
- *La* **unión** *de* m_1 *y* m_2, *que notaremos por* $m_1 + m_2$, *es la función que a cada* $x \in U$ *le asigna* $m_1(x) + m_2(x)$.

En Isabelle el tipo de los multiconjuntos finitos está definido como sigue.

typedef *'a multiset* = $\{f::'a \Rightarrow nat \mid$ *finite* $\{x \mid f\, x > 0\}\}$

En [5] aparece una descripción completa de la formalización del tipo de los multiconjuntos finitos.

A continuación, mostramos una forma de extender una relación definida sobre un conjunto U al conjunto $M(U)$ de los multiconjuntos finitos sobre U.

Sea R una relación binaria definida sobre un conjunto U. La relación entre multiconjuntos sobre $M(U)$ inducida por R, notada por *mult R*, es la clasura transitiva de la siguiente relación:

$mult1\ R =$
$\{(N, M) | \exists\, a\, M_0\, K. M = M_0 + \{\#\ a\ \#\} \wedge N = M_0 + K \wedge (\forall\, b.\, b \in\# K \longrightarrow (b, a) \in R)\}$.

Informalmente, $(N, M) \in (mult1\ R)$ si N se puede "obtener" a partir de M y un elemento fijo $a \in\# M$, reemplazando a en M por elementos "menores" que a con respecto a la relación R.

Por ejemplo, consideremos $U = \mathbb{N}$ y R la relación $\leq$. Si $M = \{\#\ 1,\ 1,\ 2,\ 3,\ 3,\ 8,\ 9\ \#\}$ y $a = 3$, al reemplazar 3 por $3, 3, 10, 10$ en M obtenemos $N = \{\#\, 1, 1, 2, 3, 3, 3, 8, 9, 10, 10\, \#\}$. De esta forma,
$M = M_0 + \{\#\ 3\ \#\}$ y $N = M_0 + \{\#\ 3, 3, 10, 10\ \#\}$ donde, $M_0 = \{\#\ 1,\ 1,\ 2,\ 3,\ 8,\ 9\ \#\}$. Por lo tanto, $(N, M) \in (mult1\ \leq)$.

Se tiene el siguiente importante teorema: Si R es bien fundamentada entonces *mult R* es bien fundamentada.

$$wf\ r \Longrightarrow wf\ (mult\ r) \qquad (wf\text{-}mult)$$

De esta forma para la formalización de la terminación de la función FNC hacemos lo siguiente:

(1) Consideramos la relación entre multiconjuntos sobre $\mathbb{N}$ inducida por la relación $<$ sobre $\mathbb{N}$.

definition *relacionmedida*:: $(nat\ multiset \times nat\ multiset)\ set$ **where**
$relacionmedida = mult\ \{(x, y{::}nat).\ x < y\}$

Utilizando el teorema *wf-mult* se demuestra que esta relación es bien fundamentada puesto que la relación $<$ sobre $\mathbb{N}$ es bien fundamentada (teorema *wf-less*).

lemma *wf-relacionmedida*:
shows $wf\ (relacionmedida)$
using *wf-mult wf-less*
by $(unfold\ relacionmedida\text{-}def)$

(2) Definimos la siguiente función de medida.

Definición 4.7.6 *La función Mcmedidas aplicada a una lista de listas de fórmulas devuelve el multiconjunto cuyos elementos son las medidas de las listas, determinadas por la función medidaLista.*

Su formalización es

primrec *Mcmedidas* :: (*'b formula1 list*) *list* ⇒ *nat multiset* **where**
Mcmedidas [] = {#}
| *Mcmedidas* (*D*#*Ds*) = *Mcmedidas Ds* + {# *medidaLista D* #}

El sistema prueba automáticamente la terminación de esta función.

La relación utilizada para la prueba de la terminación de la funcion FNC es la imagen inversa de la relación *relacionmedida* con respecto a la funcion de medida *Mcmedidas*.

definition *medida* :: ((*'b formula1 list*) *list* × (*'b formula1 list*) *list*)*set* **where**
medida = *inv-image relacionmedida Mcmedidas*

Utilizando el teorema *wf-inv-image* se demuestra que esta relación es bien fundamentada puesto que la relación *relacionmedida* es bien fundamentada.

lemma *wf-medida*:
shows *wf* (*medida*)
using *wf-inv-image wf-relacionmedida*
by (*unfold medida-def*)

Los lemas centrales en la formalización de la terminación de la función FNC son los siguientes.

Lema 4.7.7 *Sea S una lista de listas de fórmulas, D un elemento de S y S′ la lista que resulta de eliminar en S una ocurrencia de D. Entonces, Mcmedidas S = Mcmedidas S′ + {# medidaLista D # }.*

Su formalización es

lemma *propiedadMcmedidas*:
assumes *D* ∈ *set S*
shows *Mcmedidas S* = *Mcmedidas* (*borrar D S*) + {# *medidaLista D*#}

Lema 4.7.8 *Sea S una conjunción generalizada de disyunciones generalizadas tal que la fórmula F pertenence a alguna disyunción generalizada D ∈ S. Si F es una doble negación, entonces* $((reglaNoNo\ F\ S), S) \in medida$.

Su formalización es

lemma *medidanono0*:
assumes *hip1*: (*elemento F S*) ≠ [] **and** *hip2*: *FormulaNoNo F*
shows ((*reglaNoNo F S*), *S*) ∈ *medida*

Lema 4.7.9 *Sea S una conjunción generalizada de disyunciones generalizadas tal que la fórmula F pertenece a alguna disyunción generalizada D ∈ S. Si F es una fórmula alfa, entonces* $((reglaAlfa\ F\ S), S) \in medida$.

Su formalización es

lemma *medidaalfa0*:
assumes *hip1*: $(elemento\ F\ S) \neq []$ **and** *hip2*: *FormulaAlfa F*
shows $((reglaAlfa\ F\ S), S) \in medida$

Lema 4.7.10 *Sea S una conjunción generalizada de disyunciones generalizadas tal que la fórmula F pertenece a alguna disyunción generalizada $D \in S$. Si F es una fórmula beta, entonces $((reglaBeta\ F\ S), S) \in medida$.*

Su formalización es

lemma *medidabeta0*:
assumes *hip1*: $(elemento\ F\ S) \neq []$ **and** *hip2*: *FormulaBeta F*
shows $((reglaBeta\ F\ S), S) \in medida$

Usando los tres lemas anteriores, tenemos los siguientes tres teoremas que permiten demostrar que los argumentos de la función FNC decrecen en las llamadas recursivas, con respecto a la relación *medida*.

Teorema 4.7.11 *Supongamos que S es una conjunción generalizada, de disyunciones generalizadas, que contiene una disyunción generalizada D con no literales. Sea $F \in D$ un no literal. Si F es una doble negación, entonces $((reglaNoNo\ F\ S), S) \in medida$.*

Su formalización es

theorem *medidanono*:
assumes *hip1*: *tieneDisyun-con-NoLiteral S* **and**
hip2: *tipoFormula* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*))
= *NoNo*
shows
((*reglaNoNo* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*),
S) ∈ *medida*
proof −
have *selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*) ∈ *set*
(*elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*)
using *hip1 pertenenciaFundamental1* **by** *auto*
hence *elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*
≠ [] **by** *auto*
thus *?thesis* **using** *hip2 tipoNoNo* **by**(*rule-tac medidanono0*)
qed

Teorema 4.7.12 *Supongamos que S es una conjunción generalizada, de disyunciones generalizadas, que contiene una disyunción generalizada D con no literales. Sea $F \in D$ un no literal. Si F es una fórmula alfa, entonces $((reglaAlfa\ F\ S), S) \in medida$.*

Su formalización es

theorem *medidaalfa*:
assumes
hip1: *tieneDisyun-con-NoLiteral S* **and**
hip2: *tipoFormula* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) =
Alfa
shows
((*reglaAlfa* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*), *S*)
∈ *medida*
proof −
have *selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*) ∈ *set*
(*elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*)
using *hip1 pertenenciaFundamental1* **by** *auto*
hence *elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*
≠ [] **by** *auto*
thus *?thesis* **using** *hip2 tipoAlfa* **by**(*rule-tac medidaalfa0*)
qed

Teorema 4.7.13 *Supongamos que S es una conjunción generalizada, de disyunciones generalizadas, que contiene una disyunción generalizada D con no literales. Sea F ∈ D un no literal. Si F es una fórmula beta, entonces* $((reglaBeta\ F\ S), S) \in medida$.

Su formalización es

theorem *medidabeta*:
assumes
hip1: *tieneDisyun-con-NoLiteral S* **and**
hip2: *tipoFormula* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) =
Beta
shows
((*reglaBeta* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*), *S*) ∈
medida
proof −
have *selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*) ∈ *set*
(*elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*)
using *hip1 pertenenciaFundamental1* **by** *auto*
hence *elemento* (*selecNoLiteralDisyuncion* (*selecDisyun-con-NoLiteral S*)) *S*
≠ [] **by** *auto*
thus *?thesis* **using** *hip2 tipoBeta* **by**(*rule-tac medidabeta0*)
qed

Por último, usando estos teoremas tenemos la prueba de la terminación de la función FNC.

termination *FNC*
proof −
show *?thesis* **using** *wf-medida medidaalfa medidabeta medidanono*
by (*relation medida*, *auto*)
qed

4.8 Corrección del algoritmo FNC

Puesto que el significado pretendido de la definición de la función FNC es, dada una conjunción generalizada S, $FNC\ S$ es una forma normal conjuntiva de S; para demostrar su *corrección*, de acuerdo a la definición 4.4.4, debemos probar que:

1. $FNC\ S$ está en forma normal conjuntiva.
2. $FNC\ S$ es equivalente a S.

La formalización de que $FNC\ S$ está en forma normal conjuntiva, se desarrolla de la siguiente forma:

Lema 4.8.1 *Supongamos que la conjunción generalizada S, de disyunciones generalizadas, no contiene disyunciones con no literales. Entonces S está en forma normal conjuntiva.*

Su formalización es

lemma *estaFNC0*:
assumes ¬ *tieneDisyun-con-NoLiteral S*
shows *esta-en-FNC S*

Lema 4.8.2 *Supongamos que la conjunción generalizada S, de disyunciones generalizadas, contiene alguna disyunción con no literales, entonces FNC S no contiene disyunciones con no literales.*

Su formalización es

lemma *FNC2*:
assumes *tieneDisyun-con-NoLiteral S*
shows ¬ *tieneDisyun-con-NoLiteral* (*FNC S*)

Lema 4.8.3 *FNC S está en forma normal conjuntiva.*

Su formalización es

lemma *estaFNC*: *esta-en-FNC* (*FNC S*)
using *FNC2 estaFNC0*
by *auto*

Para demostrar que S es equivalente a $FNC\ S$, se demuestran en primer lugar los siguientes lemas:

Lema 4.8.4 *Si S es una conjunción generalizada de disyunciones generalizadas que contiene disyunciones con no literales, entonces S y FNC S son equivalentes.*

Su formalización es

lemma *Equi2*:
assumes *tieneDisyun-con-NoLiteral S*
shows *equivalentesG S* (*FNC S*)

Lema 4.8.5 *Si S es una conjunción generalizada de disyunciones generalizadas que no contiene disyunciones con no literales, entonces* $FNC\ S = S$.

Su formalización es

lemma *FNC4*:
assumes ¬ *tieneDisyun-con-NoLiteral S*
shows *FNC S* = *S*

Lema 4.8.6 *Sea S una conjunción generalizada de disyunciones generalizadas y* S' *la conjunción obtenida aplicándole el algoritmo FNC a S. Entonces S y* S' *son equivalentes.*

Su formalización es

lemma *equivalentesGS-FNC*: *equivalentesG S* (*FNC S*)
proof (*cases*)
assume *tieneDisyun-con-NoLiteral S*
thus *?thesis* **using** *Equi2* **by** *blast*
next
assume ¬ *tieneDisyun-con-NoLiteral S*
hence *FNC S* = *S* **using** *FNC4* **by** *simp*
hence *S* = *FNC S* **by** *simp*
thus *equivalentesG S* (*FNC S*)
by (*unfold equivalentesG-def*, *simp*)
qed

Usando los lemas *estaFNC* y *equivalentesGS-FNC* el siguiente teorema nos garantiza la correción de la función FNC.

Teorema 4.8.7 *Para cualquier conjunción generalizada S, de disyunciones generalizadas, FNC S es una forma normal conjuntiva de S.*

Su formalización es

theorem *CorreccionFNCG*: *es-FNCG* (*FNC S*) *S*
using *estaFNC equivalentesGS-FNC*
by (*unfold es-FNCG-def*, *blast*)

En particular, tenemos el siguiente corolario.

Corolario 4.8.8 *Para cualquier fórmula F, FNC* $\langle [F] \rangle$ *es una forma normal conjuntiva de F.*

Su formalización es

corollary *CorreccionFNC*: *es-FNC* (*FNC* [[*F*]]) *F*
using *CorreccionFNCG*
by(*unfold es-FNC-def*)

Capítulo 5

Método de prueba basado en tableros semánticos

En este capítulo presentamos una formalización de un procedimiento de prueba para la lógica proposicional basado en la noción de *tableros semánticos*. Describimos el algoritmo, de la misma manera como se hizo en el capítulo anterior para formalizar el algoritmo FNC, en términos de los conceptos de *fórmulas* α y *fórmulas* β, y demostramos su terminación y corrección siguiendo la exposición del texto de Fitting [14].

5.1 Descripción de la formalización de la deducibilidad mediante tableros semánticos

El método de prueba por tableros es un sistema de pruebas por *contradicción*. Para probar una fórmula F, debemos obtener una contradicción a partir de $\neg F$. El método se basa en la construcción de un árbol (tablero), en donde los nodos se etiquetan con fórmulas, cada rama representa la conjunción de las fórmulas que aparecen en ella y el árbol representa la disyunción de sus ramas.

En esta sección definimos la función *PruebaTablero* que formaliza en Isabelle un procedimiento de prueba usando tableros semánticos.

5.1.1 Definición de tablero semántico

Consideremos un árbol finito T en donde sus nodos están etiquetados con fórmulas proposicionales, sea R una rama de T y N una fórmula (nodo) de R. Entonces, a partir de T, podemos obtener un nuevo árbol T' si aplicamos una de las siguientes reglas:

- Si N es de la forma $\neg\neg Z$, T' se obtiene a partir de T añadiendo al final de la rama R un nuevo nodo etiquetado con la fórmula Z.
- Si N es una fórmula α, T' se obtiene a partir de T añadiendo al final de la rama R un nodo con etiqueta α_1 y posteriormente adicionamos otro nodo con etiqueta α_2.
- Si N es una fórmula β, T' se obtiene a partir de T añadiendo al final de la rama R un hijo izquierdo con etiqueta β_1 y un hijo derecho con etiqueta β_2.

En este caso decimos que T' se obtuvo a partir de T por la aplicación de una regla de tableros.
Representaremos en Isabelle los árboles finitos (tableros) y ramas por medio de listas; una rama estará representada por la lista de las fórmulas que etiquetan sus nodos, *rama* = *'b formula1 list* y un árbol estará representado por una lista de sus ramas, *árbol* = *rama list*. Las siguientes funciones formalizan las reglas de tableros para las fórmulas de tipo Alfa, Beta y dobles negaciones.

fun *reglaTNoNo* :: *'b formula1* ⇒ (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* **where**
reglaTNoNo F T = *reemplazarDN F* (*Comp1 F*) *T*

fun *reglaTBeta*:: *'b formula1* => (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* **where**
reglaTBeta F T = *reemplazarAlfa F* (*Comp1 F*) (*Comp2 F*) *T*

fun *reglaTAlfa*:: *'b formula1* ⇒ (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* **where**
reglaTAlfa F T = *reemplazarBeta F* (*Comp1 F*) (*Comp2 F*) *T*

La siguiente definición establece la noción de *tablero* para un conjunto finito de fórmulas.

Definición 5.1.1 *Sea* $\{A_1, A_2, \ldots, A_n\}$ *un conjunto de fórmulas proposicionales.*
1. El siguiente árbol de una sola rama es un tablero (inicial) para $\{A_1, A_2, \ldots, A_n\}$*:*

$$A_1$$
$$\vdots$$
$$A_n$$

2. Si T es un tablero para $\{A_1, A_2, \ldots, A_n\}$ *y T′ se obtiene a partir de T mediante la aplicación de una regla de tableros, entonces T′ es un tablero para* $\{A_1, A_2, \ldots, A_n\}$*.*

A continuación definimos tres tipos especiales de tableros: estrictos (5.1.2), completos (5.1.3) y cerrados (5.1.5).

Definición 5.1.2 *Un tablero T es* **estricto** *si en su construcción a ninguna fórmula se le ha aplicado una regla de tableros dos veces sobre la misma rama.*

Definición 5.1.3 *Un tablero T es* **completo** *si no es posible obtener a partir de él un tablero estricto T' mediante la aplicación de una de las reglas de tableros.*

Definición 5.1.4 *Una rama R de un tablero T está* **cerrada** *si existe una fórmula F tal que F y $\neg F$ son nodos de R o bien R contiene a $\bot$ o contiene a $\neg\top$.*

Para su formalización definimos *Ramacerrada1* tal que *(Ramacerrada1 F R)* se verifica si R contiene a F y $\neg F$, o contiene a $\bot$ o contiene a $\neg\top$.

definition *Ramacerrada1*:: *'b formula1* ⇒ *'b formula1 list* ⇒ *bool* **where**
Ramacerrada1 F R = ((*F* ∈ *set R* ∧ (¬. *F*) ∈ *set R*) ∨ *FF* ∈ *set R* ∨ ¬.*TT* ∈ *set R*)

La formalización de las ramas cerradas es:

primrec *Ramacerrada-aux*:: (*'b formula1 list*) ⇒ (*'b formula1 list*) ⇒ *bool*
where
Ramacerrada-aux [] *L* = *False*
| *Ramacerrada-aux* (*x*#*xs*) *L* = ((*Ramacerrada1 x L*) ∨ (*Ramacerrada-aux xs L*))

definition *Ramacerrada*:: (*'b formula1 list*) ⇒ *bool* **where**
Ramacerrada R = *Ramacerrada-aux R R*

El siguiente lema prueba que *Ramacerrada* cumple las condiciones de la definición 5.1.4.

theorem *Rama1*:
assumes *Ramacerrada R*
shows ∃ *F*. (*F* ∈ *set R* ∧ (¬.*F*) ∈ *set R*) ∨ *FF* ∈ *set R* ∨ ¬.*TT* ∈ *set R*

Definición 5.1.5 *Un tablero T es* **cerrado** *si todas sus ramas están cerradas.*

Su formalización es

primrec *TableroCerrado*:: (*'b formula1 list*) *list* ⇒ *bool* **where**
TableroCerrado [] = *True*
|*TableroCerrado* (*R*#*Rs*) = (*Ramacerrada R* ∧ *TableroCerrado Rs*)

5.2 Sistema de prueba por tableros

El concepto de tablero semántico permite establecer un procedimiento para decididir satisfacibilidad, y por lo tanto validez, en la lógica proposicional. El objetivo de esta sección es formalizar este método y demostrar su adecuación.

Definición 5.2.1 *Sea S un conjunto de fórmulas. Una fórmula F es* **deducible por tableros** *a partir de S si existe un tablero cerrado para $S \cup \{\neg F\}$.*

En particular tenemos la siguiente definición.

Definición 5.2.2 *Una fórmula F tiene una* **prueba por tableros** *si existe un tablero cerrado para $\{\neg F\}$. F es un* **teorema** *del sistema deductivo por tableros si F tiene una prueba por tableros.*

El sistema deductivo por tableros es correcto y completo; es decir, F es deducible por tableros a partir de S si y sólo si F es consecuencia lógica de S. Más específicamente, F es consecuencia lógica de S si y sólo si existe un tablero completo cerrado para $S \cup \{\neg F\}$; además, si existe un tablero completo cerrado para un conjunto S, cualquier otro tablero completo para S es cerrado.

De esta forma, un método que construya un tablero completo para un conjunto de fórmulas constituye un procedimiento de prueba por tableros, basta con comprobar si dicho tablero completo está cerrado o no.

El siguiente algoritmo construye un tablero completo para un conjunto finito de fórmulas.

Entrada: Un conjunto de fórmulas S.

Salida: Un tablero completo T' para S.

Procedimiento:

Hacer T = la rama cuyos nodos están etiquetados con las fórmulas de S

Mientras que alguna rama de T contenga un nodo con etiqueta una fórmula no literal y no marcada hacer lo siguiente:

seleccionar una rama R de T que contenga un nodo etiquetado con una fórmula no literal y no marcada.

seleccionar un nodo de R etiquetado con una fórmula N no literal y no marcada.

marcar y aplicar a N en R la regla de tableros apropiada para obtener un nuevo tablero T.

Devolver T'=T

Usaremos las siguientes definiciones para formalizar en Isabelle el algoritmo anterior. Las correspondientes formalizaciones corresponden a las dadas en el capítulo 4 (sección 4.6).

Definición 5.2.3 *(tieneNoLiteralRama R) es Some F si la rama R contiene una fórmula F que no es literal y es None en caso contrario.*

Su formalización es:

definition *tieneNoLiteralRama*:: *'b formula1 list* ⇒ *'b formula1 option* **where**
tieneNoLiteralRama = *tieneNoLiteralDisyuncion*

Definición 5.2.4 *(tieneRama-con-NoLiteral T) se verifica si el tablero T contiene una rama R que contiene una fórmula que no es literal y es False en caso contrario.*

Su formalización es:

definition *tieneRama-con-NoLiteral*:: (*'b formula1 list*) *list* ⇒ *bool* **where**
tieneRama-con-NoLiteral = *tieneDisyun-con-NoLiteral*

Definición 5.2.5 *(selecRama-con-NoLiteral T) es la primera rama de T que contiene una fórmula que no es literal y es* [] *en caso contrario.*

Su formalización es:

definition *selecRama-con-NoLiteral*:: (*'b formula1 list*) *list* ⇒ *'b formula1 list* **where**
selecRama-con-NoLiteral = *selecDisyun-con-NoLiteral*

Definición 5.2.6 *(selecNoLiteralRama R) es la primera fórmula no literal de la rama R.*

Su formalización es:

definition *selecNoLiteralRama*:: *'b formula1 list* ⇒ *'b formula1* **where**
selecNoLiteralRama = *selecNoLiteralDisyuncion*

La siguiente función *PruebaTablero* formaliza el algoritmo descrito anteriormente para construir un tablero completo para un conjunto S de fórmulas.

El argumento de la función *PruebaTablero* será, en general, un tablero cualquiera. De esta forma, *PruebaTablero T* será un tablero completo construido a partir del tablero T. Así, al representar un conjunto finito de fórmulas S por medio de una lista, se tiene que *PruebaTablero* $[S]$ es un tablero completo para el conjunto de fórmulas S (construido a partir del tablero inicial $[S]$).

```
function PruebaTablero :: ('b formula1 list) list ⇒ ('b formula1 list) list  where
PruebaTablero T  =
(if ¬(tieneRama-con-NoLiteral T)
  then T
  else ((let R = (selecRama-con-NoLiteral T) in
     (let Noliteral = (selecNoLiteralRama R)
      in case tipoFormula Noliteral of
          NoNo ⇒ PruebaTablero (reglaTNoNo Noliteral T)
        | Beta ⇒ PruebaTablero (reglaTBeta Noliteral T)
        | Alfa ⇒ PruebaTablero (reglaTAlfa Noliteral T)))))
by pat-completeness auto
```

5.2.1 Terminación de la función *PruebaTablero*

En esta sección formalizamos la terminación de la función *PruebaTablero*. Usamos las mismas funciones, *rango* (def. 4.7.1), *tamaño* (def. 4.7.2), *medidaLista* (def. 4.7.3) y *Mcmedidas* (def. 4.7.6) que permitieron demostrar en el capítulo 4 la terminación de la función FNC, y los siguientes resultados.

Lema 5.2.7 *Sea T un tablero tal que la fórmula F pertenence a alguna rama $R \in T$. Si F es una doble negación, entonces $((reglaTNoNo\ F\ T), T) \in medida$.*

Su formalización es

```
lemma medidanonoT0:
 assumes hip1: (elemento F T) ≠ []
 and hip2: FormulaNoNo F
 shows ((reglaTNoNo F T), T) ∈ medida
```

Lema 5.2.8 *Sea T un tablero tal que la fórmula F pertenece a alguna rama $R \in T$. Si F es una fórmula beta, entonces $((reglaTBeta\ F\ T), T) \in medida$.*

Su formalización es:

```
lemma medidabetaT0:
 assumes hip1: (elemento F T) ≠ []
```

and *hip2*: *FormulaBeta F*
shows $((reglaTBeta\ F\ T), T) \in medida$

Lema 5.2.9 *Sea T un tablero tal que la fórmula F pertenece a alguna rama R ∈ T. Si F es una fórmula alfa, entonces* $((reglaTAlfa\ F\ T), T) \in medida$.

Su formalización es:

lemma *medidaalfaT0*:
assumes *hip1*: $(elemento\ F\ T) \neq []$
and *hip2*: *FormulaAlfa F*
shows $((reglaTAlfa\ F\ T), T) \in medida$

Usando los tres lemas anteriores, los siguientes tres teoremas permiten demostrar que los argumentos de la función *PruebaTablero*, en las llamadas recursivas, decrecen con respecto a la relación *medida*.

Lema 5.2.10 *Supongamos que T es un tablero que contiene una rama R con no literales. Sea F ∈ R un no literal. Si F es una doble negación, entonces* $((reglaTNoNo\ F\ T), T) \in medida$.

Su formalización es:

theorem *medidanonoT*:
assumes *hip1*: *tieneRama-con-NoLiteral T*
and *hip2*: *tipoFormula* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*))= *NoNo*
shows ((*reglaTNoNo* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*)) *T*),
T) ∈ *medida*

Lema 5.2.11 *Supongamos que T es un tablero que contiene una rama R con no literales. Sea F ∈ R un no literal. Si F es una fórmula alfa, entonces* $((reglaTAlfa\ F\ T), T) \in medida$.

Su formalización es:

theorem *medidaalfaT*:
assumes *hip1*: *tieneRama-con-NoLiteral T*
and *hip2*: *tipoFormula* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*))
= *Alfa*
shows ((*reglaTAlfa* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*)) *T*), *T*)
∈ *medida*

Lema 5.2.12 *Supongamos que T es un tablero que contiene una rama R con no literales. Sea F ∈ R un no literal. Si F es una fórmula beta, entonces* $((reglaBeta\ F\ R), R) \in medida$.

Su formalización es:

theorem *medidabetaT*:
assumes *hip1*: *tieneRama-con-NoLiteral T*
and *hip2*: *tipoFormula* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*))
= *Beta*
shows ((*reglaTBeta* (*selecNoLiteralRama* (*selecRama-con-NoLiteral T*)) *T*), *T*)
∈ *medida*

Por último, usando estos teoremas tenemos la prueba de la terminación de la función *PruebaTablero*

termination *PruebaTablero*
proof−
show *?thesis*
using *wf-medida medidaalfaT medidabetaT medidanonoT*
by (*relation medida*, *auto*)
qed

5.3 Semántica de los tableros semánticos

En esta sección se formalizan los conceptos semánticos relativos a los tableros.

Definición 5.3.1 *El **valor** de una rama* $[F_1, F_2, \ldots, F_n]$ *en una interpretación I se define como sigue:*

$$I'[F_1, F_2, \ldots, F_n] = \begin{cases} \mathsf{F}, & \text{si } I'(F_i) = \mathsf{F} \text{ para algún } i \\ \mathsf{V}, & \text{en caso contrario.} \end{cases}$$

Su formalización es:

primrec *valorR* :: ('*b* ⇒ *v-verdad1*) ⇒ ('*b formula1 list*) ⇒ *v-verdad1*
where
valorR I [] = *Verdad*
| *valorR I* (*F*#*Fs*) = (*if valor1 I F* = *Falso then Falso else valorR I Fs*)

Definición 5.3.2 *Un rama es **satisfacible** si es verdadera en alguna interpretación, en caso contrario se dice que es **insatisfacible**.*

Su formalización es:

definition *satisfacibleR* :: '*b formula1 list* ⇒ *bool* **where**

satisfacibleR R = (∃ *I*. *valorR I R* = *Verdad*)

El próximo teorema afirma que una rama S es satisfacible syss lo es el conjunto de sus fórmulas.

theorem *EquiSatisfacible*:
shows *satisfacibleR S* = *satisfacible* (*set S*)

Definición 5.3.3 *El* ***valor*** *de un tablero* $[R_1, R_2, \ldots, R_n]$ *en una interpretación* I *se define como sigue:*

$$I'[R_1, R_2, \ldots, R_n] = \begin{cases} \mathsf{F}, & \textit{si } I'(R_i) = \mathsf{F} \textit{ para todo } i \\ \mathsf{V}, & \textit{en caso contrario.} \end{cases}$$

Su formalización es:

primrec *valorTablero* :: ('*b* ⇒ *v-verdad1*) ⇒ ('*b formula1 list*) *list* ⇒ *v-verdad1* **where**
valorTablero I [] = *Falso*
| *valorTablero I* (*R*#*Rs*) = (*if valorR I R* = *Verdad then Verdad*
else valorTablero I Rs)
lemma *valorTablero*:
assumes *valorTablero I* (*a* # *T*) = *Verdad*
shows *valorR I a* = *Verdad* ∨ *valorTablero I T* = *Verdad*
proof(*cases valorR I a*)
assume *valorR I a* = *Verdad* **thus** *?thesis* **by** *simp*
next
assume *valorR I a* = *Falso* **thus** *?thesis* **using** *assms* **by** *auto*
qed

Definición 5.3.4 *Un tablero es* ***satisfacible*** *si verdadero en alguna interpretación, en caso contrario se dice que es* ***insatisfacible****.*

Su formalización es:

definition *satisfacibleT* :: ('*b formula1 list*) *list* ⇒ *bool* **where**
satisfacibleT T = (∃ *I*. *valorTablero I T* = *Verdad*)

Un tablero es satisfacible sy solo si tiene una rama satisfacible. Su formalización es:

lemma *satisfacibleRT*:
assumes ∃ *R*. *R* ∈ *set T* ∧ *satisfacibleR R*
shows *satisfacibleT T***lemma** *satisfacibleTR*:
assumes *satisfacibleT T*
shows ∃ *R*. *R* ∈ *set T* ∧ *satisfacibleR R*

5.3.1 Equivalencia entre tableros

Definición 5.3.5 *Los tableros T_1 y T_2 son* **equivalentes** *si para toda interpretación I el valor de T_1 es igual al valor de T_2.*

Su formalización es:

definition *equivalentesT*:: (*'b formula1 list*) *list* ⇒ (*'b formula1 list*) *list* ⇒ *bool* **where**
equivalentesT T1 T2 ≡ (∀ *I*. ((*valorTablero I T1*) = (*valorTablero I T2*)))

Las reglas que se utilizan en el algoritmo *PruebaTablero* preservan el valor de cualquier *tablero*. Más precisamente, en esta sección demostramos que todo tablero T es equivalente a *PruebaTablero T*. Para esto demostramos el siguiente teorema.

theorem *EquivalentesTablero-PruebaTablero*:
EquivalentesT T (*PruebaTablero T*)

La demostración de este teorema utiliza los siguientes resultados.

Lema 5.3.6 *Si F es una doble negación y G es su componente, entonces* $[[F]] \equiv [[G]]$.

Su formalización es

lemma *EquiDNoNo*:
assumes *tipoFormula F* = *NoNo*
shows *equivalentesT* [[*F*]] [[*Comp1 F*]]

Lema 5.3.7 *Si F es una fórmula alfa y sus componentes son F_1 y F_2,* $[[F]] \equiv [[F_1], [F_2]]$.

Su formalización es''

lemma *EquiDAlfa*:
assumes *tipoFormula F* = *Alfa*
shows *equivalentesT* [[*F*]] [[*Comp1 F*,*Comp2 F*]]

Lema 5.3.8 *Si F es una fórmula beta y sus componentes son F_1 y F_2,* $[[F]] \equiv [[F_1, F_2]]$.

Su formalización es:

lemma *EquiDBeta*:
assumes *tipoFormula F* = *Beta*
shows *equivalentesT* [[*F*]] [[*Comp1 F*], [*Comp2 F*]]

Lema 5.3.9 *Si T es un tablero que contiene ramas con no literales, entonces T y PruebaTablero T son equivalentes.*

Su formalización es:

lemma *EquiT2*:
assumes *tieneRama-con-NoLiteral T*
shows *equivalentesT T* (*PruebaTablero T*)

Lema 5.3.10 *Supongamos que T es un tablero que no contiene ramas con no literales, entonces PruebaTablero T = T.*

Su formalización es:

lemma *PruebaTablero4*:
assumes ¬ *tieneRama-con-NoLiteral T*
shows *PruebaTablero T = T*

Teorema 5.3.11 *Sea T un tablero y T′ el obtenido aplicándole el algoritmo PruebaTablero a T. Entonces T y T′ son equivalentes.*

Su formalización es:

theorem *EquivalentesTablero-PruebaTablero*:
equivalentesT T (*PruebaTablero T*)
proof (*cases*)
assume *tieneRama-con-NoLiteral T*
thus *?thesis* **using** *EquiT2* **by** *blast*
next
assume ¬ *tieneRama-con-NoLiteral T*
hence *PruebaTablero T = T* **using** *PruebaTablero4* **by** *simp*
hence *T = PruebaTablero T* **by** *simp*
thus *equivalentesT T* (*PruebaTablero T*)
by (*unfold equivalentesT-def*, *simp*)
qed

Como consecuencia del teorema anterior, se tienen los siguientes dos resultados que serán utilizados, respectivamente, para demostrar la corrección y completitud del algoritmo *PruebaTablero*.

Corolario 5.3.12 *Si el tablero T es satisfacible entonces PruebaTablero T es satisfacible.*

Su formalización es:

```
corollary satisfacible-PruebaTablero:
 assumes satisfacibleT T
 shows satisfacibleT (PruebaTablero T)
proof −
 obtain I where I: valorTablero I T = Verdad
 using assms by (auto simp add: satisfacibleT-def)
 hence valorTablero I (PruebaTablero T) = Verdad
 using EquivalentesTablero-PruebaTablero[of T]
 by (unfold equivalentesT-def, auto)
 thus ?thesis by (auto simp add: satisfacibleT-def)
qed
```

Corolario 5.3.13 *Sea T un tablero. Si PruebaTablero T es satisfacible entonces T es satisfacible.*

Su formalización es:

```
corollary PruebaTablero-satisfacible:
 assumes satisfacibleT (PruebaTablero T)
 shows satisfacibleT T
proof −
 obtain I where I: valorTablero I (PruebaTablero T) = Verdad using assms
  by (auto simp add: satisfacibleT-def)
 hence valorTablero I T = Verdad
  using EquivalentesTablero-PruebaTablero[of T]
  by (unfold equivalentesT-def, auto)
 thus satisfacibleT T by (unfold satisfacibleT-def, blast)
qed
```

5.3.2 Corrección del algoritmo *PruebaTablero*

Dado un conjunto finito S y una fórmula proposicional F, sea T el tablero (inicial) correspondiente al conjunto $S \cup \{\neg F\}$. El objetivo de la definición de la función *PruebaTablero* es: si *PruebaTablero T* es un tablero cerrado, entonces F es una consecuencia lógica de S. Para demostrar su *corrección* ([14] páginas 55-56), teniendo en cuenta que estamos representando los conjuntos finitos de fórmulas por medio de listas, tenemos que probar el siguiente teorema.

```
theorem correcionTableroConsecuencia:
assumes TableroCerrado (PruebaTablero [S @ [(¬. F)]])
shows (set S) |=1 F
```

Para probar el teorema, demostramos previamente los siguientes resultados.

Lema 5.3.14 *Cualquier tablero satifacible es no cerrado.*

Su formalización es:

```
lemma Tablero1:
 assumes satisfacibleT T
 shows ¬TableroCerrado T
```

Lema 5.3.15 *Si la rama compuesta por la fórmula $\neg F$ es insatisfacible entonces F es una tautología.*

Su formalización es:

```
lemma tautsat:
 assumes ¬(satisfacibleR [¬. F])
 shows tautologia F
```

Lema 5.3.16 *Sea S un conjunto finito de fórmulas proposicionales y S' el tablero (inicial) correspondiente. Si el tablero PruebaTablero S' es cerrrado entonces S es insatifacible.*

Su formalización es:

```
lemma TabSat:
 assumes TableroCerrado (PruebaTablero [S])
 shows ¬ satisfacible (set S)
proof −
 have ¬(satisfacibleT (PruebaTablero [S])) using assms Tablero1 by blast
 hence ¬(satisfacibleT [S]) using satisfacible-PruebaTablero by blast
 hence ¬(satisfacibleR S)
  by (auto simp add: satisfacibleT-def,
     auto simp add: satisfacibleR-def,
     split split-if-asm,
     auto)
 thus ?thesis using EquiSatisfacible by auto
qed
```

Podemos ahora probar el teorema de corrección.

Teorema 5.3.17 *Sea S un conjunto finito de fórmulas proposicionales y T el tablero (inicial) correspondiente al conjunto $S \cup \{\neg F\}$. Si el tablero PruebaTablero T es cerrrado entonces F es una consecuencia lógica de S.*

Su formalización es:

theorem *correcionTableroConsecuencia*:
assumes *TableroCerrado* (*PruebaTablero* [*S* @ [(¬. *F*)]])
shows (*set S*) ⊨*1 F*
proof −
have ¬ *satisfacible* (*set* (*S* @ [(¬. *F*)]))
using *assms TabSat*[**where** *S* = *S* @ [(¬. *F*)]] **by** *simp*
thus *?thesis* **using** *EquiConsSat* **by** *auto*
qed

Como consecuencia se obtiene el siguiente corolario.

Lema 5.3.18 *Sea T el tablero que resulta de aplicarle el algoritmo PruebaTablero al tablero (inicial) correspondiente a* $\{\neg F\}$*. Si T es cerrado entonces F es una tautología.*

Su formalización es:

corollary *correcionTableroTautologia*:
assumes *TableroCerrado* (*PruebaTablero* [[¬. *F*]])
shows *tautologia F*
using *assms correcionTableroConsecuencia*[**where** *S* = []] *CNS-tautologia*
by *auto*

Capítulo 6

Teorema de existencia de modelos proposicionales

6.1 Introducción

En el capítulo 3 se definió un sistema axiomático $\vdash$, definición (3.1.5), para la lógica proposicional y se demostró de manera constructiva la completitud del sistema utilizando el método de Kalmar (teorema 3.4.26). Otro método de demostración, utilizado por el lógico Leon Henkin, es considerar la afirmación contrarrecíproca, es decir, si una fórmula F no es deducible entonces no es tautología. La prueba utiliza el siguiente teorema que hace referencia a la noción de *consistencia*. Una fórmula F es consistente, con respecto al cálculo de prueba $\vdash$, si $\nvdash \neg F$; es decir, si $\neg F$ no es deducible en el sistema.

Teorema 6.1.1 (Henkin) *Para cualquier fórmula F, Si F es consistente entonces F tiene un modelo.*

Como una consecuencia del teorema anterior podemos demostrar que nuestro sistema axiomático es completo. Consideremos una fórmula F que no es deducible, $\nvdash F$. Por el lema 3.1.23 sabemos que $\vdash \neg\neg F \rightarrow F$. Por consiguiente, de la hipótesis $\nvdash F$ tenemos que $\nvdash \neg\neg F$ por la regla MP, es decir, $\nvdash \neg(\neg F)$. De esta forma, $\neg F$ es consistente. Luego, por el teorema 6.1.1, $\neg F$ tiene un modelo. Por lo tanto, F no es tautología.

Para demostrar el teorema 6.1.1, se extiende la noción de consistencia de una fórmula a un conjunto de fórmulas. Un conjunto finito de fórmulas $S = \{F_1, F_2, \ldots, F_n\}$ es consistente si la fórmula $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ es consistente, es decir $\nvdash \neg(F_1 \wedge F_2 \wedge \cdots \wedge F_n)$. Un conjunto arbitrario de fórmulas S es consistente si cada subconjunto finito de S es consistente.

Un conjunto consistente de fórmulas S es un *conjunto consistente maximal* si para toda fórmula $F \notin S$, $S \cup \{F\}$ no es consistente. Un resultado importante sobre estos

conjuntos es el teorema de Lindenbaum:

Teorema 6.1.2 (Lindenbaum) *Todo conjunto consistente S puede ser extendido a un conjunto consistente maximal M.*

El conjunto M se define, a partir de S y de una *enumeración* $\phi_0, \phi_1, \phi_2, \ldots$ del conjunto de fórmulas del lenguaje proposicional, como la unión $\bigcup_i S_i$ de la siguiente sucesión S_i de conjuntos consistentes:

$$\begin{aligned} S_0 &= S \\ S_{i+1} &= \begin{cases} S_i \cup \{\phi_i\} & \text{si } S_i \cup \{\phi_i\} \text{ es consistente} \\ S_i & \text{en otro caso.} \end{cases} \end{aligned}$$

Sea M un conjunto consistente maximal. Para establecer la conexión entre conjuntos consistentes e interpretaciones, definimos la siguiente interpretación I_M. Sea P un símbolo proposicional, entonces

$$I_M(P) = \begin{cases} \mathsf{V}, & \text{si } P \in M \\ \mathsf{F}, & \text{en otro caso.} \end{cases}$$

Se tiene el siguiente resultado.

Teorema 6.1.3 *Si M es un conjunto consistente maximal, entonces para toda fórmula F,* $I'_M(F) = \mathsf{V}$ *si solamente si* $F \in M$.

Como una consecuencia del teorema anterior se tiene la demostración del teorema 6.1.1:

Demostración: Sea F una fórmula consistente. Por el teorema 6.1.2, $\{F\}$ puede extenderse a un conjunto consistente maximal M. Por el teorema 6.1.3, $I'_M(F) = \mathsf{V}$ puesto que $F \in M$. Es decir, F tiene un modelo.

□

De forma más general, en este capítulo formalizaremos el enunciado y demostración del teorema de existencia de modelos en la lógica proposicional. Este resultado, que garantiza la existencia de un modelo para un conjunto de fórmulas proposicionales, relaciona la sintaxis y semántica de un lenguaje proposicional: cualquier conjunto de fórmulas *consistente* es *satisfacible*. La demostración en Isabelle de este teorema presentada en este trabajo es una formalización de la prueba descrita en el texto de Fitting [14], la cual está basada en las ideas expuestas anteriormente pero utilizando un argumento abstracto de completitud que generaliza los argumentos comunes usados en la mayoría de las demostraciones indirectas de la completitud de procedimientos de prueba como,

sistemas de Hilbert, sistemas de Gentzen, resolución, tableros semánticos, etc. Estas demostraciones de completitud hacen uso del concepto de *consistencia*, que es relativo a cada procedimiento de prueba particular. Un conjunto de fórmulas es consistente, con respecto a un procedimento de prueba, si ninguna contradicción se puede derivar a partir del conjunto usando el *cálculo de prueba específico*. Sin embargo, Fitting usa un enfoque más general basado en caracterizar el concepto de consistencia, sin hacer uso de la noción de "cálculo de prueba", por medio de las propiedades básicas que todos los conjuntos consistentes deben tener.

La prueba de existencia de un modelo para un conjunto consistente de fórmulas S pertenecientes a un lenguaje L que se presenta en el texto de Fitting, ([14] página 60), está basada en extender S a un conjunto consistente maximal M (su definición es la misma que la dada anteriormente).

El desarrollo central de la demostración que aparece en el libro de Fitting es el siguiente.

- Usando la noción de *clausura por subconjuntos* se prueba que M es maximal.
- Usando la noción de *carácter finito* se prueba que M es consistente.

Además se demuestran los siguientes dos resultados.

- El conjunto M es de *Hintikka*.
- Los conjuntos de Hintikka son satisfacibles.

Por tanto, se puede concluir que M es satisfacible y, puesto que $S \subseteq M$, se tiene que S también es satisfacible.

6.2 Conjuntos consistentes

En esta sección formalizamos el criterio abstracto, propuesto en ([14] página 59), para describir los conjuntos consistentes.

Definición 6.2.1 *Una colección de conjuntos de fórmulas $\mathcal{C}$ es una* **propiedad de consistencia proposicional**, *si todo elemento S de $\mathcal{C}$ verifica las siguientes propiedades.*

1. *Para cualquier fórmula atómica P, $P \notin S$ o $\neg P \notin S$.*
2. *$\bot \notin S$ y $\neg\top \notin S$.*
3. *Si $\neg\neg F \in S$ entonces $S \cup \{F\} \in \mathcal{C}$.*

4. *Si* $\alpha \in S$ *entonces,* $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$.

5. *Si* $\beta \in S$ *entonces,* $S \cup \{\beta_1\} \in \mathcal{C}$ *o* $S \cup \{\beta_2\} \in \mathcal{C}$.

Su formalización es:

definition *consistenciaP* :: *'b formula1 set set* ⇒ *bool* **where**
consistenciaP $\mathcal{C}$ =
($\forall S.\ S \in \mathcal{C} \longrightarrow (\forall P.\ \neg\ (Atomo\ P \in S \wedge (\neg.Atomo\ P) \in S)) \wedge$
$FF \notin S \wedge (\neg.TT) \notin S \wedge$
$(\forall F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in\ \mathcal{C}) \wedge$
$(\forall F.\ ((FormulaAlfa\ F) \wedge F{\in}S) \longrightarrow (S{\cup}\{Comp1\ F, Comp2\ F\}) \in \mathcal{C}) \wedge$
$(\forall F.\ ((FormulaBeta\ F) \wedge F{\in}S) \longrightarrow (S{\cup}\{Comp1\ F\}{\in}\mathcal{C}) \vee (S{\cup}\{Comp2\ F\}{\in}\mathcal{C})))$

6.3 Conjuntos consistentes y clausura por subconjuntos

Definición 6.3.1 *Una colección de conjuntos* $\mathcal{C}$ *es* **cerrada por subconjuntos** *si para cada* $S \in \mathcal{C}$ *los subconjuntos de* S *también son elementos de* $\mathcal{C}$.

Su formalización es:

definition *subconj-cerrada* :: *'a set set* ⇒ *bool* **where**
subconj-cerrada $\mathcal{C} = (\forall S \in \mathcal{C}.\ \forall S'.\ S' \subseteq S \longrightarrow S' \in \mathcal{C})$

En esta sección demostraremos que toda propiedad de consistencia proposicional $\mathcal{C}$ puede extenderse a una propiedad de consistencia $\mathcal{C}^+$ que es cerrada por subconjuntos. Para la demostración basta con considerar $\mathcal{C}^+$ igual a la colección de los subconjuntos de los elementos de $\mathcal{C}$,

$$\mathcal{C}^+ = \{S | \exists S' \in \mathcal{C}(S \subseteq S')\}.$$

La definición en Isabelle de $\mathcal{C}^+$ es,

definition *clausura-subconj* :: *'a set set* ⇒ *'a set set* (*-⁺*[1000]* *1000*) **where**
$\mathcal{C}^+ = \{S.\ \exists S' \in \mathcal{C}.\ S \subseteq S'\}$

Teorema 6.3.2

Sea $\mathcal{C}$ *una colección de conjuntos. entonces,*

(a) $\mathcal{C} \subseteq \mathcal{C}^+$.

(b) $\mathcal{C}^+$ *es cerrada por subconjuntos.*

(c) Si $\mathcal{C}$ *es una propiedad de consistencia proposicional entonces* $\mathcal{C}^+$ *también es una propiedad de consistencia proposicional.*

Demostración:

(a) Sea $S \in \mathcal{C}$. Puesto que $S \subseteq S$ se tiene que $S \in \mathcal{C}^+$ por la definición de $\mathcal{C}^+$. Por tanto $\mathcal{C}$ está contenido en $\mathcal{C}^+$.

(b) Supongamos que $S \in \mathcal{C}^+$ y sea $T \subseteq S$. Puesto que $S \in \mathcal{C}^+$, existe $S' \in \mathcal{C}$ tal que $S \subseteq S'$ por la definición de $\mathcal{C}^+$. Luego existe $S' \in \mathcal{C}$ tal que $T \subseteq S'$ y, por tanto, $T \in \mathcal{C}^+$. Así, $\mathcal{C}^+$ es una colección cerrada por subconjuntos.

(c) Supongamos que $\mathcal{C}$ es una propiedad de consistencia proposicional. Sea $S \in \mathcal{C}^+$, entonces existe $S' \in \mathcal{C}$ tal que $S \subseteq S'$, de esto último mostramos que se cumplen las condiciones para que $\mathcal{C}^+$ sea una propiedad de consistencia.

1. Sea P una fórmula atómica, entonces $P \notin S'$ o $\neg P \notin S'$ ya que $S' \in \mathcal{C}$ y $\mathcal{C}$ es una propiedad de consistencia. Luego, $P \notin S$ o $\neg P \notin S$ puesto que $S \subseteq S'$.

2. $\bot \notin S'$ y $\neg\top \notin S'$ puesto que $S' \in \mathcal{C}$ y $\mathcal{C}$ es una propiedad de consistencia. Luego, $\bot \notin S$ y $\neg\top \notin S$ puesto que $S \subseteq S'$.

3. Supongamos que $\neg\neg F \in S$, entonces $\neg\neg F \in S'$ ya que $S \subseteq S'$. Luego $S' \cup \{F\} \in \mathcal{C}$, ya que $S' \in \mathcal{C}$ y $\mathcal{C}$ es una propiedad de consistencia; además $S \cup \{F\} \subseteq S' \cup \{F\}$ puesto que $S \subseteq S'$. Por lo tanto, por definición de $\mathcal{C}^+$, se tiene que $S \cup \{F\} \in \mathcal{C}^+$.

4. Supongamos que $\alpha \in S$, entonces $\alpha \in S'$ ya que $S \subseteq S'$. Luego $S' \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$, ya que $S' \in \mathcal{C}$ y $\mathcal{C}$ es una propiedad de consistencia.

   Por otro lado $S \cup \{\alpha_1, \alpha_2\} \subseteq S' \cup \{\alpha_1, \alpha_2\}$ ya que, $S \subseteq S'$. Por lo tanto, por definición de $\mathcal{C}^+$, se tiene que $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}^+$.

5. Supongamos que $\beta \in S$, entonces $\beta \in S'$ puesto que que $S \subseteq S'$.

   Luego, $S' \cup \{\beta_1\} \in \mathcal{C}$ o $S' \cup \{\beta_2\} \in \mathcal{C}$ por ser $\mathcal{C}$ una propiedad de consistencia; además $S \cup \{\beta_1\} \subseteq S' \cup \{\beta_1\}$ y $S \cup \{\beta_2\} \subseteq S' \cup \{\beta_2\}$ ya que $S \subseteq S'$. Por lo tanto, por definición de $\mathcal{C}^+$, se tiene que $S \cup \{\beta_1\} \in \mathcal{C}^+$ o $S \cup \{\beta_2\} \in \mathcal{C}^+$.

□

A continuación formalizamos cada parte del teorema anterior. La formalización de la parte (a) es la siguiente:

lemma *cerrado-subset*: $\mathcal{C} \subseteq \mathcal{C}^+$
proof −
{ **fix** S
assume $S \in \mathcal{C}$
moreover

```
   have S ⊆ S by simp
   ultimately
   have S ∈ C⁺
    by (unfold clausura-subconj-def, auto) }
 thus ?thesis by auto
qed
```

El siguiente lema formaliza la parte (b).

```
lemma cerrado-cerrado: subconj-cerrada (C⁺)
proof −
 { fix S T
  assume S ∈ C⁺ and T ⊆ S
  obtain S1 where S1 ∈ C and S ⊆ S1 using ⟨S ∈ C⁺⟩
   by (unfold clausura-subconj-def, auto)
  have T ⊆ S1 using ⟨T ⊆ S⟩ and ⟨S ⊆ S1⟩ by simp
  hence T ∈ C⁺ using ⟨S1 ∈ C⟩
   by (unfold clausura-subconj-def, auto)}
 thus ?thesis by (unfold subconj-cerrada-def, auto)
qed
```

Los siguientes lemas corresponden a la formalización de los 5 casos de la parte (c) del teorema anterior.

```
lemma condiconsisP1:
 assumes consistenciaP C and T ∈ C and S ⊆ T
 shows (∀ P. ¬(Atomo P ∈ S ∧ (¬.Atomo P) ∈ S))

lemma condiconsisP2:
 assumes consistenciaP C and T ∈ C and S ⊆ T
 shows FF ∉ S ∧ (¬.TT)∉ S

lemma condiconsisP3:
 assumes consistenciaP C and T ∈ C and S ⊆ T
 shows ∀ F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈ C⁺
proof(rule allI)

lemma condiconsisP4:
 assumes consistenciaP C and T ∈ C and S ⊆ T
 shows ∀ F. ((FormulaAlfa F) ∧ F ∈ S) ⟶ (S ∪ {Comp1 F, Comp2 F}) ∈ C⁺

lemma condiconsisP5:
 assumes consistenciaP C and T ∈ C and S ⊆ T
 shows (∀ F. ((FormulaBeta F) ∧ F ∈ S) ⟶
        (S ∪ {Comp1 F} ∈ C⁺) ∨ (S ∪ {Comp2 F} ∈ C⁺))
```

Por último, la formalización de la parte (c) del teorema 6.3.2 es la siguiente:

```
theorem cerrado-consistenciaP:
 assumes hip1: consistenciaP C
 shows consistenciaP (C⁺)
proof −
 { fix S
  assume S ∈ C⁺
  hence ∃T∈C. S ⊆ T by(simp add: clausura-subconj-def)
  then obtain T where hip2: T ∈ C and hip3: S ⊆ T by auto
  have (∀P. ¬ (Atomo P ∈ S ∧ (¬.Atomo P) ∈ S)) ∧
        FF ∉ S ∧ (¬.TT) ∉ S ∧
        (∀F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈ C⁺) ∧
        (∀F. ((FormulaAlfa F) ∧ F ∈ S) ⟶
           (S ∪ {Comp1 F, Comp2 F}) ∈ C⁺) ∧
        (∀F. ((FormulaBeta F) ∧ F ∈ S) ⟶
           (S ∪ {Comp1 F} ∈ C⁺) ∨ (S ∪ {Comp2 F} ∈ C⁺))
    using
     condiconsisP1[OF hip1 hip2 hip3] condiconsisP2[OF hip1 hip2 hip3]
     condiconsisP3[OF hip1 hip2 hip3] condiconsisP4[OF hip1 hip2 hip3]
     condiconsisP5[OF hip1 hip2 hip3]
    by blast}
 thus ?thesis by (simp add: consistenciaP-def)
qed
```

6.4 Propiedad de carácter finito

La demostración del teorema de existencia de modelos está basada en poder extender una propiedad de consistencia a otra propiedad de consistencia que sea cerrada por subconjuntos y de *carácter finito*.

Definición 6.4.1 *Una colección de conjuntos $\mathcal{C}$ es de **carácter finito** si para cada conjunto S se tiene que, S pertenece a $\mathcal{C}$ si y sólo si cada subconjunto finito de S pertenece a $\mathcal{C}$.*

Su formalización es:

```
definition caracter-finito :: 'a set set ⇒ bool where
 caracter-finito C = (∀S. S ∈ C = (∀S'. finite S' ⟶ S' ⊆ S ⟶ S' ∈ C))
```

Teorema 6.4.2 *Toda colección de conjuntos de carácter finito $\mathcal{C}$ es cerrada por subconjuntos.*

Demostración: Supongamos que $\mathcal{C}$ es de carácter finito; sea $S \in \mathcal{C}$ y $T \subseteq S$, por la definición de colección cerrada por subconjuntos, hay que demostrar que $T \in \mathcal{C}$. Para esto, por hipotésis, basta con demostrar que cada subconjunto finito U de T pertenece a $\mathcal{C}$. Sea U un conjunto finito tal que $U \subseteq T$. Puesto que $T \subseteq S$ se tiene que $U \subseteq S$. Así, puesto que $S \in \mathcal{C}$ y C es de carácter finito se tiene que $U \in \mathcal{C}$.

□

La siguiente es la formalización del teorema anterior.

```
theorem caracter-finito-cerrado:
 assumes caracter-finito C
 shows subconj-cerrada C
proof −
 { fix S T
  assume S ∈ C and  T ⊆ S
  have T ∈ C using caracter-finito-def
  proof −
   { fix U
    assume finite U and U ⊆ T
    have U ∈ C
    proof −
     have U ⊆ S using ⟨U ⊆ T⟩ and ⟨T ⊆ S⟩ by simp
     thus U ∈ C using ⟨S ∈ C⟩ and ⟨finite U⟩ and assms
      by (unfold caracter-finito-def) blast
    qed}
   thus ?thesis using assms by( unfold caracter-finito-def) blast
  qed }
 thus ?thesis  by(unfold  subconj-cerrada-def) blast
qed
```

6.5 Extensión a una propiedad de carácter finito

En la sección 6.3 se demostró que toda propiedad de consistencia proposicional $\mathcal{C}$ puede extenderse a una propiedad de consistencia $\mathcal{C}^+$ que es cerrada por subconjuntos. En esta sección demostraremoss que toda propiedad de consistencia proposicional $\mathcal{C}$ que sea cerrada por subconjuntos puede extenderse a una propiedad de consistencia $\mathcal{C}^-$ que es de carácter finito. Para la demostración, basta con considerar $\mathcal{C}^-$ igual a la colección de todos los conjuntos tales que sus subconjuntos finitos están en $\mathcal{C}$: $\mathcal{C}^- = \{S \mid \forall\, S' \subseteq S\ (S' \textit{ finito} \longrightarrow S' \in \mathcal{C})\}$.

La definición en Isabelle de $\mathcal{C}^-$ es,

```
definition clausura-cfinito :: 'a set set ⇒ 'a set set (-⁻ [1000] 999) where
```

$$\mathcal{C}^{-} = \{S.\, \forall\, S'.\, S' \subseteq S \longrightarrow \textit{finite}\ S' \longrightarrow S' \in \mathcal{C}\}$$

Teorema 6.5.1 *Sea $\mathcal{C}$ una colección de conjuntos. Se verifican las siguientes propiedades:*

(a) Si $\mathcal{C}$ es cerrada por subconjuntos, entonces $\mathcal{C} \subseteq \mathcal{C}^{-}$.

(b) $\mathcal{C}^{-}$ es de carácter finito.

(c) Si $\mathcal{C}$ es una propiedad de consistencia proposicional que es cerrada por subconjuntos entonces, $\mathcal{C}^{-}$ es una propiedad de consistencia proposicional.

Demostración:

Apartado (a) Supongamos que $\mathcal{C}$ es cerrada por subconjuntos y sea $S \in \mathcal{C}$. Mostramos que $S \in \mathcal{C}^{-}$ usando la definición de $\mathcal{C}^{-}$: sea $S' \subseteq S$ y S' finito. Puesto que $S \in \mathcal{C}$ y $S' \subseteq S$, entonces $S' \in \mathcal{C}$, ya que por hipótesis $\mathcal{C}$ es cerrada por subconjuntos.

Apartado (b) Sea S un conjunto. Por la definición de propiedad de carácter finito hay que demostrar que, $S \in \mathcal{C}^{-}$ si y sólo si cada subconjunto finito de S pertenece a $\mathcal{C}^{-}$:

Si $S \in \mathcal{C}^{-}$, por la definición de $\mathcal{C}^{-}$, se tiene que cada subconjunto finito de S pertenece a $\mathcal{C}^{-}$ y recíprocamente si cada subconjunto finito de S pertenece a $\mathcal{C}^{-}$ entonces, por la definición de $\mathcal{C}^{-}$, $S \in \mathcal{C}^{-}$.

Apartado (c) Supongamos que $\mathcal{C}$ es una propiedad de consistencia proposicional que es cerrada por subconjuntos. Sea $S \in \mathcal{C}^{-}$. Entonces, por la definición de $\mathcal{C}^{-}$, se tiene que cada subconjunto finito de S pertenece a $\mathcal{C}$. A continuación mostramos que se cumplen las condiciones para que $\mathcal{C}^{-}$ sea una propiedad de consistencia.

Condición 1 Sea P una fórmula atómica, demostramos por contradicción que $P \notin S$ o $\neg P \notin S$. Supongamos que $P \in S$ y $\neg P \in S$, entonces $\{P, \neg P\} \subseteq S$. Por tanto $\{P, \neg P\} \in \mathcal{C}$, por ser $\mathcal{C}$ cerrada por subconjuntos. Luego $P \notin \{P, \neg P\}$ o $\neg P \notin \{P, \neg P\}$, ya que $\mathcal{C}$ es una propiedad de consistencia. De esta forma obtenemos una contradicción.

Condición 2 La demostración de $\bot \notin S$ es por contradicción. Supongamos que $\bot \in S$, entonces $\{\bot\} \subseteq S$. Por lo tanto $\{\bot\} \in \mathcal{C}$, por ser $\mathcal{C}$ cerrada pos subconjuntos. Luego $\bot \notin \{\bot\}$, ya que $\mathcal{C}$ es una propiedad de consistencia. De esta forma obtenemos una contradicción.

De la misma forma $\neg\top \notin S$, de lo contrario $\{\neg\top\} \subseteq S$ y por lo tanto $\{\neg\top\} \in \mathcal{C}$, por ser $\mathcal{C}$ cerrada por subconjuntos. Así, puesto que $\mathcal{C}$ es una propiedad de consistencia, se tendría que $\neg\top \notin \{\neg\top\}$, lo cual es imposible.

Condición 3 Supongamos que $\neg\neg F \in S$. Demostramos que $S \cup \{F\} \in \mathcal{C}^{-}$ usando la definición de $\mathcal{C}^{-}$: consideremos S' subconjunto finito de $S \cup \{F\}$, y mostremos que $S' \in \mathcal{C}$.

Puesto que $\neg\neg F \in S$ y $S' \subseteq S \cup \{F\}$ tenemos que $S' - \{F\} \cup \{\neg\neg F\} \subseteq S$; también tenemos que $S' - \{F\} \cup \{\neg\neg F\}$ es finito ya que S' es finito. Así, $S' - \{F\} \cup \{\neg\neg F\} \in \mathcal{C}$ por la definición de $\mathcal{C}^-$, y como $\{\neg\neg F\} \in S' - \{F\} \cup \{\neg\neg F\}$ entonces $(S' - \{F\} \cup \{\neg\neg F\}) \cup \{F\} \in \mathcal{C}$ por ser $\mathcal{C}$ una propiedad de consistencia. Además, puesto que $S' - \{F\} \cup \{\neg\neg F\}) \cup \{F\} = S' \cup \{\neg\neg F\} \cup \{F\}$, se tiene que $S' \cup \{\neg\neg F\}) \cup \{F\} \in \mathcal{C}$. De esto último y como $S' \subseteq S' \cup \{\neg\neg F\} \cup \{F\}$ se tiene que $S' \in \mathcal{C}$ ya que por hipótesis $\mathcal{C}$ es cerrada por subconjuntos.

Condición 4 Supongamos que $\alpha \in S$. Demostramos que $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}^-$ usando la definición de $\mathcal{C}^-$: consideremos S' subconjunto finito de $S \cup \{\alpha_1, \alpha_2\}$, y mostremos que $S' \in \mathcal{C}$.

Puesto que $\alpha \in S$ y $S' \subseteq S \cup \{\alpha_1, \alpha_2\}$ tenemos que $S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\} \subseteq S$; también tenemos que $S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\}$ es finito ya que S' es finito.

Así, $S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\} \in \mathcal{C}$ por la definición de $\mathcal{C}^-$, y como $\alpha \in S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\}$ entonces $S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\}) \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ por ser $\mathcal{C}$ una propiedad de consistencia.

Además, puesto que $S' - \{\alpha_1, \alpha_2\} \cup \{\alpha\}) \cup \{\alpha_1, \alpha_2\} = S' \cup \{\alpha\} \cup \{\alpha_1, \alpha_2\}$ se tiene que $S' \cup \{\alpha\} \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. De esto último, y como $S' \subseteq S' \cup \{\alpha\} \cup \{\alpha_1, \alpha_2\}$, se tiene que $S' \in \mathcal{C}$ ya que por hipótesis $\mathcal{C}$ es cerrada por subconjuntos.

Condición 5 Supongamos que $\beta \in S$. Demostramos que $S \cup \{\beta_1\} \in \mathcal{C}^-$ o $S \cup \{\beta_2\} \in \mathcal{C}^-$ por contradicción.

Supongamos que $S \cup \{\beta_1\} \notin \mathcal{C}^-$ y $S \cup \{\beta_2\} \notin \mathcal{C}^-$. Entonces, existe S_1 subconjunto finito de $S \cup \{\beta_1\}$ tal que $S_1 \notin \mathcal{C}$, y existe S_2 subconjunto finito de $S \cup \{\beta_2\}$ tal que $S_2 \notin \mathcal{C}$. Puesto que $\beta \in S$, $S_1 \subseteq S \cup \{\beta_1\}$ y $S_2 \subseteq S \cup \{\beta_2\}$ tenemos que, $(S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\} \subseteq S$.

También tenemos que $(S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}$ es finito ya que S_1 y S_2 son finitos. Así, $(S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\} \in \mathcal{C}$ por la definición de $\mathcal{C}^-$, y como

$$\beta \in (S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}$$

entonces, por ser $\mathcal{C}$ una propiedad de consistencia, se tiene que

$$((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_1\} \in \mathcal{C}$$
$$((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_2\} \in \mathcal{C}.$$

De esto último tenemos que $S_1 \in \mathcal{C}$ o $S_2 \in \mathcal{C}$:

Si $((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_1\} \in \mathcal{C}$ entonces, puesto que

$$S_1 \subseteq ((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_1\},$$

tenemos que $S_1 \in \mathcal{C}$ por ser $\mathcal{C}$ cerrada por subconjuntos.

De igual forma, si $((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_2\} \in \mathcal{C}$ entonces, puesto que

$$S_2 \subseteq ((S_1 - \{\beta_1\}) \cup (S_2 - \{\beta_2\}) \cup \{\beta\}) \cup \{\beta_2\},$$

tenemos que $S_2 \in \mathcal{C}$ por ser $\mathcal{C}$ cerrada por subconjuntos.

Esto contradice la hipótesis inicial: $S_1 \notin \mathcal{C}$ y $S_2 \notin \mathcal{C}$.

□

A continuación formalizamos cada parte de la prueba del teorema anterior. La formalización de la parte (a) es la siguiente:

```
lemma caracter-finito-subset:
 assumes subconj-cerrada C
 shows C ⊆ C⁻
proof −
 { fix S
  assume S ∈ C
  have S ∈ C⁻
  proof −
   { fix S'
    assume S' ⊆ S and finite S'
    hence S' ∈ C using ⟨subconj-cerrada C⟩ and ⟨S ∈ C⟩
     by (simp add: subconj-cerrada-def)}
   thus ?thesis by (simp add: clausura-cfinito-def)
  qed}
 thus ?thesis by auto
qed
```

El siguiente lema formaliza la parte (b) del teorema 6.5.1

```
lemma caracter-finito: caracter-finito (C⁻)
proof (unfold caracter-finito-def)
 show ∀ S. (S ∈ C⁻) = (∀ S'. finite S' ⟶ S' ⊆ S ⟶ S' ∈ C⁻)
 proof
  fix S
  { assume S ∈ C⁻
   hence ∀ S'. finite S' ⟶ S' ⊆ S ⟶ S' ∈ C⁻
    by(simp add: clausura-cfinito-def)}
  moreover
  { assume ∀ S'. finite S' ⟶ S' ⊆ S ⟶ S' ∈ C⁻
   hence S ∈ C⁻ by(simp add: clausura-cfinito-def)}
  ultimately
  show (S ∈ C⁻) = (∀ S'. finite S' ⟶ S' ⊆ S ⟶ S' ∈ C⁻)
   by blast
 qed
qed
```

Los siguientes lemas corresponden a la formalización de los 5 casos de la parte (c) del teorema 6.5.1.

```
lemma condicaracterP1:
 assumes consistenciaP C
 and subconj-cerrada C
 and hip: ∀ S'⊆S. finite S' ⟶ S' ∈ C
```

shows $(\forall P.\ \neg(Atomo\ P \in S \wedge (\neg.Atomo\ P) \in S))$

lemma *condicaracterP2*:
assumes *consistenciaP* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $FF \notin S \wedge (\neg.TT) \notin S$

lemma *condicaracterP3*:
assumes *consistenciaP* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall F.\ (\neg.\neg.F) \in S \longrightarrow\ S \cup \{F\} \in \mathcal{C}^{-}$

lemma *condicaracterP4*:
assumes *consistenciaP* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $(\forall F.\ ((FormulaAlfa\ F) \wedge F \in S) \longrightarrow (S \cup \{Comp1\ F, Comp2\ F\}) \in \mathcal{C}^{-})$

lemma *condicaracterP5*:
assumes *consistenciaP* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall F.\ FormulaBeta\ F \wedge F \in S \longrightarrow S \cup \{Comp1\ F\} \in \mathcal{C}^{-} \vee S \cup \{Comp2\ F\} \in \mathcal{C}^{-}$

Por último, se demuestra que si $\mathcal{C}$ es una propiedad de consistencia proposicional que es cerrada por subconjuntos entonces $\mathcal{C}^{-}$ es de carácter finito.

theorem *cfinito-consistenciaP*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: *subconj-cerrada* $\mathcal{C}$
shows *consistenciaP* $(\mathcal{C}^{-})$
proof −
{ **fix** S
assume $S \in \mathcal{C}^{-}$
hence *hip3*: $\forall S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
by (*simp add*: *clausura-cfinito-def*)
have $(\forall P.\ \neg(Atomo\ P \in S \wedge (\neg.Atomo\ P) \in S)) \wedge$
$FF \notin S \wedge (\neg.TT) \notin S \wedge$
$(\forall F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^{-}) \wedge$
$(\forall F.\ ((FormulaAlfa\ F) \wedge F \in S) \longrightarrow (S \cup \{Comp1\ F, Comp2\ F\}) \in \mathcal{C}^{-}) \wedge$
$(\forall F.\ ((FormulaBeta\ F) \wedge F \in S) \longrightarrow$
$(S \cup \{Comp1\ F\} \in \mathcal{C}^{-}) \vee (S \cup \{Comp2\ F\} \in \mathcal{C}^{-}))$
using
condicaracterP1[*OF hip1 hip2 hip3*] *condicaracterP2*[*OF hip1 hip2 hip3*]

condicaracterP3[*OF hip1 hip2 hip3*] *condicaracterP4*[*OF hip1 hip2 hip3*]
condicaracterP5[*OF hip1 hip2 hip3*] **by** *auto* }
thus *?thesis* **by** (*simp add*: *consistenciaP-def*)
qed

6.6 Extensión a conjuntos consistentes maximales

En esta sección precisamos en Isabelle, cómo extender un conjunto consistente S de fórmulas proposicionales a un conjunto consistente maximal de acuerdo a lo expuesto en la página 61 de [14].

Definición 6.6.1 *Sea $\mathcal{C}$ una colección de conjuntos. El conjunto S es un elemento* **maximal** *de $\mathcal{C}$ si, para cualquier $S' \in \mathcal{C}$ tal que $S \subseteq S'$ implica que $S = S'$.*

Su formalización es:

definition *maximal* :: *'a set* ⇒ *'a set set* ⇒ *bool* **where**
maximal S C = (∀ *S'*∈*C*. *S* ⊆ *S'* ⟶ *S* = *S'*)

La construcción, a partir de S de un conjunto consistente maximal está basada en la siguiente sucesión creciente (por inclusión) de conjuntos.

Definición 6.6.2 *Sean S un conjunto de fórmulas, $\mathcal{C}$ una colección de conjuntos de fórmulas y f una sucesión de fórmulas. Definimos la siguiente sucesión de conjuntos, $\{S^n_{S,\mathcal{C},f}\}_n$.*

$$S^0_{S,\mathcal{C},f} = S$$
$$S^{n+1}_{S,\mathcal{C},f} = \begin{cases} S^n_{S,\mathcal{C},f} \cup \{f_n\}, & \text{si } S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C} \\ S^n_{S,\mathcal{C},f}\,, & \text{en otro caso.} \end{cases}$$

La formalización de la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ es:

primrec *sucP* :: *'b formula1 set* ⇒ *'b formula1 set set* ⇒ (*nat* ⇒ *'b formula1*) ⇒ *nat* ⇒ *'b formula1 set*
where
sucP S C f 0 = *S*
| *sucP S C f* (*Suc n*) =
(*if sucP S C f n* ∪ {*f n*} ∈ *C*
then sucP S C f n ∪ {*f n*}
else sucP S C f n)

Consideremos ahora la unión de los elementos $S^n_{S,\mathcal{C},f}$,

$$S_{S,\mathcal{C},f} = \bigcup_n S^n_{S,\mathcal{C},f}.$$

Su formalización en Isabelle es:

definition *MsucP* :: *'b formula1 set* ⇒ *'b formula1 set set* ⇒ (*nat* ⇒ *'b formula1*) ⇒ *'b formula1 set*
where
MsucP S C f = (⋃ *n*. *sucP S C f n*)

En lo que sigue demostraremos que para cualquier lenguaje proposicional **L**, si $\mathcal{C}$ es una propiedad de consistencia proposicional de carácter finito, $S \in \mathcal{C}$ y f es una *enumeración* del conjunto de fórmulas de **L** entonces, $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$ que pertenece a $\mathcal{C}$ y extiende a S.

Por definición, se tiene que el conjunto $S_{S,\mathcal{C},f}$ es una extensión de S:

Teorema 6.6.3 $S \subseteq S_{S,\mathcal{C},f}$.

Su formalización es:

theorem *Max-subconjuntoP*: *S* ⊆ *MsucP S C f*

La demostracion de que $S_{S,\mathcal{C},f}$ es un elemento de $\mathcal{C}$ está basada en que $\mathcal{C}$ sea de carácter finito y que los elementos $S^n_{S,\mathcal{C},f}$ formen una cadena (ascendente) de elementos de $\mathcal{C}$, es decir, $S^n_{S,\mathcal{C},f} \in \mathcal{C}$ y $S^n_{S,\mathcal{C},f} \subseteq S^{n+1}_{S,\mathcal{C},f}$, para todo n. A continuación demostraremos de forma más general, que si $\mathcal{C}$ es una colección de conjuntos de carácter finito entonces la unión de los elementos de una cadena de conjuntos de $\mathcal{C}$ es también un conjunto de $\mathcal{C}$.

Definición 6.6.4 *Una cadena de conjuntos es cualquier sucesión* $\{S_n\}_n$ *de conjuntos ordenada por inclusión,* $S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$.

Su formalización es:

definition *es-cadena* :: (*nat* ⇒ *'a set*) ⇒ *bool* **where**
es-cadena S = (∀ *n*. *S n* ⊆ *S* (*Suc n*))

La unión $\bigcup_n S_n$ de los elementos de una cadena $\{S_n\}_n$, es llamada el límite de la cadena. La propiedad fundamental de las colecciones de conjuntos de carácter finito es que son cerradas bajo límites. Es decir, si $\mathcal{C}$ es una colección de conjuntos de carácter finito y $\{S_n\}_n$ es una cadena de elementos de $\mathcal{C}$ entonces, el límite $\bigcup_n S_n \in \mathcal{C}$.

Teorema 6.6.5 *Si $\mathcal{C}$ es una colección de conjuntos de carácter finito y $\{S_n\}_n$ es una cadena de elementos de $\mathcal{C}$ entonces, el límite de $\{S_n\}_n$ pertenece a $\mathcal{C}$.*

Su formalización es:

theorem *cadena-union-cerrado*:
assumes *hip1*: *caracter-finito* $\mathcal{C}$
and *hip2*:*es-cadena S*
and *hip3*: $\forall\, n.\ S\ n \in \mathcal{C}$
shows $(\bigcup n.\ S\ n) \in \mathcal{C}$

De esta forma, para demostrar que $\bigcup_n S^n_{S,\mathcal{C},f} \in \mathcal{C}$ como consecuencia del teorema anterior, basta probar que los elementos $S^n_{S,\mathcal{C},f}$ forman una cadena de elementos de $\mathcal{C}$.

Lema 6.6.6 *Sean S un conjunto de fórmulas, $\mathcal{C}$ una colección de conjuntos de fórmulas y f una sucesión de fórmulas. Entonces, la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ es una cadena.*

Su formalización es:

lemma *es-cadena-suc*: *es-cadena* (*sucP S* $\mathcal{C}$ *f*)
by (*simp add*: *es-cadena-def*) *blast*

Por último formalizamos el siguiente teorema que afirma que si $\mathcal{C}$ es una propiedad de consistencia de carácter finito y S está en $\mathcal{C}$ entonces, $S_{S,\mathcal{C},f}$ también está en $\mathcal{C}$.

Teorema 6.6.7 *Si $\mathcal{C}$ es una propiedad de consistencia proposicional de carácter finito, $S \in \mathcal{C}$ y f una sucesión de fórmulas, entonces $S_{S,\mathcal{C},f} \in \mathcal{C}$.*

Demostración: Por el lema 6.6.6, $\{S^n_{S,\mathcal{C},f}\}_n$ es una cadena. Además, de la hipótesis $S \in \mathcal{C}$, por inducción se demuestra que $S^n_{S,\mathcal{C},f} \in \mathcal{C}$, para todo n. De lo anterior y como $\mathcal{C}$ es una colección de carácter finito, usando el lema 6.6.5 se concluye que $S_{S,\mathcal{C},f} \in \mathcal{C}$.

□

Su formalización es:

theorem *MaxP-en-C*:
assumes *hip1*: *caracter-finito* $\mathcal{C}$ **and** *hip2*: $S \in \mathcal{C}$
shows *MsucP S* $\mathcal{C}$ *f* $\in \mathcal{C}$
proof (*unfold MsucP-def*)
have *es-cadena* (*sucP S* $\mathcal{C}$ *f*) **by** (*rule es-cadena-suc*)
moreover
have $\forall\, n.$ *sucP S* $\mathcal{C}$ *f n* $\in \mathcal{C}$

```
 proof (rule allI)
  fix n
  show sucP S C f n ∈ C using hip2
   by (induct n)(auto simp add: sucP-def)
 qed
 ultimately
 show (⋃ n. sucP S C f n) ∈ C by (rule cadena-union-cerrado[OF hip1])
qed
```

Para demostrar que $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$ no es necesario que $\mathcal{C}$ sea de carácter finito, sólo es necesario suponer que la colección $\mathcal{C}$ sea cerrada por subconjuntos. Sin embargo la sucesión f de fórmulas debe corresponder a una *enumeración* del conjunto de fórmulas del lenguaje proposicional que estemos considerando. Así, en el siguiente teorema demostramos que si tenemos una enumeración del conjunto de fórmulas de un lenguaje proposicional entonces, para cualquier colección de conjuntos de fórmulas $\mathcal{C}$ que sea cerrada por subconjuntos, se tiene que el límite $S_{S,\mathcal{C},f}$, es un elemento maximal de $\mathcal{C}$.

Definición 6.6.8 *Una* **enumeración** *de un conjunto A es cualquier función sobreyectiva $f\colon \mathbb{N} \to A$, donde $\mathbb{N}$ es el conjunto de los números naturales.*

Su formalización es:

```
definition enumeracion :: (nat ⇒ 'b) ⇒ bool where
 enumeracion f = (∀ y.∃ n. y = (f n))
```

Por ejemplo, la función identidad $i\colon \mathbb{N} \to \mathbb{N}$ es sobreyectiva. Luego es una enumeración de $\mathbb{N}$.

Lema 6.6.9 *Existe una enumeración g de $\mathbb{N}$.*

Su formalización es:

```
lemma enum-nat: ∃ g. enumeracion (g:: nat ⇒ nat)
proof −
 have ∀ y. ∃ n. y = (λn. n) n by simp
 hence enumeracion (λn. n) by (unfold enumeracion-def)
 thus ?thesis by auto
qed
```

Teorema 6.6.10 *Sean* **L** *un lenguaje proposicional, S un conjunto de fórmulas de* **L**, *$\mathcal{C}$ una colección de conjuntos de fórmulas de* **L** *cerrada por subconjuntos y f una enumeración del conjunto de fórmulas de* **L**. *Entonces $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$.*

Demostración:

Sea $S' \in \mathcal{C}$. Supongamos que $S_{S,\mathcal{C},f} \subseteq S'$, entonces $\forall n\ S^n_{S,\mathcal{C},f} \subseteq S'$. Demostramos por contradicción que $\bigcup_n S^n_{S,\mathcal{C},f} = S'$. Supongamos que $\bigcup_n S^n_{S,\mathcal{C},f} \neq S'$. Entonces, existe $z \in S'$ tal que $z \notin \bigcup_n S^n_{S,\mathcal{C},f}$. Luego, existe n tal que $z = f_n$, puesto que f es una enumeración del conjunto de fórmulas. Por lo tanto, $f_n \in S'$ y como $S^n_{S,\mathcal{C},f} \subseteq S'$, tenemos que $S^n_{S,\mathcal{C},f} \cup \{f_n\} \subseteq S'$. Luego, $S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C}$, puesto que $\mathcal{C}$ es cerrada por subconjuntos y $S' \in \mathcal{C}$. Así, por definición de la sucesión $S^n_{S,\mathcal{C},f}$, $f_n \in S^{n+1}_{S,\mathcal{C},f}$. Por otro lado, $f_n \notin S^{n+1}_{S,\mathcal{C},f}$, ya que $f_n = z$ y por hipótesis $z \notin \bigcup_n S^n_{S,\mathcal{C},f}$. De esta forma obtenemos una contradicción.

□

Su formalización es:

theorem *suc-maximalP*:
 assumes *hip1*: $\forall y.\ \exists n.\ y = f\ n$ **and** *hip2*: *subconj-cerrada* $\mathcal{C}$
 shows *maximal* (*MsucP S* $\mathcal{C}$ *f*) $\mathcal{C}$
proof (*simp add*: *maximal-def MsucP-def*)
 show $\forall S' \in \mathcal{C}.\ (\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x) \subseteq S' \longrightarrow (\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x) = S'$
 proof (*rule ballI impI*)+
  fix S'
  assume *h1*: $S' \in \mathcal{C}$ **and** *h2*: $(\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x) \subseteq S'$
  show $(\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x) = S'$
  proof (*rule ccontr*)
   assume $(\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x) \neq S'$
   hence $\exists z.\ z \in S' \wedge z \notin (\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x)$ **using** *h2* **by** *blast*
   then obtain *z* **where** *z*: $z \in S' \wedge z \notin (\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x)$ **by** (*rule exE*)
   have $\exists n.\ z = f\ n$ **using** *hip1 h1* **by** *simp*
   then obtain *n* **where** *n*: $z = f\ n$ **by** (*rule exE*)
   have $sucP\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \subseteq S'$
   proof −
    have $f\ n \in S'$ **using** *z n* **by** *simp*
    moreover
    have $sucP\ S\ \mathcal{C}\ f\ n \subseteq (\bigcup x.\ sucP\ S\ \mathcal{C}\ f\ x)$ **by** *auto*
    ultimately
    show *?thesis* **using** *h2* **by** *simp*
   qed
   hence $sucP\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \in \mathcal{C}$
    using *h1 hip2* **by** (*unfold subconj-cerrada-def*) *simp*
   hence $f\ n \in sucP\ S\ \mathcal{C}\ f\ (Suc\ n)$ **by** *simp*
   moreover
   have $\forall x.\ f\ n \notin sucP\ S\ \mathcal{C}\ f\ x$ **using** *z n* **by** *simp*
   hence $f\ n \notin sucP\ S\ \mathcal{C}\ f\ (Suc\ n)$ **by** (*rule-tac x* = *Suc n* **in** *allE*)
   ultimately
   show *False* **by** *simp*

```
  qed
 qed
qed
```

Corolario 6.6.11 *Sean* **L** *un lenguaje proposicional,* $\mathcal{C}$ *una colección de conjuntos de fórmulas de* **L** *de carácter finito,* S *un elemento de* $\mathcal{C}$ *y* f *una enumeración del conjunto de fórmulas de* **L**. *Entonces,*

(a) $S \subseteq S_{S,\mathcal{C},f}$.

(b) $S_{S,\mathcal{C},f} \in \mathcal{C}$.

(c) $S_{S,\mathcal{C},f}$ *es un elemento maximal de* $\mathcal{C}$.

Demostración:

(a) Por el teorema 6.6.3.

(b) Por el teorema 6.6.7, pues $\mathcal{C}$ es de carácter finito y S está en $\mathcal{C}$.

(c) Dado que $\mathcal{C}$ es de carácter finito, por el teorema 6.4.2, se tiene que $\mathcal{C}$ es cerrada por subconjuntos. De esto último y puesto que f es una enumeración del conjunto de fórmulas de **L**, por el teorema 6.6.10, obtenemos (c).

□

Su formalización es:

```
corollary ExtensionConsistenteP:
 assumes hip1: caracter-finito C
 and hip2: S ∈ C
 and hip3: ∀ y. ∃ n. y = f n
 shows S ⊆ MsucP S C f
 and MsucP S C f ∈ C
 and maximal (MsucP S C f) C
proof −
 show S ⊆ MsucP S C f using Max-subconjuntoP by auto
next
 show MsucP S C f ∈ C using  MaxP-en-C[OF hip1 hip2] by simp
next
 show maximal (MsucP S C f) C
  using caracter-finito-cerrado[OF hip1] and suc-maximalP[OF hip3]
  by simp
qed
```

6.7 Conjuntos de Hintikka y modelos de Hintikka

En esta sección introducimos el concepto de conjunto de Hintikka y demostramos que en un lenguaje proposicional, cualquier subconjunto de un *conjunto de Hintikka* es *satisfacible*.

6.7.1 Conjuntos de Hintikka

Definición 6.7.1 *Un conjunto H de fórmulas proposicionales es un* **conjunto de Hintikka** *si cumple las siguientes condiciones:*

1. *Para cualquier fórmula atómica P, $P \notin H$ o $\neg P \notin H$.*
2. *$\bot \notin H$ y $\neg\top \notin H$.*
3. *Si $\neg\neg F \in H$ entonces $F \in H$.*
4. *Si $\alpha \in H$ entonces, $\alpha_1 \in H$ y $\alpha_2 \in H$.*
5. *Si $\beta \in H$ entonces, $\beta_1 \in H$ o $\beta_2 \in H$.*

Su formalización es:

definition *hintikkaP* :: *'b formula1 set* ⇒ *bool* **where**
hintikkaP H = ((∀ *P*. ¬ (*Atomo P* ∈ *H* ∧ (¬.*Atomo P*) ∈ *H*)) ∧
FF ∉ *H* ∧ (¬.*TT*) ∉ *H* ∧
(∀ *F*. (¬.¬.*F*) ∈ *H* ⟶ *F* ∈ *H*) ∧
(∀ *F*. ((*FormulaAlfa F*) ∧ *F* ∈ *H*) ⟶
((*Comp1 F*) ∈ *H* ∧ (*Comp2 F*) ∈ *H*)) ∧
(∀ *F*. ((*FormulaBeta F*) ∧ *F* ∈ *H*) ⟶
((*Comp1 F*) ∈ *H* ∨ (*Comp2 F*) ∈ *H*)))

6.7.2 Satisfacibilidad de conjuntos de Hintikka

En esta sección mostramos que cualquier conjunto de Hintikka proposicional tiene un modelo (definición 4.1.12). Para la demostración, a cada conjunto de Hintikka le asociamos una *interpretación de Hintikka*.

Definición 6.7.2 *Sea H un conjunto de fórmulas proposicionales, definimos la interpretación I_H como sigue: sea P un símbolo proposicional, entonces*

$$I_H(P) = \begin{cases} \mathsf{V}, & \text{si } P \in H \\ \mathsf{F}, & \text{en otro caso.} \end{cases}$$

Si H es un conjunto de Hintikka, I_H es llamada una **interpretación de Hintikka**.

La formalización en Isabelle de la interpretacion I_H es la siguiente:

fun *IH* :: *'b formula1 set* ⇒ *'b* ⇒ *v-verdad1* **where**
IH H P = (*if Atomo P* ∈ *H then Verdad else Falso*)

A continuación demostramos que si H es de Hintikka entonces I_H es un modelo de H. Para la demostración probamos por inducción bien fundamentada sobre el *tamaño* de una fórmula la siguiente afirmación equivalente: *dado un conjunto de Hintikka H, si $F \in H$ entonces $I'_H(F) = \mathsf{V}$, y si $\neg F \in H$ entonces $I'_H(\neg F) = \mathsf{V}$.*

En general, si tenemos una relación $\prec$ bien fundamentada en un conjunto A y una propiedad P sobre A, la regla de inferencia por inducción bien fundamentada se enuncia de la siguiente manera:

$$\frac{\begin{array}{c}[\forall y(y \prec x \Rightarrow P(y))] \\ \vdots \\ P(x)\end{array}}{P(a)}$$

Es decir, para demostrar $P(a)$ para un elemento $a \in A$, es suficiente demostrar $P(x)$ para un x arbitrario usando como hipótesis que si $y \prec x$ entonces $P(y)$.

En Isabelle esta regla de inducción *wf-induct*, se expresa de la siguiente forma:

$$[\![wf\ r;\ \bigwedge x.\ \forall y.\ (y, x) \in r \longrightarrow P\ y \Longrightarrow P\ x]\!] \Longrightarrow P\ a$$

o también,

$$[\![wf\ r;\ \bigwedge x.\ \forall y.\ (y, x) \in r \longrightarrow P\ y \Longrightarrow P\ x]\!] \Longrightarrow P\ a$$

En donde *wf r* afirma que la relación r es *bien fundamentada*.

Teorema 6.7.3 *Sea H un conjunto de Hintikka. Si $F \in H$ entonces $I'_H(F) = \mathsf{V}$, y si $\neg F \in H$ entonces $I'_H(\neg F) = \mathsf{V}$.*

Demostración: La demostración es por inducción bien fundamentada. Para definir la relación bien fundamentada $\prec$ en el conjunto de fórmulas, consideramos la función *tamaño* definida en el capítulo 4 (definición 4.7.2) que determina el tamaño de una fórmula.

Así, la relación $\prec$ es igual a la imagen inversa de la relación en $\mathbb{N}$ inducida por la función de medida *tamaño*,

$$F \prec G \Longleftrightarrow \textit{tamaño}(F) < \textit{tamaño}(G).$$

Puesto que la relación $<$ (*menor que*) en $\mathbb{N}$ es bien fundamentada entonces, $\prec$ es bien fundamentada.

Sea F una fórmula arbitraria, la hipótesis de inducción afirma que si $G \prec F$ entonces, si $G \in H$ entonces $I'_H(G) = \mathsf{V}$, y si $\neg G \in H$ entonces $I'_H(\neg G) = \mathsf{V}$.

Hay que demostrar que, si $F \in H$, entonces $I'_H(F) = \mathsf{V}$ y si $\neg F \in H$, entonces $I'_H(\neg F) = \mathsf{V}$.

La demostración es por casos en la fórmula F.

Caso 1: Supongamos que F es la fórmula $\bot$.

Caso 1.a: Por hipótesis H es un conjunto de Hintikka, luego $\bot \notin H$. Por lo tanto la implicación, si $\bot \in H$ entonces $I'_H(\bot) = \mathsf{V}$, es verdadera.

Caso 1.b: Por definición, $I'_H(\neg\bot) = \neg I'_H(\bot) = \mathsf{V}$. Por lo tanto, la implicación, si $\neg\bot \in H$ entonces $\neg I'_H(\bot) = \mathsf{V}$, es verdadera.

Caso 2: Supongamos que F es la fórmula $\top$.

Caso 2.a: Por definición, $I'_H(\top) = \mathsf{V}$. Por lo tanto la implicación, si $\top \in H$ entonces $I'_H(\top) = \mathsf{V}$, es verdadera.

Caso 2.b: Por hipótesis H es un conjunto de Hintikka, luego $\neg\top \notin H$. Por lo tanto la implicación, si $\neg\top \in H$ entonces $I'_H(\neg\top) = \mathsf{V}$, es verdadera.

Caso 3: Supongamos que F es la fórmula atómica P.

Caso 3.a: Supongamos que $P \in H$, entonces, por definición de I_H, $I'_H(P) = \mathsf{V}$.

Caso 3.b: Supongamos que $\neg P \in H$. Puesto que H es de Hintikka, se tiene que $P \notin H$ o $\neg P \notin H$. Luego, $P \notin H$ ya que $\neg P \in H$. Por lo tanto $I'_H(P) = \mathsf{F}$, por definición de I_H. De esta forma, $I'_H(\neg P) = \neg(I'_H(P)) = \mathsf{V}$.

Caso 4: Sea F la fórmula $F_1 \wedge F_2$.

Caso 4.a: Supongamos que $F_1 \wedge F_2 \in H$. Puesto que H es de Hintikka se tiene que, $F_1 \in H$ y $F_2 \in H$. También, por la definición del tamaño de una fórmula se tiene que $F_1 \prec (F1 \wedge F_2)$ y $F_2 \prec (F_1 \wedge F_2)$, por consiguiente usando la hipótesis de inducción tenemos que las siguientes dos implicaciones son verdaderas:

Si $F_1 \in H$ entonces $I'_H(F_1) = \mathsf{V}$, y si $F_2 \in H$ entonces $I'_H(F_2) = \mathsf{V}$.

De esta forma, $I'_H(F_1) = \mathsf{V}$ y $I'_H(F_2) = \mathsf{V}$. Luego, por la definición del valor de una fórmula tenemos que, $I'_H(F_1 \wedge F_2) = I'_H(F_1) \wedge I'_H(F_2) = \mathsf{V}$.

Caso 4.b: Supongamos que $\neg(F_1 \wedge F_2) \in H$. Puesto que H es de Hintikka se tiene

que, $\neg F_1 \in H$ o $\neg F_2 \in H$. A partir de la anterior disyunción mostramos, por eliminación de la disyunción, que $I'_H(\neg(F_1 \wedge F_2)) = \mathsf{V}$.

Caso 4.b.1: Supongamos que $\neg F_1 \in H$. Por la definición del tamaño de una fórmula se tiene que $F_1 \prec \neg(F_1 \wedge F_2)$ y por lo tanto por la hipótesis de inducción tenemos que la implicación, si $\neg F_1 \in H$ entonces $I'_H(\neg F_1) = \mathsf{V}$ es verdadera. De lo anterior, se tiene que $I'_H(\neg F_1) = \mathsf{V}$, es decir, $\neg(I'_H(F_1)) = \mathsf{V}$. Por lo tanto, $I'_H(\neg(F_1 \wedge F_2)) = \mathsf{V}$.

Caso 4.b.2: Supongamos que $\neg F_2 \in H$. La demostración es igual que la del caso anterior.

Los otros casos, cuando F es $F_1 \vee F_2, F_1 \rightarrow F_2$ o $\neg F$, se demuestran en forma similar a los anteriores.

Los diferentes casos del teorema anterior se formalizan por medio de los siguientes lemas. La relación *measure tamano* en Isabelle corresponde a la formalización de la relación $\prec$.

lemma *casoP1*:
assumes *hip1*: *hintikkaP H*
and *hip2*: $\forall G.\ (G, FF) \in$ *measure tamano* $\longrightarrow$
$(G \in H \longrightarrow$ *valor1 I G* $=$ *Verdad*$) \wedge$
$((\neg.G) \in H \longrightarrow$ *valor1 I* $(\neg.G) =$ *Verdad*$)$
shows $(FF \in H \longrightarrow$ *valor1 I FF* $=$ *Verdad*$) \wedge$
$((\neg.FF) \in H \longrightarrow$ *valor1 I* $(\neg.FF) =$*Verdad*$)$

lemma *casoP2*:
assumes *hip1*: *hintikkaP H*
and *hip2*: $\forall G.\ (G, TT) \in$ *measure tamano* $\longrightarrow$
$(G \in H \longrightarrow$ *valor1 I G* $=$ *Verdad*$) \wedge$
$((\neg.G) \in H \longrightarrow$ *valor1 I* $(\neg.G) =$ *Verdad*$)$
shows $(TT \in H \longrightarrow$ *valor1 I TT* $=$ *Verdad*$) \wedge$
$((\neg.TT) \in H \longrightarrow$ *valor1 I* $(\neg.TT) =$*Verdad*$)$

lemma *casoP3*:
assumes *hip1*: *hintikkaP H*
and *hip2*: $\forall G.\ (G,$ *Atomo P*$) \in$ *measure tamano* $\longrightarrow$
$(G \in H \longrightarrow$ *valor1 I G* $=$ *Verdad*$) \wedge$
$((\neg.G) \in H \longrightarrow$ *valor1 I* $(\neg.G) =$ *Verdad*$)$
shows $($*Atomo P* $\in H \longrightarrow$ *valor1* $($*IH H*$)$ $($*Atomo P*$) =$ *Verdad*$) \wedge$
$((\neg.$*Atomo P*$) \in H \longrightarrow$ *valor1* $($*IH H*$)$ $(\neg.$*Atomo P*$) =$ *Verdad*$)$

lemma *casoP4*:
assumes *hip1*: *hintikkaP H*
and *hip2*: $\forall G.\ (G, F1 \wedge. F2) \in$ *measure tamano* $\longrightarrow$
$(G \in H \longrightarrow$ *valor1 I G* $=$ *Verdad*$) \wedge$

```
        ((¬.G) ∈ H ⟶ valor1 I (¬.G) = Verdad)
 shows ((F1 ∧. F2) ∈ H ⟶ valor1 I (F1 ∧. F2) = Verdad) ∧
     ((¬.(F1 ∧. F2)) ∈ H ⟶ valor1 I (¬.(F1 ∧. F2)) = Verdad)

lemma casoP5:
 assumes hip1: hintikkaP H
 and hip2: ∀ G. (G, F1 ∨. F2) ∈ measure tamano ⟶
        (G ∈ H ⟶ valor1 I G = Verdad) ∧
        ((¬.G) ∈ H ⟶ valor1 I (¬.G) = Verdad)
 shows ((F1 ∨. F2) ∈ H ⟶
      valor1 I (F1 ∨. F2) = Verdad) ∧
      ((¬.(F1 ∨. F2)) ∈ H ⟶ valor1 I (¬.(F1 ∨. F2)) = Verdad)

lemma casoP6:
 assumes hip1: hintikkaP H
 and hip2: ∀ G. (G, F1 →. F2) ∈ measure tamano ⟶
        (G ∈ H ⟶ valor1 I G = Verdad) ∧
        ((¬.G) ∈ H ⟶ valor1 I (¬.G) = Verdad)
 shows ((F1 →. F2) ∈ H ⟶ valor1 I (F1 →. F2) = Verdad) ∧
     ((¬.(F1 →. F2)) ∈ H ⟶ valor1 I (¬.(F1 →. F2)) = Verdad)

lemma casoP7:
 assumes hip1: hintikkaP H
 and hip2: ∀ G. (G, (¬.form)) ∈ measure tamano ⟶
        (G ∈ H ⟶ valor1 I G = Verdad) ∧
        ((¬.G) ∈ H ⟶ valor1 I (¬.G) = Verdad)
 shows ((¬.form) ∈ H ⟶
      valor1 I (¬.form) = Verdad) ∧
      ((¬.(¬.form)) ∈ H ⟶ valor1 I (¬.(¬.form)) = Verdad)
```

Usando los lemas anteriores, se tiene la formalización del teorema 6.7.3.

```
theorem hintikkaP-model-aux:
 assumes hip: hintikkaP H
 shows (F ∈ H ⟶ valor1 (IH H) F = Verdad) ∧
        ((¬.F) ∈ H ⟶ valor1 (IH H) (¬.F) = Verdad)
proof (rule-tac r=measure tamano and a=F in wf-induct)
 show wf(measure tamano) by simp
next
 fix F
 assume hip1: ∀ G. (G, F) ∈ measure tamano ⟶
        (G ∈ H ⟶ valor1 (IH H) G = Verdad) ∧
        ((¬.G) ∈ H ⟶ valor1 (IH H) (¬.G) = Verdad)
 show (F ∈ H ⟶ valor1 (IH H) F = Verdad) ∧
     ((¬.F) ∈ H ⟶ valor1 (IH H) (¬.F) = Verdad)
```

```
 proof (cases F)
  assume F=FF
  thus ?thesis using casoP1 hip hip1 by simp
 next
  assume F=TT
  thus ?thesis using casoP2 hip hip1 by auto
 next
  fix P
  assume F = Atomo P
  thus ?thesis using hip hip1 casoP3[of H P] by simp
 next
  fix F1  F2
  assume F= (F1 ∧. F2)
  thus ?thesis using hip hip1 casoP4[of H F1 F2] by simp
 next
  fix F1 F2
  assume F= (F1 ∨. F2)
  thus ?thesis using hip hip1 casoP5[of H F1 F2] by simp
 next
  fix F1 F2
  assume F= (F1 →. F2)
  thus ?thesis using hip hip1 casoP6[of H F1 F2] by simp
 next
  fix F1
  assume F=(¬.F1)
  thus ?thesis using hip hip1 casoP7[of H F1] by simp
 qed
qed
```

Por último, como un corolario del teorema anterior se demuestra que I_H es un modelo del conjunto de Hintikka H.

Corolario 6.7.4 *Sea H un conjunto de Hintikka. Si $F \in H$ entonces $I'_H(F) = \mathsf{V}$.*

Demostración: Es consecuencia inmediata del teorema 6.7.3

□

Su formalización es:

```
corollary ModeloHintikkaPa:
 assumes hintikkaP H and F ∈ H
 shows valor1 (IH H) F = Verdad
using assms hintikkaP-model-aux by auto
```

Corolario 6.7.5 *Sea H es un conjunto de Hintikka. Entonces, I_H es un modelo de H.*

Demostración: Es consecuencia inmediata de la definición de modelo y del corolario anterior.

□

Su formalización es:

```
corollary ModeloHintikkaP:
 assumes hintikkaP H
 shows (IH H) modelo H
proof (unfold modelo-def)
 show ∀ F∈H. valor1 (IH H) F = Verdad
 proof (rule ballI)
  fix F
  assume F ∈ H
  thus valor1 (IH H) F = Verdad using assms ModeloHintikkaPa  by auto
 qed
qed
```

De esta forma, cualquier conjunto de Hintikka es satisfacible.

Corolario 6.7.6 *Cualquier conjunto de Hintikka es satisfacible.*

Demostración: Es consecuencia inmediata de la definición de satisfacibilidad y del corolario 6.7.5.

□

Su formalización es:

```
corollary Hintikkasatisfacible:
 assumes hintikkaP H
 shows satisfacible H
using assms ModeloHintikkaP
by (unfold satisfacible-def, auto)
```

6.8 Completitud del método de prueba basado en tableros semánticos

Como una aplicación de que todo conjunto de Hintikka es satisfacible (corolario 6.7.6), formalizaremos en esta sección la completitud del método de prueba, basado en tableros

semánticos presentado en el capítulo 5. Es decir, demostramos que toda tautología tiene una prueba en este sistema siguiendo lo expuesto en la página 69 de [14].

Esta demostración está basada en las siguientes propiedades:

(a) para cualquier tablero T, las ramas de *PruebaTablero T* están compuestas únicamente de literales,

(b) toda rama no cerrada compuesta únicamente de literales es un conjunto de Hintikka.

Lema 6.8.1 *Sea T un tablero. Las fórmulas de cualquier rama de PruebaTablero T son literales.*

Su formalización es:

```
lemma LiteralesRamas:
 assumes R ∈ set (PruebaTablero T)
 shows ∀ F. F ∈ (set R) ⟶ FormulaLiteral F
```

Lema 6.8.2 *Sea S un conjunto de literales tal que para cualquier símbolo proposicional P, $P \notin S$ o $\neg P \notin S$, $\bot \notin S$ y $\neg\top \notin S$. Entonces, S es un conjunto de Hintikka.*

Su formalización es:

```
lemma LiteralHintikka:
 assumes hip1: (∀ P. ¬ (Atomo P ∈ S ∧ (¬. Atomo P) ∈ S))
 and hip2: FF ∉ S
 and hip3: ¬.TT ∉ S
 and hip4: ∀ F. F ∈ S ⟶ FormulaLiteral F
 shows hintikkaP S
proof −
 have (∀ P. ¬ (Atomo P ∈ S ∧ (¬. Atomo P) ∈ S)) using assms by simp
 moreover
 have FF ∉ S using assms by simp
 moreover
 have ¬.TT ∉ S using assms by simp
 moreover
 have ∀ F. (¬.¬.F) ∈ S ⟶ F ∈ S
 proof (rule allI impI)+
  fix F
  assume ¬.¬.F ∈ S
  hence FormulaLiteral (¬.¬.F) using hip4 by auto
```

```
  moreover
  have FormulaNoNo (¬.¬.F) by simp
  hence ¬ FormulaLiteral (¬.¬.F) by simp
  ultimately
  have False by simp
  thus F ∈ S by simp
 qed
 moreover
 have ∀ F. (FormulaAlfa F) ∧ F ∈ S ⟶ Comp1 F ∈ S ∧ Comp2 F ∈ S
 proof (rule allI impI)+
  fix F
  assume (FormulaAlfa F) ∧ F ∈ S
  hence a: FormulaAlfa F and b: F ∈ S by auto
  have  FormulaLiteral F  using b  hip4 by auto
  moreover
  have ¬ FormulaLiteral F using noLiteralAlfa a by auto
  ultimately
  have False by simp
  thus Comp1 F ∈ S ∧ Comp2 F ∈ S by simp
 qed
 moreover
 have ∀ F. (FormulaBeta F) ∧ F ∈ S ⟶ Comp1 F ∈ S ∨ Comp2 F ∈ S
 proof (rule allI impI)+
  fix F
  assume (FormulaBeta F) ∧ F ∈ S
  hence a: (FormulaBeta F) and b: F ∈ S by auto
  have  FormulaLiteral F using b hip4 by auto
  moreover
  have ¬ FormulaLiteral F using a noLiteralBeta by auto
  ultimately
  have False by simp
  thus Comp1 F ∈ S ∨ Comp2 F ∈ S by simp
 qed
 ultimately
 show ?thesis by (unfold hintikkaP-def, blast)
qed
```

Lema 6.8.3 *Toda rama no cerrada compuesta únicamente de literales es un conjunto de Hintikka.*

Su formalización es:

lemma *RamaNoCerradaHintikka1*:

```
 assumes hip1: ∀ F. F ∈ set R ⟶ FormulaLiteral F
 and hip2: ¬ Ramacerrada R
 shows hintikkaP (set R)
proof −
 have ∀ F. F ∈ set R ⟶ FormulaLiteral F using hip1
  by (auto simp add: multiset-eq-iff)
 moreover
 have (∀ P. ¬(Atomo P ∈ set R ∧ (¬. Atomo P) ∈ set R))
  and ¬(FF ∈ set R)
  and ¬(¬.TT ∈ set R)
  using hip2 Ramacerrada by auto
 hence (∀ P. ¬ (Atomo P ∈ set R ∧ (¬. Atomo P) ∈ set R))
  and (FF ∉ set R)
  and (¬.TT ∉ set R)
  by(auto simp add: multiset-eq-iff)
 ultimately
 show hintikkaP (set R) using LiteralHintikka by auto
qed
```

Los anteriores dos lemas permiten demostrar la siguiente propiedad.

Corolario 6.8.4 *Sea S un conjunto finito de fórmulas proposicionales y T el tablero (inicial) correspondiente al conjunto S. Entonces, las ramas no cerradas del tablero PruebaTablero T son conjuntos de Hintikka.*

Su formalización es:

```
corollary RamaNoCerradaHintikka:
 assumes hip1: R ∈ set (PruebaTablero [S])
 and hip2: ¬ Ramacerrada R
 shows hintikkaP (set R)
proof −
 have ∀ F. F ∈ set R ⟶ FormulaLiteral F using hip1 LiteralesRamas by simp
 thus ?thesis using hip2 RamaNoCerradaHintikka1 by auto
qed
```

Como una consecuencia del corolario anterior y del corolario 6.7.6 se demuestra el siguiente teorema:

Teorema 6.8.5 *Sea S un conjunto finito de fórmulas proposicionales y T el tablero (inicial) correspondiente al conjunto S. Si PruebaTablero T no es un tablero cerrado, entonces S es satisfacible.*

Su formalización es:

```
theorem TabSat1:
 assumes ¬TableroCerrado (PruebaTablero [S])
 shows satisfacible (set S)
proof −
 have ∃ R. R ∈ set (PruebaTablero [S]) ∧ ¬ Ramacerrada R using assms tabSat1 by auto
 then obtain R where R1: R ∈ set (PruebaTablero [S]) and R2: ¬ Ramacerrada R
  by auto
 hence hintikkaP (set R) using RamaNoCerradaHintikka by simp
 hence satisfacible (set R) using Hintikkasatisfacible by auto
 hence satisfacibleR R using EquiSatisfacible by auto
 hence ∃R. R ∈ set (PruebaTablero [S]) ∧ satisfacibleR R using R1 by blast
 hence satisfacibleT (PruebaTablero [S]) using satisfacibleRT by blast
 hence satisfacibleT [S] using PruebaTablero-satisfacible by auto
 hence ∃ R. R ∈ set [S] ∧ satisfacibleR R using satisfacibleTR by blast
 then obtain R where S1: R ∈ set [S] and S2: satisfacibleR R by auto
 have R=S using S1 perteneceIgual by simp
 hence satisfacibleR S using S2 by simp
 thus satisfacible (set S) using EquiSatisfacible by auto
qed
```

Por último se tiene el teorema de completitud del sistema de prueba basado en tableros.

Teorema 6.8.6 *Sea S un conjunto finito de fórmulas proposicionales y T el tablero (inicial) correspondiente al conjunto $S \cup \{\neg F\}$. Si F es una consecuencia lógica de S entonces el tablero PruebaTablero T es cerrado.*

Su formalización es la siguiente:

```
theorem CompletitudConsecuenciaTablero:
 assumes (set S)|=1 F
 shows TableroCerrado(PruebaTablero[S@[¬. F]])
proof −
 { assume ¬ TableroCerrado(PruebaTablero[S@[¬. F]])
  hence satisfacible (set (S@[¬. F])) using TabSat1 by blast
  hence ¬ (set S)|=1 F using EquiConsSat by auto
  hence False using assms by simp }
 thus ?thesis by blast
qed
```

Como una consecuencia del teorema anterior se demuestra que toda tautología tienen una prueba por tableros.

Corolario 6.8.7 *Supongamos que F es una tautología y T el tablero (inicial) correspondiente al conjunto* $\{\neg F\}$*. Entonces, el tablero PruebaTablero T es cerrado.*

Su formalización es:

corollary *CompletitudTautologiaTablero*:
assumes *tautologia F*
shows *TableroCerrado*(*PruebaTablero*[[¬. *F*]])
using *assms CompletitudConsecuenciaTablero*[*of* [] *F*] *CNS-tautologia*[*of F*]
by *simp*

6.9 Conjuntos maximales y conjuntos de Hintikka

En esta sección mostramos que si $\mathcal{C}$ es una propiedad de consistencia proposicional que además es cerrada por subconjuntos, M es un conjunto maximal de $\mathcal{C}$ y M pertenence a $\mathcal{C}$ entonces, M es un conjunto de Hintikka.

Teorema 6.9.1 *Sea* $\mathcal{C}$ *una colección de conjuntos de fórmulas tal que,*

- *(hip1)* $\mathcal{C}$ *es una propiedad de consistencia proposicional.*
- *(hip2)* M *es un elemento maximal de* $\mathcal{C}$*.*
- *(hip3)* $M \in \mathcal{C}$*.*

Entonces M *es un conjunto de Hintikka.*

Demostración:

A continuación mostramos que M cumple las propiedades que definen un conjunto de Hintikka (definición 6.7.1). Las propiedades (1) y (2) se cumplen por las hipótesis $(hip1)$ y $(hip3)$; las propiedades (3) a (5) se demuestran usando las hipótesis $(hip1)$, $(hip2)$ y $(hip3)$:

3. Supongamos que $\neg\neg F \in M$, hay que demostrar que $F \in M$. Tenemos que, si $\neg\neg F \in M$ entonces $M \cup \{F\} \in \mathcal{C}$, puesto que $\mathcal{C}$ es una propiedad de consistencia. También se tiene que para cualquier $S' \in \mathcal{C}$, si $M \subseteq S'$ entonces $M = S'$, puesto que M es un elemento maximal de $\mathcal{C}$.

De lo anterior y puesto que $M \subseteq M \cup \{F\}$ tenemos que $M = M \cup \{F\}$, y por lo tanto, $F \in M$.

4. Supongamos que $\alpha \in M$. Hay que demostrar que $\alpha_1 \in M$ y $\alpha_2 \in M$. Tenemos que, si $\alpha \in M$ entonces $M \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$, ya que $\mathcal{C}$ es propiedad de consistencia.

También se tiene que para cualquier $S' \in \mathcal{C}$, si $M \subseteq S'$ entonces $M = S'$, puesto que M es un elemento maximal de $\mathcal{C}$.

De lo anterior y puesto que $M \subseteq M \cup \{\alpha_1, \alpha_2\}$ tenemos que, $M = M \cup \{\alpha_1, \alpha_2\}$. Luego $\alpha_1 \in M$ y $\alpha_2 \in M$.

5. Supongamos que $\beta \in M$. Hay que demostrar que $\beta_1 \in M$ o $\beta_2 \in M$. Puesto que $\mathcal{C}$ es una propiedad de consistencia tenemos que, si $\beta \in M$ entonces, $M \cup \{\beta_1\} \in \mathcal{C}$ o $M \cup \{\beta_2\} \in \mathcal{C}$. A partir de la anterior disyunción demostramos por el método de eliminación de la disyunción que, $\beta_1 \in M$ o $\beta_2 \in M$.

(a) Supongamos que $M \cup \{\beta_1\} \in \mathcal{C}$. Puesto que M es un elemento maximal de $\mathcal{C}$ tenemos que para cualquier $S' \in \mathcal{C}$, si $M \subseteq S'$ entonces $M = S'$. De lo anterior y puesto que $M \subseteq M \cup \{\beta_1\}$ tenemos que, $M = M \cup \{\beta_1\}$. Luego $\beta_1 \in M$, y por lo tanto, $\beta_1 \in M$ o $\beta_2 \in M$.

(b) Supongamos que $M \cup \{\beta_2\} \in \mathcal{C}$. La demostración es igual que la del caso anterior.

□

Los siguientes lemas corresponden a la formalización de cada una de las partes del teorema anterior.

lemma *exten-hintikkaP1*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $\forall p. \neg (Atomo\ p \in M \wedge (\neg.Atomo\ p) \in M)$

lemma *exten-hintikkaP2*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $FF \notin M$

lemma *exten-hintikkaP3*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $(\neg.TT) \notin M$

lemma *exten-hintikkaP4*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: *maximal* $M\ \mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F. (\neg.\neg.F) \in M \longrightarrow F \in M$

lemma *exten-hintikkaP5*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: *maximal* $M\ \mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F. (FormulaAlfa\ F) \wedge F \in M \longrightarrow (Comp1\ F \in M \wedge Comp2\ F \in M)$

lemma *exten-hintikkaP6*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: *maximal* $M\ \mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F. (FormulaBeta\ F) \wedge F \in M \longrightarrow Comp1\ F \in M \vee Comp2\ F \in M$

Por último, tenemos la formalización del teorema 6.9.1.

theorem *MaximalHintikkaP*:
assumes *hip1*: *consistenciaP* $\mathcal{C}$ **and** *hip2*: *maximal* M $\mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows *hintikkaP* M
proof (*unfold hintikkaP-def*)
show $(\forall P.\ \neg\ (Atomo\ P \in M \wedge \neg.Atomo\ P \in M)) \wedge$
$FF \notin M \wedge$
$\neg.TT \notin M \wedge$
$(\forall F.\ \neg.\neg.F \in M \longrightarrow F \in M) \wedge$
$(\forall F.\ FormulaAlfa\ F \wedge F \in M \longrightarrow Comp1\ F \in M \wedge Comp2\ F \in M) \wedge$
$(\forall F.\ FormulaBeta\ F \wedge F \in M \longrightarrow Comp1\ F \in M \vee Comp2\ F \in M)$
using *exten-hintikkaP1*[*OF hip1 hip3*]
exten-hintikkaP2[*OF hip1 hip3*]
exten-hintikkaP3[*OF hip1 hip3*]
exten-hintikkaP4[*OF hip1 hip2 hip3*]
exten-hintikkaP5[*OF hip1 hip2 hip3*]
exten-hintikkaP6[*OF hip1 hip2 hip3*]
by *blast*
qed

6.10 Enumeración de fórmulas proposicionales

En esta sección mostramos una manera de enumerar las fórmulas de cualquier lenguaje de la lógica proposicional. La enumeración (definición 6.6.8) está basada en representar las fórmulas por medio de árboles binarios de números maturales. De esta forma, si se tiene una enumeración del conjunto de árboles binarios entonces se tiene una enumeración del conjunto de fórmulas del lenguaje dado.

Recordemos que una enumeración de un conjunto A es cualquier función sobreyectiva $f : \mathbb{N} \to A$ definición (6.6.8).

En forma equivalente, los siguientes dos lemas demuestran que, una función $f : \mathbb{N} \to A$ es una enumeración si existe una función g inversa por derecha de f.

Lema 6.10.1 *Si f es una funcíon sobreyectiva entonces, existe una función g inversa por derecha de f.*

Su formalización es

lemma *enum1*:
assumes $(\forall y.\exists x.\ y = (f\ x))$
shows $\exists g.\ \forall y.\ f\ (g\ y) = y$

Lema 6.10.2 *Si la función f tiene una inversa por derecha entonces, f es sobreyectiva.*

Su formalización es

lemma *enum2*:
assumes $\forall x. f(g\ x) = x$
shows $\forall y. \exists x.\ y = f\ x$

Así, tenemos otra forma equivalente de definir la noción de enumeración la cual se utilizará en esta sección:

Lema 6.10.3 $f : \mathbb{N} \to A$ *es una enumeración siysi tiene una inversa por derecha.*

Su formalización es

lemma *enumeracion*: *enumeracion f* $= (\exists g.\ \forall y. f(g\ y) = y)$
using *enum1 enum2*
by (*unfold enumeracion-def*) *blast*

6.10.1 Enumeración de árboles binarios

En esta sección mostramos una manera de enumerar los árboles binarios. Consideraremos formalmente el tipo de dato *árbol binario* donde sus elementos son árboles binarios cuyas hojas son números naturales.

datatype *arbolb* = *Hoja nat* | *Arbol arbolb arbolb*

La enumeración del tipo de dato árbol binario está basada en una enumeración del producto cartesiano $\mathbb{N} \times \mathbb{N}$ de los números naturales.

Enumeración del producto cartesiano $\mathbb{N} \times \mathbb{N}$

Una forma de ordenar el conjunto de los pares de números naturales es la siguiente:
$(0,0)$,
$(0,1)$, $(1,0)$,
$(0,2)$, $(1,1)$, $(2,0)$,
$(0,3)$, $(1,2)$, $(2,1)$, $(3,0)$,
....

Es decir, se ordenan por la suma de sus componentes y los que tienen la misma suma se ordenan por su primera componente. La función *diag*, que le asigna a cada número natural el par que ocupa la posición n en la anterior ordenación, se puede definir por recursión como sigue:

1. $diag(0) = (0,0)$
2. $diag(n+1) = \begin{cases} (0, x+1), & \text{si } diag(n) = (x,0) \\ (x+1, y), & \text{si } diag(n) = (x, y+1) \end{cases}$

Su formalización es

```
primrec diag :: nat ⇒ (nat × nat) where
  diag 0 = (0, 0)
| diag (Suc n) =
    (let (x, y) = diag n
     in case y of
        0 ⇒ (0, Suc x)
      | Suc y ⇒ (Suc x, y))
```

Para demostrar que *diag* es sobreyectiva, definimos la siguiente función *undiag* inversa (por derecha) de *diag*,

1. $undiag(0,0) = 0$
2. $undiag(0, y+1) = undiag(y,0) + 1$
3. $undiag(x+1, y) = undiag(x, y+1) + 1$

```
function undiag :: nat × nat ⇒ nat where
  undiag (0, 0) = 0
| undiag (0, Suc y) = Suc (undiag (y, 0))
| undiag (Suc x, y) = Suc (undiag (x, Suc y))
by pat-completeness auto

termination
  by (relation measure (λ(x, y). ((x + y) * (x + y + 1)) div 2 + x)) auto
```

Nótese que la función de medida de la función *undiag* está definida justamente por la expresión que define explícitamente a la función *undiag*.

El siguiente resultado demuestra que *undiag* es inversa por derecha de *diag* .

```
lemma diag-undiag [simp]: diag (undiag (x, y)) = (x, y)
by (rule undiag.induct) (simp add: Let-def)+
```

De esta forma, se tiene que la función *diag* es una enumeración de $\mathbb{N} \times \mathbb{N}$. Formalmente:

```
lemma enumeracion-natxnat: enumeracion (diag::nat ⇒ (nat × nat))
proof −
```

have $\forall\, x\ y.\ diag\ (undiag\ (x, y)) = (x, y)$ **using** *diag-undiag* **by** *auto*
hence $\exists\, undiag.\ \forall\, x\ y.\ diag\ (undiag\ (x, y)) = (x, y)$ **by** *blast*
thus *?thesis* **using** *enumeracion*[*of diag*] **by** *auto*
qed

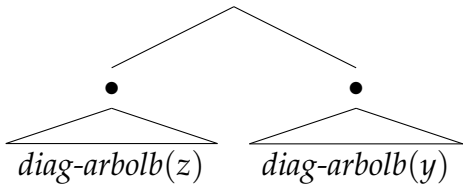
Enumeración del tipo de dato árbol binario

Con base a la enumeración *diag* del producto cartesiano $\mathbb{N} \times \mathbb{N}$, definimos la siguiente enumeración *diag-arbolb* : $\mathbb{N} \to$ *arbolb* del tipo de dato *arbolb*.

Dado $n \in \mathbb{N}$, supogamos que $diag(n) = (x, y)$.

1. Si $x = 0$ entonces $diag\text{-}arbolb(n)$ es el árbol

2. Si $x = z + 1$ para algún $z \in \mathbb{N}$ entonces, $diag\text{-}arbolb(n)$ es el árbol



function *diag-arbolb* :: *nat* ⇒ *arbolb* **where**
diag-arbolb n = (*case fst* (*diag n*) *of*
0 ⇒ *Hoja* (*snd* (*diag n*))
| *Suc z* ⇒ *Arbol* (*diag-arbolb z*) (*diag-arbolb* (*snd* (*diag n*))))
by *auto*

La siguiente función *undiag-arbolb* corresponde a una inversa (por derecha) de la función *diag-arbolb*.

primrec *undiag-arbolb* :: *arbolb* ⇒ *nat* **where**
undiag-arbolb (*Hoja n*) = *undiag* (*0*, *n*)
| *undiag-arbolb* (*Arbol t1 t2*) =
undiag (*Suc* (*undiag-arbolb t1*), *undiag-arbolb t2*)

El siguiente lema demuestra que *undiag-arbolb* es inversa por derecha de *diag-arbolb*.

lemma *diag-undiag-arbolb* [*simp*]: *diag-arbolb* (*undiag-arbolb t*) = *t*
by (*induct t*) (*simp-all add*: *Let-def*)

Por consiguiente, la función *diag-arbolb* es una enumeración del tipo de dato *arbolb*:

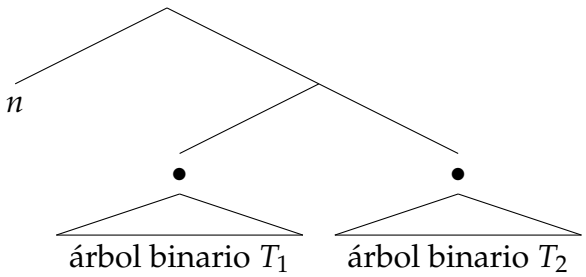
lemma *enumeracion-arbolb*: *enumeracion* (*diag-arbolb* :: *nat* ⇒ *arbolb*)
proof −
have ∀ *x*. *diag-arbolb* (*undiag-arbolb x*) = *x*
using *diag-undiag-arbolb* **by** *blast*
hence ∃ *undiag-arbolb*. ∀ *x* . *diag-arbolb* (*undiag-arbolb x*) = *x* **by** *blast*
thus *?thesis* **using** *enumeracion*[*of diag-arbolb*] **by** *auto*
qed

6.10.2 Enumeración de fórmulas proposicionales

Para enumerar las fórmulas de un lenguaje proposicional, mostramos inicialmente cómo asignar una fórmula a cada árbol binario que sea una hoja con valor $n \geq 0$ y, de forma recursiva, a cada árbol binario cuyo hijo izquierdo esté compuesto por una hoja con valor entre 0 y 4, del cual dependerá el tipo de fórmula que se le asocie.

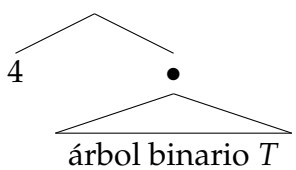
Sea g una sucesión de símbolos proposicionales:

1. (a) Al árbol compuesto por la hoja 0 le corresponde la fórmula $\bot$.
   (b) Al árbol compuesto por la hoja 1 le corresponde la fórmula $\top$.
   (c) Al árbol compuesto por la hoja $n + 2$ le corresponde la fórmula atómica P, donde $g(n) = P$.
2. (a) Al árbol,



   le corresponde una de las fórmulas $F \wedge G$, $F \vee G$, $F \rightarrow G$ dependiendo si $n = 1$, $n = 2$ ó $n = 3$, respectivamente. Donde F, G son las fórmulas correspondientes a los árboles T_1, T_2.

   (b) Al árbol,



le corresponde la fórmula $\neg F$. Donde F es la fórmula correspondiente al árbol T.

Obsérvese que no todos los árboles binarios tienen una fórmula asociada. La siguiente función *parcial* formaliza la correspondencia anterior.

```
fun formulaP-del-arbolb :: (nat ⇒ 'b) ⇒ arbolb ⇒ 'b formula1 where
 formulaP-del-arbolb g (Hoja 0) = FF
| formulaP-del-arbolb g (Hoja (Suc 0)) = TT
| formulaP-del-arbolb g (Hoja (Suc (Suc n))) = (Atomo (g n))
| formulaP-del-arbolb g (Arbol (Hoja (Suc 0)) (Arbol T1 T2)) =
   ((formulaP-del-arbolb g T1) ∧. (formulaP-del-arbolb g T2))
| formulaP-del-arbolb g (Arbol (Hoja (Suc (Suc 0))) (Arbol T1 T2)) =
   ((formulaP-del-arbolb g T1) ∨. (formulaP-del-arbolb g T2))
| formulaP-del-arbolb g (Arbol (Hoja (Suc (Suc (Suc 0)))) (Arbol T1 T2)) =
   ((formulaP-del-arbolb g T1) →. (formulaP-del-arbolb g T2))
| formulaP-del-arbolb g (Arbol (Hoja (Suc (Suc (Suc (Suc 0))))) T) =
   (¬. (formulaP-del-arbolb g T))
```

De manera recíproca, la siguiente función formaliza la inversa de la correspodencia anterior: dada una función g del conjunto de símbolos proposicionales al conjunto de números naturales, a cada fórmula proposicional se le asigna un árbol binario.

```
primrec arbolb-de-la-formulaP :: ('b ⇒ nat) ⇒  'b formula1 ⇒ arbolb where
 arbolb-de-la-formulaP  g FF = Hoja 0
| arbolb-de-la-formulaP g TT = Hoja (Suc 0)
| arbolb-de-la-formulaP g (Atomo P) = Hoja (Suc (Suc (g P)))
| arbolb-de-la-formulaP g (F ∧. G) = Arbol (Hoja (Suc 0))
   (Arbol (arbolb-de-la-formulaP g F) (arbolb-de-la-formulaP g G))
| arbolb-de-la-formulaP g (F ∨. G) = Arbol (Hoja (Suc (Suc 0)))
   (Arbol (arbolb-de-la-formulaP g F) (arbolb-de-la-formulaP g G))
| arbolb-de-la-formulaP g (F →. G) = Arbol (Hoja (Suc (Suc (Suc 0))))
   (Arbol (arbolb-de-la-formulaP g F) (arbolb-de-la-formulaP g G))
| arbolb-de-la-formulaP g (¬. F) = Arbol (Hoja (Suc (Suc (Suc (Suc 0)))))
   (arbolb-de-la-formulaP g F)
```

Con base a lo anterior, y usando la enumeración *diag-arbolb* del conjunto de árboles binarios, tenemos que en cualquier lenguaje proposicional y por cada enumeración g del conjunto $\mathcal{P}$ de símbolos proposicionales, construimos una enumeración Δ_g del correspondiente conjunto de fórmulas $\mathcal{F}$:

Definición 6.10.4 *La función $\Delta_g : \mathbb{N} \to \mathcal{F}$ está definida por,*

$$\Delta_g(n) = (formulaP\text{-}de\text{-}arbolb\ g)(diag\text{-}arbolb\ n).$$

Teorema 6.10.5 *Sea $g : \mathbb{N} \to \mathcal{P}$ una enumeración de los símbolos proposicionales del lenguaje* **L**. *Entonces, Δ_g es una enumeración del conjunto de fórmulas de* **L**.

Demostración: Por hipótesis y el lema 9.7.3, existe $g' : \mathcal{P} \to \mathbb{N}$ inversa a derecha de g. Entonces, la función $\Delta'_{g'} : \mathcal{F} \to \mathbb{N}$ definida por,

$$\Delta'_{g'}(F) = undiag\text{-}arbolb\ (arbolb\text{-}de\text{-}la\text{-}formulaP\ g'\ F),$$

es inversa por derecha de Δ_g. Es decir, $\Delta_g(\Delta'_{g'}(F)) = F$ para toda fórmula F.

□

Para la formalización del teorema anterior, definimos en Isabelle la función Δ_g y su inversa Δ'_g de la siguiente manera.

definition $\Delta P :: (nat \Rightarrow 'b) \Rightarrow nat \Rightarrow 'b\ formula1$ **where**
$\Delta P\ g\ n = formulaP\text{-}del\text{-}arbolb\ g\ (diag\text{-}arbolb\ n)$

definition $\Delta P' :: ('b \Rightarrow nat) \Rightarrow 'b\ formula1 \Rightarrow nat$ **where**
$\Delta P'\ g'\ F = undiag\text{-}arbolb\ (arbolb\text{-}de\text{-}la\text{-}formulaP\ g'\ F)$

theorem *enumeracionformulasP*[*simp*]:
assumes $\forall x.\ g(g'\ x) = x$
shows $\Delta P\ g\ (\Delta P'\ g'\ F) = F$
using *assms*
by (*induct F*)(*simp-all add*: Δ*P-def* $\Delta P'$*-def*)

Los siguientes corolarios muestran otras formas de expresar la propiedad de enumerabilidad anterior.

Corolario 6.10.6 *Sea $g : \mathbb{N} \to \mathcal{P}$ tal que, $\forall P \exists n\ (P = g(n))$. Entonces, $\forall F \exists n\ (F = \Delta_g(n))$.*

Demostración: Sea $g' : \mathcal{P} \to \mathbb{N}$ definida de la siguiente manera: dado $P \in \mathcal{P}$, definimos $g'(P) = n$, donde n es tal que $P = g(n)$.

De las definición anterior se tiene que $g(g'(P)) = P$ para todo $P \in \mathcal{P}$. Luego, por el teorema anterior, $\Delta_g(\Delta'_{g'}(F)) = F$ para toda formula F. De esta forma, para toda fórmula F existe n tal que, $F = \Delta_g(n)$, donde $n = \Delta'_{g'}(F)$.

□

Su formalización es:

corollary *EnumeracionFormulasP*:
assumes $\forall P.\ \exists\ n.\ P = g\ n$
shows $\forall F.\ \exists n.\ F = \Delta P\ g\ n$
proof (*rule allI*)

```
fix F
{ have ∀ P. P = g (SOME n. P = (g n))
  proof(rule allI)
    fix P
    obtain n where n: P=g(n) using assms by auto
    thus P = g (SOME n. P = (g n)) by (rule someI)
  qed }
hence ∀ P. g((λP. SOME n. P = (g n)) P) = P by simp
hence F = ΔP g (ΔP' (λP. SOME n. P = (g n)) F)
  using enumeracionformulasP by simp
thus ∃ n. F = ΔP g n
  by (rule-tac x= (ΔP' (λP. SOME n. P = (g n)) F) in exI)
qed
```

Corolario 6.10.7 *Si $g : \mathbb{N} \to \mathcal{P}$ es una enumeración del conjunto de símbolos proposicionales $\mathcal{P}$ de un lenguaje* **L**, *entonces la función $\Delta_g : \mathbb{N} \to \mathcal{F}$ (definición 6.10.6) es una enumeración del correspondiente conjuntos de fórmulas proposicionales $\mathcal{F}$.*

Demostración: Es consecuencia del corolario anterior y la definición de enumeración. □

Su formalización es:

```
corollary EnumeracionFormulasP1:
 assumes enumeracion (g:: nat ⇒ 'b)
 shows enumeracion ((ΔP g):: nat ⇒ 'b formula1)
proof −
 have ∀ P. ∃ n. P = g n using assms by(unfold enumeracion-def)
 hence ∀ F. ∃ n. F = ΔP g n using EnumeracionFormulasP by auto
 thus ?thesis by(unfold enumeracion-def)
qed
```

6.11 Formalización de la existencia de modelos proposicionales

En esta sección formalizamos la demostración del teorema de existencia de modelos en la lógica proposicional. Es decir, demostramos en Isabelle que cualquier conjunto *consistente S* de fórmulas proposicionales es *satisfacible*.

Teniendo en cuenta que el corolario 6.7.6 garantiza que cualquier conjunto de Hintikka es satisfacible, la demostración del teorema de existencia de modelos para lenguajes proposicionales se puede hacer basándonos en que todo conjunto consistente puede extenderse a un conjunto de Hintikka. Para la demostración de esto último se utilizará el teorema 6.9.1 que afirma que en una propiedad de consistencia proposicional $\mathcal{C}$ que sea cerrada por subconjuntos, si M es un conjunto maximal de $\mathcal{C}$ y pertenece a $\mathcal{C}$ entonces M es un conjunto de Hintikka. En particular, si el límite de la sucesión $S^n_{S,\mathcal{C},f}$ pertenece a $\mathcal{C}$ y es un conjunto maximal de $\mathcal{C}$, entonces dicho límite es un conjunto de Hintikka. Además, el corolario 6.6.11 afirma que estas dos últimas condiciones se tienen si la propiedad de consistencia $\mathcal{C}$ es de carácter finito y $S \in \mathcal{C}$.

De acuerdo con lo anterior, necesitamos extender una propiedad de consistencia proposicional a una de carácter finito (y por lo tanto cerrada por subconjuntos). Los teoremas 6.5.1 y 6.3.2 permiten hacer esta extensión:

Teorema 6.11.1 *Si $\mathcal{C}$ es una colección de conjuntos entonces se tiene lo siguiente.*

(a) $\mathcal{C} \subseteq \mathcal{C}^{+-}$.

(b) $\mathcal{C}^{+-}$ *es de carácter finito.*

(c) Si $\mathcal{C}$ es una propiedad de consistencia proposicional entonces, $\mathcal{C}^{+-}$ es una propiedad de consistencia proposicional.

Demostración:

(a) Por el teorema 6.3.2 tenemos que $\mathcal{C} \subseteq \mathcal{C}^{+}$ y que $\mathcal{C}^{+}$ es cerrada por subconjuntos. Además, por el teorema 6.5.1, $\mathcal{C}^{+} \subseteq \mathcal{C}^{+-}$. Por lo tanto, $\mathcal{C} \subseteq \mathcal{C}^{+-}$.

(b) Por el teorema 6.5.1 tenemos que $\mathcal{C}^{+-}$ es de carácter finito.

(c) Supongamos que $\mathcal{C}$ es una propiedad de consistencia proposicional. Entonces, por el teorema 6.3.2, $\mathcal{C}^{+}$ es una propiedad de consistencia proposicional cerrada por subconjuntos. Finalmente, por el teorema 6.5.1, $\mathcal{C}^{+-}$ es una propiedad de consistencia proposicional.

□

Su formalización es:

theorem *ExtensionCaracterFinitoP*:
shows $\mathcal{C} \subseteq \mathcal{C}^{+-}$
and *caracter-finito* $(\mathcal{C}^{+-})$

```
 and consistenciaP C ⟶ consistenciaP (C^{+-})
proof −
show C ⊆ C^{+-}
 proof −
  have C ⊆ C^+ using cerrado-subset by auto
  also
  have ... ⊆ C^{+-}
  proof −
   have subconj-cerrada (C^+) using cerrado-cerrado by auto
   thus ?thesis using caracter-finito-subset  by auto
  qed
  finally show ?thesis by simp
 qed
next
 show caracter-finito (C^{+-}) using caracter-finito by auto
next
 show consistenciaP C ⟶ consistenciaP (C^{+-})
 proof(rule impI)
  assume consistenciaP C
  hence  consistenciaP (C^+) using cerrado-consistenciaP by auto
  moreover
  have subconj-cerrada (C^+) using  cerrado-cerrado by auto
  ultimately
  show consistenciaP (C^{+-}) using cfinito-consistenciaP
   by auto
 qed
qed
```

Usando el teorema anterior se demuestra que $S_{S,\mathcal{C}^{+-},\Delta_g}$ pertenece a $\mathcal{C}^{+-}$ y es un elemento maximal de $\mathcal{C}^{+-}$ que contiene a S.

Lema 6.11.2 *Sean* **L** *un lenguaje proposicional, g una enumeración del conjunto de símbolos proposicionales de* **L** *y Δ_g la correspondiente enumeración de las fórmulas de* **L** *(definición 6.10.6). Si $\mathcal{C}$ es una propiedad de consistencia proposicional tal que $S \in \mathcal{C}$, entonces*

(a) $S \subseteq S_{S,\mathcal{C}^{+-},\Delta_g}$.

(b) $S_{S,\mathcal{C}^{+-},\Delta_g}$ *es un elemento maximal de* $\mathcal{C}^{+-}$.

(c) $S_{S,\mathcal{C}^{+-},\Delta_g} \in \mathcal{C}^{+-}$.

Demostración: Por el teorema 6.11.1 tenemos que, $\mathcal{C}^{+-}$ es una propiedad de consistencia de carácter finito, y también que $S \in \mathcal{C}^{+-}$. Además, por el corolario 6.10.6 se tiene

que la sucesión Δ_g es una enumeración del conjunto de fórmulas de **L**, De lo anterior y por el corolario 6.6.11, se tienen (a), (b) y (c).

□

Su formalización es:

```
lemma ExtensionConsistenteP1:
 assumes h'0: ∀ y. ∃ n. y = g n
 and h1: consistenciaP C
 and h2: S ∈ C
 shows S ⊆ MsucP S (C^+-) (ΔP g)
 and maximal (MsucP S (C^+-) (ΔP g)) (C^+-)
 and MsucP S (C^+-) (ΔP g) ∈ C^+-
proof −
 have consistenciaP (C^+-)
  using h1 and ExtensionCaracterFinitoP by auto
 moreover
 have caracter-finito (C^+-) using ExtensionCaracterFinitoP by auto
 moreover
 have S ∈ C^+-
  using h2 and ExtensionCaracterFinitoP by auto
 moreover
 have ∀ y. ∃ n. y = (ΔP g) n using EnumeracionFormulasP[OF h'0] by simp
 ultimately
 show S ⊆ MsucP S (C^+-) (ΔP g)
  and maximal (MsucP S (C^+-) (ΔP g)) (C^+-)
  and MsucP S (C^+-) (ΔP g) ∈ C^+-
  using ExtensionConsistenteP[of C^+-] by auto
qed
```

El siguiente resultado demuestra que si S es un conjunto consistente entonces $S_{S,\mathcal{C}^{+-},\Delta_g}$ es un conjunto de Hintikka.

Teorema 6.11.3 *Sean* **L** *un lenguaje proposicional, g una enumeración del conjunto de símbolos proposicionales de* **L** *y Δ_g la correspondiente enumeración de las fórmulas de* **L** *(definición 6.10.6). Si $\mathcal{C}$ es una propiedad de consistencia proposicional tal que $S \in \mathcal{C}$, entonces $S_{S,\mathcal{C}^{+-},\Delta_g}$ es un conjunto de Hintikka.*

Demostración: Por el teorema 6.11.1, se tiene que $\mathcal{C}^{+-}$ es una propiedad de consistencia y es de carácter finito; también es subconjunto cerrada (por el teorema 6.4.2). Luego por el teorema 6.11.2 se tiene que $S_{S,\mathcal{C}^{+-},\Delta_g}$ es un elemento maximal de $\mathcal{C}^{+-}$ y $S_{S,\mathcal{C}^{+-},\Delta_g} \in \mathcal{C}^{+-}$. De lo anterior y por el teorema 6.9.1 se tiene que $S_{S,\mathcal{C}^{+-},\Delta_g}$ es un conjunto de Hintikka.

□

Su formalización es:

theorem *HintikkaP*:
assumes *h0*: $\forall\ y.\ \exists\ n.\ y = g\ n$ **and** *h1*: *consistenciaP* $\mathcal{C}$ **and** *h2*: $S \in \mathcal{C}$
shows *hintikkaP* (*MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$))
proof −
have *1*: *consistenciaP* ($\mathcal{C}^{+-}$)
using *h1 ExtensionCaracterFinitoP* **by** *auto*
have *2*: *subconj-cerrada* ($\mathcal{C}^{+-}$)
proof −
have *caracter-finito* ($\mathcal{C}^{+-}$)
using *ExtensionCaracterFinitoP* **by** *auto*
thus *subconj-cerrada* ($\mathcal{C}^{+-}$) **by** (*rule caracter-finito-cerrado*)
qed
have *3*: *maximal* (*MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$)) ($\mathcal{C}^{+-}$)
and *4*: *MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$) $\in \mathcal{C}^{+-}$
using *ExtensionConsistenteP1*[*OF h0 h1 h2*] **by** *auto*
show *?thesis*
using *1* **and** *2* **and** *3* **and** *4* **and** *MaximalHintikkaP*[*of* $\mathcal{C}^{+-}$] **by** *simp*
qed

Por último, el siguiente teorema garantiza la existencia de modelos.

Teorema 6.11.4 *Sean* **L** *un lenguaje proposicional, g una enumeración del conjunto de símbolos proposicionales de* **L** *y* Δ_g *la correspondiente enumeración de las fórmulas de* **L** *(definición 6.10.6). Si* $\mathcal{C}$ *es una propiedad de consistencia proposicional tal que* $S \in \mathcal{C}$ *y* $F \in S$*, sean* $H = S_{S,\mathcal{C}^{+-},\Delta_g}$ *e* I_H *la interpretación de Hintikka definida en 6.7.2. Entonces,* $I'_H(F) = \mathsf{V}$.

Demostración: Por el teorema 6.11.3, $S_{S,\mathcal{C}^{+-},\Delta_g}$ es de Hintikka. Además por el teorema 6.6.3 se tiene que $F \in S_{S,\mathcal{C}^{+-},\Delta_g}$. De lo anterior y por el corolario 6.7.5 se tiene el teorema. □

Su formalización es:

theorem *ExistenciaModeloP*:
assumes *h0*: $\forall y.\ \exists n.\ y = g\ n$
and *h1*: *consistenciaP* $\mathcal{C}$
and *h2*: $S \in \mathcal{C}$
and *h3*: $F \in S$
shows *valor1* (*IH* (*MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$))) F = *Verdad*
proof (*rule ModeloHintikkaPa*)
show *hintikkaP* (*MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$))
using *h0* **and** *h1* **and** *h2* **by**(*rule HintikkaP*)
next
show $F \in$ *MsucP S* ($\mathcal{C}^{+-}$) ($\Delta P\ g$)

using *h3 Max-subconjuntoP* **by** *auto*
qed

En el caso del lenguaje proposicional **L** en el que identificamos los símbolos proposicionales con los números naturales, tenemos que la función identidad $i(n) = n$, es una enumeración del conjunto de símbolos proposicionales. En este caso la función Δ_i es una enumeración de las fórmulas de este lenguaje y se tiene el siguiente corolario.

Corolario 6.11.5 *Sea* **L** *el lenguaje proposicional en el que los símbolos proposicionales son los números naturales. Si $\mathcal{C}$ una propiedad de consistencia proposicional tal que $S \in \mathcal{C}$ y $F \in S$, sean i la función identidad en $\mathbb{N}$, $H = S_{S,\mathcal{C}^{+-},\Delta_i}$ e I_H la interpretación de Hintikka definida en 6.7.2, entonces, $I'_H(F) = \mathsf{V}$.*

Demostración: Por hipótesis, la función identidad *i* es una enumeración de los símbolos proposicionales de **L**. De esta forma, por el teorema 6.11.4 se tiene que $I'_H(F) = \mathsf{V}$.

□

Su formalización es:

corollary *CasoExistenciaModeloP*:
assumes *h1*: *consistenciaP* $\mathcal{C}$ **and** *h2*: $S \in \mathcal{C}$ **and** *h3*: $F \in S$
shows *valor1* (*IH* (*MsucP S* ($\mathcal{C}^{+-}$) (*ΔP* (λ*n*. *n*)))) F = *Verdad*
proof −
have *h0*: ∀ *y*. ∃ *n*. *y* = (λ*n*. *n*) *n* **by** *simp*
show *?thesis* **using** *ExistenciaModeloP*[*OF h0 h1 h2 h3*] **by** *simp*
qed

Por último, como una consecuencia del teorema 6.11.4 tenemos que todo conjunto consistente es satisfacible.

Corolario 6.11.6 *Sea* **L** *un lenguaje proposicional tal que existe una enumeración del conjunto de símbolos proposicionales. Si $\mathcal{C}$ es una propiedad de consistencia proposicional y $S \in \mathcal{C}$, entonces S es satisfacible.*

Demostración: Es consecuencia del teorema 6.11.4 y de la definición de satisfacibilidad.

□

Su formalización es:

corollary *ConjuntosatisfacibleP*:
assumes *h1*: ∃*g*. *enumeracion* (*g*:: *nat* ⇒ *'b*)
and *h2*: *consistenciaP* $\mathcal{C}$
and *h3*: (*S*:: *'b formula1 set*) ∈ $\mathcal{C}$
shows *satisfacible S*

```
proof −
 obtain g where g: ∀ y. ∃ n. y = (g::nat ⇒ 'b) n
  using h1 by (unfold enumeracion-def) auto
 { fix F
  assume hip: F ∈ S
  have valor1 (IH (MsucP S (C^+-) (ΔP g))) F = Verdad
   using g h2 h3 ExistenciaModeloP hip by blast }
 hence ∀ F∈S. valor1 (IH (MsucP S (C^+-) (ΔP g))) F = Verdad
  by (rule ballI)
 hence ∃ I. ∀ F ∈ S. valor1 I F = Verdad by auto
 thus satisfacible S by(unfold satisfacible-def, unfold modelo-def)
qed
```

En el caso del lenguaje proposicional en el que identificamos los símbolos proposicionales con los números naturales, tenemos el siguiente corolario.

Corolario 6.11.7 *Sean* **L** *el lenguaje proposicional en el que los símbolos proposicionales son los números naturales y* $\mathcal{C}$ *una propiedad de consistencia proposicional. Si* $S \in \mathcal{C}$*, entonces* S *es satisfacible.*

Demostración: Puesto que existe una enumeración de $\mathbb{N}$ (lema 6.6.9), entonces por el coralario anterior, se tiene el resultado.

□

Su formalización es:

```
corollary Conjuntosatisfacible1P:
 assumes consistenciaP C and (S:: nat formula1 set) ∈ C
 shows satisfacible S
   using enum-nat assms ConjuntosatisfacibleP by auto
```

Capítulo 7

Aplicaciones del teorema de existencia de modelos proposicionales

En este capítulo ilustramos la utilización del teorema de existencia de modelos en la formalización de importantes metateoremas de la lógica proposicional, específicamente formalizamos las pruebas del *teorema de compacidad* y el *teorema de interpolación de Craig* en la lógica proposicional siguiendo la exposición del libro de Fitting [14] (páginas 62-63).

7.1 Teorema de compacidad

La demostración del teorema de compacidad que se presenta en los textos clásicos de introducción a la lógica utiliza el *lema de König*. El teorema de existencia de modelos permite otra forma de demostrar este teorema a partir del siguiente lema:

Lema 7.1.1 *La colección de conjuntos de fórmulas proposicionales,*

$$\mathcal{C} = \{W | \forall A\, (A \subseteq W \wedge A \text{ es finito } \rightarrow A \text{ es satisfacible})\}$$

es una propiedad de consistencia.

Demostración: Sea $W \in \mathcal{C}$. A continuación demostramos que W verifica las condiciones que definen una propiedad de consistencia proposicional.

(1). Sea P un símbolo proposicional. Puesto que el conjunto $\{P, \neg P\}$ es finito y no satisfacible entonces, $\{P, \neg P\} \nsubseteq W$. Así $P \notin W$ o $\neg P \notin W$.

(2) y (3). Puesto que los conjuntos $\{\bot\}$ y $\{\neg\top\}$ son finitos y no satisfacibles entonces, $\{\bot\} \nsubseteq W$ y $\{\neg\top\} \nsubseteq W$. Así $\bot, \neg\top \notin W$.

(4). Sea $\neg\neg F \in W$. Hay que demostrar que $W \cup \{F\} \in \mathcal{C}$; es decir, hay que demostrar que todo subconjunto finito de $W \cup \{F\}$ es satisfacible. Supongamos que A es un subconjunto finito de $W \cup \{F\}$. Entonces $A - \{F\}$ es un subconjunto finito de W y por tanto $(A - \{F\}) \cup \{\neg\neg F\}$ tambi'en lo es. Por consiguiente, $(A - \{F\}) \cup \{\neg\neg F\}$ es satisfacible. Luego existe una interpretación I, tal que para toda fórmula G de $(A - \{F\}) \cup \{\neg\neg F\}$, $I'(G) = \mathsf{V}$. Además, puesto que F y $\neg\neg F$ son equivalentes, se tiene que $I'(F) = I'(\neg\neg F) = \mathsf{V}$. Así, $I'(G) = \mathsf{V}$ para toda fórmula G de $(A - \{F\}) \cup \{F\}$. Luego $(A - \{F\}) \cup \{F\}$ es satisfacible y como $A \subseteq (A - \{F\}) \cup \{F\}$, se tiene que A es satisfacible.

(5). Supongamos que $\alpha \in W$. Hay que demostrar que $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$; es decir, hay que demostrar que todo subconjunto finito de $W \cup \{\alpha_1, \alpha_2\}$ es satisfacible. Supongamos que A es un subconjunto finito de $W \cup \{\alpha_1, \alpha_2\}$. Entonces, $A - \{\alpha_1, \alpha_2\}$ es un subconjunto finito de W y por tanto $(A - \{\alpha_1, \alpha_2\}) \cup \{\alpha\}$ también lo es. finito de W. Luego $(A - \{\alpha_1, \alpha_2\}) \cup \{\alpha\}$ es satisfacible. Por consiguiente, existe una interpretación I tal que para toda fórmula F de $(A - \{\alpha_1, \alpha_2\}) \cup \{\alpha\}$, $I'(F) = \mathsf{V}$. Además, puesto que α y $\alpha_1 \wedge \alpha_2$ son equivalentes, se tiene que $I'(\alpha_1) = I'(\alpha_2) = I'(\alpha) = \mathsf{V}$. Así, $I'(F) = \mathsf{V}$ para toda fórmula F de $(A - \{\alpha_1, \alpha_2\}) \cup \{\alpha_1, \alpha_2\}$. Luego $(A - \{\alpha_1, \alpha_2\}) \cup \{\alpha_1, \alpha_2\}$ es satisfacible y por tanto, A es satisfacible.

(6). Supongamos que $\beta \in W$. Hay que demostrar que $W \cup \{\beta_1\} \in \mathcal{C}$ o $W \cup \{\beta_2\} \in \mathcal{C}$; es decir, hay que demostrar que todo subconjunto finito de $W \cup \{\beta_1\}$ es satisfacible o que todo subconjunto finito de $W \cup \{\beta_2\}$ es satisfacible. La demostración es por contradicción. Supongamos que, existe un subconjunto finito A de $W \cup \{\beta 1\}$ no satisfacible y un subconjunto finito B de $W \cup \{\beta_2\}$ no satisfacible. Entonces $A - \{\beta_1\}$ y $B - \{\beta_2\}$ son subconjuntos finitos de W y por tanto $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta\}$ es un subconjunto finito de W. Luego, $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta\}$ es satisfacible. Por consiguiente, existe una interpretación I tal que para toda fórmula F de $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta\}$, $I(F) = \mathsf{V}$. En particular, $I(\beta) = \mathsf{V}$. Luego, puesto que β y $\beta_1 \vee \beta_2$ son equivalentes, se tiene que $I(\beta_1) = \mathsf{V}$ o $I(\beta_2) = \mathsf{V}$.

Así, para toda fórmula F de $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_1\}$, $I'(F) = \mathsf{V}$ o para toda fórmula F de $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_2\}$, $I'(F) = \mathsf{V}$.

Luego, $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_1\}$ o $(A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_2\}$ es satisfacible. De esta forma, puesto que $A \subseteq (A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_1\}$ y $B \subseteq (A - \{\beta_1\}) \cup (B - \{\beta_2\}) \cup \{\beta_2\}$, se tiene que A es satisfacible o B es satisfacible. Esto último contradice la hipótesis de que A y B no son satifacibles.

□

Los siguientes lemas permiten formalizar la prueba del lema anterior.

En la demostración de (1) se utilizó la propiedad "$\{F, \neg F\}$ es insatisfacible para cualquier fórmula F". En Isabelle,

lemma *NosatisfacibleAtom*:
shows $\neg$(*satisfacible* {*F*, $\neg$.*F*})

La formalización de la prueba de la propiedad (1) es la siguiente:

lemma *comp1*:
assumes $\forall$ (*A*::'*b formula1 set*). (*A*$\subseteq$ *W* $\wedge$ *finite A*) $\longrightarrow$ *satisfacible A*
shows ($\forall$ *P*. $\neg$ (*Atomo P* $\in$ *W* $\wedge$ ($\neg$. *Atomo P*) $\in$ *W*))
proof (*rule allI notI*)+
fix *P*
assume *h1*: *Atomo P* $\in$ *W* $\wedge$ ($\neg$.*Atomo P*) $\in$ *W*
show *False*
proof −
have {*Atomo P*, ($\neg$.*Atomo P*)} $\subseteq$ *W* **using** *h1* **by** *simp*
moreover
have *finite* {*Atomo P*, ($\neg$.*Atomo P*)} **by** *simp*
ultimately
have {*Atomo P*, ($\neg$.*Atomo P*)} $\subseteq$ *W* $\wedge$ *finite* {*Atomo P*, ($\neg$.*Atomo P*)} **by** *simp*
moreover
have ({*Atomo P*, ($\neg$.*Atomo P*)}$\subseteq$ *W* $\wedge$ *finite* {*Atomo P*, ($\neg$.*Atomo P*)}) $\longrightarrow$
satisfacible {*Atomo P*, ($\neg$.*Atomo P*)}
using *assms* **by**(*rule-tac x* = {*Atomo P*, ($\neg$.*Atomo P*)} **in** *allE*, *auto*)
ultimately
have *satisfacible* {*Atomo P*, ($\neg$.*Atomo P*)} **by** *simp*
thus *False* **using** *NosatisfacibleAtom* **by** *auto*
qed
qed

En la demostración de (2) se utilizó la propiedad, "{$\perp$} es insatisfacible". En Isabelle,

lemma *NosatisfacibleFF*:
shows $\neg$ (*satisfacible* {*FF*})

La formalización de la prueba de la propiedad (2) es la siguiente:

lemma *comp2*:
assumes $\forall$ (*A*::'*b formula1 set*). (*A*$\subseteq$ *W* $\wedge$ *finite A*) $\longrightarrow$ *satisfacible A*
shows *FF* $\notin$ *W*
proof (*rule notI*)
assume *hip*: *FF* $\in$ *W*
show *False*
proof −
have {*FF*} $\subseteq$ *W* **using** *hip* **by** *simp*
moreover
have *finite* {*FF*} **by** *simp*

```
  ultimately
  have {FF} ⊆ W ∧ finite {FF} by simp
  moreover
  have ({FF::'b formula1} ⊆ W ∧ finite {FF}) ⟶
      satisfacible {FF::'b formula1}
   using assms by(rule-tac x = {FF::'b formula1} in allE, auto)
  ultimately
  have satisfacible {FF::'b formula1} by simp
  thus False using NosatisfacibleFF by auto
 qed
qed
```

En la demostración de (3) se utilizó la propiedad, "$\{\neg\top\}$ es insatisfacible". Su formalización en Isabelle es,

```
lemma NosatisfacibleFFa:
 shows ¬ (satisfacible {¬.TT})
```

La formalización de la prueba de la propiedad (3) es la siguiente:

```
lemma comp3:
 assumes ∀ (A::'b formula1 set). (A⊆ W ∧ finite A) ⟶ satisfacible A
 shows ¬.TT ∉ W
proof (rule notI)
 assume hip: ¬.TT ∈ W
 show False
 proof −
  have {¬.TT} ⊆ W using hip by simp
  moreover
  have finite {¬.TT} by simp
  ultimately
  have {¬.TT} ⊆ W ∧ finite {¬.TT} by simp
  moreover
  have ({¬.TT::'b formula1} ⊆ W ∧ finite {¬.TT}) ⟶
      satisfacible {¬.TT::'b formula1}
   using assms by(rule-tac x = {¬.TT::'b formula1} in allE, auto)
  ultimately
  have satisfacible {¬.TT::'b formula1} by simp
  thus False using NosatisfacibleFFa by auto
 qed
qed
```

En la prueba de (4) se utilizaron la siguientes propiedades: "los subconjuntos de un conjunto satisfacible son satisfacibles" y "si $S \cup \{\neg\neg F\}$ es satisfacible, entonces $S \cup \{F\}$ es satisfacible". En Isabelle se expresan como sigue:

lemma *SubSatis*:
assumes *hip1*: *satisfacible S* **and** *hip2*: $S' \subseteq S$
shows *satisfacible* S'

lemma *satisfacibleUnion1*:
assumes *satisfacible* $(A \cup \{\neg.\neg.F\})$
shows *satisfacible* $(A \cup \{F\})$

La formalización de la prueba de la propiedad (4) es la siguiente:

lemma *comp4*:
assumes *hip1*: $\forall$ $(A::'b\ formula1\ set).\ (A \subseteq W \wedge finite\ A) \longrightarrow satisfacible\ A$
and *hip2*: $\neg.\neg.F \in W$
shows $\forall$ $(A::'b\ formula1\ set).\ (A \subseteq W \cup \{F\} \wedge finite\ A) \longrightarrow satisfacible\ A$
proof (*rule allI*, *rule impI*)+
fix A
assume *hip*: $A \subseteq W \cup \{F\} \wedge finite\ A$
show *satisfacible A*
proof −
have $A - \{F\} \subseteq W \wedge finite\ (A - \{F\})$ **using** *hip* **by** *auto*
hence $(A - \{F\}) \cup \{\neg.\neg.F\} \subseteq W \wedge finite\ ((A - \{F\}) \cup \{\neg.\neg.F\})$
using *hip2* **by** *auto*
hence *satisfacible* $((A - \{F\}) \cup \{\neg.\neg.F\})$ **using** *hip1* **by** *auto*
hence *satisfacible* $((A - \{F\}) \cup \{F\})$ **using** *satisfacibleUnion1* **by** *blast*
moreover
have $A \subseteq (A - \{F\}) \cup \{F\}$ **by** *auto*
ultimately
show *satisfacible A* **using** *SubSatis* **by** *auto*
qed
qed

En la prueba de (5) se utilizó la propiedad "si $A \cup \{\alpha\}$ es satisfacible entonces $A \cup \{\alpha_1, \alpha_2\}$ también lo es". En Isabelle,

lemma *satisfacibleUnion2*:
assumes *hip1*: *FormulaAlfa F* **and** *hip2*: *satisfacible* $(A \cup \{F\})$
shows *satisfacible* $(A \cup \{Comp1\ F, Comp2\ F\})$

La formalización de la prueba de la propiedad (5) es la siguiente:

lemma *comp5*:
assumes *hip0*: *FormulaAlfa F*
and *hip1*: $\forall$ $(A::'b\ formula1\ set).\ (A \subseteq W \wedge finite\ A) \longrightarrow satisfacible\ A$
and *hip2*: $F \in W$
shows $\forall$ $(A::'b\ formula1\ set).\ (A \subseteq W \cup \{Comp1\ F, Comp2\ F\} \wedge finite\ A) \longrightarrow$
satisfacible A

```
proof (rule allI, rule impI)+
 fix A
 assume hip: A ⊆ W ∪ {Comp1 F, Comp2 F} ∧ finite A
 show satisfacible A
 proof −
  have A−{Comp1 F, Comp2 F} ⊆ W ∧ finite (A−{Comp1 F, Comp2 F})
   using hip by auto
  hence (A−{Comp1 F, Comp2 F}) ∪ {F} ⊆ W ∧
      finite ((A−{Comp1 F, Comp2 F}) ∪ {F})
   using hip2 by auto
  hence satisfacible ((A−{Comp1 F, Comp2 F}) ∪ {F})
   using hip1 by auto
  hence satisfacible ((A−{Comp1 F, Comp2 F}) ∪ {Comp1 F, Comp2 F})
   using hip0 satisfacibleUnion2 by auto
  moreover
  have A ⊆ (A−{Comp1 F, Comp2 F}) ∪ {Comp1 F, Comp2 F} by auto
  ultimately
  show satisfacible A using SubSatis by auto
 qed
qed
```

En la prueba de (6) se utilizó la propiedad "si $A \cup \{\beta\}$ es satisfacible entonces $A \cup \{\beta_1\}$ o $A \cup \{\beta_2\}$ es satisfacible". En Isabelle,

```
lemma satisfacibleUnion3:
 assumes hip1: FormulaBeta F and hip2: satisfacible (A ∪ {F})
 shows satisfacible (A ∪ {Comp1 F}) ∨ satisfacible (A ∪ {Comp2 F})
```

La formalización de la prueba de la propiedad (6) es la siguiente:

```
lemma comp6:
 assumes hip0: FormulaBeta F
 and hip1: ∀ (A::'b formula1 set). (A⊆ W ∧ finite A) ⟶ satisfacible A
 and hip2: F ∈ W
 shows (∀ (A::'b formula1 set). (A⊆ W ∪ {Comp1 F} ∧ finite A) ⟶
 satisfacible A) ∨
 (∀ (A::'b formula1 set). (A⊆ W ∪ {Comp2 F} ∧ finite A) ⟶
 satisfacible A)
proof −
 { assume hip3:¬((∀ (A::'b formula1 set). (A⊆ W ∪ {Comp1 F} ∧ finite A) ⟶
  satisfacible A) ∨
  (∀ (A::'b formula1 set). (A⊆ W ∪ {Comp2 F} ∧ finite A) ⟶
  satisfacible A))
  have False
  proof −
```

```
    obtain A B where A1: A ⊆ W ∪ {Comp1 F}
      and A2: finite A
      and A3: ¬ satisfacible A
      and B1: B ⊆ W ∪ {Comp2 F}
      and B2: finite B
      and B3: ¬ satisfacible B
      using hip3 by auto
    have a1: A − {Comp1 F} ⊆ W
      and a2: finite (A − {Comp1 F})
      using A1 and A2 by auto
    hence satisfacible (A − {Comp1 F}) using hip1 by simp
    have b1: B − {Comp2 F} ⊆ W
      and b2: finite (B − {Comp2 F})
      using B1 and B2 by auto
    hence satisfacible (B − {Comp2 F}) using hip1 by simp
    moreover
    have (A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {F} ⊆ W
      and finite ((A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {F})
      using a1 a2 b1 b2 hip2 by auto
    hence satisfacible ((A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {F})
      using hip1 by simp
    hence satisfacible ((A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {Comp1 F}) ∨
        satisfacible ((A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {Comp2 F})
      using hip0 satisfacibleUnion3 by auto
    moreover
    have A ⊆ (A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {Comp1 F}
      and B ⊆ (A − {Comp1 F}) ∪ (B − {Comp2 F}) ∪ {Comp2 F}
      by auto
    ultimately
    have satisfacible A ∨ satisfacible B using SubSatis by auto
    thus False using A3 B3 by simp
   qed }
  thus ?thesis by auto
qed
```

Por último, la formalización del lema 7.1.1 es la siguiente.

```
lemma ConsistenciaCompacidad:
  shows consistenciaP{W::'b formula1 set. ∀ A. (A⊆ W ∧ finite A) ⟶
  satisfacible A}
proof (unfold consistenciaP-def , rule allI, rule impI)
  let ?C = {W::'b formula1 set.  ∀ A. (A⊆ W ∧ finite A) ⟶ satisfacible A}
  fix W:: 'b formula1 set
  assume W ∈ ?C
```

```
 hence hip: ∀ A. (A⊆ W ∧ finite A) ⟶ satisfacible A by simp
 show (∀ P. ¬ (Atomo P ∈ W ∧ ¬.Atomo P ∈ W)) ∧
      FF ∉ W ∧
      ¬.TT ∉ W ∧
      (∀ F. ¬.¬.F ∈ W ⟶ W ∪ {F} ∈ ?C) ∧
      (∀ F. (FormulaAlfa F) ∧ F ∈ W ⟶
      (W ∪ {Comp1 F, Comp2 F} ∈ ?C)) ∧
      (∀ F. (FormulaBeta F) ∧ F ∈ W ⟶
      (W ∪ {Comp1 F} ∈ ?C ∨ W ∪ {Comp2 F} ∈ ?C))
 proof −
  have (∀ P. ¬ (Atomo P ∈ W ∧ (¬. Atomo P) ∈ W))
   using hip comp1 by simp
  moreover
  have FF ∉ W using hip comp2 by auto
  moreover
  have ¬. TT ∉ W using hip comp3 by auto
  moreover
  have ∀ F. (¬.¬.F) ∈ W ⟶ W ∪ {F} ∈ ?C
  proof (rule allI impI)+
   fix F
   assume hip1: ¬.¬.F ∈ W
   show W ∪ {F} ∈ ?C using hip hip1 comp4 by simp
  qed
  moreover
  have ∀ F. (FormulaAlfa F) ∧ F ∈ W ⟶ (W ∪ {Comp1 F, Comp2 F} ∈ ?C)
  proof (rule allI impI)+
   fix F
   assume FormulaAlfa F ∧ F ∈ W
   thus W ∪ {Comp1 F, Comp2 F} ∈ ?C using hip comp5[of F] by blast
  qed
  moreover
  have ∀ F. (FormulaBeta F) ∧ F ∈ W ⟶
        (W ∪ {Comp1 F} ∈ ?C ∨ W ∪ {Comp2 F} ∈ ?C)
  proof (rule allI impI)+
   fix F
   assume (FormulaBeta F) ∧ F ∈ W
   thus W ∪ {Comp1 F} ∈ ?C ∨ W ∪ {Comp2 F} ∈ ?C
    using hip comp6[of F] by blast
  qed
  ultimately
  show ?thesis by auto
 qed
qed
```

Teorema 7.1.2 (Teorema de compacidad) *Sean* **L** *un lenguaje proposicional tal que existe una enumeración del conjunto de símbolos proposicionales, y S un conjunto de fórmulas proposicionales. Si todo subconjunto finito de S es satisfacible, entonces S es satisfacible.*

Demostración: Sea $\mathcal{C} = \{W | \forall A\, (A \subseteq W \wedge A \text{ es finito } \rightarrow A \text{ es satisfacible})\}$. Entonces por hipótesis $S \in \mathcal{C}$. Luego, por el lemma 7.1.1 y el corolario 6.11.6 se concluye que S es satisfacible.

□

La formalización del teorema de compacidad es la siguiente:

```
theorem TeoremaCompacidadProposicional1:
 assumes hip1: ∃ g. enumeracion (g:: nat ⇒ 'b)
 and hip2: ∀ A. (A ⊆ (S:: 'b formula1 set) ∧ finite A) ⟶ satisfacible A
 shows satisfacible S
proof −
 obtain g where g: ∀ y. ∃ n. y = (g::nat => 'b) n
  using hip1 by(unfold enumeracion-def ) auto
 let ?C = {W:: 'b formula1 set. ∀ A. (A ⊆ W ∧ finite A) ⟶ satisfacible A}
 have consistenciaP ?C
  using ConsistenciaCompacidad by simp
 moreover
 have S ∈ ?C using hip2 by simp
 ultimately
 show satisfacible S using g hip1 ConjuntosatisfacibleP by blast
qed
```

En el caso del lenguaje proposicional en el que identificamos los símbolos proposicionales con los números naturales, tenemos el siguiente corolario.

Corolario 7.1.3 *Sean* **L** *el lenguaje proposicional en el que los símbolos proposicionales son los números naturales y S un conjunto de fórmulas proposicionales. Si todo subconjunto finito de S es satisfacible, entonces S es satisfacible.*

Demostración: Consideremos la siguiente colección de conjuntos de fórmulas con símbolos proposicionales pertenecientes a $\mathbb{N}$.

$$\mathcal{C} = \{W | \forall A\, (A \subseteq W \wedge A \text{ es finito } \rightarrow A \text{ es satisfacible})\}.$$

Entonces por hipótesis $S \in \mathcal{C}$. Además, existe una enumeración de $\mathbb{N}$ (lema 6.6.9). Luego, por el teorema anterior, S es satisfacible.

□

```
corollary CompacidadProposicional2:
```

assumes $\forall A.\ (A \subseteq (S::\ nat\ formula1\ set) \wedge finite\ A) \longrightarrow satisfacible\ A$
shows *satisfacible S*
using *enum-nat assms TeoremaCompacidadProposicional1*
by *auto*

7.2 Teorema de interpolación de Craig

Nuestra segunda aplicación del teorema de existencia de modelos en la lógica proposicional es la formalización del teorema de interpolación de Craig. Este teorema afirma que si $F \to G$ es una tautología, entonces tiene un *interpolante*.

Definición 7.2.1 *Una fórmula H es un* **interpolante** *de la implicación $F \to G$, si cada símbolo proposicional que ocurre en H también ocurre en F y G, y $F \to H$ y $H \to G$ son tautologías.*

Para formalizar en Isabelle el concepto de interpolante se define la función *ocurrencias* tal que *ocurrencias F* es el conjunto de las fórmulas atómicas que ocurren en *F*.

primrec *ocurrencias* :: *'b formula1* ⇒ (*'b formula1 set*) **where**
ocurrencias FF = {}
| *ocurrencias TT* = {}
| *ocurrencias* (*Atomo P*) = {(*Atomo P*)}
| *ocurrencias* (¬. *F*) = *ocurrencias F*
| *ocurrencias* (*F* ∧. *G*) = (*ocurrencias F*) ∪ (*ocurrencias G*)
| *ocurrencias* (*F* ∨. *G*) = (*ocurrencias F*) ∪ (*ocurrencias G*)
| *ocurrencias* (*F* →. *G*) = (*ocurrencias F*) ∪ (*ocurrencias G*)

La formalización del concepto de interpolante es:

definition *interpolante*:: *'b formula1* ⇒ *'b formula1* ⇒ *'b formula1* ⇒ *bool*
where
interpolante H F G =
(*ocurrencias H* ⊆ (*ocurrencias F* ∩ *ocurrencias G*) ∧
(*tautologia* (*F* →. *H*) ∧ *tautologia*(*H* →. *G*)))

Definición 7.2.2 *Sea $S = \{F_1, F_2, \ldots, F_n\}$ un conjunto de fórmulas. La conjunción de las fórmulas de S es la fórmula $F_1 \wedge F_2 \cdots \wedge F_n$. Denotamos esta fórmula por $\langle S \rangle$.*

Su formalización es:

primrec *Conj*:: *'b formula1 list* ⇒ *'b formula1* **where**
Conj [] = *TT*
| *Conj* (*F*#*Fs*) = *F* ∧. (*Conj Fs*)

Para la demostración del teorema de interpolación de Craig utilizamos la siguiente noción de *conjunto consistente de Craig*.

Definición 7.2.3 *Un conjunto finito S es* **consistente de Craig** *si existe una partición de S en subconjuntos S_1 y S_2, es decir, $S = S_1 \cup S_2$ y $S_1 \cap S_2 = \emptyset$, tal que $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$ no tiene interpolante.*

Para formalizar el concepto anterior definimos la relación *Particion* tal que *Particion* S_1, S_2, S se verifica si $\{S_1, S_2\}$ es una partición de S. En Isabelle,

definition *Particion* :: *'a set* ⇒ *'a set* ⇒ *'a set* ⇒ *bool* **where**
Particion S1 S2 S = ((*S* = *S1* ∪ *S2*) ∧ (*S1* ∩ *S2* = {}))

La formalización de los conjuntos consistentes de Craig es la siguiente:

definition *ConsistenteCraig*:: *'b formula1 set* ⇒ *bool* **where**
ConsistenteCraig S =
(∃ (*S1*:: *'b formula1 list*) (*S2*:: *'b formula1 list*). *finite S* ∧
(*Particion* (*set S1*) (*set S2*) *S*) ∧
(∀ *H*. ¬*interpolante H* (*Conj S1*) (¬.(*Conj S2*))))

La demostración del teorema de interpolación de Craig se fundamenta en que la colección $\mathcal{C}$ de conjuntos consistentes de Craig es una propiedad de consistencia.

Lema 7.2.4 *$\mathcal{C} = \{S | S$ es un conjunto consistente de Craig $\}$ es una propiedad de consistencia.*

Demostración: Sea $S \in \mathcal{C}$. A continuación demostramos las condiciones que definen una propiedad de consistencia.

(1). Sea P un símbolo proposicional. Demostramos que $P \notin S$ o $\neg P \notin S$ en forma contrarecíproca. Supongamos que $P \in S$ y $\neg P \in S$ y demostremos que S no es un conjunto consistente de Craig. Sea $\{S_1, S_2\}$ una partición de S. Por casos mostramos que existe una fórmula H que es interpolante de $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$: si $P \in S_1$ y $\neg P \in S_1$ entonces $H = \bot$; si $P \in S_1$ y $\neg P \in S_2$ entonces $H = P$ es; si $P \in S_2$ y $\neg P \in S_1$ entonces $H = \neg P$; si $P \in S_2$ y $\neg P \in S_2$ entonces $H = \top$.

(2). Demostramos que $\bot \notin S$ en forma contrarecíproca. Supongamos que $\bot \in S$ y demostremos que S no es un conjunto consistente de Craig. Sea $\{S_1, S_2\}$ una partición de S. Por casos mostramos que existe una fórmula H que es interpolante de $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$: si $\bot \in S_1$ entonces, $H = \bot$; si $\bot \in S_2$ entonces, $H = \top$.

(3). Demostramos que $\neg\top \notin S$ en forma contrarecíproca. Supongamos que $\neg\top \in S$ y demostremos que S no es un conjunto consistente de Craig. Sea $\{S_1, S_2\}$ una partición

de S. Por casos mostramos que existe una fórmula H que es interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$: si $\neg\top \in S_1$ entonces, $H = \bot$; si $\neg\top \in S_2$ entonces, $H = \top$.

(4). Supongamos que $\neg\neg F \in S$, hay que demostrar que $S \cup \{F\}$ es un conjunto consistente de Craig. Consideremos los casos $F \in S$ y $F \notin S$. En el caso $F \in S$ se tiene que $S \cup \{F\} = S$ y por lo tanto $S \cup \{F\}$ es un conjunto consistente de Craig. Supongamos $F \notin S$. Puesto que S es consistente de Craig, se tiene que S es finito, y por lo tanto $S \cup \{F\}$ es finito, y existe una partición $\{S_1, S_2\}$ de S tal que $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante. De esto último demostramos que existe una partición $\{S_1{}', S_2{}'\}$ de $S \cup \{F\}$ tal que $\langle S_1{}'\rangle \rightarrow \neg\langle S_2{}'\rangle$ no tiene interpolante. Consideremos los casos $\neg\neg F \in S_1$ y $\neg\neg F \in S_2$. Para cada caso demostramos que existe una partición $\{S_1{}', S_2{}'\}$ de $S \cup \{F\}$ tal que $\langle S_1{}'\rangle \rightarrow \neg\langle S_2{}'\rangle$ no tiene interpolante. Las correspondientes particiones son:

$\{S_1 \cup \{F\}, S_2\}$, si $\neg\neg F \in S_1$ y $\{S_1, S_2 \cup \{F\}\}$, si $\neg\neg F \in S_2$.

Ninguna de las correspondientes implicaciones tiene interpolante, de lo contrario $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ tendría interpolante.

(5). Sea $\alpha \in S$, hay que demostrar que $S \cup \{\alpha_1, \alpha_2\}$ es un conjunto consistente de Craig. Consideremos los casos $\alpha_1 \in S$ y $\alpha_2 \in S$, y $\alpha_1 \notin S$ o $\alpha_2 \notin S$. En el caso $\alpha_1 \in S$ y $\alpha_2 \in S$ se tiene que $S \cup \{\alpha_1, \alpha_2\} = S$ y por lo tanto $S \cup \{\alpha_1, \alpha_2\}$ es un conjunto consistente de Craig. Si $\alpha_1 \notin S$ o $\alpha_2 \notin S$, se tienen los casos: (a) $\alpha_1 \notin S$ y $\alpha_2 \in S$; (b) $\alpha_1 \in S$ y $\alpha_2 \notin S$; y (c) $\alpha_1 \notin S$ y $\alpha_2 \notin S$.

Puesto que S es un conjunto consistente de Craig, se tiene que S es finito, y por lo tanto $S \cup \{\alpha_1, \alpha_2\}$ es finito, y existe una partición $\{S_1, S_2\}$ de S tal que $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante. De esto último demostramos, para cada caso, que existe una partición $\{S_1{}', S_2{}'\}$ de $S \cup \{\alpha_1, \alpha_2\}$ tal que $\langle S_1{}'\rangle \rightarrow \neg\langle S_2{}'\rangle$ no tiene interpolante. Las correspondientes particiones son:

(a) $\{S_1 \cup \{\alpha_1\}, S_2\}$, si $\alpha \in S_1$; $\{S_1, S_2 \cup \{\alpha_1\}\}$, si $\alpha \in S_2$.

(b) $\{S_1 \cup \{\alpha_2\}, S_2\}$, si $\alpha \in S_1$; $\{S_1, S_2 \cup \{\alpha_2\}\}$, si $\alpha \in S_2$.

(c) $\{S_1 \cup \{\alpha_1, \alpha_2\}, S_2\}$, si $\alpha \in S_1$; $\{S_1, S_2 \cup \{\alpha_1, \alpha_2\}\}$, si $\alpha \in S_2$.

Ninguna de las correspondientes implicaciones tiene interpolante, de lo contrario $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ tendría interpolante.

(6). Sea $\beta \in S$, hay que demostrar que $S \cup \{\beta_1\}$ es un conjunto consistente de Craig o $S \cup \{\beta_2\}$ es un conjunto consistente de Craig. Consideremos los casos $\beta_1 \in S$ o $\beta_2 \in S$, y $\beta_1 \notin S$ y $\beta_2 \notin S$. En el caso $\beta_1 \in S$ o $\beta_2 \in S$ se tiene que $S \cup \{\beta_1\} = S$ o $S \cup \{\beta_2\} = S$ y por lo tanto $S \cup \{\beta_1\}$ o $S \cup \{\beta_2\}$ es un conjunto consistente de Craig. Si $\beta_1 \notin S$ y $\beta_2 \notin S$ demostramos que $S \cup \{\beta_1\}$ o $S \cup \{\beta_2\}$ es un conjunto consistente de Craig en forma contrarecíproca. Supongamos que $S \cup \{\beta_1\}$ no es un conjunto consistente de Craig y $S \cup \{\beta_2\}$ no es un conjunto consistente de Craig, y demostremos que S no es un conjunto consistente de Craig. Sea $\{S_1, S_2\}$ una partición de S, entonces $\beta \in S_1$ o $\beta \in S_2$.

Por casos probamos que existe una fórmula H que interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$:

(a) Si $\beta \in S_1$ entonces, existe H_1 interpolante de $\langle S_1 \cup \{\beta_1\}\rangle \rightarrow \neg\langle S_2\rangle$ y H_2 interpolante de $\langle S_1 \cup \{\beta_2\}\rangle \rightarrow \neg\langle S_2\rangle$ tales que $H_1 \vee H_2$ es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.

(b) Si $\beta \in S_2$, entonces existe H_1 interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_1\}\rangle$ y H_2 interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_2\}\rangle$ tales que $H_1 \wedge H_2$ es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$. □

La formalización de la propiedad (1) utiliza los siguientes lemas.

Lema 7.2.5 *Si G y $\top$ son equivalentes entonces $\top$ es un interpolante de $F \rightarrow G$, para cualquier fórmula F.*

Su formalización es:

lemma *interpolanteTT*:
assumes *equivalentes G TT*
shows *interpolante TT F G*

Lema 7.2.6 *Si F y $\bot$ son equivalentes, entonces $\bot$ es un interpolante de $F \rightarrow G$, para cualquier fórmula G.*

Su formalización es:

lemma *interpolanteFF*:
assumes *equivalentes F FF*
shows *interpolante FF F G*

Lema 7.2.7 *Si $F \in S_1$ y $\neg F \in S_2$, entonces $\bot$ es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es:

lemma *Interpolante1*:
assumes *F ∈ set S1 ∧ ¬.F ∈ set S1*
shows *interpolante FF* (*Conj S1*) (*¬.Conj S2*)

Lema 7.2.8 *Sea P un símbolo proposicional. Si $P \in S_1$ y $\neg P \in S_2$, entonces P es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es:

lemma *Interpolante2*:
assumes *hip1*: (*Atomo P*) *∈ set S1* **and** *hip2*: (*¬.Atomo P*) *∈ set S2*
shows *interpolante* (*Atomo P*) (*Conj S1*) (*¬.*(*Conj S2*))

Lema 7.2.9 *Sea P un símbolo proposicional. Si $\neg P \in S_1$ y $P \in S_2$, entonces $\neg P$ es un interpolante de $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$.*

Su formalización es:

```
lemma Interpolante3:
 assumes hip1: (¬.Atomo P) ∈ set S1 and hip2: (Atomo P) ∈ set S2
 shows interpolante (¬.Atomo P) (Conj S1) (¬.(Conj S2))
```

Lema 7.2.10 *Si $F \in S_2$ y $\neg F \in S_2$, entonces $\top$ es un interpolante de $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$.*

Su formalización es:

```
lemma Interpolante4:
 assumes F ∈ set S2 ∧ ¬.F ∈ set S2
 shows interpolante TT (Conj S1) (¬.Conj S2)
```

Las últimas cuatro propiedades acerca de interpolantes permiten demostrar el siguiente lema.

Lema 7.2.11 *Sea P un símbolo proposicional. Si $P, \neg P \in S$ y $\{S_1, S_2\}$ es una una partición de S, entonces $\langle S_1 \rangle \rightarrow \neg\langle S_2 \rangle$ tiene un interpolante.*

Su formalización es:

```
lemma craig1a:
 assumes hip1: (Atomo P) ∈ S ∧ ¬.(Atomo P) ∈ S
 and hip2: Particion (set S1) (set S2) S
 shows (∃ H. interpolante H (Conj S1) (¬.(Conj S2)))
proof −
 have (Atomo P) ∈ set S1 ∪ set S2 ∧ ¬.(Atomo P) ∈ set S1 ∪ set S2
  using assms
  by (unfold Particion-def, simp)
 hence (Atomo P) ∈ set S1 ∧ ¬.(Atomo P) ∈ set S1 ∨
       (Atomo P) ∈ set S1 ∧ ¬.(Atomo P) ∈ set S2 ∨
       (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S1 ∨
       (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S2 using assms by auto
 thus ?thesis
 proof(rule disjE)
  assume Atomo P ∈ set S1 ∧ ¬.Atomo P ∈ set S1
  hence interpolante FF (Conj S1) (¬. Conj S2)
   using Interpolante1[of Atomo P] by auto
  thus (∃ H. interpolante H (Conj S1) (¬.Conj S2)) by auto
```

```
 next
  assume (Atomo P) ∈ set S1 ∧ ¬.(Atomo P) ∈ set S2 ∨
         (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S1 ∨
         (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S2
  thus ?thesis
  proof(rule disjE)
   assume (Atomo P) ∈ set S1 ∧ ¬.(Atomo P) ∈ set S2
   hence interpolante (Atomo P) (Conj S1) (¬.(Conj S2))
    using Interpolante2[of P] by auto
   thus (∃ H. interpolante H (Conj S1) (¬.Conj S2)) by auto
  next
   assume (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S1 ∨
          (Atomo P) ∈ set S2 ∧ ¬.(Atomo P) ∈ set S2
   thus ?thesis
   proof(rule disjE)
    assume (Atomo P) ∈ set S2 ∧ (¬.Atomo P ∈ set S1)
    hence interpolante (¬.Atomo P) (Conj S1) (¬.(Conj S2))
     using Interpolante3[of P] by auto
    thus (∃ H. interpolante H (Conj S1) (¬.Conj S2)) by auto
   next
    assume (Atomo P) ∈ set S2 ∧ (¬.Atomo P) ∈ set S2
    hence interpolante TT (Conj S1) (¬.(Conj S2))
     using Interpolante4[of Atomo P] by auto
    thus (∃ H. interpolante H (Conj S1) (¬.Conj S2)) by auto
   qed
  qed
 qed
qed
```

A partir de lema anterior tenemos el siguiente resultado

Lema 7.2.12 *Si existe un símbolo proposicional P tal que $P, \neg P \in S$, entonces S no es un conjunto consistente de Craig.*

Su formalización es:

```
lemma craig1b:
 assumes ¬ (∀ P. ¬ (Atomo P ∈ S ∧ ¬.Atomo P ∈ S))
 shows ¬ ConsistenteCraig (S :: 'b formula1 set)
```

Por último, como una consecuencia del lema anterior podemos formalizar la prueba de la propiedad (1).

```
lemma craig1:
```

```
 assumes ConsistenteCraig (S :: 'b formula1 set)
 shows (∀ P. ¬ (Atomo P ∈ S ∧ ¬.Atomo P ∈ S))
using assms craig1b[of S]
by auto
```

En la prueba de la propiedad (2) se usan los tres lemas siguientes.

Lema 7.2.13 *Si $\bot \in S$ y $\{S_1, S_2\}$ es una partición de S, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ tiene un interpolante.*

Su formalización es:

```
lemma craig2a:
 assumes hip1: FF ∈ S and hip2: Particion (set S1) (set S2) S
 shows (∃ H. interpolante H (Conj S1) (¬.(Conj S2)))
```

Lema 7.2.14 *Si $\bot \in S$, entonces S no es consitente Craig.*

Su formalización es:

```
lemma craig2b:
 assumes FF ∈ S
 shows ¬ ConsistenteCraig (S :: 'b formula1 set)
```

La formalización de la prueba de la propiedad (2) es la siguiente:

```
lemma craig2:
 assumes ConsistenteCraig (S :: 'b formula1 set)
 shows FF ∉ S
using assms craig2b[of S]
by auto
```

En la prueba de la propiedad (3) se usan los tres lemas siguientes.

Lema 7.2.15 *Si $\neg\top \in S$ y $\{S_1, S_2\}$ es una partición de S, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$ tiene interpolante.*

Su formalización es:

```
lemma craig3a:
 assumes hip1: ¬.TT ∈ S and hip2: Particion (set S1) (set S2) S
 shows (∃ H. interpolante H (Conj S1) (¬.(Conj S2)))
```

Lema 7.2.16 *Si $\neg\top \in S$, entonces S no es un conjunto consistente de Craig.*

Su formalización es:

```
lemma craig3b:
 assumes ¬.TT ∈ S
 shows ¬ ConsistenteCraig (S :: 'b formula1 set)
```

La formalización de la prueba de la propiedad (3) es la siguiente:

```
lemma craig3:
 assumes ConsistenteCraig (S :: 'b formula1 set)
 shows ¬.TT ∉ S
using assms craig3b[of S]
by auto
```

En la demostración de la propiedad (4) se usan los siguientes lemas.

Lema 7.2.17 *Si* $\neg\neg F \in S_1$ *y* H *es un interpolante de* $\langle S_1 \cup \{F\}\rangle \rightarrow \neg\langle S_2\rangle$*, entonces* H *es un interpolante de* $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.

Su formalización es:

```
lemma craig4a:
 assumes hip1: (¬. ¬.F) ∈ set S1
 and hip2: interpolante H (Conj ([F] @ S1)) (¬.(Conj S2))
 shows interpolante H (Conj S1) (¬.(Conj S2))
proof −
 have 1:ocurrencias H ⊆
       (ocurrencias (Conj ([F] @ S1)) ∩ ocurrencias (¬.(Conj S2)))
  and 2: (tautologia ((Conj ([F] @ S1)) →. H))
  and 3: (tautologia (H →.(¬.(Conj S2))))
  using hip2 by(unfold interpolante-def, auto)
 have ocurrencias H ⊆ ocurrencias (Conj S1) ∩ ocurrencias (¬.(Conj S2))
 proof −
  have 4: ocurrencias F = ocurrencias (¬. ¬.F) by simp
  moreover
  have (¬. ¬.F) ∈ set S1 using hip1 by(auto simp add: set-eq-iff)
  hence ocurrencias (¬. ¬.F) ⊆ ocurrencias (Conj S1)
   using OcurrenciasFormula by blast
  hence ocurrencias F ⊆ ocurrencias (Conj S1) using 4 by blast
  moreover
  have ocurrencias (Conj ([F] @ S1)) =
       ocurrencias (Conj [F]) ∪ ocurrencias (Conj S1)
   using ocurrenConcat by auto
  ultimately
  have ocurrencias (Conj ([F] @ S1)) = ocurrencias (Conj S1) by auto
```

```
  thus ocurrencias H ⊆ ocurrencias (Conj S1) ∩ ocurrencias (¬.(Conj S2))
   using 1 by auto
 qed
 moreover
 have tautologia ((Conj S1 ) →. H) ∧ tautologia (H →.(¬.(Conj S2)))
 proof −
  have tautologia ((Conj S1 ) →. H)
  proof(unfold tautologia-def, rule allI)
   fix I
   show valor1 I (Conj S1 →. H) = Verdad
   proof(cases valor1 I (Conj S1))
    assume 5: valor1 I (Conj S1) = Verdad
    have valor1 I (Conj ([F] @ S1)) = valor1 I (Conj S1)
     using hip1 valorConjNoNob[of F S1 I] by(auto simp add: set-eq-iff)
    hence 6: valor1 I (Conj ([F] @ S1)) = Verdad using 5 by simp
    have valor1 I H = Verdad
    proof −
     { assume valor1 I H ≠ Verdad
      hence valor1 I H = False using CasosValor by auto
      hence valor1 I ((Conj ([F] @ S1)) →. H) = False
       using 6 by(auto, unfold v-implicacion1-def, auto)
      hence False using 2 by(auto, unfold tautologia-def, auto)}
     thus valor1 I H = Verdad by auto
    qed
    thus valor1 I (Conj S1 →. H) = Verdad
     using 3 by(auto, unfold v-implicacion1-def, auto)
   next
    assume valor1 I (Conj S1) = False
    thus valor1 I (Conj S1 →. H) = Verdad
     by(auto, unfold v-implicacion1-def, auto)
   qed
  qed
  thus tautologia ((Conj S1 ) →. H) ∧ tautologia (H →.(¬.(Conj S2)))
   using 3 by auto
 qed
 ultimately
 show interpolante H (Conj S1) (¬.(Conj S2))
  by (unfold interpolante-def, auto)
qed
```

Lema 7.2.18 *Si $\neg\neg F \in S_2$ y H es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{F\}\rangle$, entonces H es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es la siguiente; la demostración es análoga a la anterior.

```
lemma craig4b:
 assumes hip1: (¬. ¬.F) ∈ set S2
 and hip2: interpolante H (Conj S1) (¬.(Conj ([F] @ S2)))
 shows interpolante H (Conj S1) (¬.(Conj S2))
```

La formalización de la prueba de la propiedad (4) es la siguiente:

```
lemma craig4:
 assumes hip1: ¬.¬.F ∈ S and hip2: ConsistenteCraig (S :: 'b formula1 set)
 shows ConsistenteCraig ({F} ∪ S )
proof(cases)
 assume F ∈ S
 hence {F} ∪ S = S by auto
 thus ConsistenteCraig ({F} ∪ S) using hip2 by simp
next
 assume hipa: F ∉ S
 show ConsistenteCraig ({F} ∪ S)
 proof(unfold ConsistenteCraig-def)
  have hip: (∃ (S1:: 'b formula1 list) (S2:: 'b formula1 list). finite S ∧
        (Particion (set S1) (set S2)  S) ∧
        (∀ F. ¬interpolante F (Conj S1) (¬.(Conj S2))))
   using hip2 by (unfold ConsistenteCraig-def)
  then obtain S1 S2 where fin: finite S
   and par: (Particion (set S1) (set S2)  S)
   and int: (∀ F. ¬interpolante F (Conj S1) (¬.(Conj S2)))
   by auto
  show (∃ (S1':: 'b formula1 list) (S2':: 'b formula1 list).
      finite ({F} ∪ S) ∧
      (Particion (set S1') (set S2')  ({F} ∪ S)) ∧
      (∀ F. ¬interpolante F (Conj S1') (¬.(Conj S2'))))
  proof(cases)
   assume hipb: ¬.¬. F ∈ set S1
   have finite ({F} ∪ S) using fin by simp
   moreover
   have  Particion ({F} ∪ set S1) (set S2)  ({F} ∪ S)
   proof −
    have (S = set S1 ∪ set S2) ∧ (set S1 ∩ set S2 = {})
     using par by(unfold Particion-def)
    hence  {F} ∪ S = (({F} ∪ set S1) ∪ set S2) ∧
        (({F} ∪ set S1) ∩ set S2 = {})
     using hipa by auto
    thus Particion ({F} ∪ set S1) (set S2) ({F} ∪ S)
     by(unfold Particion-def)
```

```
  qed
  moreover
  have (∀ H. ¬interpolante H (Conj ([F] @ S1)) (¬.(Conj S2)))
  proof −
   { assume ¬(∀ H. ¬interpolante H (Conj ([F] @ S1)) (¬.(Conj S2)))
    then obtain H where interpolante H (Conj ([F] @ S1)) (¬.(Conj S2))
     by auto
    hence interpolante H (Conj S1) (¬.(Conj S2))
     using hipb craig4a by auto
    hence False  using int by auto}
   thus ?thesis by auto
  qed
  ultimately
  have finite ({F} ∪ S) ∧ Particion ({F} ∪ set S1) (set S2) ({F} ∪ S) ∧
      (∀H. ¬ interpolante H (Conj ([F]@ S1)) (¬.Conj S2))
   by auto
  hence ∃ S2′. finite ({F} ∪ S) ∧
          Particion ({F} ∪ set S1) (set S2′) ({F} ∪ S) ∧
          (∀H. ¬ interpolante H (Conj ([F] @ S1)) (¬.Conj S2′))
   by blast
  hence ∃ S2′. finite ({F} ∪ S) ∧
          Particion (set([F] @ S1)) (set S2′) ({F} ∪ S) ∧
          (∀H. ¬ interpolante H (Conj ([F] @ S1)) (¬.Conj S2′))
   by auto
  thus ?thesis by blast
 next
  assume ¬.¬.F ∉ set S1
  hence  hipb: ¬.¬.F ∈ set S2
   using par hip1 by(unfold Particion-def, auto)
  have finite ({F} ∪ S) using fin by simp
  moreover
  have Particion (set S1) ({F} ∪ set S2) ({F} ∪ S)
  proof −
   have (S = set S1 ∪ set S2) ∧ (set S1 ∩ set S2 = {})
    using par by(unfold Particion-def)
   hence {F} ∪ S = ((set S1) ∪ ({F} ∪ set S2)) ∧
       ((set S1) ∩ ({F} ∪ set S2) = {})
    using hipa by auto
   thus Particion (set S1) ({F} ∪ set S2) ({F} ∪ S)
    by(unfold Particion-def)
  qed
  moreover
  have (∀ H. ¬interpolante H (Conj S1) (¬.(Conj ([F] @ S2))))
```

```
    proof −
     { assume ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj ([F] @ S2))))
       then obtain H where interpolante H (Conj S1) (¬.(Conj ([F] @ S2)))
         by auto
       hence interpolante H (Conj S1) (¬.(Conj S2))
         using hipb craig4b by auto
       hence False  using int by auto}
      thus ?thesis by auto
    qed
    ultimately
    have finite ({F} ∪ S) ∧ Particion (set S1) ({F} ∪ set S2) ({F} ∪ S) ∧
        (∀ H. ¬ interpolante H (Conj S1) (¬.(Conj ([F] @ S2))))
      by auto
    hence finite ({F} ∪ S) ∧ Particion (set S1) (set([F] @ S2)) ({F} ∪ S) ∧
        (∀ H. ¬ interpolante H (Conj S1) (¬.(Conj ([F] @ S2))))
      by auto
    hence ∃ S2'. finite ({F} ∪ S) ∧ Particion (set S1) (set S2') ({F} ∪ S) ∧
        (∀ H. ¬ interpolante H (Conj S1) (¬.(Conj S2')))
      by blast
    thus ?thesis by blast
   qed
  qed
qed
```

En la prueba de la propiedad (5) se usan los siguientes lemas.

Lema 7.2.19 *Si $\alpha \in S_1$ y H es un interpolante de $\langle S_1 \cup \{\alpha_1\}\rangle \rightarrow \neg\langle S_2\rangle$, entonces H es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es

```
lemma craig5a1:
 assumes hip1: FormulaAlfa F
 and hip2: F ∈ set S1
 and hip3: interpolante H (Conj ([Comp1 F] @  S1)) (¬.(Conj S2))
 shows interpolante H (Conj S1) (¬.(Conj S2))
```

Lema 7.2.20 *Si $\alpha \in S_2$ y H es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\alpha_2\}\rangle$, entonces H es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es:

```
lemma craig5a2:
```

```
 assumes hip1: FormulaAlfa F
 and hip2: F ∈ set S2
 and hip3: interpolante H (Conj S1) (¬.(Conj ([Comp1 F] @ S2)))
 shows interpolante H (Conj S1) (¬.(Conj S2))
```

Lema 7.2.21 *Si S es un conjunto consistente de Craig, $\alpha \in S$, $\alpha_1 \notin S$ y $\alpha_2 \in S$, entonces $S \cup \{\alpha_1, \alpha_2\}$ es un conjunto consistente de Craig.*

Su formalización es:

```
lemma craig5a:
 assumes hip0: FormulaAlfa F
 and hip1: ConsistenteCraig (S :: 'b formula1 set)
 and hip2: F ∈ S
 and hipb: Comp1 F ∉ S
 and hipc: Comp2 F ∈ S
 shows ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S )
proof −
 have (∃ (S1:: 'b formula1 list) (S2:: 'b formula1 list).
       finite S ∧
       (Particion (set S1) (set S2) S) ∧
       (∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2))))
  using hip1 by (unfold ConsistenteCraig-def)
 then obtain S1 S2 where fin: finite S
  and par: (Particion (set S1) (set S2) S)
  and int: (∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2)))
  by auto
 show ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S)
 proof(unfold ConsistenteCraig-def)
  show (∃ (S1':: 'b formula1 list) (S2':: 'b formula1 list).
       finite ({Comp1 F, Comp2 F} ∪ S) ∧
       (Particion (set S1') (set S2') ({Comp1 F, Comp2 F} ∪ S)) ∧
       (∀H. ¬interpolante H (Conj S1') (¬.(Conj S2'))))
  proof −
   have F ∈ set S1 ∨ F ∈ set S2
    using hip2 par by(unfold Particion-def) auto
   thus ?thesis
   proof(rule disjE)
    assume hipd: F ∈ set S1
    have finite ({Comp1 F, Comp2 F} ∪ S) ∧
         (Particion ({Comp1 F} ∪ (set S1))
                    (set S2)
                    ({Comp1 F, Comp2 F} ∪ S)) ∧
```

```
    (∀ H. ¬interpolante H (Conj ([Comp1 F] @ S1)) (¬.(Conj S2)))
proof −
 have finite ({Comp1 F, Comp2 F} ∪ S) using fin by simp
 moreover
 have Particion ({Comp1 F} ∪ (set S1))
             (set S2)
             ({Comp1 F, Comp2 F} ∪ S)
  using hipb hipc par by(unfold Particion-def, auto)
 moreover
 have (∀ H. ¬interpolante H (Conj ([Comp1 F] @ S1)) (¬.(Conj S2)))
 proof −
  { assume ¬ (∀ H. ¬interpolante H (Conj ([Comp1 F] @ S1))
                          (¬.(Conj S2)))
    then obtain H where interpolante H (Conj ([Comp1 F] @ S1))
                            (¬.(Conj S2))
     by auto
    hence interpolante H (Conj S1) (¬.(Conj S2))
     using hip0 hipd craig5a1 by auto
    hence False  using int by auto}
  thus ?thesis by auto
 qed
 ultimately
 show finite ({Comp1 F, Comp2 F} ∪ S) ∧
     (Particion ({Comp1 F} ∪ set S1)
             (set S2)
             ({Comp1 F, Comp2 F} ∪ S)) ∧
     (∀ H. ¬interpolante H (Conj ([Comp1 F] @ S1)) (¬.(Conj S2)))
  by auto
qed
hence ∃ S2'. finite ({Comp1 F, Comp2 F} ∪ S) ∧
        (Particion ({Comp1 F} ∪ set S1)
                (set S2')
                ({Comp1 F, Comp2 F} ∪ S)) ∧
        (∀ H. ¬interpolante H (Conj ([Comp1 F] @ S1))
                        (¬.Conj S2'))
 by blast
hence ∃ S2'. finite ({Comp1 F, Comp2 F} ∪ S) ∧
        Particion (set([Comp1 F] @ S1))
               (set S2')
               ({Comp1 F, Comp2 F} ∪ S) ∧
        (∀ H. ¬ interpolante H
                        (Conj ([Comp1 F] @ S1))
                        (¬.Conj S2'))
```

```
    by auto
  thus ?thesis by blast
next
  assume hipd: F ∈ set S2
  have finite ({Comp1 F, Comp2 F} ∪ S) ∧
      (Particion (set S1) ({Comp1 F} ∪ (set S2)) ({Comp1 F, Comp2 F} ∪ S)) ∧
      (∀ H. ¬interpolante H (Conj S1) (¬.Conj([Comp1 F] @ S2)))
  proof −
   have finite ({Comp1 F, Comp2 F} ∪ S) using fin by simp
   moreover
   have Particion ((set S1)) ({Comp1 F} ∪ set S2) ({Comp1 F, Comp2 F} ∪ S)
    using hipb hipc par by(unfold Particion-def, auto)
   moreover
   have (∀ H. ¬interpolante H (Conj S1) (¬.(Conj([Comp1 F] @ S2))))
   proof −
    { assume ¬ (∀ H. ¬interpolante H (Conj S1)
                              (¬.(Conj ([Comp1 F] @ S2))))
     then obtain H where interpolante H
                              (Conj S1)
                              (¬.(Conj ([Comp1 F] @ S2)))
      by auto
     hence interpolante H (Conj S1) (¬.(Conj S2))
      using hip0 hipd craig5a2 by auto
     hence False  using int by auto}
    thus ?thesis by auto
   qed
   ultimately
   show finite ({Comp1 F, Comp2 F} ∪ S) ∧
      (Particion (set S1)
            ({Comp1 F} ∪ set S2)
            ({Comp1 F, Comp2 F} ∪ S)) ∧
      (∀ H. ¬interpolante H
                  (Conj S1)
                  (¬.(Conj ([Comp1 F] @ S2))))
    by auto
  qed
  hence finite ({Comp1 F, Comp2 F} ∪ S) ∧
      (Particion (set S1)
           (set ([Comp1 F] @ S2))
           ({Comp1 F, Comp2 F} ∪ S)) ∧
      (∀ H. ¬ interpolante H
                  (Conj S1)
                  (¬.(Conj ([Comp1 F] @ S2))))
```

```
      by auto
    hence ∃ S2'. finite ({Comp1 F, Comp2 F} ∪ S) ∧
            (Particion (set S1) (set S2') ({Comp1 F, Comp2 F} ∪ S)) ∧
            (∀ H. ¬ interpolante H (Conj S1) (¬.(Conj S2')))
     by blast
    thus ?thesis by blast
   qed
  qed
 qed
qed
```

La formalización de las pruebas de los siguientes dos lemas son similares a la anterior.

Lema 7.2.22 *Si S es un conjunto consistente de Craig, $\alpha \in S$, $\alpha_2 \notin S$ y $\alpha_1 \in S$, entonces $S \cup \{\alpha_1, \alpha_2\}$ también lo es.*

Su formalización es:

```
lemma craig5b:
 assumes hip0: FormulaAlfa F
 and hip1: ConsistenteCraig (S :: 'b formula1 set)
 and hip2: F ∈ S
 and hipb: Comp2 F ∉ S
 and hipc: Comp1 F ∈ S
 shows ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S)
```

Lema 7.2.23 *Si S es un conjunto consistente de Craig, $\alpha \in S$, $\alpha_1 \notin S$ y $\alpha_2 \notin S$, entonces $S \cup \{\alpha_1, \alpha_2\}$ es un conjunto consistente de Craig.*

Su formalización es:

```
lemma craig5c:
 assumes hip0: FormulaAlfa F
 and hip1: ConsistenteCraig (S :: 'b formula1 set)
 and hip2: F ∈ S
 and hipb: Comp1 F ∉ S
 and hipc: Comp2 F ∉ S
 shows ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S )
```

La formalización de la prueba de la propiedad (5) es la siguiente:

```
lemma craig5:
```

```
 assumes hip0: FormulaAlfa F
 and hip1: F ∈ S
 and hip2: ConsistenteCraig (S :: 'b formula1 set)
 shows ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S )
proof(cases)
 assume Comp1 F ∈ S ∧ Comp2 F ∈ S
 hence {Comp1 F, Comp2 F} ∪ S = S by auto
 thus ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S) using hip2 by auto
next
 assume ¬(Comp1 F ∈ S ∧ Comp2 F ∈ S)
 hence hipa: Comp1 F ∉ S ∨ Comp2 F ∉ S by auto
 thus ConsistenteCraig ({Comp1 F, Comp2 F} ∪ S)
 proof(rule disjE)
  assume hipa: Comp1 F ∉ S
  show ?thesis
  proof(cases)
   assume Comp2 F ∈ S
   thus ?thesis using hip0 hip1 hip2 hipa craig5a by simp
  next
   assume Comp2 F ∉ S
   thus ?thesis using hip0 hip1 hip2 hipa craig5c by simp
  qed
 next
  assume hipb: Comp2 F ∉ S
  show ?thesis
  proof(cases)
   assume Comp1 F ∈ S
   thus ?thesis using hip0 hip1 hip2 hipa craig5b by simp
  next
   assume Comp1 F ∉ S
   thus ?thesis using hip0 hip1 hip2 hipb craig5c by simp
  qed
 qed
qed
```

En la demostración de la propiedad (6) se usan los siguientes lemas.

Lema 7.2.24 *Supongamos que $\beta \in S_1$, H_1 es un interpolante de $\langle S_1 \cup \{\beta_1\}\rangle \to \neg\langle S_2\rangle$ y H_2 es un interpolante de $\langle S_1 \cup \{\beta_2\}\rangle \to \neg\langle S_2\rangle$, entonces $H_1 \vee H_2$ es un interpolante de $\langle S_1\rangle \to \neg\langle S_2\rangle$.*

Su formalización es:

```
lemma craig6a1:
```

assumes *hip1*: *FormulaBeta F*
and *hip2*: $F \in$ *set S1*
and *hip3*: *interpolante H1* (*Conj* ([*Comp1 F*] @ *S1*)) (¬.(*Conj S2*))
and *hip4*: *interpolante H2* (*Conj* ([*Comp2 F*] @ *S1*)) (¬.(*Conj S2*))
shows *interpolante* (*H1* ∨. *H2*) (*Conj S1*) (¬.*Conj S2*)

Lema 7.2.25 *Sean S un conjunto y β una fórmula tales que se tiene lo siguiente:*

$\beta_1 \notin S$ y $\beta_2 \notin S$.

Si $\{S_1 \cup \{\beta_1\}, S_2\}$ es una partición de $S \cup \{\beta_1\}$ entonces $\langle S_1 \cup \{\beta_1\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante.

Si $\{S_1 \cup \{\beta_2\}, S_2\}$ es una partición de $S \cup \{\beta_2\}$, entonces $\langle S_1 \cup \{\beta_2\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante; $\beta \in S_1$.

Entonces, si $\{S_1, S_2\}$ es una partición de S se cumple que $\langle S_1 \cup \{\beta_2\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante.

Su formalización es:

lemma *craig6a*:
assumes *hip0*: *FormulaBeta F*
and *hip1*: *Comp1 F* $\notin$ *S*
and *hip2*: *Comp2 F* $\notin$ *S*
and *hip3*: (*Particion* (({*Comp1 F*} ∪ (*set S1*))) (*set S2*) ({*Comp1 F*} ∪ *S*)) ⟶
¬(∀ *H*. ¬*interpolante H* (*Conj* ([*Comp1 F*] @ *S1*)) (¬.(*Conj S2*)))
and *hip4*: (*Particion* (({*Comp2 F*} ∪ (*set S1*))) (*set S2*) ({*Comp2 F*} ∪ *S*)) ⟶
¬(∀ *H*. ¬*interpolante H* (*Conj* ([*Comp2 F*] @ *S1*)) (¬.(*Conj S2*)))
and *hip5*: $F \in$ *set S1*
shows (*Particion* (*set S1*) (*set S2*) *S*) ⟶
¬(∀ *H*. ¬*interpolante H* (*Conj S1*) (¬.(*Conj S2*)))

Lema 7.2.26 *Sea S finito y β una fórmula tal que los conjuntos $S \cup \{\beta_1\}$, $S \cup \{\beta_2\}$ no son consistentes Craig.*

Si $\{S_1 \cup \{\beta_1\}, S_2\}$ es una partición de $S \cup \{\beta_1\}$, entonces $\langle S_1 \cup \{\beta_1\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante.

Si $\{S_1 \cup \{\beta_2\}, S_2\}$ es una partición de $S \cup \{\beta_2\}$, entonces $\langle S_1 \cup \{\beta_2\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante.

Su formalización es:

```
lemma craig6a2:
 assumes hip1: finite S
 and hip2: ¬ ConsistenteCraig ({Comp1 F} ∪ S)
 and hip3: ¬ ConsistenteCraig ({Comp2 F} ∪ S)
 shows (¬(Particion (set ([Comp1 F ] @ S1)) (set S2) ({Comp1 F} ∪ S)) ∨
      ¬(∀ H. ¬interpolante H (Conj ([Comp1 F ] @ S1)) (¬.(Conj S2))))∧
     (¬(Particion (set ([Comp2 F ] @ S1)) (set S2) ({Comp2 F} ∪ S)) ∨
      ¬(∀ H. ¬interpolante H (Conj ([Comp2 F ] @ S1)) (¬.(Conj S2))))
```

Lema 7.2.27 *Si $\beta \in S_2$, H_1 es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_1\}\rangle$ y H_2 es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_2\}\rangle$ entonces, $H_1 \wedge H_2$ es un interpolante de $\langle S_1\rangle \rightarrow \neg\langle S_2\rangle$.*

Su formalización es:

```
lemma craig6b1:
 assumes hip1: FormulaBeta F
 and hip2: F ∈ set S2
 and hip3: interpolante H1 (Conj S1) (¬.(Conj ([Comp1 F] @ S2)))
 and hip4: interpolante H2 (Conj S1) (¬.(Conj ([Comp2 F] @ S2)))
 shows interpolante (H1 ∧. H2) (Conj S1) (¬.Conj S2)
```

Lema 7.2.28 *Sean S un conjunto y β una fórmula tales que:*

1. *$\beta_1 \notin S$ y $\beta_2 \notin S$.*
2. *Si $\{S_1, S_2 \cup \{\beta_1\}\}$ es una partición de $S \cup \{\beta_1\}$, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_1\}\rangle$ no tiene interpolante*
3. *Si $\{S_1, S_2 \cup \{\beta_2\}\}$ es una partición de $S \cup \{\beta_2\}$, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_2\}\rangle$ no tiene interpolante.*
4. *$\beta \in S_2$.*

Entonces, si $\{S_1, S_2\}$ es una partición de S se cumple que $\langle S_1 \cup \{\beta_2\}\rangle \rightarrow \neg\langle S_2\rangle$ no tiene interpolante.

Su formalización es:

```
lemma craig6b:
 assumes hip0: FormulaBeta F
 and hip1: Comp1 F ∉ S
 and hip2: Comp2 F ∉ S
```

```
 and hip3: (Particion ((set S1)) ({Comp1 F} ∪ set S2) ({Comp1 F} ∪ S)) ⟶
       ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj ([Comp1 F] @ S2))))
 and hip4: (Particion ((set S1)) ({Comp2 F} ∪ set S2) ({Comp2 F} ∪ S)) ⟶
       ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj ([Comp2 F] @ S2))))
 and hip5: F ∈ set S2
 shows (Particion (set S1) (set S2) S) ⟶
     ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2)))
proof(rule impI)
 assume hip: Particion (set S1) (set S2) S
 show ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2)))
 proof −
  have (Particion (set S1) (({Comp1 F} ∪ (set S2))) ({Comp1 F} ∪ S))
   using hip hip1 by(unfold Particion-def, auto)
  hence ∃ H1. interpolante H1 (Conj S1) (¬.(Conj ([Comp1 F] @ S2)))
   using hip3 by auto
  then obtain H1
   where H1: interpolante H1 (Conj S1) (¬.(Conj ([Comp1 F] @ S2)))
   by auto
  have (Particion (set S1) (({Comp2 F} ∪ (set S2))) ({Comp2 F} ∪ S))
   using hip hip2 by(unfold Particion-def, auto)
  hence ∃ H2. interpolante H2 (Conj S1) (¬.(Conj ([Comp2 F] @ S2)))
   using hip4 by auto
  then obtain H2
   where H2: interpolante H2 (Conj S1) (¬.(Conj ([Comp2 F] @ S2)))
   by auto
  have interpolante (H1 ∧. H2) (Conj S1) (¬.(Conj S2))
   using hip0 hip5 H1 H2 craig6b1
   by auto
  thus ¬ (∀ H. ¬ interpolante H (Conj S1) (¬.Conj S2)) by auto
 qed
qed
```

Lema 7.2.29 *Sean S un conjunto finito y β una fórmula tales que los conjuntos $S \cup \{\beta_1\}$, $S \cup \{\beta_2\}$ no son consistentes Craig.*

1. *Si $\{S_1, S_2 \cup \{\beta_1\}\}$ es una partición de $S \cup \{\beta_1\}$, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_1\}\rangle$ no tiene interpolante.*
2. *Si $\{S_1, S_2 \cup \{\beta_2\}\}$ es una partición de $S \cup \{\beta_2\}$, entonces $\langle S_1\rangle \rightarrow \neg\langle S_2 \cup \{\beta_2\}\rangle$ no tiene interpolante.*

```
lemma craig6b2:
 assumes hip1: finite S
```

```
 and hip2: ¬ ConsistenteCraig ({Comp1 F} ∪ S)
 and hip3: ¬ ConsistenteCraig ({Comp2 F} ∪ S)
 shows (¬(Particion (set S1) (set ([Comp1 F ] @ S2)) ({Comp1 F} ∪ S)) ∨
       ¬(∀ H. ¬interpolante H (Conj S1) (¬.((Conj ([Comp1 F ] @ S2)))))) ∧
      (¬(Particion (set S1) (set ([Comp2 F ] @ S2)) ({Comp2 F} ∪ S)) ∨
       ¬(∀ H. ¬interpolante H (Conj S1) (¬.((Conj ([Comp2 F ] @ S2))))))
```

Lema 7.2.30 *Sea S un conjunto y β una fórmula tales que $\beta \in S$, $\beta_1 \notin S$, $\beta_2 \notin S$ y los conjuntos $S \cup \{\beta_1\}$, $S \cup \{\beta_2\}$, S no son conjuntos consistentes de Craig. Si S es finito y $\{S_1, S_2\}$ es una partición de S, entonces $\langle S_1 \rangle \rightarrow \neg\langle S_2 \cup \{\beta_2\}\rangle$ no tiene interpolante.*

Su formalización es:

```
lemma craig6c:
 assumes hip0: FormulaBeta F
 and hip1: F ∈ S
 and hip2: Comp1 F ∉ S
 and hip3: Comp2 F ∉ S
 and hip4: ¬ ConsistenteCraig ({Comp1 F} ∪ S)
 and hip5: ¬ ConsistenteCraig ({Comp2 F} ∪ S)
 shows ¬ ConsistenteCraig S
proof(unfold ConsistenteCraig-def)
 show ¬(∃ S1 S2. finite S ∧
     (Particion (set S1) (set S2) S) ∧
     (∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2))))
 proof −
  have ∀ S1 S2. ¬ finite S ∨
      ¬(Particion (set S1) (set S2) S) ∨
      ¬(∀ H. ¬interpolante H (Conj S1) (¬.(Conj S2)))
  proof((rule allI)+)
   fix S1 S2
   have finite S ⟶ (¬ Particion (set S1) (set S2) S ∨
                ¬ (∀H. ¬ interpolante H (Conj S1) (¬.Conj S2)))
   proof(rule impI)
    assume hipa: finite S
    { have (Particion (set S1) (set S2) S) ⟶
         ¬ (∀H. ¬ interpolante H (Conj S1) (¬.Conj S2))
     proof(rule impI)
      assume hipb: Particion (set S1) (set S2) S
      show ¬(∀H. ¬ interpolante H (Conj S1) (¬.Conj S2))
      proof −
       have S = (set S1) ∪ (set S2)
        using hipb by(auto, unfold Particion-def, auto)
```

hence $F \in$ *set S1* $\vee$ $F \in$ *set S2* **using** *hip1* **by** *simp*
thus $\neg(\forall H.\ \neg$ *interpolante H* (*Conj S1*) ($\neg$.*Conj S2*))
proof(*rule disjE*)
assume *1*: $F \in$ *set S1*
have ($\neg$(*Particion* (*set* ([*Comp1 F*] @ *S1*))
(*set S2*)
({*Comp1 F*} $\cup$ *S*)) $\vee$
$\neg(\forall H.\ \neg$*interpolante H*
(*Conj* ([*Comp1 F*] @ *S1*))
($\neg$.(*Conj S2*)))) $\wedge$
($\neg$(*Particion* (*set* ([*Comp2 F*] @ *S1*))
(*set S2*)
({*Comp2 F*} $\cup$ *S*)) $\vee$
$\neg(\forall\ H.\ \neg$*interpolante H*
(*Conj* ([*Comp2 F*] @ *S1*))
($\neg$.(*Conj S2*))))
using *hip4 hip5 hipa craig6a2* **by** *blast*
hence (*Particion* (*set S1*) (*set S2*) *S*) $\longrightarrow$
$\neg$ ($\forall H.\ \neg$ *interpolante H* (*Conj S1*) ($\neg$.*Conj S2*))
using *hip0 hip1 hip2 hip3 1 craig6a*[*of F*] **by** *auto*
thus $\neg$ ($\forall H.\ \neg$ *interpolante H* (*Conj S1*) ($\neg$.*Conj S2*))
using *hipb* **by** *auto*
next
assume *2*: $F \in$ *set S2*
have ($\neg$(*Particion* (*set S1*)
(*set* ([*Comp1 F*] @ *S2*))
({*Comp1 F*} $\cup$ *S*)) $\vee$
$\neg(\forall H.\ \neg$*interpolante H*
(*Conj S1*)
($\neg$.((*Conj* ([*Comp1 F*] @ *S2*)))))) $\wedge$
($\neg$(*Particion* (*set S1*)
(*set* ([*Comp2 F*] @ *S2*))
({*Comp2 F*} $\cup$ *S*)) $\vee$
$\neg(\forall H.\ \neg$*interpolante H*
(*Conj S1*)
($\neg$.((*Conj* ([*Comp2 F*] @ *S2*))))))
using *hip4 hip5 hipa craig6b2* **by** *blast*
hence (*Particion* (*set S1*) (*set S2*) *S*) $\longrightarrow$
$\neg$ ($\forall H.\ \neg$ *interpolante H*
(*Conj S1*)
($\neg$.*Conj S2*))
using *hip0 hip1 hip2 hip3 2 craig6b*[*of F*] **by** *auto*
thus $\neg$ ($\forall H.\ \neg$ *interpolante H* (*Conj S1*) ($\neg$.*Conj S2*))

```
            using hipb by auto
          qed
        qed
      qed}
    thus ¬(Particion (set S1) (set S2) S) ∨
        ¬(∀ H. ¬ interpolante H (Conj S1) (¬.Conj S2))
     by auto
   qed
   thus ¬ finite S ∨ ¬(Particion (set S1) (set S2) S) ∨
       ¬ (∀ H. ¬ interpolante H (Conj S1) (¬.Conj S2))
    by auto
  qed
  thus ?thesis by auto
 qed
qed
```

La formalización de la prueba de la propiedad (6) es la siguiente.

```
lemma craig6:
 assumes hip0: FormulaBeta F
 and hip1: F ∈ S
 and hip2: ConsistenteCraig S
 shows ConsistenteCraig ({Comp1 F} ∪ S) ∨ ConsistenteCraig ({Comp2 F} ∪ S)
proof(cases)
 assume Comp1 F ∈ S ∨ Comp2 F ∈ S
 hence {Comp1 F} ∪ S = S ∨ {Comp2 F} ∪ S = S by auto
 thus ConsistenteCraig ({Comp1 F} ∪ S) ∨ ConsistenteCraig ({Comp2 F} ∪ S)
  using hip2 by auto
next
 assume  ¬(Comp1 F ∈ S ∨ Comp2 F ∈ S)
 hence hip3: Comp1 F ∉ S and hip4: Comp2 F ∉ S by auto
 thus ConsistenteCraig ({Comp1 F} ∪ S) ∨ ConsistenteCraig ({Comp2 F} ∪ S)
  using hip0 hip1 hip2 craig6c by auto
qed
```

Finalmente, la formalización de la prueba del lema 7.2.4 es la siguiente.

```
lemma ConsistenciaCraig:
 shows consistenciaP{S::'b formula1 set. ConsistenteCraig S}
proof(unfold consistenciaP-def, rule allI, rule impI)
 let ?C = {S::'b formula1 set. ConsistenteCraig S}
 fix S:: 'b formula1 set
 assume S ∈ ?C
 hence hip: ConsistenteCraig S by auto
 show (∀ P. ¬ (Atomo P ∈ S ∧ ¬.Atomo P ∈ S)) ∧
```

```
    FF ∉ S ∧
    ¬.TT ∉ S ∧
    (∀ F. ¬.¬.F ∈ S ⟶ S ∪ {F} ∈ ?C) ∧
    (∀ F. (FormulaAlfa F) ∧ F ∈ S ⟶
       (S ∪ {Comp1 F, Comp2 F} ∈ ?C)) ∧
    (∀ F. (FormulaBeta F) ∧ F ∈ S ⟶
       (S ∪ {Comp1 F} ∈ ?C ∨ S ∪ {Comp2 F} ∈ ?C))
 proof −
  have (∀ P. ¬ (Atomo P ∈ S ∧ (¬. Atomo P) ∈ S)) using hip craig1 by simp
  moreover
  have FF ∉ S using hip craig2 by auto
  moreover
  have ¬. TT ∉ S using hip craig3 by auto
  moreover
  have ∀ F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈ ?C
  proof(rule allI impI)+
   fix F
   assume hip1: ¬.¬.F ∈ S
   show S ∪ {F} ∈ ?C using hip hip1 craig4 by simp
  qed
  moreover
  have ∀ F. (FormulaAlfa F) ∧ F ∈ S ⟶ (S ∪ {Comp1 F, Comp2 F} ∈ ?C)
  proof(rule allI impI)+
   fix F
   assume FormulaAlfa F ∧ F ∈ S
   thus S ∪ {Comp1 F, Comp2 F} ∈ ?C using hip craig5[of F] by auto
  qed
  moreover
  have ∀ F. (FormulaBeta F) ∧ F ∈ S ⟶
        (S ∪ {Comp1 F} ∈ ?C ∨ S ∪ {Comp2 F} ∈ ?C)
  proof(rule allI impI)+
   fix F
   assume (FormulaBeta F) ∧ F ∈ S
   thus S ∪ {Comp1 F} ∈ ?C ∨ S ∪ {Comp2 F} ∈ ?C
    using hip craig6[of F] by auto
  qed
  ultimately
  show ?thesis by auto
 qed
qed
```

Teorema 7.2.31 (de interpolacion de Craig) *Sea* **L** *un lenguaje proposicional tal que existe una enumeración del conjunto de símbolos proposicionales. Si* $F \to G$ *es una tautología, en-*

tonces tiene un interpolante.

Demostración: La demostración es en forma contrarrecíproca. Supongamos que, $F \to G$ no tiene un interpolante y demostremos que $F \to G$ no es una tautología.

Consideremos los casos $F = \neg G$ y $F \neq \neg G$. Si $F = \neg G$ entonces, $F \to G$ no es tautología. Supongamos que $F \neq \neg G$. Sea $S = \{F, \neg G\}$ y $S_1 = \{F\}$, $S_2 = \{\neg G\}$ una partición de S. Entonces $\langle\{F\}\rangle \to \neg\langle\{\neg G\}\rangle$ no tiene interpolante. En caso contrario, si H es un interpolante de $\langle\{F\}\rangle \to \neg\langle\{\neg G\}\rangle$ entonces, H también sería un interpolante de $F \to G$. Por lo tanto, S es un conjunto consistente de Craig. Así por el lema 7.2.4 y el corolario 6.11.6, S es satisfacible, y por lo tanto $F \to G$ no es tautología.

□

En la prueba del teorema de Craig se usan los siguentes resultados:

Lema 7.2.32 *Si $\neg G \to G$ es una tautología entonces, tiene un interpolante.*

En Isabelle,

```
lemma InterpolacionCraiga:
  assumes tautologia (¬.G →.G)
  shows ∃ H. interpolante H (¬.G) G
```

Lema 7.2.33 *Si H es un interpolante de $\langle\{F\}\rangle \to \neg\langle\{\neg G\}\rangle$, entonces H es interpolante de $F \to G$.*

En Isabelle,

```
lemma InterpolacionCraigb:
  assumes interpolante H (Conj [F]) (¬. Conj [¬.G])
  shows interpolante H F G
```

Lema 7.2.34 *Si $F \neq G$ y $F \to G$ no tiene interpolante, entonces $\{F, \neg G\}$ es un conjunto consistente de Craig.*

Su formalización es:

```
lemma InterpolacionCraigc:
  assumes hip1: F ≠ ¬.G and hip2: ∀ H. ¬ interpolante H F G
  shows ConsistenteCraig {F, ¬.G}
proof −
  have finite {F, ¬.G} by simp
  moreover
```

```
 have Particion {F} {¬.G} {F, ¬.G} using hip1 by(unfold Particion-def, auto)
 moreover
 have ¬(∃ H. interpolante H (Conj [F]) (¬. Conj [¬.G ]))
 proof −
  { assume ∃ H. interpolante H (Conj [F]) (¬. Conj [¬.G ])
    hence False using InterpolacionCraigb hip2 by auto }
  thus ¬(∃ H. interpolante H (Conj [F]) (¬. Conj [¬.G ])) by auto
 qed
 ultimately
 have finite {F, ¬.G} ∧ Particion {F} {¬.G} {F, ¬.G} ∧
     (∀ H. ¬ interpolante H (Conj [F]) (¬. Conj [¬.G ]))
  by auto
 hence finite {F, ¬.G} ∧ Particion (set [F]) (set [¬.G]) {F, ¬.G} ∧
      (∀ H. ¬ interpolante H (Conj [F]) (¬. Conj [¬.G ]))
  by auto
 hence ∃ S1 S2. finite {F, ¬.G} ∧ Particion (set S1) (set S2) {F, ¬.G} ∧
      (∀ H. ¬ interpolante H (Conj S1) (¬. (Conj S2)))
  by blast
 thus ConsistenteCraig {F, ¬.G}
  by(auto, unfold ConsistenteCraig-def, auto)
qed
```

La formalización del teorema de interpolación de Craig 7.2.31 es la siguiente.

```
theorem InterpolacionCraig1:
 assumes hip1: ∃ g. enumeracion (g:: nat ⇒ 'b)
 and hip2: tautologia ((F::'b formula1) →.G)
 shows ∃ H. interpolante H F G
proof(cases)
 assume F = ¬.G
 thus ∃ H. interpolante H F G using hip2 InterpolacionCraiga by auto
next
 assume hip3: F ≠ ¬.G
 show ∃ H. interpolante H F G
 proof −
  let ?C = {S::'b formula1 set. ConsistenteCraig S}
  { assume ¬(∃ H. interpolante H F G)
    hence ConsistenteCraig {F, ¬.G}
     using hip3 hip2 InterpolacionCraigc by auto
    hence {F, ¬.G} ∈ ?C by simp
    hence satisfacible {F, ¬.G}
     using hip1 ConsistenciaCraig ConjuntosatisfacibleP by auto
    hence ¬ tautologia (F →.G) using  TautSatis by auto}
  thus ?thesis using assms by auto
```

```
  qed
qed
```

En el caso del lenguaje proposicional en el que identificamos los símbolos proposicionales con los números naturales, tenemos el siguiente corolario.

Corolario 7.2.35 *Sea* **L** *el lenguaje proposicional en el que los símbolos proposicionales son los números naturales. Si* $F \to G$ *es una tautología, entonces tiene un interpolante.*

Demostración: Puesto que existe una enumeración de $\mathbb{N}$ (lema 6.6.9) entonces, por el teorema anterior, se tiene el resultado.

□

Su formalización es:

```
corollary InterpolacionCraig2:
  assumes tautologia ((F:: nat formula1) →.G)
  shows ∃ H. interpolante H F G
using enum-nat assms InterpolacionCraig1
by auto
```

Capítulo 8

Sintaxis y semántica de la lógica de primer orden

En este capítulo presentamos la sintaxis y la semántica de la lógica de primer orden y su formalización en Isabelle.

8.1 Sintaxis de la lógica de primer orden

Definición 8.1.1 *El **alfabeto** de un lenguaje de primer orden se compone de los siguientes símbolos.*

1. ***Símbolos lógicos:***
   - *variables:* $x_0,\ x_1,\ x_2,\ \ldots$
   - *conectivas:* $\bot, \top, \neg, \wedge, \vee, \rightarrow$
   - *cuantificadores:* $\forall, \exists$
   - *símbolos de puntuación:* $'('\ ,\ ')'\ ,\ ','$
2. ***Símbolos no lógicos:***
   - *símbolos de relación*
   - *símbolos de función*

Los lenguajes de primer orden tienen en común los símbolos lógicos, sin embargo cada lenguaje está caracterizado por sus propios símbolos de relación y de función.

Definición 8.1.2 *Un* **lenguaje de primer orden** *está determinado por:*

1. *Un conjunto numerable* **F** *de símbolos de función tal que cada elemento de* **F** *tiene asociado un número entero no negativo, su aridad. Si* $f \in \mathbf{F}$ *y* n *es el número asociado a* f*, decimos que* f *es un símbolo de función* n*-aria. En el caso particular* $n = 0$*, los símbolos de función* 0*-arias se denominan* **símbolos de constante**.

2. *Un conjunto numerable* **R** *de símbolos de relación o predicado tal que cada elemento de* **R** *tiene asociado un número entero positivo, su aridad. Si* $P \in \mathbf{R}$ *y* n *es el número asociado a* P*, decimos que* P *es un símbolo de relación* n*-aria.*

El lenguaje de primer orden determinado por **F** y **R** lo denotaremos por $\mathbf{L}(\mathbf{R}, \mathbf{F})$ o simplemente por **L** cuando no sea necesario hacer énfasis en los conjuntos **F** y **R**.

Los *términos* y las *fórmulas* de un lenguaje de primer orden $\mathbf{L}(\mathbf{R}, \mathbf{F})$ se definen de la siguiente manera.

Definición 8.1.3 *El conjunto de* **términos** *es el conjunto más pequeño que satisface las siguientes condiciones:*

1. *Las variables son términos.*

2. *Si* $f \in \mathbf{F}$ *es un símbolo de función* n*-aria y* $t_1, t_2, \ldots, t_n$ *son términos entonces,* $f(t_1, t_2, \ldots, t_n)$ *es un término.*

Nota 8.1.4 *En el caso* $n = 0$ *el término* $f(\,)$ *correspondiente al símbolo de constante* f*, se denotará simplemente por* f*.*

Definición 8.1.5 *Las* **fórmulas atómicas** *de* **L** *están definidas de la siguiente manera.*

Si $P \in \mathbf{R}$ *es un símbolo de relación* n*-aria y* $t_1, t_2, \ldots, t_n$ *son términos entonces,* $P(t_1, t_2, \ldots, t_n)$ *es una fórmula atómica.*

Definición 8.1.6 *El conjunto de* **fórmulas** *de* **L** *es el conjunto más pequeño que satisface las siguientes condiciones:*

1. *Los símbolos* $\bot$ *y* $\top$ *son fórmulas.*

2. *Cualquier fórmula atómica es una fórmula de* **L**.

3. *Si* F *es una fórmula entonces* $\neg F$ *es una fórmula de* **L**.

4. *Si* F *y* G *son fórmulas, entonces* $(F \wedge G)$, $(F \vee G)$ *y* $(F \to G)$ *son fórmulas.*

5. *Si* F *es una fórmula y* x *es una variable, entonces* $\forall\, x\, F$ *y* $\exists\, x\, F$ *son fórmulas.*

8.1.1 Índices de de Bruijn

En la definición anterior, si para la construcción de los términos identificamos las variables x_i con los números naturales, obtenemos lo que se denominan *términos con índices de de Bruijn*. De esta forma, cada índice de de Bruijn es un número natural $n \geq 0$ que representa una ocurrencia de una variable en un término. Para la representación de fórmulas cuantificadas utilizando índices de de Bruijn se omite la variable que acompaña a cada cuantificador, y en este caso un índice de de Bruijn en una fórmula denota el número de cuantificadores que aparecen entre la variable respectiva y el cuantificador que liga a la variable. Por ejemplo, las siguientes fórmulas

- $\forall\, x\, P(x, z) \wedge \exists\, y\, P(y, x)$
- $\forall\, x\, (P(x, y) \wedge \exists\, x\, P(x, y))$
- $\forall\, z \exists\, y\, (P(h(z), y) \wedge \forall\, x\, (Q(x,y) \rightarrow \forall\, x\, R(z,x)))$

en donde usamos las letras x, y, z para denotar las variables x_0, x_1, x_2, se pueden escribir, utilizando índices de de Bruijn de la siguiente forma

- $\forall\ P(0, 1) \wedge \exists\ P(0, 2)$
- $\forall\ (P(0, 1) \wedge \exists\ P(0, 2))$
- $\forall \exists\ (P(h(1), 0) \wedge \forall\, (Q(0,1) \rightarrow \forall\ R(3,0)))$

En este trabajo, en la formalización de las propiedades que se estudian sobre la metateoría de la lógica de primer orden se representarán las fórmulas utilizando la notación de de Bruijn.

La ventaja de esta representación, como se observará más adelante, es que facilita la mecanización de las reglas y propiedades que tienen que ver con la introducción y eliminación de los cuantificadores. En particular, permite automatizar de manera simple y eficiente el proceso de sustituir en una fórmula las "ocurrencias libres" de una variable x, por un "término t libre" para x; además permite considerar como idénticas las fórmulas que difieren únicamente en los nombres de sus "variables cuantificadas", por ejemplo, las fórmulas $\forall\, x\, P(x, z) \wedge \exists\, y\, P(y, x)$ y $\forall\, y\, P(y, z) \wedge \exists\, x\, P(x, y)$ tienen la misma representación, $\forall\ P(0, 1) \wedge \exists\ P(0, 2)$, usando la notación de de Bruijn.

Los siguientes tipos de datos definen en Isabelle los términos y las fórmulas del lenguaje $\mathbf{L}(\mathbf{R}, \mathbf{F})$ usando la notación de de Bruijn. En la definición se usan los tipos *'a* y *'b* para representar los conjuntos $\mathbf{F}$ y $\mathbf{R}$. Así, *'a* denotará el tipo de los símbolos de función y *'b* el tipo de los símbolos de predicado.

datatype *'a term = Var nat | Term 'a 'a term list*

datatype $('a, 'b)$ *form* =
FF'
| *TT'*
| *Atom 'b 'a term list*
| *And* $('a, 'b)$ *form* $('a, 'b)$ *form* (**infixl** $\wedge$. *35*)
| *Or* $('a, 'b)$ *form* $('a, 'b)$ *form* (**infixl** $\vee$. *30*)
| *Impl* $('a, 'b)$ *form* $('a, 'b)$ *form* (**infixl** $\rightarrow$. *25*)
| *Neg* $('a, 'b)$ *form* ($\neg$.(-) [*40*] *40*)
| *Forall* $('a, 'b)$ *form* ($\forall$.(-) [*9*] *10*)
| *Exists* $('a, 'b)$ *form* ($\exists$.(-) [*9*] *10*)

Por ejemplo, la fórmula

$$\forall \exists\ (P(h(1), 0) \rightarrow \forall\ R(3,0)))$$

se representa por,

$$\forall . \exists .\ Atom\ P\ [Term\ h\ [Var\ 1], Var\ 0] \rightarrow . \forall .\ Atom\ R\ [Var\ 3, Var\ 0].$$

Los términos como $(Term\ f\ ts1)$ y $(Term\ f\ ts2)$ con el mismo nombre de función y las longitudes de las listas *ts1* y *ts1* distintas, hacen referencia a símbolos de función diferentes. Análogamente para símbolos de predicado.

8.1.2 Símbolos de función de una fórmula

En algunas definiciones se hace referencia a que un símbolo de constante no pertenezca a una fórmula, por ejemplo en los cálculos de prueba que incluyen las reglas de introducción del cuantificador universal y de eliminación del cuantificador existencial, se requiere que la variable cuantificada sea reemplazada por un símbolo de constante *arbitrario* (parámetro) que no ocurra en ninguna de las premisas de las reglas. Al considerar los símbolos de constante como símbolos de función sin argumentos, la anterior condición se puede verificar mostrando que el símbolo de constante dado no es ninguno de los símbolos de función que aparecen en cada una de las respectivas premisas. Para la verificación mecánica definimos la función *(simbfun F)* que determina el conjunto de símbolos de función de una fórmula *F*. Inicialmente se definen las siguientes funciones auxiliares *(simbfunt t)* y *(simbfunts ts)* que permiten hallar el conjunto de símbolos de función de un término *t* y de una lista de términos *ts* respectivamente.

primrec *simbfunt* :: *'a term* $\Rightarrow$ *'a set* **and** *simbfunts* :: *'a term list* $\Rightarrow$ *'a set* **where**
simbfunt (*Var n*) = {}
| *simbfunt* (*Term f ts*) = {*f*} $\cup$ *simbfunts ts*
| *simbfunts* [] = {}
| *simbfunts* (*t* # *ts*) = (*simbfunt t* $\cup$ *simbfunts ts*)

La siguiente es la definición de la función que halla el conjunto de símbolos de

función de una fórmula.

primrec *simbfun* :: ('*a*, '*b*) *form* ⇒ '*a set* **where**
simbfun FF′ = {}
| *simbfun TT′* = {}
| *simbfun* (*Atom P ts*) = *simbfunts ts*
| *simbfun* (*F* ∧. *G*) = *simbfun F* ∪ *simbfun G*
| *simbfun* (*F* ∨. *G*) = *simbfun F* ∪ *simbfun G*
| *simbfun* (*F* →.*G*) = *simbfun F* ∪ *simbfun G*
| *simbfun* (¬. *F*) = *simbfun F*
| *simbfun* (∀ . *F*) = *simbfun F*
| *simbfun* (∃ . *F*) = *simbfun F*

Por ejemplo, $(simbfun\ \ P\ (h\ (1), g\ (0)) \wedge R\ (f\ (3, 0, c))) = \{h, g, f, c\}$.

8.1.3 Sustituciones

En esta sección se formaliza el proceso de sustituir en una fórmula las ocurrencias libres de una variable x por un término libre para x.

Si se tiene una fórmula de la forma $\forall x_i F$ se dice que: la ocurrencia de la variable x_i en $\forall x_i$ está *universalmente cuantificada*, F es el *alcance* de $\forall x_i$, y cada ocurrencia de x_i en F está *ligada* por $\forall x_i F$. Similarmente se tienen los conceptos anteriores para una fórmula de la forma $\exists x_i F$.

Cada ocurrencia de una variable x_i, en una fórmula, que no está cuantificada o ligada se dice que es una *ocurrencia libre*. Una variable x_i es una *variable libre de una fórmula* si hay ocurrencias libres de x_i en la fórmula. Una fórmula es *cerrada* si no tiene variables libres.

Para indicar que todas las ocurrencias libres de la variable x son sustituidas por un término t en una fórmula F, se denota F por $F(x)$ y se utiliza la notación $F(t)$ para la fórmula que resulta de la sustitución.

Se dice que un *término t es libre para la variable x en una fórmula F* si, la sustitución de todas las ocurrencias libres de x por t en F no introduce ocurrencias nuevas de variables ligadas.

En seguida se formaliza el proceso de sustituir en una fórmula las ocurrencias libres de una variable por un término, usando índices de de Bruijn. El método garantiza que el término respectivo sea libre para la variable que sustituye.

Primero se definen las funciones auxiliares ϑ y Θ para sustituir en términos y listas de términos respectivamente, variables por términos: en la sustitucion de la variable con índice k por el término s, el valor k es reemplazado por s, el índice j es reemplazado por $j - 1$ para cada $j > k$ y no se modifican los índices menores que i.

```
primrec
  sustt :: 'a term ⇒ 'a term ⇒ nat ⇒ 'a term (-[-'ϑ-] [300, 0, 0] 300) and
  sustts :: 'a term list ⇒ 'a term ⇒ nat ⇒ 'a term list (-[-'Θ -] [300, 0, 0] 300)
where
  (Var i)[s ϑ k] = (if k < i then Var (i − 1) else if i = k then s else Var i)
| (Term f ts)[s ϑ k] = Term f (ts[s Θ k])
| [][s Θ k] = []
| (t # ts)[s Θ k] = t[s ϑ k] # ts[s Θ k]
```

Así, por ejemplo

$$(f(g(0), 1, 2, 3))(s\theta 1) = f(g(0), s, 1, 2)$$

La siguiente función *elevat* incrementa en uno el valor de los índices de las variables que aparecen en un término. Para su definición se usa la función auxiliar *elevats* que realiza el mismo proceso sobre lista de términos.

```
primrec elevat :: 'a term ⇒ 'a term and  elevats :: 'a term list ⇒ 'a term list
where
  elevat (Var i) = Var (Suc i)
| elevat (Term f ts) = Term f (elevats ts)
| elevats [] = []
| elevats (t # ts) = elevat t # elevats ts
```

Por ejemplo, $(elevat f[g[0], 1, 2, 3]) = f[g[1], 2, 3, 4]$.

La siguiente es la definición de la función que sustituye en una fórmula las ocurrencias libres de una variable por un término libre para la variable.

```
primrec sust :: ('a, 'b) form ⇒ 'a term ⇒ nat ⇒ ('a, 'b) form (-[-'\-] [300, 0, 0] 300) where
  FF'[s\k] = FF'
| TT'[s\k] = TT'
| (Atom P ts)[s\k] = Atom P (ts[s Θ k])
| (F ∧. G)[s\k] = ((F[s\k]) ∧.(G[s\k]))
| (F∨. G)[s\k] = ((F[s\k]) ∨.(G[s\k]))
| (F→. G)[s\k] = ((F[s\k]) →.(G[s\k]))
| (¬.F)[s\k] = (¬.(F[s\k]))
| (∀ . F)[s\k] = (∀ . (F[elevat s\Suc k]))
| (∃ .F)[s\k] = (∃ . (F[elevat s\Suc k]))
```

Nótese que $F[s/k]$ es la fórmula que resulta de sustituir en F los índices $k + i$ por s para todo i tal que i es el número de cuantificadores bajo cuyo alcance está el índice $k + i$. En cada uno de los siguientes ejemplos realizamos la sustitución en notación estándar y de de Bruijn, y la fórmula que resulta en notación de de Bruijn la escribimos de nuevo en notación estándar.

1.a $(\forall x(P(x,y,z,v) \wedge \exists y(P(x,y,z,v) \wedge \forall z(P(x,y,z,v)))))[w/y] =$
$\forall x(P(x,w,z,v) \wedge \exists y(P(x,y,z,v) \wedge \forall z(P(x,y,z,v))))$

1.b $(\forall(P(0,1,2,3) \wedge \exists(P(1,0,3,4) \wedge \forall(P(2,1,0,5)))))[6/0] =$
$\forall(P(0,7,1,2) \wedge \exists(P(1,0,2,3) \wedge \forall(P(2,1,0,4))))$

1.c $\forall x(P(x,x_6,x_0,x_1) \wedge \exists y(P(x,y,x_0,x_1) \wedge \forall z(P(x,y,z,x_1)))$

2.a $(\forall x(P(x,y,z,v) \wedge \exists y(P(x,y,z,v) \wedge \forall z(P(x,y,z,v)))))[w/z] =$
$\forall x(P(x,y,w,v) \wedge \exists y(P(x,y,w,v) \wedge \forall z(P(x,y,z,v))))$

2.b $(\forall(P(0,1,2,3) \wedge \exists(P(1,0,3,4) \wedge \forall(P(2,1,0,5)))))[6/1] =$
$\forall(P(0,1,7,2) \wedge \exists(P(1,0,8,3) \wedge \forall(P(2,1,0,4))))$

2.c $\forall x(P(x,x_0,x_6,x_1) \wedge \exists y(P(x,y,x_6,x_1) \wedge \forall z(P(x,y,z,x_1))))$

3.a $(\forall x(P(x,y,z,v) \wedge \exists y(P(x,y,z,v) \wedge \forall z(P(x,y,z,v)))))[w/v] =$
$\forall x(P(x,y,z,w) \wedge \exists y(P(x,y,z,w) \wedge \forall z(P(x,y,z,w))))$

3.b $(\forall(P(0,1,2,3) \wedge \exists(P(1,0,3,4) \wedge \forall(P(2,1,0,5)))))[6/2] =$
$\forall(P(0,1,2,7) \wedge \exists(P(1,0,3,8) \wedge \forall(P(2,1,0,9))))$

3.c $\forall x(P(x,x_0,x_1,x_6) \wedge \exists y(P(x,y,x_1,x_6) \wedge \forall z(P(x,y,z,x_6)))).$

Obsérvese que por la propiedad de i, si el índice $k+i$ ocurre en la fórmula F entonces corresponde a una variable libre.

Nota 8.1.7 *En este trabajo las únicas sustituciones que se realizarán son aquellas que permiten la eliminación de los cuantificadores, es decir, en el proceso de eliminación de los cuantificadores correspondiente a las fórmulas $\forall x F(x), \exists x F(x)$, se reemplazan en F las ocurrencias libres de x por un término t, libre para x. La fórmula $F(t)$ que resulta se obtiene formalmente, utilizando la anterior definición y la notación de de Bruijn , mediante la sustitución $F[t/0]$.*

Por ejemplo, si en la fórmula $\forall x \exists y P(x,y)$ queremos eliminar el cuantificador universal reemplazando la variable x en $\exists y P(x,y)$ por el término $h(x,y)$, podemos hacerlo mediante un renombramiento de variables, por ejemplo y por y' en $h(x,y)$, para que el término resulte libre para x en la sustitucion.

Así, al hacer la sustitución $(\exists y P(x,y))[h(x,y')/x]$ obtenemos $\exists y P(h(x,y'),y)$.

Por otro lado, usando la notación de de Bruijn y la definición de sustitución tenemos que, $\exists P(1,0)[h(0,1)/0] = \exists P(h(1,2),0)$; nótese que la última fórmula escrita en notación estándar es $\exists x P(h(x_0,x_1),x)$.

8.2 Semántica de la lógica de primer orden

En esta sección estudiamos la semántica de la lógica de primer orden.

8.2.1 Valor de verdad de una fórmula

El significado de las fórmulas de un lenguaje de primer orden está determinado por la elección de un *dominio* o universo y la *interpretación* de los símbolos no lógicos. Un dominio junto con una interpretación forman una *estructura*.

En esta sección se formaliza la semántica de la lógica de primer orden. Esta formalización está basada en las siguientes definiciones.

Definición 8.2.1 *Una* **estructura** *para un lenguaje de primer orden* $\mathbf{L}(\mathbf{R}, \mathbf{F})$*, o* $\mathbf{L}$*-estructura, es una terna* $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$*, en donde:*

> $\mathcal{D}$ *es un conjunto no vacío, llamado el* **dominio** *de* $\mathcal{M}$*.*
>
> $\mathcal{R}$ *es una función, llamada* **interpretación de símbolos de relación** *que asocia a cada símbolo de relación n-aria* $P \in \mathbf{R}$*, una relación n-aria,* $P^{\mathcal{R}} \subseteq \mathcal{D}^n$*.*
>
> $\mathcal{F}$ *es una función, llamada* **interpretación de símbolos de función***, que asocia a cada símbolo de función n-aria* $f \in \mathbf{F}$ *una función n-aria,* $f^{\mathcal{F}} : \mathcal{D}^n \to \mathcal{D}$*.*

La estructura $\mathcal{M}(\mathcal{D}, \mathcal{R}, \mathcal{F})$ la denotaremos por $\mathcal{M}$ cuando no sea necesario hacer énfasis en las funciones $\mathcal{F}$ y $\mathcal{R}$.

La representación en Isabelle de los conceptos de interpretación de símbolos de función y de símbolos de relación son funciones de la forma,
$I_F :: 'a \Rightarrow 'c\ list \Rightarrow 'c$ e $I_R :: 'b \Rightarrow 'c\ list \Rightarrow bool$.
Donde $'a$, $'b$ y $'c$ son los tipos de **F**, **R** y $\mathcal{D}$ respectivamente.

Para poder asignar a cada fórmula de un lenguaje **L** un *valor de verdad* en una **L**-estructura $\mathcal{M}$, es necesario asignar a cada término de **L** un valor en el dominio de la estructura $\mathcal{M}$. Esto último se hace de manera progresiva comenzando por la asignación de valores a las variables.

Definición 8.2.2 *Una asignación de valores a variables en una estructura* $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ *es una función A del conjunto de variables al conjunto* $\mathcal{D}$*. El valor de la variable v con respecto a la asignación A se denota por* v^A*.*

La representación en Isabelle de una asignación A es una función de la forma,
$I_A :: nat \Rightarrow 'c$.

El valor de un término se define recursivamente de la siguiente manera.

Definición 8.2.3 *Sea $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una* **L***-estructura y A una asignación en $\mathcal{M}$. A cada término t de* **L***, se le asocia un valor $t^{\mathcal{F},A}$ en $\mathcal{D}$ como sigue:*

Si v es una variable, $v^{\mathcal{F},A} = v^A$.

Si f es un símbolo de función n-aria, $n \geq 0$, $[f(t_1, \ldots, t_n)]^{\mathcal{F},A} = f^{\mathcal{F}}(t_1^{\mathcal{F},A}, \ldots, t_n^{\mathcal{F},A})$.

Este concepto se formaliza de la siguiente forma:

```
primrec
 evalt  :: ('a ⇒ 'c list ⇒ 'c) ⇒ (nat ⇒ 'c) ⇒ 'a term ⇒ 'c and
 evalts :: ('a ⇒ 'c list ⇒ 'c)⇒ (nat ⇒ 'c)⇒ 'a term list ⇒ 'c list
where
 evalt  I_F I_A (Var n) = I_A  n
| evalt  I_F I_A (Term f ts) = I_F f (evalts I_F I_A ts)
| evalts I_F I_A [] = []
| evalts I_F I_A (t # ts) = evalt I_F I_A t # evalts I_F I_A ts
```

Por ejemplo, $[f(x, h(y, x))]^{\mathcal{F},A} = f^{\mathcal{F}}(x^A, h^{\mathcal{F}}(y^A, x^A))$ o también expresado en notación de de Bruijn $(evalt\ I_F\ I_A\ f(3, h(2, 3))) = I_F\, f(I_A\, 3, I_F\, h(I_A\, 2, I_A\, 3))$.

Una vez formalizada la asociación de valores a términos, precisamos ahora el concepto de valor de verdad de una fórmula con respecto a una estructura y una asignación.

Definición 8.2.4 *Los **valores de verdad** que consideraremos son,* V *(que se interpreta como **verdadero**) y* F *(que se interpreta como **falso**).*

Para definir el valor de verdad de fórmulas cuantificadas usaremos el siguiente concepto.

Definición 8.2.5 *Sean A y A' dos asignaciones en una estructura $\mathcal{M}$ y x una variable. Decimos que A' es una x-***variante** *de A, si A y A' asignan el mismo valor a cada variable excepto, posiblemente, a x.*

Definición 8.2.6 *Sea $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una* **L***-estructura y A una asignación en $\mathcal{M}$. A cada fórmula F de* **L***, le asociamos un valor de verdad $F^{\mathcal{R},\mathcal{F},A}$ como sigue:*

1. *Casos atómicos,*
   - $\top^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$

- $\bot^{\mathcal{R},\mathcal{F},A} = \mathsf{F}$
- $[P(t_1,\ldots,t_n)]^{\mathcal{R},\mathcal{F},A} = \mathsf{V} \Longleftrightarrow P^{\mathcal{R}}(t_1^{\mathcal{F},A},\ldots,t_n^{\mathcal{F},A})$

2. $[\neg F]^{\mathcal{R},\mathcal{F},A} = \begin{cases} \mathsf{V}, & \textit{si } F^{\mathcal{R},\mathcal{F},A} = \mathsf{F} \\ \mathsf{F}, & \textit{en caso contrario.} \end{cases}$

3. $[F \wedge G]^{\mathcal{R},\mathcal{F},A} = \begin{cases} \mathsf{V}, & \textit{si } F^{\mathcal{R},\mathcal{F},A} = G^{\mathcal{R},\mathcal{F},A} = \mathsf{V} \\ \mathsf{F}, & \textit{en caso contrario.} \end{cases}$

4. $[F \vee G]^{\mathcal{R},\mathcal{F},A} = \begin{cases} \mathsf{F}, & \textit{si } F^{\mathcal{R},\mathcal{F},A} = G^{\mathcal{R},\mathcal{F},A} = \mathsf{F} \\ \mathsf{V}, & \textit{en caso contrario.} \end{cases}$

5. $[F \to G]^{\mathcal{R},\mathcal{F},A} = \begin{cases} \mathsf{F}, & \textit{si } F^{\mathcal{R},\mathcal{F},A} = \mathsf{V} \textit{ y } G^{\mathcal{R},\mathcal{F},A} = \mathsf{F} \\ \mathsf{V}, & \textit{en caso contrario.} \end{cases}$

6. $(\forall x F)^{\mathcal{R},\mathcal{F},A} = \mathsf{V} \Longleftrightarrow F^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$ *para toda x-variante* A' *de* A.

7. $(\exists x F)^{\mathcal{R},\mathcal{F},A} = \mathsf{V} \Longleftrightarrow F^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$ *para alguna x-variante* A' *de* A.

Para la formalización de las definiciones anteriores utilizamos los valores *False* y *True* del tipo *bool* y la implementación predefinida en Isabelle de la semántica de las conectivas y cuantificadores.

En la formalización del valor de verdad de una fórmula cuantificada, al interpretar el índice de la variable cuantificada, este se reemplaza por la correspondiente variable y recursivamente se interpreta la fórmula que está bajo el alcance del cuantificador. En este proceso se usa una función, denotada por $I_A\langle i{:}a\rangle$ que modifica la asignación I_A, reemplazando el valor $I_A(i)$ por a y el valor $I_A(j)$ por $I_A(j-1)$ para cada variable con índice $j > i$, y dejando sin modificar los valores de las variables con índices menores que i.

definition *despl* :: $(nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow nat \Rightarrow 'a$ $(\text{-}\langle\text{-:-}\rangle\ [90, 0, 0]\ 91)$ **where**
$I_A\langle i{:}a\rangle = (\lambda j.\ \textit{if } j < i \textit{ then } I_A\ j \textit{ else if } j = i \textit{ then } a \textit{ else } I_A\ (j-1))$

Por ejemplo, $(evalt\ I_A\langle 2{:}t\rangle\ If(1, h(2,3))) = If\ (I_A\ 1\ , I\,h(t, I_A\ 2))$.

La siguiente es la formalización de la función que asigna un valor de verdad a las fórmulas con respecto a cualesquier estructura $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ y asignación de valores a variables A.

primrec *eval* :: $('b \Rightarrow 'c\ list \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c\ list \Rightarrow 'c) \Rightarrow (nat \Rightarrow 'c) \Rightarrow ('a, 'b)\ form \Rightarrow bool$
where
$eval\ I_R\ I_F\ I_A\ \ FF' = False$

$| \mathit{eval}\ I_R\ I_F\ I_A\ TT' = \mathit{True}$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (\mathit{Atom}\ P\ ts) = I_R\ P\ (\mathit{evalts}\ I_F\ I_A\ ts)$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (F \wedge. G) = ((\mathit{eval}\ I_R\ I_F\ I_A\ F) \wedge (\mathit{eval}\ I_R\ I_F\ I_A\ G))$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (F \vee. G) = ((\mathit{eval}\ I_R\ I_F\ I_A\ F) \vee (\mathit{eval}\ I_R\ I_F\ I_A\ G))$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (F \rightarrow. G) = ((\mathit{eval}\ I_R\ I_F\ I_A\ F) \longrightarrow (\mathit{eval}\ I_R\ I_F\ I_A\ G))$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (\neg. F) = (\neg\ (\mathit{eval}\ I_R\ I_F\ I_A\ F))$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (\forall\,. F) = (\forall x.\ \mathit{eval}\ I_R\ I_F\ (I_A\langle 0{:}x\rangle)\ F)$
$| \mathit{eval}\ I_R\ I_F\ I_A\ (\exists\,. F) = (\exists x.\ \mathit{eval}\ I_R\ I_F\ (I_A\langle 0{:}x\rangle)\ F)$

Por ejemplo, el valor de la fórmula

$$\forall(P(0, c, h(2,1)) \rightarrow \exists P(1, c, 3, 0)) \tag{8.1}$$

en $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ con la asignación A es,

$$\forall x(P^{\mathcal{R}}(x, c^{\mathcal{F}}, h^{\mathcal{F}}(1^A, 0^A)) \rightarrow (\exists y P^{\mathcal{R}}(x, c^{\mathcal{F}}, 1^A, y)) \tag{8.2}$$

Obsérvese que se obtiene una expresión del metalenguaje. Para verlo más detalladamente, la representación de (8.1) en Isabelle es,
$\forall.\ \mathit{Atom}\ P[\mathit{Var}\ 0,\ \mathit{Term}\ c[\,],\ \mathit{Term}\ h[\mathit{Var}\ 2,\ \mathit{Var}\ 1]] \rightarrow .$
$(\exists.\ \mathit{Atom}\ P[\mathit{Var}\ 1, \mathit{Term}\ c[\,], \mathit{Var}\ 3, \mathit{Var}\ 0])$
y la de (8.2) es,
$\forall x.\ I_R\ P[x, I_F\ c[\,], I_F\ h[I_A\ 1, I_A\ 0] \rightarrow (\exists y.\ I_R\ P[x, I_F\ c[\,], I_A\ 1, y])$.

En el siguiente ejemplo se demuestra en Isar que la fórmula,
$\exists y \forall x P(y, x) \rightarrow \forall y \exists x P(x, y)$ es verdadera en cualquier interpretación.

lemma *eval-conmuta-existe-paratodo1*:
eval $I_R\ I_F\ I_A$ $((\exists.(\forall.(\mathit{Atom}\ P\ [\mathit{Var}\ 1, \mathit{Var}\ 0]))) \rightarrow. (\forall.(\exists.(\mathit{Atom}\ P\ [\mathit{Var}\ 0, \mathit{Var}\ 1]))))$
proof −
have *eval* $I_R\ I_F\ I_A$ $((\exists.(\forall.(\mathit{Atom}\ P\ [\mathit{Var}\ 1, \mathit{Var}\ 0]))) \rightarrow.$
$(\forall.(\exists.(\mathit{Atom}\ P\ [\mathit{Var}\ 0, \mathit{Var}\ 1])))) \equiv$
$(\exists y.\ \forall x.\ I_R\ P\ [(I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 1, (I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 0]) \longrightarrow$
$(\forall y.\ \exists x.\ I_R\ P\ [(I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 0, (I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 1])$
by *auto*
moreover
have $(\exists y.\ \forall x.\ I_R\ P\ [(I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 1, (I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 0]) \longrightarrow$
$(\forall y.\ \exists x.\ I_R\ P\ [(I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 0, (I_A\langle 0{:}y\rangle\langle 0{:}x\rangle)\ 1]) \equiv$
$(\exists y.\ \forall x.\ I_R\ P\ [y, x]) \longrightarrow (\forall y.\ \exists x.\ I_R\ P\ [x, y])$
by *simp*
moreover
have $(\exists y.\ \forall x.\ I_R\ P\ [y, x]) \longrightarrow (\forall y.\ \exists x.\ I_R\ P\ [x, y])$
by *auto*
ultimately
show *?thesis* **by** *blast*

qed

En forma más automatizada tenemos otra demostración del ejemplo anterior.

lemma *eval-conmuta-existe-paratodo2*:
eval I_R I_F I_A $((\exists .(\forall .(Atom\ P\ [Var\ 1, Var\ 0]))) \rightarrow .\ (\forall .(\exists .(Atom\ P\ [Var\ 0, Var\ 1]))))$
proof −
have $(\exists\, y.\ \forall\, x.\ I_R\ P\ [y, x]) \longrightarrow (\forall\, y.\ \exists\, x.\ I_R\ P\ [x, y])$ **by** *auto*
thus *?thesis* **by** *simp*
qed

8.2.2 Satisfacibilidad y Consecuencia lógica

Definición 8.2.7 *Sean F una fórmula y S un conjunto de fórmulas del lenguaje* **L**, $\mathcal{M} = (\mathcal{D}, \mathcal{F}, \mathcal{R})$ *una* **L***-estructura y A una asignación en* $\mathcal{M}$.

1. *F es* **válida en** $\mathcal{M}$ **respecto de** *A, si* $F^{\mathcal{F},\mathcal{R},A} = \mathsf{V}$, *y se representa por* $\models_{\mathcal{M},A} F$.
2. *F es* **válida en** $\mathcal{M}$, *si* $\models_{\mathcal{M},A} F$ *para toda asignación A, y se representa por* $\models_{\mathcal{M}} F$. *En este caso se dice que* $\mathcal{M}$ *es un* **modelo** *de F.*
3. *F es* **válida**, *si* $\models_{\mathcal{M}} F$ *para toda estructura* $\mathcal{M}$, *y se representa por* $\models F$.
4. *S es* **válido en** $\mathcal{M}$ **respecto de** *A, si para toda fórmula* $F \in S$ *se tiene que* $\models_{\mathcal{M},A} F$, *y se representa por* $\models_{\mathcal{M},A} S$. *S es* **satisfacible en** $\mathcal{M}$, *si existe A tal que* $\models_{\mathcal{M},A} S$.
5. *S es* **válido en** $\mathcal{M}$, *si* $\models_{\mathcal{M},A} S$ *para toda asignación A, y se representa por* $\models_{\mathcal{M}} S$.
6. *F es* **consecuencia lógica de** *S* **en** $\mathcal{M}$ **respecto de** *A, si* $\models_{\mathcal{M},A} S$ *implica que* $\models_{\mathcal{M},A} F$, *y se representa por* $S \models_{\mathcal{M},A} F$.
7. *F* **es consecuencia lógica de** *S* **en** $\mathcal{M}$, *si* $\models_{\mathcal{M}} S$ *implica que* $\models_{\mathcal{M}} F$ *y se representa por* $S \models_{\mathcal{M}} F$.
8. *F* **es consecuencia lógica de** *S si* $S \models_{\mathcal{M}} F$ *para toda estructutura* $\mathcal{M}$ *y se representa por* $S \models F$.

El concepto de conjunto satisfacible en $\mathcal{M}$ se formaliza de la siguiente manera.

definition *Satisfacible* :: $('a, 'b)\ form\ set \Rightarrow ('b \Rightarrow 'c\ list \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c\ list \Rightarrow 'c) \Rightarrow bool$ **where**
Satisfacible S I_R I_F $= (\exists\ I_A.\ \forall\, F{\in}S.\ eval\ I_R\ I_F\ I_A\ F)$

En la siguiente definición se formaliza la noción de $S \models_{\mathcal{M},A} F$.

definition *Consecuencia* :: $('b \Rightarrow 'c\ list \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c\ list \Rightarrow 'c) \Rightarrow (nat \Rightarrow 'c) \Rightarrow ('a, 'b)\ form\ list \Rightarrow ('a, 'b)\ form \Rightarrow bool$ (-,-,-,- ⊨ - [50,50] 50) **where**

$$(I_R, I_F, I_A, S \models F) = (\textit{list-all}\ (\textit{eval}\ I_R\ I_F\ I_A)\ S \longrightarrow \textit{eval}\ I_R\ I_F\ I_A\ F)$$

Donde $(\textit{list-all}\ \ P\ xs)$ se verifica si todos los elementos de xs cumplen el predicado P.

Capítulo 9

Teorema de existencia de modelos de primer orden

En esta sección extendemos a la lógica de primer orden la formalización del teorema de existencia de modelos proposicionales que se estudió en el capítulo 6, siguiendo la exposición del libro de Fitting [14].

9.1 Conjuntos consistentes

Inicialmente extendemos a la lógica de primer orden el concepto de *consistencia* estudiado en la sección 6.2.

Las condiciones que deben cumplir algunas de las fórmulas cuantificadas en esta definicion, y en los principales resultados que estudiaremos en las próximas secciones, dependen del concepto *término cerrado*.

Definición 9.1.1 *Un término es* **cerrado** *si no contiene variables.*

Para la formalización de este concepto hacemos uso de la notación de de Bruijn y definimos lo que significa un término cerrado y un conjunto de términos cerrados, en el *nivel n*.

Definición 9.1.2 *Un término es cerrado en el nivel n si no contiene índices de variable mayores o iguales que n.*

Su formalización es:

primrec *cerradot* :: *nat* ⇒ *'a term* ⇒ *bool* **and** *cerradots* :: *nat* ⇒ *'a term list* ⇒ *bool* **where**

```
 cerradot m (Var n) = (n < m)
| cerradot m (Term a ts) = cerradots m ts
| cerradots m [] = True
| cerradots m (t # ts) = (cerradot m t ∧ cerradots m ts)
```

Así, *un término es cerrado si es cerrado en el nivel 0.*

Definición 9.1.3 *Una colección $\mathcal{C}$ de conjuntos de fórmulas es una* **propiedad de consistencia**, *si todos los elementos S de $\mathcal{C}$ verifican las siguientes propiedades.*

1. *Para toda fórmula atómica $P(t_1, \ldots, t_n)$, $P(t_1, \ldots, t_n) \notin S$ o $\neg P(t_1, \ldots, t_n) \notin S$.*
2. *$\bot \notin S$ y $\neg\top \notin S$.*
3. *Si $\neg\neg F \in S$, entonces $S \cup \{F\} \in \mathcal{C}$.*
4. *Si $F \wedge G \in S$, entonces $S \cup \{F, G\} \in \mathcal{C}$.*
5. *Si $\neg(F \vee G) \in S$, entonces $S \cup \{\neg F, \neg G\} \in \mathcal{C}$.*
6. *Si $F \vee G \in S$, entonces $S \cup \{F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$.*
7. *Si $\neg(F \wedge G) \in S$, entonces $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{\neg G\} \in \mathcal{C}$.*
8. *Si $F \to G \in S$, entonces $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$.*
9. *Si $\neg(F \to G) \in S$, entonces $S \cup \{F, \neg G\} \in \mathcal{C}$.*
10. *Si $\forall x F(x) \in S$, entonces $S \cup \{F(t)\} \in \mathcal{C}$, para cualquier término cerrado t.*
11. *Si $\neg\exists x F(x) \in S$, entonces $S \cup \{\neg F(t)\} \in \mathcal{C}$, para cualquier término cerrado t.*
12. *Si $\exists x F(x) \in S$, entonces $S \cup \{F(c)\} \in \mathcal{C}$, para algún símbolo de constante c.*
13. *Si $\neg\forall x F(x) \in S$, entonces $S \cup \{\neg F(c)\} \in \mathcal{C}$, para algún símbolo de constante c.*

Su formalización es:

```
definition consistencia :: ('a, 'b) form set set ⇒ bool where
 consistencia C = (∀ S. S ∈ C ⟶
   (∀ P ts. ¬ (Atom P ts ∈ S ∧ (¬.Atom P ts) ∈ S)) ∧
   FF' ∉ S ∧ (¬.TT') ∉ S ∧
   (∀ F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈ C) ∧
   (∀ F G. (F ∧. G) ∈ S ⟶ S ∪ {F, G} ∈ C) ∧
   (∀ F G. (¬.(F ∨. G)) ∈ S ⟶ S ∪ {¬.F, ¬.G} ∈ C) ∧
   (∀ F G. (F ∨. G) ∈ S ⟶ S ∪ {F} ∈ C ∨ S ∪ {G} ∈ C) ∧
```

$(\forall F\, G.\, (\neg.(F \wedge. G)) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C} \vee S \cup \{\neg.G\} \in \mathcal{C}) \wedge$
$(\forall F\, G.\, (F \rightarrow. G) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C} \vee S \cup \{G\} \in \mathcal{C}) \wedge$
$(\forall F\, G.\, (\neg.(F \rightarrow. G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in \mathcal{C}) \wedge$
$(\forall F\, t.\, cerradot\ 0\ t \longrightarrow (\forall .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}) \wedge$
$(\forall F\, t.\, cerradot\ 0\ t \longrightarrow (\neg.(\exists .F)) \in S \longrightarrow S \cup \{\neg.F[t\backslash 0]\} \in \mathcal{C}) \wedge$
$(\forall F.\, (\exists .F) \in S \longrightarrow (\exists c.\, S \cup \{F[Term\ c\ []\backslash 0]\} \in \mathcal{C})) \wedge$
$(\forall F.\, (\neg.(\forall .F)) \in S \longrightarrow (\exists c.\, S \cup \{\neg.F[Term\ c\ []\backslash 0]\} \in \mathcal{C})))$

9.1.1 Conjuntos consistentes y clausura por subconjuntos

Recordemos que una colección $\mathcal{C}$ de conjuntos es **cerrada por subconjuntos** si para cada $S \in \mathcal{C}$ los subconjuntosde S también son elementos de $\mathcal{C}$ (definicion 6.3.1).

En esta sección extendemos a la lógica de primer orden el teorema 6.3.2 que afirma que toda propiedad de consistencia $\mathcal{C}$ puede extenderse a la propiedad de consistencia $\mathcal{C}^+$, la cual es cerrada por subconjuntos.

Teorema 9.1.4 *Sea $\mathcal{C}$ una colección de conjuntos y $\mathcal{C}^+ = \{S | \exists S' \in \mathcal{C}(S \subseteq S')\}$. Si $\mathcal{C}$ es una propiedad de consistencia, entonces $\mathcal{C}^+$ es una propiedad de consistencia.*

Demostracion:
Supongamos que $\mathcal{C}$ es una propiedad de consistencia. Sea $S \in \mathcal{C}^+$, entonces existe $T \in \mathcal{C}$ tal que $S \subseteq T$. De lo anterior se demuestra que se cumplen las condiciones para que $\mathcal{C}^+$ sea una propiedad de consistencia. Las primeras nueve condiciones se demuestran de manera similar al caso proposicional (teorema 6.3.2).

10. Supongamos que $\forall x F(x) \in S$ entonces, $\forall x F(x) \in T$. Luego, $T \cup \{F(t)\} \in \mathcal{C}$ para cualquier término cerrado t. Además $S \cup \{F(t)\} \subseteq T \cup \{F(t)\}$. Por lo tanto, se tiene que $S \cup \{F(t)\} \in \mathcal{C}^+$.

11. Supongamos $\neg\exists x F(x) \in S$ entonces, $\neg\exists x F(x) \in T$. Luego $T \cup \{\neg F(t)\} \in \mathcal{C}$ para cualquier término cerrado t. Además, $S \cup \{\neg F(t)\} \subseteq T \cup \{\neg F(t)\}$. Por lo tanto, $S \cup \{\neg F(t)\} \in \mathcal{C}^+$.

12. Supongamos que $\exists x F(x) \in S$ entonces, $\exists x F(x) \in T$. Luego $T \cup \{F(c)\} \in \mathcal{C}$ para algún símbolo de constante c. Además, $S \cup \{F(c)\} \subseteq T \cup \{F(c)\}$. Por lo tanto, $S \cup \{F(c)\} \in \mathcal{C}^+$.

13. Supongamos que $\neg\forall x F(x) \in S$ entonces, $\neg\forall x F(x) \in T$. Luego $T \cup \{\neg F(c)\} \in \mathcal{C}$ para algún símbolo de constante c. Además, $S \cup \{\neg F(c)\} \subseteq T \cup \{\neg F(c)\}$. Por lo tanto, $S \cup \{\neg F(c)\} \in \mathcal{C}^+$.

Los siguientes lemas corresponden a la formalización de las anteriores propiedades. Las pruebas de las condiciones (11) y (13) son análogas a las de (10) y (12) respectivamente.

lemma *condiconsis10*:
assumes *consistencia* $\mathcal{C}$ **and** $T \in \mathcal{C}$ **and** $S \subseteq T$
shows $\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\forall .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}^+$
proof (*rule allI*)+
fix $F\ t$
show $cerradot\ 0\ t \longrightarrow (\forall .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in (\mathcal{C}^+)$
proof (*rule impI*)+
assume $cerradot\ 0\ t$ **and** $(\forall .F) \in S$
hence $cerradot\ 0\ t$ **and** $(\forall .F) \in T$ **using** ‹$S \subseteq T$› **by** *auto*
hence $T \cup \{F[t\backslash 0]\} \in \mathcal{C}$ **using** ‹(*consistencia* $\mathcal{C}$)› **and** ‹$T \in \mathcal{C}$›
by (*auto simp add*: *consistencia-def*)
moreover
have $S \cup \{F[t\backslash 0]\} \subseteq T \cup \{F[t\backslash 0]\}$ **using** ‹$S \subseteq T$› **by** *auto*
ultimately
show $S \cup \{F[t\backslash 0]\} \in \mathcal{C}^+$
by (*auto simp add*: *clausura-subconj-def*)
qed
qed

lemma *condiconsis11*:
assumes *consistencia* $\mathcal{C}$ **and** $T \in \mathcal{C}$ **and** $S \subseteq T$
shows $\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists .F)) \in S \longrightarrow S \cup \{\neg.F[t\backslash 0]\} \in \mathcal{C}^+$

lemma *condiconsis12*:
assumes *consistencia* $\mathcal{C}$ **and** $T \in \mathcal{C}$ **and** $S \subseteq T$
shows $\forall F.\ (\exists .F) \in S \longrightarrow (\exists x.\ S \cup \{F[Term\ x\ []\backslash 0]\} \in \mathcal{C}^+)$
proof (*rule allI*)
fix F
show $(\exists .F) \in S \longrightarrow (\exists x.\ S \cup \{F[Term\ x\ []\backslash 0]\} \in \mathcal{C}^+)$
proof (*rule impI*)
assume $(\exists .F) \in S$
hence $(\exists .F) \in T$ **using** ‹$S \subseteq T$› **by** *auto*
hence $\exists x.\ T \cup \{F[Term\ x\ []\backslash 0]\} \in \mathcal{C}$ **using** ‹*consistencia* $\mathcal{C}$› **and** ‹$T \in \mathcal{C}$›
by (*auto simp add*: *consistencia-def*)
then obtain x **where** $T \cup \{F[Term\ x\ []\backslash 0]\} \in \mathcal{C}$ **by** *auto*
moreover
have $S \cup \{F[Term\ x\ []\backslash 0]\} \subseteq T \cup \{F[Term\ x\ []\backslash 0]\}$ **using** ‹$S \subseteq T$› **by** *auto*
ultimately
have $S \cup \{F[Term\ x\ []\backslash 0]\} \in \mathcal{C}^+$
by (*auto simp add*: *clausura-subconj-def*)

```
   thus ∃ x. S ∪ {F[Term x []\0]} ∈ C⁺ by auto
  qed
qed

lemma condiconsis13:
  assumes consistencia C and T ∈ C and S ⊆ T
  shows ∀ F. (¬.(∀ .F)) ∈ S ⟶ (∃ x. S ∪ {¬.F[Term x []\0]} ∈ C⁺)
```

Por último, la formalización del teorema 9.1.4 es la siguiente.

```
theorem cerrado-consistencia:
 assumes hip1: consistencia C
 shows consistencia (C⁺)
proof −
 { fix S
   assume S ∈ C⁺
   hence ∃ T∈C. S ⊆ T by (simp add: clausura-subconj-def)
   then obtain T where hip2: T∈C and hip3: S ⊆ T by auto
   have
    (∀ P ts. ¬ (Atom P ts ∈ S ∧ (¬.Atom P ts) ∈ S)) ∧
    FF' ∉ S ∧ (¬.TT') ∉ S ∧
    (∀ F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈ C⁺) ∧
    (∀ F G. (F ∧. G) ∈ S ⟶ S ∪ {F, G} ∈ C⁺) ∧
    (∀ F G. (¬.(F ∨. G)) ∈ S ⟶ S ∪ {¬.F, ¬.G} ∈ C⁺) ∧
    (∀ F G. (F ∨. G) ∈ S ⟶ S ∪ {F} ∈ C⁺ ∨ S ∪ {G} ∈ C⁺) ∧
    (∀ F G. (¬.(F ∧. G)) ∈ S ⟶ S ∪ {¬.F} ∈ C⁺ ∨ S ∪ {¬.G} ∈ C⁺) ∧
    (∀ F G. (F →. G) ∈ S ⟶ S ∪ {¬.F} ∈ C⁺ ∨ S ∪ {G} ∈ C⁺) ∧
    (∀ F G. (¬.(F →. G)) ∈ S ⟶ S ∪ {F, ¬.G} ∈ C⁺) ∧
    (∀ F t. cerradot 0 t ⟶ (∀ .F) ∈ S ⟶ S ∪ {F[t\0]} ∈ C⁺) ∧
    (∀ F t. cerradot 0 t ⟶ (¬.(∃ .F)) ∈ S ⟶ S ∪ {¬.F[t\0]} ∈ C⁺) ∧
    (∀ F. (∃ .F) ∈ S ⟶ (∃ c. S ∪ {F[Term c []\0]} ∈ C⁺)) ∧
    (∀ F. (¬.(∀ .F)) ∈ S ⟶ (∃ c. S ∪ {¬.F[Term c []\0]} ∈ C⁺))
    using
  condiconsis1[OF hip1 hip2 hip3] condiconsis2[OF hip1 hip2 hip3] condiconsis3[OF hip1 hip2 hip3]
  condiconsis4[OF hip1 hip2 hip3] condiconsis5[OF hip1 hip2 hip3] condiconsis6[OF hip1 hip2 hip3]
  condiconsis7[OF hip1 hip2 hip3] condiconsis8[OF hip1 hip2 hip3] condiconsis9[OF hip1 hip2 hip3]
  condiconsis10[OF hip1 hip2 hip3] condiconsis11[OF hip1 hip2 hip3] condiconsis12[OF hip1 hip2 hip3]
  condiconsis13[OF hip1 hip2 hip3] by blast }
  thus ?thesis by (simp add: consistencia-def)
qed
```

9.2 Propiedad de consistencia y consistencia alternativa

Como se observó en la sección 6.4, la demostración del teorema de existencia de modelos está basada en poder extender una propiedad de consistencia a una propiedad de consistencia que sea cerrada por subconjuntos y de *carácter finito* (definición 6.4.1).

Para el caso de la lógica de predicados no es fácil construir una extensión de carácter finito de una propiedad de consistencia y que cumpla las condiciones (12) y (13) de la definición 9.1.3 de propiedad de consistencia. Por lo tanto, Fitting [14], (página 130) propone una variante a esta extensión, modificando las condiciones (12) y (13).

9.2.1 Propiedad de consistencia alternativa

Definición 9.2.1 *Una colección $\mathcal{C}$ de conjuntos de fórmulas es una* **propiedad de consistencia alternativa**, *si todos los elementos S de $\mathcal{C}$ verifican las siguientes propiedades.*

1. *Para toda fórmula atómica $P(t_1, \ldots, t_n)$, $P(t_1, \ldots, t_n) \notin S$ o $\neg P(t_1, \ldots, t_n) \notin S$.*
2. *$\bot \notin S$ y $\neg \top \notin S$.*
3. *Si $\neg\neg F \in S$, entonces $S \cup \{F\} \in \mathcal{C}$.*
4. *Si $F \wedge G \in S$, entonces $S \cup \{F, G\} \in \mathcal{C}$.*
5. *Si $\neg(F \vee G) \in S$, entonces $S \cup \{\neg F, \neg G\} \in \mathcal{C}$.*
6. *Si $F \vee G \in S$, entonces $S \cup \{F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$.*
7. *Si $\neg(F \wedge G) \in S$, entonces $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{\neg G\} \in \mathcal{C}$.*
8. *Si $F \rightarrow G \in S$, entonces $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$.*
9. *Si $\neg(F \rightarrow G) \in S$, entonces $S \cup \{F, \neg G\} \in \mathcal{C}$.*
10. *Si $\forall x F(x) \in S$, entonces $S \cup \{F(t)\} \in \mathcal{C}$, para cualquier término cerrado t.*
11. *Si $\neg\exists x F(x) \in S$, entonces $S \cup \{\neg F(t)\} \in \mathcal{C}$, para cualquier término cerrado t.*
12. *Si $\exists x F(x) \in S$, entonces $S \cup \{F(c)\} \in \mathcal{C}$, para cualquier símbolo de constante c que no ocurre en ninguna de las fórmulas de S.*
13. *Si $\neg\forall x F(x) \in S$, entonces $S \cup \{\neg F(c)\} \in \mathcal{C}$, para cualquier símbolo de constante c que no ocurre en ninguna de las fórmulas de S.*

Es decir, en las afirmaciones (12) y (13) de la definición 9.1.3 donde aparecen las fórmulas de la forma $\exists x F(x)$, $\neg\forall x F(x)$ se ha cambiado, en la nueva definición, la condición de "**existencia** de un símbolo de constante c" por la de "**cualquier** símbolo de constante c que no ocurra en ninguna de las fórmulas de S". Su formalización es,

definition *alt-consistencia* :: $('a, 'b)$ *form set set* $\Rightarrow$ *bool* **where**
$alt\text{-}consistencia\ \mathcal{C} = (\forall S.\ S \in \mathcal{C} \longrightarrow$
$(\forall P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S)) \wedge$
$FF' \notin S \wedge$
$(\neg.TT') \notin S \wedge$
$(\forall F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (F \wedge. G) \in S \longrightarrow S \cup \{F, G\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (\neg.(F \vee. G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (F \vee. G) \in S \longrightarrow S \cup \{F\} \in \mathcal{C} \vee S \cup \{G\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (\neg.(F \wedge. G)) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C} \vee S \cup \{\neg.G\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (F \rightarrow. G) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C} \vee S \cup \{G\} \in \mathcal{C}) \wedge$
$(\forall F\ G.\ (\neg.(F \rightarrow. G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in \mathcal{C}) \wedge$
$(\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\forall .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}) \wedge$
$(\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists .F)) \in S \longrightarrow S \cup \{\neg.(F[t\backslash 0])\} \in \mathcal{C}) \wedge$
$(\forall F\ c.\ (\forall G{\in}S.\ c \notin simbfun\ G) \longrightarrow (\exists .F) \in S \longrightarrow S \cup \{F[Term\ c\ []\backslash 0]\} \in \mathcal{C}) \wedge$
$(\forall F\ c.\ (\forall G{\in}S.\ c \notin simbfun\ G) \longrightarrow (\neg.(\forall .F)) \in S \longrightarrow$
$S \cup \{\neg.(F[Term\ c\ []\backslash 0])\} \in \mathcal{C}))$

Nuestro propósito en lo que sigue es demostrar que una propiedad de consistencia puede extenderse a una propiedad de consistencia alternativa de carácter finito (y por lo tanto cerrada por subconjuntos). La definición de esta extensión está basada en el concepto de *sustitución de símbolos de función* en términos y fórmulas.

La función sustitución de símbolos de función en términos, *psustt*, está definida de la siguiente manera.

Definición 9.2.2 *Sean A y C conjuntos de símbolos de función. Dada una función $T\colon A \to C$, llamada* **sustitución de parámetros**, *y un término t con símbolos de funcion en A, ($psustt\ T\ t$) es el término con símbolos de función en C, que resulta de sustituir cada símbolo de función f en t por $T(f)$.*

primrec *psustt* :: $('a \Rightarrow 'c) \Rightarrow 'a\ term \Rightarrow 'c\ term$ **and** *psustts* :: $('a \Rightarrow 'c) \Rightarrow 'a\ term\ list \Rightarrow 'c\ term\ list$ **where**
$psustt\ T\ (Var\ i) = Var\ i$
$|\ psustt\ T\ (Term\ f\ ts) = Term\ (T\ f)\ (psustts\ T\ ts)$
$|\ psustts\ T\ [] = []$
$|\ psustts\ T\ (t\ \#\ ts) = psustt\ T\ t\ \#\ psustts\ T\ ts$

La función sustitución de parámetros en fórmulas, *psust*, está definida de la siguiente forma.

Definición 9.2.3 *Sean A y C conjuntos de símbolos de función. Dada una sustitución de parámetros $T\colon A \to C$ y F una fórmula con símbolos de función en A, $(psust\ T\ F)$ es la fórmula con símbolos de función en C, que resulta de sustituir cada símbolo de función f en F por $T(f)$.*

primrec *psust* :: $('a \Rightarrow 'c) \Rightarrow ('a, 'b)\ form \Rightarrow ('c, 'b)\ form$ **where**
$psust\ T\ FF' = FF'$
$|\ psust\ T\ TT' = TT'$
$|\ psust\ T\ (Atom\ P\ ts) = Atom\ P\ (psustts\ T\ ts)$
$|\ psust\ T\ (F \wedge. G) = ((psust\ T\ F) \wedge. (psust\ T\ G))$
$|\ psust\ T\ \ (F \vee. G) = ((psust\ T\ F) \vee. (psust\ T\ G))$
$|\ psust\ T\ (F \rightarrow. G) = ((psust\ T\ F) \rightarrow. (psust\ T\ G))$
$|\ psust\ T\ (\neg.F) = (\neg.(psust\ T\ F))$
$|\ psust\ T\ (\forall .F) = \ (\forall .(psust\ T\ F))$
$|\ psust\ T\ (\exists .F) = (\exists .(psust\ T\ F))$

La función de sustitución de parámetros en fórmulas $(psust\ T)$, permite extender cualquier propiedad de consistencia $\mathcal{C}$ (cerrada por subconjuntos) a una propiedad de consistencia alternativa $\mathcal{C}^0$ (cerrada por subconjuntos). Para la demostración, basta con considerar la colección $\mathcal{C}^0$ de conjuntos de fórmulas definida por:

Definición 9.2.4 *Sea $\mathcal{C}$ una colección de conjuntos de fórmulas. $S \in \mathcal{C}^0$ si y sólo si existe una sustitución de parámetros T tal que, la imagen directa de S con respecto a $(psust\ T)$ pertenece a $\mathcal{C}$.*

Su formalización es:

definition *ext-consistencia-alt* :: $('a, 'b)\ form\ set\ set \Rightarrow ('a, 'b)\ form\ set\ set$ $(\text{-}^0\ [1000]\ 1000)$ **where**
$\mathcal{C}^0\ = \{S.\ \exists T.\ psust\ T\ `\ S \in \mathcal{C}\}$

Teorema 9.2.5 *Sea $\mathcal{C}$ una colección de conjuntos de fórmulas. Entonces,*

(a) $\mathcal{C} \subseteq \mathcal{C}^0$.

(b) Si $\mathcal{C}$ es cerrada por subconjuntos, entonces $\mathcal{C}^0$ es cerrada por subconjuntos.

(c) Si $\mathcal{C}$ es una propiedad de consistencia, entonces $\mathcal{C}^0$ es una propiedad de consistencia alternativa.

Demostración:

(a) Sea $S \in \mathcal{C}$. Consideremos T la sustitución de parámetros identidad, es decir, $T(f) = f$ para todo símbolo de función $f \in A$, en donde A el conjunto de símbolos de

función que ocurren en las fórmulas de los elementos de $\mathcal{C}$. Entonces, $(psust\ T)[S] = S \in \mathcal{C}$ por definición de *psust*. Por lo tanto, $S \in \mathcal{C}^0$.

(b) Supongamos que $\mathcal{C}$ es cerrada por subconjuntos. Sea $S \in \mathcal{C}^0$ y $S' \subseteq S$. Puesto que $S \in \mathcal{C}^0$, tenemos que existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$. Además, $(psust\ T)[S'] \subseteq (psust\ T)[S] \in \mathcal{C}$. Luego, $(psust\ T)[S'] \in \mathcal{C}$. Por tanto, $S' \in \mathcal{C}^0$.

(c) Supongamos que $\mathcal{C}$ es una propiedad de consistencia. Mostremos que $\mathcal{C}^0$ es una propiedad de consistencia alternativa. Sea $S \in \mathcal{C}^0$, entonces existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, de este hecho mostramos que se cumplen las condiciones para que $\mathcal{C}^0$ sea una propiedad de consistencia alternativa.

1. Sea P una fórmula atómica, hay que demostrar que $P \notin S$ o $\neg P \notin S$. Puesto que $\mathcal{C}$ es una propiedad de consistencia y $(psust\ T)[S] \in \mathcal{C}$ entonces, para toda fórmula atómica Q se tiene que $Q \notin (psust\ T)[S]$ o $\neg Q \notin (psust\ T)[S]$. En particular para $Q = (psust\ T\ P$ obtenemos, $(psust\ T\ P) \notin (psust\ T)[S]$ o $\neg(psust\ T\ P) \notin (psust\ T)[S]$, es decir, $(psust\ T\ P) \notin (psust\ T)[S]$ o $(psust\ T\ \neg P) \notin (psust\ T)[S]$. Por lo tanto, por definición de imagen directa, $P \notin S$ o $\neg P \notin S$.

2. La demostración de $\bot \notin S$ es por contradicción. Supongamos que $\bot \in S$, entonces $(psust\ T\ \bot) \in (psust\ T)[S]$ por definición de imagen directa. Por lo tanto, puesto que $psust\ T\ \bot = \bot$, se tiene que $\bot \in (psust\ T)[S]$. Además, puesto que $(psustT)[S] \in \mathcal{C}$ y $\mathcal{C}$ es propiedad de consistencia, tenemos que $\bot \notin (psust\ T)[S]$. De esta forma obtenemos una contradicción.

De la misma forma $\neg\top \notin S$, de lo contrario $(psust\ T\ \neg\top) \in (psust\ T)[S]$ por definición de imagen directa y por lo tanto, puesto que $psust\ T\ \neg\top = \neg\top$, se tendría que $\neg\top \in (psust\ T)[S]$, lo cual es imposible ya que, por las hipótesis $(psust\ T)[S] \in \mathcal{C}$ y $\mathcal{C}$ es propiedad de consistencia, tenemos que $\neg\top \notin (psust\ T)[S]$.

3. Supongamos que $\neg\neg F \in S$. Hay que demostrar que $S \cup \{F\} \in \mathcal{C}^0$. Es decir, hay que probar que existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{F\}] \in \mathcal{C}$, es decir, por propiedades de la imagen directa, hay que demostrar que existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$.

Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, se tiene que si $\neg\neg G \in (psust\ T)[S]$ entonces $(psust\ T)[S] \cup \{G\} \in \mathcal{C}$, para toda fórmula G.

En particular, para $G = (psust\ T\ F)$, si $\neg\neg(psust\ T\ F) \in (psust\ T)[S]$ entonces $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$.

Además, por la hipótesis $\neg\neg F \in S$ y la definición de imagen directa, se tiene que $(psust\ T\ \neg\neg F) \in (psust\ T)[S]$, es decir, $\neg\neg(psust\ T\ F) \in (psust\ T)[S]$.

Por lo tanto, $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$.

De esta forma, existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{psust\ T\ F\} \in$

$\mathcal{C}$.

4. Supongamos que $F \wedge G \in S$. Hay que demostrar que $S \cup \{F, G\} \in \mathcal{C}^0$. Es decir, hay que probar que existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{F, G\}] \in \mathcal{C}$, es decir, por propiedades de la imagen directa, hay que demostrar que existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ G\} \in \mathcal{C}$. Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, se tiene que si $H \wedge I \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{H, I\} \in \mathcal{C}$, para cualesquier fórmulas H, I. En particular, para $H = psust\ T\ F$ y $I = psust\ T\ G$, si $(psust\ T\ F\ ) \wedge (psust\ T\ G) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ G\} \in \mathcal{C}$. Además, por la hipótesis $F \wedge G \in S$ y la definición de imagen directa, se tiene que, $(psust\ T\ F \wedge G) \in (psust\ T)[S]$, es decir, $(psust\ T\ F) \wedge (psust\ T\ G) \in (psust\ T)[S]$. Por lo tanto, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ G\} \in \mathcal{C}$. De esta forma, existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ G\} \in \mathcal{C}$.

Las demostraciones de las propiedades (5) y (9) son similares a la anterior.

6. Supongamos que $F \vee G \in S$. Hay que demostrar que $S \cup \{F\} \in \mathcal{C}^0$ o $S \cup \{G\} \in \mathcal{C}^0$. Es decir, hay que probar que, existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{F\}] \in \mathcal{C}$ o existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{G\}] \in \mathcal{C}$. Es decir, por propiedades de la imagen directa, hay que demostrar que, existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$ o existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{psust\ T\ G\} \in \mathcal{C}$. Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, se tiene que, si $H \vee I \in (psust\ T)[S]$ entonces, $S \cup \{H\} \in \mathcal{C}$ o $S \cup \{I\} \in \mathcal{C}$, para cualesquier fórmulas H, I. En particular, para $H = psust\ T\ F$ y $I = psust\ T\ G$, si $(psust\ f\ F) \vee (psust\ f\ G) \in (psust\ f)[S]$ entonces, $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$ o $(psust\ T)[S] \cup \{psust\ T\ G\} \in \mathcal{C}$. Además, por la hipótesis $F \vee G \in S$ y la definición de imagen directa, se tiene que $(psust\ T\ F \vee G) \in (psust\ T)[S]$, es decir, $(psust\ T\ F) \vee (psust\ T\ G) \in (psust\ T)[S]$. Por lo tanto, $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$ o $(psust\ T)[S] \cup \{psust\ T\ G\} \in \mathcal{C}$. De esta forma, existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{psust\ T\ F\} \in \mathcal{C}$ o existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{psust\ T\ G\} \in \mathcal{C}$.

Las demostraciones de las propiedades (7) y (8) son análogas a la anterior.

9. Supongamos que $\neg(F \rightarrow G) \in S$. Hay que demostrar que $S \cup \{F, \neg G\} \in \mathcal{C}^0$. Es decir, hay que probar que existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{F, \neg G\}] \in \mathcal{C}$, es decir, por propiedades de la imagen directa, hay que demostrar que existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ \neg G\} \in \mathcal{C}$.

Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$ se tiene que, si $\neg(H \rightarrow I) \in (psust\ T)[S]$, entonces

$(psust\ T)[S] \cup \{H, \neg I\} \in \mathcal{C}$, para cualesquier fórmulas H, I.

En particular, para $H = psust\ T\ F$ y $I = psust\ T\ G$, si $\neg(psust\ T\ F \rightarrow psust\ T\ G) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{psust\ T\ F, \neg(psust\ T\ G)\} \in \mathcal{C}$.

Además, por la hipótesis $\neg(F \rightarrow G) \in S$ y la definición de imagen directa se tiene que $psust\ T\ \neg(F \rightarrow G) \in (psust\ T)[S]$, es decir, $\neg(psust\ T\ F \rightarrow psust\ T\ G) \in (psust\ T)[S]$. Por lo tanto, $(psust\ T)[S] \cup \{psust\ T\ F, \neg(psust\ T\ G)\} \in \mathcal{C}$, es decir, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ \neg G\} \in \mathcal{C}$.

De esta forma, existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F, psust\ T\ \neg G\}$ $\mathcal{C}$.

10. Sea F una fórmula. Supongamos que $\forall x F(x) \in S$. Hay que demostrar que $S \cup \{F(t)\} \in \mathcal{C}^0$ para cualquier término cerrado t. Es decir, hay que probar que existe T sustitución de parámetros tal que $(psust\ T)[S \cup \{F(t)\}] \in \mathcal{C}$ para cualquier término cerrado t. Es decir, por propiedades de la imagen directa, hay que demostrar que existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{psust\ T\ F(t)\} \in \mathcal{C}$ para cualquier término cerrado t. Es decir, por propiedades de la función de sustitución de parametros, hay que demostrar que existe T sustitución de parámetros tal que, $(psust\ T)[S] \cup \{(psust\ T\ F)(psustt\ T\ t)\} \in \mathcal{C}$ para cualquier término cerrado t. Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, se tiene que si $\forall x G(x) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{G(u)\} \in \mathcal{C}$, para cualquier fórmula G y cualquier término cerrado u. En particular, para $G = psust\ T\ F$ y $u = psustt\ T\ t$, si $\forall x(psust\ T\ F)(x) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{(psust\ T\ F)(psustt\ T\ t)\} \in \mathcal{C}$. Además, por la hipótesis $\forall x F(x) \in S$ y la definición de imagen directa, se tiene que, $(psust\ T\ \forall x F(x)) \in (psust\ T)[S]$, es decir, $\forall x(psust\ T\ F)(x) \in (psust\ T)[S]$. Por lo tanto, $(psust\ T)[S] \cup \{(psust\ T\ F)(psustt\ T\ t)\} \in \mathcal{C}$. De esta forma, existe T sustitución de parámetros tal que $(psust\ T)[S] \cup \{(psust\ T\ F)(psustt\ T\ t)\} \in \mathcal{C}$ para cualquier término cerrado t.

La demostración de la propiedad (11) es análoga a la anterior.

12. Sea F una fórmula y c un símbolo de constante. Supongamos que $\exists x F(x) \in S$ y c no ocurre en ninguna de las fórmulas de S, hay que demostrar que $S \cup \{F(c)\} \in \mathcal{C}^0$, es decir, hay que probar que existe T' sustitución de parámetros tal que, $(psust\ T')[S \cup \{F(c)\}] \in \mathcal{C}$. Es decir, por propiedades de la imagen directa, hay que demostrar que existe T' sustitución de parámetros tal que, $(psust\ T')[S] \cup \{psust\ T'\ F(c)\} \in \mathcal{C}$. Es decir, por propiedades de la función de sustitución de parametros, hay que demostrar que existe T' sustitución de parámetros tal que, $(psust\ T')[S] \cup \{(psust\ T'\ F)(psustt\ T'\ c)\} \in \mathcal{C}$, es decir, puesto que $psustt\ T'\ c = T'(c)$ ya que c es un símbolo de constante, hay que demostrar que existe T' sustitución de parámetros tal que, $(psust\ T')[S] \cup \{(psust\ T\ F)(T(c))\} \in \mathcal{C}$. Puesto que por hipótesis, $\mathcal{C}$ es propiedad de consistencia y existe T sustitución de parámetros tal que $(psust\ T)[S] \in \mathcal{C}$, se tiene que existe T

sustitución de parámetros tal que para cualquier fórmula H, si $\exists x H(x) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{H(d)\} \in \mathcal{C}$, para algún símbolo de constante d.

En particular, para $H = psust\ T\ F$, si $\exists x(psust\ T\ F)(x) \in (psust\ T)[S]$ entonces, $(psust\ T)[S] \cup \{(psust\ T\ F)(d)\} \in \mathcal{C}$, para algún símbolo de constante d. Además, por las hipótesis $\exists x F(x) \in S$ y la definición de imagen directa, se tiene que $(psust\ T\ \exists x F(x)) \in (psust\ T)[S]$, es decir, $\exists x(psust\ T\ F)(x) \in (psust\ T)[S]$. Por lo tanto, $(psust\ T)[S] \cup \{(psust\ T\ F)(d)\} \in \mathcal{C}$, para algún símbolo de constante d. Por otro lado, por las hipótesis $\exists x F(x) \in S$ y c no ocurre en ninguna fórmula de S, se tiene que c no ocurre en $\exists x F(x)$ y por lo tanto c no ocurre en F. Ahora, consideremos la sustitución de parámetros $T(c : d)$ definida por, $T(c : d)(x) = T(x)$ si $x \neq c$ y $T(c : d)(c) = d$. Entonces, por la definición de *psust*, puesto que c no ocurre en ninguna de las fórmulas de $S \cup \{F\}$ se tiene que, $(psust\ T(c : d))[S] = (psust\ T)[S]$ y $psust\ T(c : d)\ F = psust\ T\ F$. Luego, $(psust\ T(c : d))[S] \cup \{(psust\ T(c : d)\ F)(T(c : d)(c))\} = (psust\ T)[S] \cup \{(psust\ T\ F)(d)\} \in \mathcal{C}$. De este modo, existe $T' = T(c : d))$ sustitución de parámetros tal que, $(psust\ T')[S] \cup \{(psust\ T'\ F)(g(c))\} \in \mathcal{C}$.

La demostración de la propiedad (13) es análoga a la anterior.

La formalización de la prueba de la parte (a) es la siguiente:

```
lemma ext-consistencia-alt-subset: C ⊆ C⁰
proof(rule subsetI)
 fix S
 assume S ∈ C
 let ?T = λx. x
 have psust ?T ' S = S by simp
 hence (psust ?T ' S) ∈ C using ⟨S ∈ C⟩ by simp
 hence ∃ T. psust T ' S ∈ C by (rule-tac x= ?T in exI)
 thus S ∈ C⁰ by (unfold ext-consistencia-alt-def) simp
qed
```

La formalización de la prueba de la parte (b) es la siguiente.

```
lemma ext-consistencia-alt-cerrado:
 assumes subconj-cerrada C
 shows subconj-cerrada (C⁰)
proof −
 { fix S S'
  assume S ∈ C⁰
   and S' ⊆ S
  obtain T where psust T ' S ∈ C using ⟨S ∈ C⁰⟩
   by (auto simp: ext-consistencia-alt-def)
  moreover
  have psust T ' S' ⊆ psust T ' S using ⟨S' ⊆ S⟩ by auto
```

ultimately
have *psust T ' S'* $\in \mathcal{C}$ **using** *assms* **by** (*unfold subconj-cerrada-def*) *simp*
hence $S' \in \mathcal{C}^0$ **by** (*auto simp*: *ext-consistencia-alt-def*)}
thus *?thesis* **by** (*unfold subconj-cerrada-def*) *simp*
qed

Los siguientes lemas corresponden a la formalización de la parte (c).

lemma *condiconalt1*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall P\ t.\ \neg((Atom\ P\ t) \in S \wedge (\neg.(Atom\ P\ t)) \in S))$

lemma *condiconalt2*:
assumes *consistencia* $\mathcal{C}$ **and** *psust T ' S* $\in \mathcal{C}$
shows $FF' \notin S \wedge (\neg.TT') \notin S$

lemma *condiconalt3*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $\forall F.\ (\neg.\neg.F) \in S \longrightarrow\ S \cup \{F\} \in \mathcal{C}^0$

lemma *condiconalt4*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $\forall F\ G.(F \wedge.\ G) \in S \longrightarrow S \cup \{F, G\} \in \mathcal{C}^0$

lemma *condiconalt5*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall F\ G.\ (\neg.\ (F \vee.\ G)) \in S \longrightarrow S \cup \{\neg.\ F, \neg.\ G\} \in \mathcal{C}^0)$

lemma *condiconalt6*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall F\ G.\ (F \vee.\ G) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^0 \vee S \cup \{G\} \in \mathcal{C}^0)$

lemma *condiconalt7*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall F\ G.\ (\neg.\ (F \wedge.\ G)) \in S \longrightarrow S \cup \{\neg.\ F\} \in \mathcal{C}^0 \vee S \cup \{\neg.\ G\} \in \mathcal{C}^0)$

lemma *condiconalt8*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall F\ G.\ ((F \rightarrow.\ G) \in S) \longrightarrow S \cup \{\neg.\ F\} \in \mathcal{C}^0 \vee S \cup \{G\} \in \mathcal{C}^0)$

lemma *condiconalt9*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$
shows $(\forall F\ G.\ (\neg.\ (F \rightarrow.\ G)) \in S \longrightarrow S \cup \{F, \neg.\ G\} \in$ *ext-consistencia-alt* $\mathcal{C})$

lemma *condiconalt10*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust T ' S* $\in \mathcal{C}$

shows $(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\forall\, .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}^0)$

lemma *condiconalt11*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust* T ' $S \in \mathcal{C}$
shows $(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.\ (\exists\, .F)) \in S \longrightarrow S \cup \{\neg.\ (F[t\backslash 0])\} \in \mathcal{C}^0)$

lemma *condiconalt12*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust* T ' $S \in \mathcal{C}$
shows $(\forall\, F\ c.\ (\forall\ G{\in}S.\ c \notin simbfun\ G) \longrightarrow$
$(\exists\, .F) \in S \longrightarrow S \cup \{F[Term\ c\ []\backslash 0]\} \in \mathcal{C}^0)$**lemma** *condiconalt13*:
assumes *hip1*: *consistencia* $\mathcal{C}$ **and** *hip2*: *psust* T ' $S \in \mathcal{C}$
shows $(\forall\, F\ c.\ (\forall\ G{\in}S.\ c \notin simbfun\ G) \longrightarrow$
$(\neg.\ (\forall\, .F)) \in S \longrightarrow S \cup \{\neg.\ (F[Term\ c\ []\backslash 0])\} \in \mathcal{C}^0)$

Por último, la formalización de la prueba de la parte (c) es,

theorem *alt-consistencia*:
assumes *hip1*: *consistencia* $\mathcal{C}$
shows *alt-consistencia* $(\mathcal{C}^0)$
proof −
{ **fix** S
assume $S \in \mathcal{C}^0$
hence $\exists\ T.\ psust\ T$ ' $S \in \mathcal{C}$ **by**(*simp add*: *ext-consistencia-alt-def*)
then obtain T **where** *hip2*: *psust* T ' $S \in \mathcal{C}$ **by** *auto*
have
$(\forall\, P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S)) \wedge$
$FF' \notin S \wedge$
$(\neg.TT') \notin S \wedge$
$(\forall\, F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in (\mathcal{C}^0)) \wedge$
$(\forall\, F\ G.\ (F \wedge.\ G) \in S \longrightarrow S \cup \{F, G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ G.\ (\neg.(F \vee.\ G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ G.\ (F \vee.\ G) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^0 \vee S \cup \{G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ G.\ (\neg.(F \wedge.\ G)) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C}^0 \vee S \cup \{\neg.G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ G.\ (F \rightarrow.\ G) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C}^0 \vee S \cup \{G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ G.\ (\neg.(F \rightarrow.\ G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\forall\, .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists\, .F)) \in S \longrightarrow S \cup \{\neg.(F[t\backslash 0])\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ c.\ (\forall\, G{\in}S.\ c \notin simbfun\ G) \longrightarrow (\exists\, .F) \in S \longrightarrow S \cup \{F[Term\ c\ []\backslash 0]\} \in \mathcal{C}^0) \wedge$
$(\forall\, F\ c.\ (\forall\, G{\in}S.\ c \notin simbfun\ G) \longrightarrow (\neg.(\forall\, .F)) \in S \longrightarrow$
$S \cup \{\neg.(F[Term\ c\ []\backslash 0])\} \in \mathcal{C}^0)$
using
condiconalt1[*OF hip1 hip2*] *condiconalt2*[*OF hip1 hip2*] *condiconalt3*[*OF hip1 hip2*]
condiconalt4[*OF hip1 hip2*] *condiconalt5*[*OF hip1 hip2*] *condiconalt6*[*OF hip1 hip2*]
condiconalt7[*OF hip1 hip2*] *condiconalt8*[*OF hip1 hip2*] *condiconalt9*[*OF hip1 hip2*]
condiconalt10[*OF hip1 hip2*] *condiconalt11*[*OF hip1 hip2*] *condiconalt12*[*OF hip1 hip2*]

condiconalt13[*OF hip1 hip2*] **by** *auto* }
thus *?thesis* **by** (*simp add*: *alt-consistencia-def*)
qed

9.3 Extensión de una propiedad de consistencia alternativa a una de carácter finito

Recordemos que en la definición 6.4.1 se estableció que, una colección $\mathcal{C}$ de conjuntos es de *carácter finito* si para cada conjunto S se tiene que, S pertenece a $\mathcal{C}$ si y solamente si cada subconjunto *finito* de S pertenece a $\mathcal{C}$. También se demostró que, $\mathcal{C}^-$ es de carácter finito y si $\mathcal{C}$ es una propiedad de consistencia proposicional que es cerrada por subconjuntos entonces $\mathcal{C}^-$ es una propiedad de consistencia que extiende a $\mathcal{C}$ (teorema 6.5.1). En esta sección se demuestra que si $\mathcal{C}$ es una propiedad de consistencia alternativa que es cerrada por subconjuntos entonces $\mathcal{C}^-$ es una propiedad de consistencia alternativa (de carácter finito que extiende a $\mathcal{C}$).

Teorema 9.3.1
Sea $\mathcal{C}$ una colección de conjuntos y $\mathcal{C}^- = \{S|\forall S' \subseteq S(S'\ finito \longrightarrow S' \in \mathcal{C})\}$. Si $\mathcal{C}$ es una propiedad de consistencia alternativa que es cerrada por subconjuntos entonces, $\mathcal{C}^-$ es una propiedad de consistencia alternativa.

Demostración:

Supongamos que $\mathcal{C}$ es una propiedad de consistencia alternativa que es cerrada por subconjuntos. Sea $S \in \mathcal{C}^-$. Entonces, por la defínción de $\mathcal{C}^-$, se tiene que cada subconjunto finito de S pertenece a $\mathcal{C}$. De este hecho mostramos que se cumplen las condiciones para que $\mathcal{C}^-$ sea una propiedad de consistencia alternativa. Las primeras nueve condiciones se demuestran de manera similar al caso proposicional (6.5.1).

10. Sea F una fórmula. Supongamos que $\forall xF(x) \in S$. Probamos que $S \cup \{F(t)\} \in \mathcal{C}^-$, para cualquier término cerrado t, usando la definición de $\mathcal{C}^-$: consideremos t un término cerrado, $S' \subseteq S \cup \{F(t)\}$ y S' finito, y mostremos que $S' \in \mathcal{C}$.

Puesto que $\forall xF(x) \in S$ y $S' \subseteq S \cup \{F(t)\}$ tenemos que, $S' - \{F(t)\} \cup \{\forall xF(x)\} \subseteq S$; también tenemos que, $S' - \{F(t)\} \cup \{\forall xF(x)\}$ es finito ya que S' es finito.

Así, por hipótesis, $S' - \{F(t)\} \cup \{\forall xF(x)\} \in \mathcal{C}$, y como $\forall xF(x) \in S' - \{F(t)\} \cup \{\forall xF(x)\}$ donde t es un término cerrado, entonces

$$(S' - \{F(t)\} \cup \{\forall xF(x)\}) \cup \{F(t)\} \in \mathcal{C}$$

por ser $\mathcal{C}$ una propiedad de consistencia alternativa.

De esto último y como $S' \subseteq (S' - \{F(t)\} \cup \{\forall x F(x)\}) \cup \{F(t)\}$ se tiene que $S' \in \mathcal{C}$ ya que por hipótesis $\mathcal{C}$ es cerrada por subconjuntos.

12. Sea F una fórmula y c un símbolo de constante. Supongamos que $\exists x F(x) \in S$ y c no ocurre en ninguna de las fórmulas de S. Demostramos que $S \cup \{F(c)\} \in \mathcal{C}^-$ usando la definición de $\mathcal{C}^-$: consideremos $S' \subseteq S \cup \{F(c)\}$ y S' finito, y mostremos que $S' \in \mathcal{C}$.

Puesto que $\exists x F(x) \in S$ y $S' \subseteq S \cup \{F(c)\}$ tenemos que, $S' - \{F(c)\} \cup \{\exists x F(x)\} \subseteq S$; también tenemos que, $S' - \{F(c)\} \cup \{\exists x F(x)\}$ es finito ya que S' es finito. Así, por hipótesis, $S' - \{F(c)\} \cup \{\exists x F(x)\} \in \mathcal{C}$. Por otro lado, tenemos que c no ocurre en ninguna de las fórmulas de $S' - \{F(c)\} \cup \{\exists x F(x)\}$, ya que $S' - \{F(c)\} \cup \{\exists x F(x)\} \subseteq S$ y c no ocurre en ninguna de las fórmulas de S; también tenemos que $\exists x F(x) \in S' - \{F(c)\} \cup \{\exists x F(x)\}$. De lo anterior, $(S' - \{F(c)\} \cup \{\exists x F(x)\}) \cup \{F(c)\} \in \mathcal{C}$ por ser $\mathcal{C}$ una propiedad de consistencia alternativa.

De esto último y como $S' \subseteq (S' - \{F(c)\} \cup \{\exists x F(x)\}) \cup \{F(c)\}$ se tiene que $S' \in \mathcal{C}$ ya que por hipótesis $\mathcal{C}$ es cerrada por subconjuntos.

Las pruebas de las condiciones (11) y (13) son análogas a las de (10) y (12) respectivamente.

Los siguientes lemas corresponden a la formalización de las anteriores propiedades.

lemma *condicaracter1*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
shows $(\forall P\ t.\ \neg((Atom\ P\ t) \in S \wedge (\neg.(Atom\ P\ t)) \in S))$

lemma *condicaracter2*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
shows $FF' \notin S \wedge (\neg.TT') \notin S$

lemma *condicaracter3*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
shows $\forall F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^-$

lemma *condicaracter4*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$

and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.(F \wedge.\ G) \in S \longrightarrow S \cup \{F, G\}\ \in \mathcal{C}^{-}$

lemma *condicaracter5*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.\ (\neg.(F \vee.\ G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in \mathcal{C}^{-}$

lemma *condicaracter6*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.(F \vee.\ G) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^{-} \vee S \cup \{G\} \in\ \mathcal{C}^{-}$

lemma *condicaracter7*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.\ (\neg.(F \wedge.\ G)) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C}^{-} \vee S \cup \{\neg.G\} \in\ \mathcal{C}^{-}$

lemma *condicaracter8*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.\ (F \rightarrow.\ G) \in S \longrightarrow S \cup \{\neg.F\}\ \in \mathcal{C}^{-}\ \vee S \cup \{G\}\ \in \mathcal{C}^{-}$

lemma *condicaracter9*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ G.\ (\neg.(F \rightarrow.\ G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in \mathcal{C}^{-}$

lemma *condicaracter10*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$ **and**
hip: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ t.\ \mathit{cerradot}\ 0\ t \longrightarrow (\forall .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in \mathcal{C}^{-}$

lemma *condicaracter11*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ \mathit{finite}\ S' \longrightarrow S' \in \mathcal{C}$
shows $\forall\, F\ t.\ \mathit{cerradot}\ 0\ t \longrightarrow\ (\neg.(\exists\, .F)) \in S \longrightarrow S \cup \{\neg.F[t\backslash 0]\} \in \mathcal{C}^{-}$

lemma *condicaracter12*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $(\forall\, F\ c.\ (\forall\ G \in S.\ c \notin simbfun\ G) \longrightarrow$
$(\exists\, .F) \in S \longrightarrow S \cup \{F[Term\ c\ [] \backslash 0]\} \in \mathcal{C}^{-})$

lemma *condicaracter13*:
assumes *alt-consistencia* $\mathcal{C}$
and *subconj-cerrada* $\mathcal{C}$
and *hip*: $\forall\, S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
shows $(\forall\, F\ c.\ (\forall\ G \in S.\ c \notin simbfun\ G) \longrightarrow$
$(\neg.(\forall\, .F)) \in S \longrightarrow S \cup \{\neg.(F[Term\ c\ [] \backslash 0])\} \in\ \mathcal{C}^{-})$

Por último, la formalización de la prueba del teorema anterior es la siguiente.

theorem *alt-consistencia-cfinita*:
assumes *hip1*: *alt-consistencia* $\mathcal{C}$
and *hip2*: *subconj-cerrada* $\mathcal{C}$
shows *alt-consistencia* $(\mathcal{C}^{-})$
proof −
{ **fix** S
assume $S \in \mathcal{C}^{-}$
hence *hip3*: $\forall\, S' \subseteq S.\ finite\ S' \longrightarrow S' \in \mathcal{C}$
by(*simp add*: *clausura-cfinito-def*)
have $(\forall\, P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S)) \wedge$
$FF' \notin S \wedge$
$(\neg.TT') \notin S \wedge$
$(\forall\, F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (F \wedge.\ G) \in S \longrightarrow S \cup \{F, G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \vee.\ G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (F \vee.\ G) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^{-} \vee S \cup \{G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \wedge.\ G)) \in S \longrightarrow S \cup \{\neg.F\} \in clausura\text{-}cfinito\ \ \mathcal{C} \vee$
$S \cup \{\neg.G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (F \rightarrow.\ G) \in S \longrightarrow S \cup \{\neg.F\} \in \mathcal{C}^{-} \vee S \cup \{G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \rightarrow.\ G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\forall\, .F) \in S \longrightarrow S \cup \{F[t \backslash 0]\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists\, .F)) \in S \longrightarrow S \cup \{\neg.(F[t \backslash 0])\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ c.\ (\forall\, G \in S.\ c \notin simbfun\ G) \longrightarrow (\exists\, .F) \in S \longrightarrow S \cup \{F[Term\ c\ [] \backslash 0]\} \in \mathcal{C}^{-}) \wedge$
$(\forall\, F\ c.\ (\forall\, G \in S.\ c \notin simbfun\ G) \longrightarrow (\neg.(\forall\, .F)) \in S \longrightarrow$
$S \cup \{\neg.(F[Term\ c\ [] \backslash 0])\} \in\ \mathcal{C}^{-})$
using
condicaracter1[*OF hip1 hip2 hip3*] *condicaracter2*[*OF hip1 hip2 hip3*]
condicaracter3[*OF hip1 hip2 hip3*] *condicaracter4*[*OF hip1 hip2 hip3*]
condicaracter5[*OF hip1 hip2 hip3*] *condicaracter6*[*OF hip1 hip2 hip3*]

condicaracter7[*OF hip1 hip2 hip3*] *condicaracter8*[*OF hip1 hip2 hip3*]
condicaracter9[*OF hip1 hip2 hip3*] *condicaracter10*[*OF hip1 hip2 hip3*]
condicaracter11[*OF hip1 hip2 hip3*] *condicaracter12*[*OF hip1 hip2 hip3*]
condicaracter13[*OF hip1 hip2 hip3*] **by** *auto* }
thus *?thesis* **by** (*simp add*: *alt-consistencia-def*)
qed

9.4 Extensión a conjuntos consistentes maximales

En forma análoga a lo estudiado en la sección 6.6, en esta sección precisamos en Isabelle, cómo extender un conjunto consistente S de fórmulas de primer orden a un conjunto consistente maximal de acuerdo a lo expuesto en [14] (páginas 131-132). La construcción, a partir de S, del conjunto consistente maximal está basada en la siguiente sucesión creciente (por inclusión) de conjuntos consistentes.

Definición 9.4.1 *Sea S un conjunto de fórmulas, $\mathcal{C}$ una colección de conjuntos de fórmulas y f una sucesión de fórmulas. Definimos la siguiente sucesión de conjuntos, $\{S^n_{S,\mathcal{C},f}\}_n$.*

1. $S^0_{S,\mathcal{C},f} = S$.
2. *El valor de $S^{n+1}_{S,\mathcal{C},f}$ es,*
   - $S^n_{S,\mathcal{C},f}$, *si* $S^n_{S,\mathcal{C},f} \cup \{f_n\} \notin \mathcal{C}$.
   - $S^n_{S,\mathcal{C},f} \cup \{f_n\} \cup \{F(k)\}$, *si* $S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C}$ *y* $f_n = \exists x F(x)$ *para alguna fórmula* F.
   - $S^n_{S,\mathcal{C},f} \cup \{f_n\} \cup \{\neg F(k)\}$, *si* $S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C}$ *y* $f_n = \neg\forall x F(x)$ *para alguna fórmula* F.
   - $S^n_{S,\mathcal{C},f} \cup \{f_n\}$, *en otro caso.*

Donde k es algún símbolo de constante que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{f_n\}$.

Para la formalización en Isabelle de la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ definimos las siguientes funciones, *elim-Neg*, *elim-Forall* y *elim-Exits* que eliminan la negación y los cuantificadores de las respectivas fórmulas. También hacemos uso del operador ϵ de Hilbert $\epsilon x.P(x)$, el cual denota *algún* x tal que $P(x)$ es verdadero (asumiendo que existe un tal x); en Isabelle se utiliza la palabra *SOME* en vez de la letra Griega ϵ.

primrec *elim-Neg* :: $('a, 'b)$ *form* $\Rightarrow$ $('a, 'b)$ *form* **where**
elim-Neg $(\neg.F) = F$

primrec *elim-Forall* :: $('a, 'b)$ *form* $\Rightarrow$ $('a, 'b)$ *form* **where**
elim-Forall $(\forall\,.\,F) = F$

primrec *elim-Exists* :: $('a, 'b)$ *form* $\Rightarrow$ $('a, 'b)$ *form* **where**
elim-Exists $(\exists\,.\,F) = F$

La formalización en Isabelle de la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ es,

primrec
suc :: $('a, 'b)$ *form set* $\Rightarrow$ $('a, 'b)$ *form set set* $\Rightarrow$ $(nat \Rightarrow ('a, 'b)\ form) \Rightarrow nat \Rightarrow ('a, 'b)$ *form set*
where
suc S $\mathcal{C}$ *f* 0 = *S*
| *suc S* $\mathcal{C}$ *f* (*Suc n*) =
(*if suc S* $\mathcal{C}$ *f n* $\cup$ {*f n*} $\in$ $\mathcal{C}$
then if $\exists F.\, f\ n = (\exists .F)$
then suc S $\mathcal{C}$ *f n* $\cup$ {*f n*} $\cup$
{*elim-Exists* (*f n*)
[*Term* (*SOME k*. *k* $\notin$ ($\bigcup F \in$ *suc S* $\mathcal{C}$ *f n* $\cup$ {*f n*}. *simbfun F*)) []\0]}
else if $\exists F.\, f\ n = (\neg.(\forall .F))$
then suc S $\mathcal{C}$ *f n* $\cup$ {*f n*} $\cup$
{¬.*elim-Forall* (*elim-Neg* (*f n*))
[*Term* (*SOME k*.
k $\notin$ ($\bigcup F \in$ *suc S* $\mathcal{C}$ *f n* $\cup$ {*f n*}. *simbfun F*)) [] \0]}
else suc S $\mathcal{C}$ *f n* $\cup$ {*f n*}
else suc S $\mathcal{C}$ *f n*)

Consideremos ahora la unión de los elementos $S^n_{S,\mathcal{C},f}$,

$$S_{S,\mathcal{C},f} = \bigcup_n S^n_{S,\mathcal{C},f}.$$

Su formalización es:

definition
Msuc :: $('a, 'b)$ *form set* $\Rightarrow$ $('a, 'b)$ *form set set* $\Rightarrow$ $(nat \Rightarrow ('a, 'b)\ form) \Rightarrow ('a, 'b)$ *form set*
where *Msuc S* $\mathcal{C}$ *f* = ($\bigcup n.$ *suc S* $\mathcal{C}$ *f n*)

En lo que sigue demostramos que en cualquier lenguaje **L** de primer orden, si $\mathcal{C}$ es una propiedad de consistencia alternativa de carácter finito, $S \in \mathcal{C}$ y f es una enumeración del conjunto de fórmulas de **L** entonces, $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$ que extiende a S.

Por definición, se tiene que el conjunto $S_{S,\mathcal{C},f}$ es una extensión de S:

Teorema 9.4.2 $S \subseteq S_{S,\mathcal{C},f}$.

Su formalización es:

theorem *Max-subconj*: $S \subseteq Msuc\ S\ \mathcal{C}\ f$

Al igual que en el caso proposicional (sección 6.6), la demostracion de la afirmación $S_{S,\mathcal{C},f} \in \mathcal{C}$ está basada en que $\mathcal{C}$ es de carácter finito y que los conjuntos $S^n_{S,\mathcal{C},f}$ forman una cadena (ascendente) de elementos de $\mathcal{C}$, es decir, $S^n_{S,\mathcal{C},f} \in \mathcal{C}$ y $S^n_{S,\mathcal{C},f} \subseteq S^{n+1}_{S,\mathcal{C},f}$, para todo n.

Lema 9.4.3 *La sucesión* $\{S^n_{S,\mathcal{C},f}\}_n$ *es una cadena.*

lemma *es-cadena-suc*: *es-cadena* $(suc\ S\ C\ f)$

Para demostrar que los elementos de la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ son elementos de $\mathcal{C}$ es importante observar que el conjunto de símbolos de constante que ocurren en cada una de las fórmulas de S es finito y por consiguiente el conjunto de símbolos de constante que ocurren en cada una de las fórmulas de los elementos de la sucesión $\{S^n_{S,\mathcal{C},f}\}_n$ es finito.

Nota 9.4.4 *En lo que sigue, el complemento de un conjunto A lo denotaremos por $-A$, es decir,* $-A = \{x | x \notin A\}$.

El siguiente lema afirma que si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito, de igual forma para cada uno de los elementos $S^n_{S,\mathcal{C},f}$.

Lema 9.4.5 *Si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito entonces, para cualquier n, el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas de $S^n_{S,\mathcal{C},f}$ es infinito; es decir, si $-\bigcup_{F\in S} simbfun(F)$ es infinito entonces $-\bigcup_{F\in S^n_{S,\mathcal{C},f}} simbfun(F)$ es infinito.*

Demostración: Una afirmación equivalente a la anterior es, si $\bigcap_{F\in S} -simbfun(F)$ es infinito entonces, $\bigcap_{F\in S^n_{S,\mathcal{C},f}} -simbfun(F)$ es infinito.

La demostración es por inducción sobre n.

□

Su formalización es:

lemma *finito-simbfun-suc*:
assumes *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows *infinite* $(-(\bigcup F \in suc\ S\ \mathcal{C}\ f\ n.\ simbfun\ F))$

Como una consecuencia del lema anterior tenemos que si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito entonces, para cada n y cualquier fórmula G existen símbolos de función que no ocurren en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{G\}$.

Corolario 9.4.6 *Si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito entonces, para cada n y cualquier fórmula G existen símbolos de función que no ocurren en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{G\}$:*
si $-\bigcup_{F\in S} simbfun(F)$ es infinito entonces, existe $k \notin \bigcup_{F\in S^n_{S,\mathcal{C},f}\cup\{G\}} simbfun(F)$.

Demostración: Por hipótesis, usando el lema 9.4.5 se tiene que para cualquier n,

$$-\bigcup_{F\in S^n_{S,\mathcal{C},f}} simbfun(F)$$

es infinito y por lo tanto para cualquier fórmula G el conjunto, $-\bigcup_{F\in S^n_{S,\mathcal{C},f}\cup\{G\}} simbfun(F)$ es infinito y en consecuencia existe $k \notin \bigcup_{F\in S^n_{S,\mathcal{C},f}\cup\{G\}} simbfun(F)$.

□

En la prueba anterior se utilizó el hecho de que todo conjunto infinito es no vacío. Su formalización es la siguiente.

lemma *infinito-novacio*:
assumes *infinite A*
shows $\exists x.\ x \in A$

corollary *cfinito-simbfun-suc1*:
assumes *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows $\exists k.\ k \notin (\bigcup x \in suc\ S\ \mathcal{C}\ f\ n\ \cup \{F\}.\ simbfun\ x)$
proof −
have *infinite* $(-\ (\bigcup x \in suc\ S\ \mathcal{C}\ f\ n.\ simbfun\ x))$

using *assms finito-simbfun-suc* **by** *simp*
hence *infinite* $(-(\bigcup x \in suc\ S\ \mathcal{C}\ f\ n \cup \{F\}.\ simbfun\ x))$ **by** *simp*
hence $\exists k'.\ k' \in -(\bigcup x \in suc\ S\ \mathcal{C}\ f\ n\ \cup \{F\}.\ simbfun\ x)$
by (*rule infinito-novacio*)
thus *?thesis* **by** *auto*
qed

El corolario anterior se puede escribir en términos del ϵ-operador de Hilbert, usando el siguiente lema.

lemma *operadorHilbert*: $(\exists x.\ P\ x) = P\ (SOME\ x.\ P\ x)$

corollary *cfinito-simbfun-suc2*:
assumes *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows $(SOME\ k.\ k \notin (\bigcup x \in suc\ S\ \mathcal{C}\ f\ n\ \cup \{F\}.\ simbfun\ x)) \notin (\bigcup x \in suc\ S\ \mathcal{C}\ f\ n\ \cup \{F\}.\ simbfun\ x)$

La afirmación $c \notin \bigcup_{F \in S^n_{S,\mathcal{C},f}} simbfun(F)$, c es un símbolo de función que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f}$, se puede escribir en forma equivalente $c \notin simbfun(G)$ para toda $G \in S^n_{S,\mathcal{C},f}$.

Lema 9.4.7 $c \notin \bigcup_{F \in S^n_{S,\mathcal{C},f}} simbfun(F) \equiv (\forall G \in S^n_{S,\mathcal{C},f})\ c \notin simbfun(G)$.

Su formalización es:

lemma *noparametro*:
$c \notin (\bigcup F \in suc\ S\ C\ f\ n.\ simbfun\ F) = (\forall G \in suc\ S\ C\ f\ n\ .\ c \notin simbfun\ G)$
by *simp*

De esta forma, en términos del ϵ-operador de Hilbert tenemos otra forma de enunciar en Isabelle el corolario 9.4.6, que garantiza que existen símbolos de función que no ocurren en el conjunto $S^n_{S,\mathcal{C},f} \cup \{G\}$.

corollary *cfinito-simbfun-suc3*:
assumes *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows $\forall G \in suc\ S\ \mathcal{C}\ f\ n \cup \{F\}.$
$(SOME\ k.\ k \notin (\bigcup x \in suc\ S\ \mathcal{C}\ f\ n \cup \{F\}.\ simbfun\ x)) \notin simbfun\ G$
using *assms cfinito-simbfun-suc2 noparametro*
by *simp*

El corolario anterior se utiliza en la demostración de los siguientes dos lemas que sirven para demostrar de manera más simple que los elementos $S^n_{S,\mathcal{C},f}$ son elementos de $\mathcal{C}$.

Lema 9.4.8 *Supongamos que $\mathcal{C}$ es una propiedad de consistencia alternativa, $S \in \mathcal{C}$ y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito. Si $S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C}$ y $f_n = \exists x F(x)$ para alguna fórmula F entonces,*

$$S^n_{S,\mathcal{C},f} \cup \{f_n\} \cup \{F(k)\} \in \mathcal{C}$$

donde k es algún símbolo de constante que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{f_n\}$.

lemma *suc1*:
assumes
hip1: *alt-consistencia* $\mathcal{C}$
and *hip2*: $S \in \mathcal{C}$
and *hip3*: *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows
$suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \in \mathcal{C} \wedge (\exists F.\ f\ n = (\exists .F)) \longrightarrow$
$suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \cup$
$\{elim\text{-}Exists\ (f\ n)[Term\ (SOME\ k.$
$k \notin (\bigcup F \in suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\}.\ simbfun\ F))\ []\backslash 0]\} \in \mathcal{C}$

Lema 9.4.9 *Supongamos que, $\mathcal{C}$ es una propiedad de consistencia alternativa, $S \in \mathcal{C}$ y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito. Si $S^n_{S,\mathcal{C},f} \cup \{f_n\} \in \mathcal{C}$ y $f_n = \neg\forall x F(x)$ para alguna fórmula F entonces,*

$$S^n_{S,\mathcal{C},f} \cup \{f_n\} \cup \{F(k)\} \in \mathcal{C}$$

donde k es algún símbolo de constante que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{f_n\}$.

Su formalización es:

lemma *suc2*:
assumes *hip1*: *alt-consistencia* $\mathcal{C}$
and *hip2*: $S \in \mathcal{C}$
and *hip3*: *infinite* $(-(\bigcup F \in S.\ simbfun\ F))$
shows $suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \in \mathcal{C}\ \wedge\ (\exists F.\ f\ n = (\neg.(\forall .F)))$
$\longrightarrow suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \cup$
$\{\neg.elim\text{-}Forall\ (elim\text{-}Neg\ (f\ n))$
$[Term\ (SOME\ k.\ k \notin (\bigcup F \in suc\ S\ \mathcal{C}\ f\ n \cup \{f\ n\}.\ simbfun\ F))\ []\backslash 0]\}$
$\in \mathcal{C}$

Lema 9.4.10 *Sean $\mathcal{C}$ una propiedad de consistencia alternativa y $S \in \mathcal{C}$. Si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito entonces, para cualquier n, $S^n_{S,\mathcal{C},f} \in \mathcal{C}$.*

Su formalización es:

lemma *suc-en-C*:
assumes *hip1*: *alt-consistencia* $\mathcal{C}$
and *hip2*: $S \in \mathcal{C}$
and *hip3*: *infinite* $(- (\bigcup F \in S.\ simbfun\ F))$
shows *suc* $S\ \mathcal{C}\ f\ n \in \mathcal{C}$

Por último, tenemos el siguiente teorema.

Teorema 9.4.11 *Sea $\mathcal{C}$ una propiedad de consistencia alternativa de carácter finito y $S \in \mathcal{C}$. Si el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito entonces, $S_{S,\mathcal{C},f} \in \mathcal{C}$.*

Demostración: Por el lema 9.4.3, $\{S^n_{S,\mathcal{C},f}\}_n$ es una cadena. Además, de las hipótesis $\mathcal{C}$ es una propiedad de consistencia alternativa, $S \in \mathcal{C}$ y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito, usando el lema 9.4.10, también tenemos que para cada n, $S^n_{S,\mathcal{C},f} \in \mathcal{C}$. De lo anterior y como por hipótesis $\mathcal{C}$ es una colección de carácter finito, usando el teorema 6.6.5 se concluye que $S_{S,\mathcal{C},f} \in \mathcal{C}$. □

Su formalización es:

theorem *Max-en-C*:
assumes
hip1: *alt-consistencia* $\mathcal{C}$
and *hip2*: *caracter-finito* $\mathcal{C}$
and *hip3*: $S \in \mathcal{C}$
and *hip4*: *infinite* $(- (\bigcup F \in S.\ simbfun\ F))$
shows *Msuc* $S\ \mathcal{C}\ f \in \mathcal{C}$
proof(*unfold Msuc-def*)
have *es-cadena* (*suc* $S\ \mathcal{C}\ f$) **by**(*rule es-cadena-suc*)
moreover
have $\forall\, n.$ *suc* $S\ \mathcal{C}\ f\ n \in \mathcal{C}$
proof(*rule allI*)
fix *n*
show *suc* $S\ \mathcal{C}\ f\ n \in \mathcal{C}$ **using** *hip1 hip3 hip4* **by**(*rule suc-en-C*)
qed
ultimately

show $(\bigcup\ n.\ suc\ S\ \mathcal{C}\ f\ n) \in \mathcal{C}$ **by**(*rule union-cadena-cerrada*[*OF hip2*])
qed

Tenemos que $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$, con las mismas condiciones que en el caso proposicional, (teorema 6.6.10):

Teorema 9.4.12 *Sean* **L** *un lenguaje de primer orden, S un conjunto de fórmulas de* **L**, $\mathcal{C}$ *una colección de conjuntos de fórmulas de* **L** *cerrada por subconjuntos y f una enumeración del conjunto de fórmulas de* **L**. *Entonces* $S_{S,\mathcal{C},f}$ *es un elemento maximal de* $\mathcal{C}$.

La demostración es igual al caso proposicional.

Su formalización es:

theorem *suc-maximal*:
assumes *hip1*: $\forall\, y.\ \exists\, n.\ y = f\ n$ **and** *hip2*: *subconj-cerrada* $\mathcal{C}$
shows *maximal* $(Msuc\ S\ \mathcal{C}\ f)\ \mathcal{C}$

Corolario 9.4.13 *Sean* **L** *un lenguaje de primer orden,* $\mathcal{C}$ *una colección de conjuntos de fórmulas de* **L**, *S un conjunto de fórmulas y f una sucesión de fórmulas. Supongamos que,*

- *(hip1)* $\mathcal{C}$ *es una propiedad de consistencia alternativa.*
- *(hip2)* $\mathcal{C}$ *es de carácter finito.*
- *(hip3)* $S \in \mathcal{C}$.
- *(hip4) El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito.*
- *(hip5) f es una enumeración del conjunto de fórmulas de L.*

Entonces $S_{S,\mathcal{C},f} = \bigcup_n S^n_{S,\mathcal{C},f}$ *tiene las siguientes propiedades:*

(a) $S \subseteq S_{S,\mathcal{C},f}$.

(b) $S_{S,\mathcal{C},f} \in \mathcal{C}$.

(c) $S_{S,\mathcal{C},f}$ *es un elemento maximal de* $\mathcal{C}$.

Demostración:

(a) Por el teorema 9.4.2.

(b) Por el teorema 9.4.11, usando las hipótesis $(hip1)$, $(hip2)$, $(hip3)$ y $(hip4)$.

(c) Usando la hipótesis $(hip2)$, por el teorema 6.4.2, se tiene que $\mathcal{C}$ es subconjunto cerrada. De esto último y la hipótesis $(hip5)$, por el teorema 9.4.12, obtenemos (c).

□

corollary *ExtensionConsistente*:
assumes *hip1*: *alt-consistencia* $\mathcal{C}$
and *hip2*: *caracter-finito* $\mathcal{C}$
and *hip3*: $S \in \mathcal{C}$
and *hip4*: *infinite* $(- (\bigcup F \in S.\ simbfun\ F))$
and *hip5*: $\forall y.\ \exists n.\ y = f\ n$
shows $S \subseteq Msuc\ S\ \mathcal{C}\ f$ **and** $Msuc\ S\ \mathcal{C}\ f \in \mathcal{C}$ **and** *maximal* $(Msuc\ S\ \mathcal{C}\ f)\ \mathcal{C}$
proof −
show $S \subseteq Msuc\ S\ \mathcal{C}\ f$ **using** *Max-subconj* **by** *auto*
next
show $Msuc\ S\ \mathcal{C}\ f \in \mathcal{C}$ **using** *Max-en-C*[*OF hip1 hip2 hip3 hip4*] **by** *simp*
next
show *maximal* $(Msuc\ S\ \mathcal{C}\ f)\ \mathcal{C}$
using *caracter-finito-cerrado*[*OF hip2*] **and** *suc-maximal*[*OF hip5*]
by *simp*
qed

9.5 Conjuntos de Hintikka y modelos de Herbrand

En esta sección demostramos que si **L** es un lenguaje de primer orden, cualquier subconjunto de *sentencias* de un *conjunto de Hintikka* con respecto a **L**, es *satisfacible* en una *estructura de Herbrand*.

9.5.1 Sentencias

Definición 9.5.1 *Una fórmula es una* **sentencia** *si no contiene variables libres.*

Para la formalización de este concepto hacemos uso de la notación de de Bruijn y definimos lo que significa una fórmula *cerrada en el nivel n*.

Definición 9.5.2 *Una fórmula es* **cerrada** *en el nivel n si no tiene un índice de variable k mayor o igual que n+j, donde j es el número de cuantificadores bajo cuyo alcance está el índice k.*

Por ejemplo, la fórmula $\forall P(0,1) \wedge \forall Q(0)$ es cerrada en el nivel 1 pero no es cerrada en el nivel 0.

La siguiente función $(sentencia\ m\ F)$ determina si una fórmula F es cerrada en el nivel m.

```
primrec sentencia :: nat ⇒ ('a, 'b) form ⇒ bool
where
 sentencia n FF' = True
| sentencia n TT' = True
| sentencia n (Atom P ts) = cerradots n ts
| sentencia n (F ∧. G) = (sentencia n F ∧ sentencia n G)
| sentencia n (F ∨. G) = (sentencia n F ∧ sentencia n G)
| sentencia n (F →. G) = (sentencia n F ∧ sentencia n G)
| sentencia n (¬. F) = sentencia n F
| sentencia n (∀ . F) = sentencia (Suc n) F
| sentencia n (∃ . F) = sentencia (Suc n) F
```

Por ejemplo, $(sentencia\ 0\ \forall P(0,1) \wedge \forall Q(0))$ es falso y $(sentencia\ 1\ \forall P(0,1) \wedge \forall Q(Var0))$ es verdadero.

Así, *una fórmula es una sentencia si es cerrada en el nivel 0*. Por ejemplo, puesto que $(sentencia\ 0\ \forall\forall P(0,1))$ es verdadera, entonces $\forall\forall P(0,1)$ es una sentencia.

Tenemos el siguiente lema sobre términos cerrados y fórmulas cerradas en el nivel i.

```
lemma sust-sentencia [simp]:
 assumes cerradot 0 t
 and sentencia (Suc i) p
 shows sentencia i (p[t\i])
using assms
by (induct p arbitrary: i t) simp-all
```

9.5.2 Conjuntos de Hintikka

Definición 9.5.3 *Un conjunto H de fórmulas es un* **conjunto de Hintikka**, *si se tiene lo siguiente.*

1. *Para toda fórmula atómica $P(t_1, \ldots, t_n)$, $P(t_1, \ldots, t_n) \notin H$ o $\neg P(t_1, \ldots, t_n) \notin H$.*
2. *$\bot \notin H$ y $\neg\top \notin H$.*
3. *Si $\neg\neg F \in H$, entonces $F \in H$.*

4. *Si* $F \wedge G \in H$, *entonces,* $F \in H$ *y* $G \in H$.
5. *Si* $\neg(F \vee G) \in H$, *entonces* $\neg F \in H$ *y* $\neg G \in H$.
6. *Si* $F \vee G \in H$, *entonces* $F \in H$ *o* $G \in H$.
7. *Si* $\neg(F \wedge G) \in H$, *entonces* $\neg F \in H$ *o* $\neg G \in H$.
8. *Si* $F \rightarrow G \in H$, *entonces* $\neg F \in H$ *o* $G \in H$.
9. *Si* $\neg(F \rightarrow G) \in H$, *entonces* $F \in H$ *y* $\neg G \in H$.
10. *Si* $\forall x F(x) \in H$, *entonces* $F(t) \in H$, *para cualquier término cerrado* $t \in L$.
11. *Si* $\neg\exists x F(x) \in H$, *entonces* $\neg F(t) \in H$, *para cualquier término cerrado* $t \in L$.
12. *Si* $\exists x F(x) \in H$, *entonces* $F(t) \in H$, *para algún término cerrado* $t \in L$.
13. *Si* $\neg\forall x F(x) \in H$, *entonces* $\neg F(t) \in H$, *para algún término cerrado* $t \in L$.

definition
hintikka :: (*'a*, *'b*) *form set* ⇒ *bool* **where**
hintikka H =
((∀ *P ts*. ¬ (*Atom P ts* ∈ *H* ∧ (¬.*Atom P ts*) ∈ *H*)) ∧
FF' ∉ *H* ∧ (¬.*TT'*) ∉ *H* ∧
(∀ *F*. (¬.¬.*F*) ∈ *H* ⟶ *F* ∈ *H*) ∧
(∀ *F G*. (*F* ∧. *G*) ∈ *H* ⟶ *F* ∈ *H* ∧ *G* ∈ *H*) ∧
(∀ *F G*. (¬.(*F* ∨. *G*)) ∈ *H* ⟶ (¬.*F*) ∈ *H* ∧ (¬.*G*) ∈ *H*) ∧
(∀ *F G*. (*F* ∨. *G*) ∈ *H* ⟶ *F* ∈ *H* ∨ *G* ∈ *H*) ∧
(∀ *F G*. (¬.(*F* ∧. *G*)) ∈ *H* ⟶ (¬.*F*) ∈ *H* ∨ (¬.*G*) ∈ *H*) ∧
(∀ *F G*. (*F* →. *G*) ∈ *H* ⟶ (¬.*F*) ∈ *H* ∨ *G* ∈ *H*) ∧
(∀ *F G*. (¬.(*F* →. *G*)) ∈ *H* ⟶ *F* ∈ *H* ∧ (¬.*G*) ∈ *H*) ∧
(∀ *F t*. *cerradot 0 t* ⟶ (∀ .*F*) ∈ *H* ⟶ *F*[*t**0*] ∈ *H*) ∧
(∀ *F t*. *cerradot 0 t* ⟶ (¬.(∃ .*F*)) ∈ *H* ⟶ (¬.*F*[*t**0*]) ∈ *H*) ∧
(∀ *F*. (∃ .*F*) ∈ *H* ⟶ (∃ *t*. *cerradot 0 t* ∧ *F*[*t**0*] ∈ *H*)) ∧
(∀ *F*. (¬.(∀ .*F*)) ∈ *H* ⟶ (∃ *t*. *cerradot 0 t* ∧ (¬.*F*[*t**0*]) ∈ *H*)))

9.5.3 Estructuras de Herbrand

Recordemos que dada una **L**-estructura $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$, en la definición 8.2.3 a cada término t se le asoció, por cada asignación A de valores a variables, un valor $t^{\mathcal{I},A}$ en el dominio $\mathcal{D}$. Sin embargo, para términos cerrados, este valor no depende de la asignación A; en este caso, si el dominio $\mathcal{D}$ es igual al conjunto de términos cerrados del lenguaje **L**, podemos hacer que el valor asignado a cada término cerrado t sea igual

a t, es decir, $t^{\mathcal{F},\mathcal{A}} = t$ (para cualquier asignación A). Una estructura $\mathcal{M}$ con esta característica se llama un *estructura de Herbrand*.

Definición 9.5.4 *Una* **L***-estructura* $\mathcal{M} = (\mathcal{D}h, \mathcal{R}, \mathcal{F}h)$*, se llama una* **L-estructura de Herbrand** *si el dominio* $\mathcal{D}h$ *y la interpretación* $\mathcal{F}h$ *de símbolos de función se definen de la siguiente manera.*

- $\mathcal{D}h$ *es el conjunto más pequeño tal que, si* f *es un símbolo de función n-aria,* $n \geq 0$*, y* $t_1, t_2, \ldots, t_n$ *son elementos de* $\mathcal{D}h$ *entonces* $f(t_1, t_2, \ldots, t_n) \in \mathcal{D}h$*.*

  Los elementos de $\mathcal{D}h$ *se llaman* **términos de Herbrand**.

- *La interpretación* $\mathcal{F}h$ *se define de la siguiente manera.*

  Si f *es un símbolo de función n-aria,* $n \geq 0$*,* $f^{\mathcal{F}h} : \mathcal{D}h^n \to \mathcal{D}h$ *está definida por,* $f^{\mathcal{F}h}(t_1, t_2, \ldots, t_n) = f(t_1, t_2, \ldots, t_n)$*.*

Con el fin de formalizar en Isabelle las estructuras de Herbrand, se define un nuevo tipo de dato *hterm* para representar los términos de Herbrand.

datatype *'a hterm = HTerm 'a 'a hterm list*

La representación en Isabelle de la interpretación $\mathcal{F}h$ de símbolos de función es la siguiente.

fun *Fh :: 'a ⇒ 'a hterm list ⇒ 'a hterm* **where**
Fh f ts = HTerm f ts

La siguiente función *hterm-term* : $\mathcal{D}h \to T$, T el conjunto de términos de **L**, establece una correspondencia entre los términos de Herbrand y los términos cerrados.

$$\textit{hterm-term}(f(t_1, t_2, \ldots, t_n)) = f(\textit{hterm-term}(t_1), \textit{hterm-term}(t_2), \ldots, \textit{hterm-term}(t_n)).$$

Para la definición en Isabelle utilizamos la función auxiliar $(\textit{hterms-terms}\ ts)$ que halla el conjunto de términos correspondiente a un conjunto de términos de Herbrand.

primrec *hterm-term :: 'a hterm ⇒ 'a term* **and** *hterms-terms :: 'a hterm list ⇒ 'a term list*
where
hterm-term (HTerm f hts) = Term f (hterms-terms hts)
| *hterms-terms [] = []*
| *hterms-terms (ht # hts) = hterm-term ht # hterms-terms hts*

El siguiente lema verifica que el término que la función *hterm-term* le asigna a cada término de Herbrad es cerrado.

Lema 9.5.5 *Sea t un término de Herbrand, entonces hterm-term*(t) *es un término cerrado.*

Su formalización es:

lemma *cerrado-hterm* [*simp*]:
cerradot 0 (*hterm-term* (*ht*::*'a hterm*))
cerradots 0 (*hterms-terms* (*hts*::*'a hterm list*))

Los siguientes dos lemas garantizan que en estas estructuras el valor de cada término cerrado coincide con su correspondiente término de Herbrand.

Lema 9.5.6 *Si t es un término cerrado entonces, hterm-term*$(t^{\mathcal{F}h,A}) = t$.

Su formalización es:

lemma *herbrand-evalt* [*simp*]:
cerradot 0 t $\Longrightarrow$ *hterm-term* (*evalt Fh A t*) $= t$
cerradots 0 ts $\Longrightarrow$ *hterms-terms* (*evalts Fh A ts*) $=$ *ts*

Lema 9.5.7 *Si t es un término de Herbrand, entonces* $[hterm\text{-}term(t)]^{\mathcal{F}h,A} = t$

lemma *herbrand-evalt'* [*simp*]:
evalt Fh A (*hterm-term ht*) $=$ *ht*
evalts Fh A (*hterms-terms hts*) $=$ *hts*

9.5.4 Satisfacibilidad de conjuntos de Hintikka

Un conjunto S de fórmulas de un lenguaje **L** es *satisfacible en una* **L**-estructura $\mathcal{M}$ si las fórmulas de S son verdaderas en $\mathcal{M}$ para alguna asignación de valores a variables.

Definición 9.5.8 *Sea S un conjunto de fórmulas del lenguaje* **L**. *S es* **satisfacible en la L**-*estructura* $\mathcal{M}$ *si existe una asignación A de valores a variables tal que,* $F^{\mathcal{F},\mathcal{I},A} = \mathsf{V}$ *para toda fórmula* $F \in S$. *S es* **satisfacible** *si es satisfacible en alguna* **L**-*estructura.*

Las **L**-estructuras de Herbrand $\mathcal{M} = (\mathcal{D}h, \mathcal{R}, \mathcal{F}h)$ se diferencian únicamente en la interpretación de los símbolos de relación. En particular, dado un conjunto H de Hintikka con respecto a un lenguaje **L**, consideremos la **L**-estructura de Herbrand

$$\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$$

donde, si P es un símbolo de relación n-aria, $P^{\mathcal{R}_H}$ está definida por,

$$P^{\mathcal{R}_H}(t_1, t_2, \ldots, t_n) \Longleftrightarrow P(\mathit{hterm\text{-}term}(t_1), \mathit{hterm\text{-}term}(t_2), \ldots, \mathit{hterm\text{-}term}(t_n)) \in H.$$

En este caso se tiene que:

$$\text{si } F \in H \text{ y } F \text{ es una sentencia, entonces } F^{\mathcal{R}_H, \mathcal{F}h, A} = \mathsf{V} \qquad (\mathit{ModeloHintikka})$$

Obsérvese que este valor no depende de la asignación A ya que F es una sentencia.

La formalización en Isabelle de la interpretación $\mathcal{R}_H$ de símbolos de relación es como sigue.

fun *Rh* :: (*'a*, *'b*) *form set* ⇒ *'b* ⇒ *'a hterm list* ⇒ *bool* **where**
Rh H P ts = (*Atom P* (*hterms-terms ts*) ∈ *H*)

En el caso particular en el que F es una fórmula atómica $P(t_1, t_2, \ldots t_n)$, se tiene el siguiente resultado.

Lema 9.5.9 *Sea H un conjunto de Hintikka y $\mathcal{M}_H$ la estructra de Herbrand correspondiente. Si $P(t_1, t_2, \ldots, t_n)$ es una sentencia, entonces $[P(t_1, t_2, \ldots, t_n)]^{\mathcal{R}_H, \mathcal{F}h, A} = \mathsf{V}$ si y solamente si $P(t_1, t_2, \ldots t_n) \in H$.*

Para demostrar el teorema de Hintikka 9.5.11, demostramos en primer lugar el siguiente lema.

Lema 9.5.10 *Sea H un conjunto de Hintikka y $\mathcal{M}_H$ su estructura de Herbrand. Si $F \in H$ y F es una sentencia entonces $F^{\mathcal{R}_H, \mathcal{F}h, A} = \mathsf{V}$, y si $\neg F \in H$ y F es una sentencia entonces $(\neg F)^{\mathcal{R}_H, \mathcal{F}h, A} = \mathsf{V}$.*

Demostración: La demostración es por inducción bien fundamentada. Para definir la relación bien fundamentada $\prec$ en el conjunto de fórmulas, consideremos la siguiente función s que determina el tamaño de una fórmula. Sean F y G fórmulas, entonces

- $s(F) = 1$ si F es una fórmula atómica, $\bot$ o $\top$
- $s(\neg F) = 1 + s(F)$
- $s(F \wedge G) = s(F \vee G) = s(F \to G) = 1 + s(F) + s(G)$
- $\forall x\, F = 1 + s(F)$

- $\exists x\, F = 1 + s(F)$

Definimos la relación $\prec$ igual a la imagen inversa de la relación en $\mathbb{N}$ inducida por la función de medida s,

$$F \prec G \Longleftrightarrow s(F) < s(G).$$

Sea F una fórmula arbitraria, la hipótesis de inducción afirma lo siguiente.

Hipótesis de inducción: si $G \prec F$ entonces, si $G \in H$ y G es una sentencia entonces $G^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, y si $\neg G \in H$ y G es una sentencia entonces $(\neg G)^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$.

Hay que demostrar que,

(a) si $F \in H$ y F es una sentencia entonces $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, y

(b) si $\neg F \in H$ y F es una sentencia entonces $(\neg F)^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$.

La demostración es por casos en la fórmula F. Los casos proposicionales se demuestran de la misma manera que se hizo en la prueba del teorema 6.7.3. En seguida, demostramos los casos de las fórmulas con cuantificadores.

Caso8 Supongamos que F es la fórmula $\forall w\, G(w)$.

(a) Supogamos que $\forall w\, G(w) \in H$ y $\forall w\, G(w)$ es una sentencia.

Hay que demostrar que $[\forall w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Es decir, hay que demostrar que $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, para toda asignación A' de valores a variables que es w-variante de A (los valores de A y A' son iguales excepto, posiblemente, en w). Sea A' una w-variante de A tal que $A'(w) = z$, donde z es un término de Herbrand. Por el lema 9.5.5 tenemos que $\mathit{hterm\text{-}term}(z)$ es un término cerrado. Por lo tanto, puesto que por hipótesis H es de Hintikka y $\forall w\, G(w) \in H$, se tiene que $G(\mathit{hterm\text{-}term}(z)) \in H$ y también se tiene que $G(\mathit{hterm\text{-}term}(z))$ es una sentencia ya que $\forall w\, G(w)$ es una sentencia. Además, por la definición del tamaño de una fórmula, $G(\mathit{hterm\text{-}term}(z)) \prec \forall w\, G(w)$. Por consiguiente, usando la hipótesis de inducción, tenemos que la siguiente implicación es verdadera, si $G(\mathit{hterm\text{-}term}(z)) \in H$ y $G(\mathit{hterm\text{-}term}(z))$ es una sentencia entonces $[G(\mathit{hterm\text{-}term}(z)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. De esta forma, $[G(\mathit{hterm\text{-}term}(z)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Por lo tanto, puesto que por el lema 9.5.7 se tiene que $[\mathit{hterm\text{-}term}(z)]^{\mathcal{F}h,A} = z$, es decir, $[\mathit{hterm\text{-}term}(z)]^{\mathcal{F}h,A} = A'(w)$ y A' es una w-variante de A entonces, $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$.

(b) Supongamos que $\neg\forall w\, G(w) \in H$ y $\forall w\, G(w)$ es una sentencia.

Hay que demostrar que $[\neg\forall w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Es decir, hay que demostrar que $\neg[\forall w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, es decir, $[\forall w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{F}$, o lo que es lo mismo, hay que demostrar que $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{F}$, para alguna asignación A' de valores a variables que es w-variante de A. Puesto que por hipótesis H es de Hintikka y $\neg\forall w\, G(w) \in H$ se tiene que $\neg G(t) \in H$, para algún término cerrado $t \in \mathbf{L}$. Entonces $G(t)$ es una sentencia ya que $\forall w\, G(w)$ es una sentencia. Además, por la definición del tamaño de

una fórmula se tiene que $G(t) \prec \forall w\, G(w)$. Por consiguiente, usando la hipótesis de inducción, tenemos que la siguiente implicación es verdadera, si $\neg G(t) \in H$ y $G(t)$ es una sentencia entonces $[\neg G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. De esta forma, $[\neg G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, es decir, $\neg[G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, es decir, $[G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{F}$. Sea A' la w-variante de A, donde $A'(w) = t^{\mathcal{F}h,A}$, entonces $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{F}$.

Caso9 Supongamos que F es la fórmula $\exists w\, G(w)$.

(a) Supongamos que $\exists w\, G(w) \in H$ y $\exists w\, G(w)$ es una sentencia.

Hay que demostrar que $[\exists w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Es decir, hay que demostrar que $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, para alguna asignación A' de valores a variables que es w-variante de A. Puesto que por hipótesis H es de Hintikka y $\exists w\, G(w) \in H$ se tiene que $G(t) \in H$, para algún término cerrado $t \in \mathbf{L}$. Entonces, $G(t)$ es una sentencia ya que $\exists w\, G(w)$ es una sentencia. Además, por la definición del tamaño de una fórmula se tiene que $G(t) \prec \forall w\, G(w)$. Por consiguiente, usando la hipótesis de inducción, tenemos que la siguiente implicación es verdadera, si $G(t) \in H$ y $G(t)$ es una sentencia entonces $[G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. De esta forma, $[G(t)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Sea A' la w-variante de A, donde $A'(w) = t^{\mathcal{F}h,A}$, entonces $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$.

(b) Supongamos que $\neg\exists w\, G(w) \in H$ y $\exists w\, G(w)$ es una sentencia.

Hay que demostrar que $[\neg\exists w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Es decir, hay que demostrar que $\neg[\exists w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$, es decir, $[\exists w\, G(w)]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{F}$, o lo que es lo mismo, hay que demostrar que $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{F}$, para toda asignación A' de valores a variables que es w-variante de A. Sea A' una w-variante de A tal que $A'(w) = z$, donde z es un término de Herbrand. Por el lema 9.5.5 tenemos que $\mathit{hterm\text{-}term}(z)$ es un término cerrado. Por lo tanto, puesto que por hipótesis H es de Hintikka y $\neg\exists w\, G(w) \in H$ se tiene que $\neg G(\mathit{hterm\text{-}term}(z)) \in H$ y también se tiene que $G(\mathit{hterm\text{-}term}(z))$ es una sentencia ya que $\exists w\, G(w)$ es una sentencia. Además, por la definición del tamaño de una fórmula, $G(\mathit{hterm\text{-}term}(z)) \prec \exists w\, G(w)$. Por consiguiente, usando la hipótesis de inducción, tenemos que la siguiente implicación es verdadera, si $\neg G(\mathit{hterm\text{-}term}(z)) \in H$ y $G(\mathit{hterm\text{-}term}(z))$ es una sentencia entonces $[\neg G(\mathit{hterm\text{-}term}(z))]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. De esta forma, $[\neg G(\mathit{hterm\text{-}term}(z))]^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Por lo tanto, puesto que por el lema 9.5.7 se tiene que $[\mathit{hterm\text{-}term}(z)]^{\mathcal{F}h,A} = z$, es decir, $[\mathit{hterm\text{-}term}(z)]^{\mathcal{F}h,A} = A'(w)$ y A' una w-variante de A entonces, $[\neg G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$ es decir, $\neg[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$, o lo que es lo mismo, $[G(w)]^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{F}$.

La formalización en Isabelle utiliza los siguientes lemas. La relación *measure size* en Isabelle corresponde a la formalización de la relación $\prec$.

lemma *caso1*:
assumes *hip1*: *hintikka H*
and *hip2*: $\forall$ *G*. (*G*, *FF'*) $\in$ *measure size* $\longrightarrow$
($G \in H \longrightarrow$ *sentencia 0 G* $\longrightarrow$ *eval* I_R I_F I_A *G*) $\wedge$

$((\neg.G) \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ (\neg.G))$
shows $(FF' \in H \longrightarrow sentencia\ 0\ FF' \longrightarrow eval\ I_R\ I_F\ I_A\ FF') \wedge$
$((\neg.FF') \in H \longrightarrow sentencia\ 0\ FF' \longrightarrow eval\ I_R\ I_F\ I_A\ (\neg.FF'))$

lemma *caso2*:
assumes *hip1*: *hintikka H*
and *hip2*: $\forall G.\ (G, TT') \in measure\ size \longrightarrow$
$(G \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ G) \wedge$
$((\neg.G) \in H \longrightarrow sentencia\ 0\ G$
$\longrightarrow eval\ (\lambda a\ ts.\ Atom\ a\ (terms\text{-}of\text{-}hterms\ ts) \in H)\ I_F\ A\ (\neg.G))$
shows $(TT' \in H \longrightarrow sentencia\ 0\ TT' \longrightarrow eval\ I_R\ I_F\ I_A\ TT') \wedge$
$((\neg.TT') \in H \longrightarrow sentencia\ 0\ TT' \longrightarrow eval\ I_R\ I_F\ I_A\ (\neg.TT'))$

lemma *caso3*:
assumes *hip1*: *hintikka H*
and *hip2*: $\forall G.\ (G, Atom\ P\ list) \in measure\ size \longrightarrow$
$(G \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ G) \wedge$
$((\neg.G) \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ (\neg.G))$
shows $(Atom\ P\ list \in H \longrightarrow sentencia\ 0\ (Atom\ P\ list) \longrightarrow$
$eval\ (Rh\ H)\ Fh\ I_A\ (Atom\ P\ list)) \wedge$
$((\neg.Atom\ P\ list) \in H \longrightarrow sentencia\ 0\ (Atom\ P\ list) \longrightarrow$
$eval\ (Rh\ H)\ Fh\ I_A\ (\neg.Atom\ P\ list))$

lemma *caso4*:
assumes *hip1*: *hintikka H*
and *hip2*: $\forall G.\ (G, form1 \wedge. form2) \in measure\ size \longrightarrow$
$(G \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ G) \wedge$
$((\neg.G) \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ (\neg.G))$
shows $((form1 \wedge. form2) \in H \longrightarrow$
$sentencia\ 0\ (form1 \wedge. form2) \longrightarrow$
$eval\ I_R\ I_F\ I_A\ (form1 \wedge. form2)) \wedge$
$((\neg.(form1 \wedge. form2)) \in H \longrightarrow$
$sentencia\ 0\ (form1 \wedge. form2) \longrightarrow$
$eval\ I_R\ I_F\ I_A\ (\neg.(form1 \wedge. form2)))$

lemma *caso5*:
assumes *hip1*: *hintikka H*
and *hip2*: $\forall G.\ (G, form1 \vee. form2) \in measure\ size \longrightarrow$
$(G \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ G) \wedge$
$((\neg.G) \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ I_R\ I_F\ I_A\ (\neg.G))$
shows $((form1 \vee. form2) \in H \longrightarrow$
$sentencia\ 0\ (form1 \vee. form2) \longrightarrow$
$eval\ I_R\ I_F\ I_A\ (form1 \vee. form2)) \wedge$
$((\neg.(form1 \vee. form2)) \in H \longrightarrow$

sentencia 0 (form1 ∨. form2) ⟶
eval I_R I_F I_A (¬.(form1 ∨. form2)))

lemma *caso6*:
assumes *hip1*: *hintikka H*
and *hip2*: ∀ *G*. (*G*, *form1* →. *form2*) ∈ *measure size* ⟶
(*G* ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R I_F I_A G*) ∧
((¬.*G*) ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R I_F I_A* (¬.*G*))
shows ((*form1* →. *form2*) ∈ *H* ⟶
sentencia 0 (*form1* →. *form2*) ⟶
eval I_R I_F I_A (*form1* →. *form2*)) ∧
((¬.(*form1* →. *form2*)) ∈ *H* ⟶
sentencia 0 (*form1* →. *form2*) ⟶
eval I_R I_F I_A (¬.(*form1* →. *form2*)))

lemma *caso7*:
assumes *hip1*: *hintikka H*
and *hip2*: ∀ *G*. (*G*, (¬.*form*)) ∈ *measure size* ⟶
(*G* ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R I_F I_A G*) ∧
((¬.*G*) ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R I_F I_A* (¬.*G*))
shows ((¬.*form*) ∈ *H* ⟶ *sentencia 0* (¬.*form*) ⟶ *eval I_R I_F I_A* (¬.*form*)) ∧
((¬.(¬.*form*)) ∈ *H* ⟶
sentencia 0 (¬.*form*) ⟶ *eval I_R I_F I_A* (¬.(¬.*form*)))

lemma *caso8*:
assumes *hip1*: *hintikka H*
and *hip2*: ∀ *G*. (*G*, (∀ .*form*)) ∈ *measure size* ⟶
(*G* ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R Fh I_A G*) ∧
((¬.*G*) ∈ *H* ⟶
sentencia 0 G ⟶ *eval I_R Fh I_A* (¬.*G*))
shows ((∀ .*form*) ∈ *H* ⟶ *sentencia 0* (∀ .*form*) ⟶ *eval I_R Fh I_A* (∀ .*form*)) ∧
((¬.(∀ .*form*)) ∈ *H* ⟶ *sentencia 0* (∀ .*form*) ⟶
eval I_R Fh I_A (¬.(∀ .*form*)))

lemma *caso9*:
assumes *hip1*: *hintikka H*
and *hip2*: ∀ *G*. (*G*, (∃ .*form*)) ∈ *measure size* ⟶
(*G* ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R Fh I_A G*) ∧
((¬.*G*) ∈ *H* ⟶ *sentencia 0 G* ⟶ *eval I_R Fh I_A* (¬.*G*))
shows ((∃ .*form*) ∈ *H* ⟶ *sentencia 0* (∃ .*form*) ⟶ *eval I_R Fh I_A*(∃ .*form*)) ∧
((¬.(∃ .*form*)) ∈ *H* ⟶ *sentencia 0* (∃ .*form*) ⟶
eval I_R Fh I_A (¬.(∃ .*form*)))

Usando los lemas anteriores, se tiene la formalización del teorema 9.5.10.

lemma *hintikka-model-aux*:
assumes *hip*: *hintikka H*
shows $(F \in H \longrightarrow sentencia\ 0\ F \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ F) \wedge$
$((\neg.F) \in H \longrightarrow sentencia\ 0\ F \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ (\neg.F))$
proof(*rule-tac r=measure size* **and** *a=F* **in** *wf-induct*)
show *wf*(*measure size*) **by** *simp*
next
fix *F*::('*a*, '*b*)*form*
assume
hip1: $\forall\, G.\ (G, F) \in measure\ size \longrightarrow$
$(G \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ G) \wedge$
$((\neg.G) \in H \longrightarrow sentencia\ 0\ G \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ (\neg.G))$
show $(F \in H \longrightarrow sentencia\ 0\ F \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ F) \wedge$
$((\neg.F) \in H \longrightarrow sentencia\ 0\ F \longrightarrow eval\ (Rh\ H)\ Fh\ I_A\ (\neg.F))$
proof(*cases F*)
assume $F = FF'$
thus *?thesis* **using** *caso1 hip hip1* **by** *simp*
next
assume $F = TT'$
thus *?thesis* **using** *caso2 hip hip1* **by** *auto*
next
fix *b list*
assume $F = Atom\ b\ list$
thus *?thesis* **using** *hip hip1 caso3*[*of H b list*] **by** *simp*
next
fix *form1 form2*
assume $F = (form1 \wedge. form2)$
thus *?thesis* **using** *hip hip1 caso4*[*of H form1 form2*] **by** *simp*
next
fix *form1 form2*
assume $F = (form1 \vee. form2)$
thus *?thesis* **using** *hip hip1 caso5*[*of H form1 form2*] **by** *simp*
next
fix *form1 form2*
assume $F = (form1 \rightarrow. form2)$
thus *?thesis* **using** *hip hip1 caso6*[*of H form1 form2*] **by** *simp*
next
fix *form*
assume $F = (\neg.form)$
thus *?thesis* **using** *hip hip1 caso7*[*of H form*] **by** *auto*
next
fix *form*
assume $F = (\forall\, .form)$

```
  thus ?thesis using hip hip1 caso8[of H form] by simp
 next
  fix form
  assume F = (∃ .form)
  thus ?thesis using hip hip1 caso9[of H form] by simp
 qed
qed
```

Por último, como un corolario del teorema anterior se tiene la demostración de la existencia de un modelo de Herbrand para el conjunto de sentencias de un conjunto de Hintikka cualquiera.

Corolario 9.5.11 (ModeloHintikka) *Sea H un conjunto de Hintikka y $\mathcal{M}_H$ su estructura de Herbrand. Si $F \in H$ y F es una sentencia entonces, $F^{\mathcal{R}_H,\mathcal{F}_h,A} = \mathsf{V}$.*

Demostración: Es consecuencia inmediata del teorema 9.5.10

□

```
corollary ModeloHintikka:
 assumes hintikka H and  F ∈ H and  sentencia 0 F
 shows eval (Rh H) Fh I_A F
using assms hintikka-model-aux
by auto
```

9.6 Conjuntos maximales y conjuntos de Hintikka

Sea $\mathcal{C}$ una propiedad de consistencia alternativa que es cerrada por subconjuntos. En esta sección mostramos que si $S_{S,\mathcal{C},f} \in \mathcal{C}$ y es un conjunto maximal entonces es un conjunto de Hintikka.

Teorema 9.6.1 *Dado un lenguaje* **L** *de primer orden. Sea $\mathcal{C}$ una colección de conjuntos de fórmulas, S un conjunto de fórmulas y f una sucesión de fórmulas. Consideremos $S_{S,\mathcal{C},f} = \bigcup_n S^n_{S,\mathcal{C},f}$. Supongamos que,*

- *(hip1) $\mathcal{C}$ es una propiedad de consistencia alternativa.*
- *(hip2) $\mathcal{C}$ es cerrada por subconjuntos.*
- *(hip3a) Para toda fórmula F existe n tal que $\exists x F(x) = f_n$.*

- *(hip3b) Para toda fórmula F existe n tal que* $\neg\forall x F(x) = f_n$.
- *(hip4)* $S_{S,\mathcal{C},f}$ *es un elemento maximal de* $\mathcal{C}$.
- *(hip5)* $S_{S,\mathcal{C},f} \in \mathcal{C}$.

Entonces $S_{S,\mathcal{C},f}$ *es un conjunto de Hintikka.*

Demostración:

A continuación mostramos que $S_{S,\mathcal{C},f}$ cumple las propiedades que definen un conjunto de Hintikka (definición 9.5.3).

Las primeras tres propiedades se cumplen por las hipótesis $(hip1)$ y $(hip5)$.

Las propiedades (4)-(12) se demuestran usando las hipótesis $(hip1)$, $(hip4)$ y $(hip5)$. En particular, las propiedades (4) a (10) se demuestran en forma similar al caso proposicional (teorema 6.9.1).

11. Supongamos que $\forall x F(x) \in S_{S,\mathcal{C},f}$. Hay que demostrar que $F(t) \in S_{S,\mathcal{C},f}$, para cualquier término cerrado $t \in \mathbf{L}$. Sea t un término cerrado, si $\forall x F(x) \in S_{S,\mathcal{C},f}$ entonces $S_{S,\mathcal{C},f} \cup \{F(t)\} \in \mathcal{C}$, ya que $\mathcal{C}$ es propiedad de consistencia alternativa. Por lo tanto, $S_{S,\mathcal{C},f} \cup \{F(t)\} \in \mathcal{C}$. También se tiene que para cualquier $S' \in \mathcal{C}$, si $S_{S,\mathcal{C},f} \subseteq S'$ entonces $S_{S,\mathcal{C},f} = S'$, puesto que $S_{S,\mathcal{C},f}$ es un elemento maximal de $\mathcal{C}$.

Así, y puesto que $S_{S,\mathcal{C},f} \subseteq S_{S,\mathcal{C},f} \cup \{F(t)\}$ tenemos que, $S_{S,\mathcal{C},f} = S_{S,\mathcal{C},f} \cup \{F(t)\}$. Luego $F(t) \in S_{S,\mathcal{C},f}$.

La propiedad (12) se demuestra en forma análoga a la anterior.

Para demostrar las últimas dos propiedades que definen un conjunto de Hintikka se utilizan las hipótesis $(hip2)$, $(hip3)$, $(hip5)$:

13. Supongamos que $\exists x F(x) \in S_{S,\mathcal{C},f}$. Hay que demostrar que $F(t) \in S_{S,\mathcal{C},f}$, para algún término cerrado $t \in \mathbf{L}$.

Por la hipótesis $(hip3a)$, existe n tal que $\exists x F(x) = f_n$. Para demostrar que $F(t) \in S_{S,\mathcal{C},f}$ para algún término cerrado $t \in L$, primero probamos la siguiente igualdad, $S^{n+1}_{S,\mathcal{C},f} = S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\} \cup \{F(k)\}$, en donde k es algún símbolo de constante que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\}$.

De la hipótesis $(hip1)$ se tiene que, $S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\} \in \mathcal{C}$: en efecto, puesto que estamos suponiendo que $\exists x F(x) \in S_{S,\mathcal{C},f}$, entonces por la hipótesis $(hip5)$ tenemos que $\exists x F(x) \in S_{S,\mathcal{C},f} = (\bigcup_i S^i_{S,\mathcal{C},f}) \in \mathcal{C}$.

Por lo tanto, $S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\} \subseteq (\bigcup i.\, S^i_{S,\mathcal{C},f}) \in \mathcal{C}$. Así, $S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\} \in \mathcal{C}$, ya que $\mathcal{C}$ es cerrada por subconjuntos.

Por otro lado, por la hipótesis $(hip3a)$ se tiene que $f_n = \exists x F(x)$. Por lo tanto, por la definición de la sucesión $S^n_{S,\mathcal{C},f}$ tenemos que, $S^{n+1}_{S,\mathcal{C},f} = S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\} \cup \{F(k)\}$, en donde k es algún símbolo de constante que no ocurre en ninguna de las fórmulas del conjunto $S^n_{S,\mathcal{C},f} \cup \{\exists x F(x)\}$.

De la igualdad anterior tenemos que $F(k) \in S^{n+1}_{S,\mathcal{C},f}$ y por lo tanto, por definición, de la unión de una familia de conjuntos, tenemos que, $F(k) \in \bigcup_i S^i_{S,\mathcal{C},f}$, es decir, $F(k) \in S_{S,\mathcal{C},f}$.

Así, $F(t) \in S_{S,\mathcal{C},f}$, en donde t es el símbolo de constante (término cerrado) k.

La propiedad (14) se demuestra en forma similar a la anterior.

Los siguientes lemas corresponden a la formalización de cada una de las partes del teorema anterior.

Obsérvese que en la demostración de las propiedades (1) a (12) donde se hizo uso de las hipótesis $(hip4)$ y $(hip5)$ no se utilizó la definición de $S_{S,\mathcal{C},f}$. Esto significa que estas propiedades se cumplen para cualquier conjunto maximal M elemento de $\mathcal{C}$. Sin embargo, para la demostración de las propiedades (13) y (14) fue fundamental la definición del conjunto $S_{S,\mathcal{C},f}$. Lo anterior aparece reflejado en la siguiente formalización del respectivo teorema.

lemma *exten-hintikka1*:
 assumes *hip1*: *alt-consistencia C*
 and *hip2*: $M \in C$
 shows $\forall P\ ts.\ \neg\ (Atom\ P\ ts \in M \wedge (\neg.Atom\ P\ ts) \in M)$
proof −
 show *?thesis* **using** *hip1 hip2* **by**(*unfold alt-consistencia-def*) *simp*
qed

lemma *exten-hintikka2*:
 assumes *hip1*: *alt-consistencia C*
 and *hip2*: $M \in C$
 shows $FF' \notin M$
proof −
 show *?thesis* **using** *hip1 hip2* **by**(*unfold alt-consistencia-def*) *simp*
qed

lemma *exten-hintikka3*:
 assumes *hip1*: *alt-consistencia C*
 and *hip2*: $M \in C$
 shows $(\neg.TT') \notin M$
proof −
 show *?thesis* **using** *hip1 hip2* **by**(*unfold alt-consistencia-def*) *simp*

qed

lemma *exten-hintikka4*:
assumes *hip1*: *alt-consistencia C*
and *hip2*: *maximal M C*
and *hip3*: $M \in C$
shows $\forall F.\ (\neg.\neg.F) \in M \longrightarrow F \in M$
proof(*rule allI impI*)+
fix *F*
assume *h1*: $(\neg.\neg.F) \in M$
show $F \in M$
proof −
have $(\neg.\neg.F) \in M \longrightarrow M \cup \{F\} \in C$
using *hip1 hip3* **by**(*unfold alt-consistencia-def*) *simp*
hence $M \cup \{F\} \in C$ **using** *h1* **by** *simp*
moreover
have $\forall M' \in C.\ M \subseteq M' \longrightarrow M = M'$ **using** *hip2* **by**(*unfold maximal-def*)
moreover
have $M \subseteq M \cup \{F\}$ **by** *auto*
ultimately
have $M = M \cup \{F\}$ **by** *auto*
thus $F \in M$ **by** *auto*
qed
qed

lemma *exten-hintikka5*:
assumes *hip1*: *alt-consistencia C*
and *hip2*: *maximal M C*
and *hip3*: $M \in C$
shows $\forall F\ G.\ (F \wedge.\ G) \in M \longrightarrow F \in M \wedge G \in M$
proof(*rule allI impI*)+
fix *F G*
assume *h1*: $(F \wedge.\ G) \in M$
show $F \in M \wedge G \in M$
proof −
have $(F \wedge.\ G) \in M \longrightarrow M \cup \{F, G\} \in C$
using *hip1 hip3* **by**(*unfold alt-consistencia-def*) *simp*
hence $M \cup \{F, G\} \in C$ **using** *h1* **by** *simp*
moreover
have $\forall M' \in C.\ M \subseteq M' \longrightarrow M = M'$ **using** *hip2* **by**(*unfold maximal-def*)
moreover
have $M \subseteq M \cup \{F, G\}$ **by** *auto*
ultimately

```
  have M = M ∪ {F, G}  by simp
  thus F ∈ M ∧ G ∈ M by auto
 qed
qed

lemma exten-hintikka6:
 assumes hip1: alt-consistencia C
 and hip2: maximal M C
 and hip3: M ∈ C
 shows ∀ F G. (¬.(F ∨. G)) ∈ M ⟶ (¬.F) ∈ M ∧ (¬.G) ∈ M
proof(rule allI impI)+
 fix F G
 assume h1: (¬.(F ∨. G)) ∈ M
 show (¬.F) ∈ M ∧ (¬.G) ∈ M
 proof −
  have (¬.(F ∨. G)) ∈ M ⟶ M ∪ {¬.F, ¬.G} ∈ C
   using hip1 hip3 by(unfold alt-consistencia-def ) simp
  hence M ∪ {¬.F, ¬.G} ∈ C using h1 by simp
  moreover
  have ∀ M′∈C. M ⊆ M′ ⟶ M = M′ using hip2 by(unfold maximal-def )
  moreover
  have  M ⊆ M ∪ {¬.F, ¬.G}  by auto
  ultimately
  have M = M ∪ {¬.F, ¬.G}  by simp
  thus (¬.F) ∈ M ∧ (¬.G) ∈ M by auto
 qed
qed

lemma exten-hintikka7:
 assumes hip1: alt-consistencia C
 and hip2: maximal M C
 and hip3: M ∈ C
 shows ∀ F G. (F ∨. G) ∈ M ⟶ F ∈ M ∨ G ∈ M
proof(rule allI impI)+
 fix F G
 assume
 h1: (F ∨. G) ∈ M
 show F ∈ M ∨ G ∈ M
 proof −
  have (F ∨. G) ∈ M ⟶ M ∪ {F} ∈ C ∨ M ∪ {G} ∈ C
   using hip1 hip3 by(unfold alt-consistencia-def ) simp
  hence  M ∪ {F} ∈ C ∨ M ∪ {G} ∈ C using h1 by simp
  thus ?thesis
```

```
   proof(rule disjE)
    assume M ∪ {F} ∈ C
    moreover
    have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
    have M ⊆ M ∪ {F} by auto
    ultimately
    have M = M ∪ {F} by simp
    hence F ∈ M by auto
    thus F ∈ M ∨ G ∈ M by simp
   next
    assume M ∪ {G} ∈ C
    moreover
    have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
    have M ⊆ M ∪ {G} by auto
    ultimately
    have M = M ∪ {G} by simp
    hence G ∈ M by auto
    thus F ∈ M ∨ G ∈ M by simp
   qed
  qed
qed

lemma exten-hintikka8:
  assumes hip1: alt-consistencia C
  and hip2: maximal M C
  and hip3: M ∈ C
  shows ∀ F G. (¬.(F ∧. G)) ∈ M ⟶ (¬.F) ∈ M ∨ (¬.G) ∈ M
proof(rule allI impI)+
  fix F G
  assume h1: (¬.(F ∧. G)) ∈ M
  show (¬.F) ∈ M ∨ (¬.G) ∈ M
  proof −
   have (¬.(F ∧. G)) ∈ M ⟶ M ∪ {¬.F} ∈ C ∨ M ∪ {¬.G} ∈ C
    using hip1 hip3 by(unfold alt-consistencia-def) simp
   hence M ∪ {¬.F} ∈ C ∨ M ∪ {¬.G} ∈ C using h1 by simp
   thus ?thesis
   proof(rule disjE)
    assume M ∪ {¬.F} ∈ C
    moreover
    have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
```

```
    have M ⊆ M ∪ {¬.F} by auto
    ultimately
    have M = M ∪ {¬.F} by simp
    hence (¬.F) ∈ M by auto
    thus (¬.F) ∈ M ∨ (¬.G) ∈ M by simp
   next
    assume M ∪ {¬.G} ∈ C
    moreover
    have ∀M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
    have M ⊆ M ∪ {¬.G} by auto
    ultimately
    have M = M ∪ {¬.G} by simp
    hence (¬.G) ∈ M by auto
    thus (¬.F) ∈ M ∨ (¬.G) ∈ M by simp
   qed
  qed
qed

lemma exten-hintikka9:
  assumes hip1: alt-consistencia C
  and hip2: maximal M C
  and hip3: M ∈ C
  shows ∀F G. (F →. G) ∈ M ⟶ (¬.F) ∈ M ∨ G ∈ M
proof(rule allI impI)+
  fix F G
  assume h1: (F →. G) ∈ M
  show (¬.F) ∈ M ∨ G ∈ M
  proof −
   have (F →. G) ∈ M ⟶ M ∪ {¬.F} ∈ C ∨ M ∪ {G} ∈ C
    using hip1 hip3 by(unfold alt-consistencia-def) simp
   hence M ∪ {¬.F} ∈ C ∨ M ∪ {G} ∈ C using h1 by simp
   thus ?thesis
   proof(rule disjE)
    assume M ∪ {¬.F} ∈ C
    moreover
    have ∀M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
    have M ⊆ M ∪ {¬.F} by auto
    ultimately
    have M = M ∪ {¬.F} by simp
    hence (¬.F) ∈ M by auto
    thus (¬.F) ∈ M ∨ G ∈ M by simp
```

```
   next
    assume M ∪ {G} ∈ C
    moreover
    have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
    moreover
    have M ⊆ M ∪ {G} by auto
    ultimately
    have M = M ∪ {G} by simp
    hence G ∈ M by auto
    thus (¬.F) ∈ M ∨ G ∈ M by simp
   qed
  qed
qed

lemma exten-hintikka10:
  assumes hip1: alt-consistencia C
  and hip2: maximal M C
  and hip3: M ∈ C
  shows ∀ F G. (¬.(F →. G)) ∈ M ⟶ F ∈ M ∧ (¬.G) ∈ M
proof(rule allI impI)+
  fix F G
  assume h1: (¬.(F →. G)) ∈ M
  show F ∈ M ∧ (¬.G) ∈ M
  proof −
   have (¬.(F  →. G)) ∈ M ⟶ M ∪ {F, ¬.G} ∈ C
    using hip1 hip3 by(unfold alt-consistencia-def) simp
   hence M ∪ {F, ¬.G} ∈ C using h1 by simp
   moreover
   have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
   moreover
   have  M ⊆ M ∪ {F, ¬.G}  by auto
   ultimately
   have M = M ∪ {F, ¬.G}  by simp
   thus F ∈ M ∧ (¬.G) ∈ M by auto
  qed
qed

lemma exten-hintikka11:
  assumes hip1: alt-consistencia C
  and hip2: maximal M C
  and hip3: M ∈ C
  shows ∀ F t. cerradot 0 t ⟶ (∀ .F) ∈ M ⟶ F[t\0] ∈ M
proof(rule allI impI)+
```

```
 fix F
 fix t:: 'a term
 assume h1: cerradot 0 t and h2: (∀ .F) ∈ M
 show F[t\0] ∈ M
 proof −
  have cerradot 0 t ⟶ (∀ .F) ∈ M ⟶ M ∪ {F[t\0]} ∈ C
   using hip1 hip3 by(unfold alt-consistencia-def) simp
  hence M ∪ {F[t\0]} ∈ C using h1 h2 by simp
  moreover
  have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
  moreover
  have M ⊆ M ∪ {F[t\0]} by auto
  ultimately
  have M = M ∪ {F[t\0]} by simp
  thus F[t\0] ∈ M by auto
 qed
qed

lemma exten-hintikka12:
 assumes hip1: alt-consistencia C
 and hip2: maximal M C
 and hip3: M ∈ C
 shows ∀ F t. cerradot 0 t ⟶ (¬.(∃ .F)) ∈ M ⟶ (¬.F[t\0]) ∈ M
proof(rule allI impI)+
 fix F
 fix t:: 'a term
 assume h1: cerradot 0 t and h2: (¬.(∃ .F)) ∈ M
 show (¬.F[t\0]) ∈ M
 proof −
  have cerradot 0 t ⟶ (¬.(∃ .F)) ∈ M ⟶ M ∪ {¬.F[t\0]} ∈ C
   using hip1 hip3 by(unfold alt-consistencia-def) simp
  hence M ∪ {¬.F[t\0]} ∈ C using h1 h2 by simp
  moreover
  have ∀ M'∈C. M ⊆ M' ⟶ M = M' using hip2 by(unfold maximal-def)
  moreover
  have M ⊆ M ∪ {¬.F[t\0]} by auto
  ultimately
  have M = M ∪ {¬.F[t\0]} by simp
  thus (¬.F[t\0]) ∈ M by auto
 qed
qed

lemma exten-hintikka13a:
```

```
 assumes hip1: subconj-cerrada C
 and hip2: (∃ .F) = f n
 and hip3: (∃ .F) ∈ Msuc S C f
 and hip4: Msuc S C f ∈ C
 shows suc S C f (Suc n) =
      suc S C f n ∪ {∃ .F} ∪
      {F[Term (SOME k. k ∉ (⋃ G ∈ suc S C f n ∪ {F}. simbfun G)) []\0]}
proof −
 have suc S C f n ∪ {∃ .F} ∈ C
 proof −
  have (∃ .F) ∈ (⋃ i. suc S C f i) using hip3 by(simp only: Msuc-def)
  hence suc S C f n ∪ {∃ .F} ⊆ (⋃ i. suc S C f i) by auto
  moreover
  have (⋃ i. suc S C f i) ∈ C using hip4 by(simp only: Msuc-def)
  ultimately
  show ?thesis using hip1 by(simp add: subconj-cerrada-def)
 qed
 moreover
 have f n = (∃ .F) using hip2 by simp
 ultimately
 show ?thesis by (simp add: suc-def)
qed

lemma exten-hintikka13:
 assumes hip1: subconj-cerrada C
 and hip2: ∀ F. ∃ n. (∃ .F) = f n
 and hip3: Msuc S C f ∈ C
 shows ∀ F. (∃ .F) ∈ Msuc S C f ⟶
      (∃ t. cerradot 0 t ∧ F[t\0] ∈ Msuc S C f)
proof(rule allI impI)+
 fix F
 assume 1: (∃ .F) ∈ Msuc S C f
 show ∃ t. cerradot 0 t ∧ F[t\0] ∈ Msuc S C f
 proof −
  have ∃ n. (∃ .F) = f n using hip2 by simp
  then obtain n where n: (∃ .F) = f n  by (rule exE)
  have
  cerradot 0 (Term (SOME k. k ∉ (⋃ G ∈ suc S C f n ∪ {∃ .F}. simbfun G)) []) ∧
  F[Term (SOME k. k ∉ (⋃ G ∈ suc S C f n ∪ {∃ .F}. simbfun G)) []\0] ∈
    (⋃ n. suc S C f n)
  proof(rule conjI)
   show cerradot 0 (Term (SOME k. k ∉ (⋃ G ∈ suc S C f n ∪ {∃ .F}. simbfun G)) [])
    by simp
```

```
  next
   show
    F[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {∃.F}. simbfun G)) []\0] ∈
    (⋃n. suc S C f n)
   proof(rule-tac a=Suc n in UN-I[OF UNIV-I] )
    show
    F[Term (SOME k. k ∉(⋃G ∈ suc S C f n ∪ {∃.F}. simbfun G))[]\0]∈
     suc S C f (Suc n)
    proof −
     have suc S C f (Suc n) = suc S C f n ∪ {(∃.F)} ∪
   {F[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(∃.F)}. simbfun G)) []\0]}
       using exten-hintikka13a[OF hip1 n 1 hip3] by auto
     thus ?thesis  by simp
    qed
   qed
  qed
  thus ?thesis by (simp only: Msuc-def) blast
 qed
qed

lemma exten-hintikka14a:
 assumes hip1: subconj-cerrada C
 and hip2: (¬.(∀.F)) = f n
 and hip3: (¬.(∀.F)) ∈ Msuc S C f
 and hip4: Msuc S C f ∈ C
 shows
 suc S C f (Suc n) = suc S C f n ∪ {¬.(∀.F)} ∪
 {(¬.F)[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G)) []\0]}
proof −
 have suc S C f n ∪ {(¬.(∀.F))} ∈ C
 proof −
  have (¬.(∀.F)) ∈ (⋃i. suc S C f i) using hip3 by(simp only: Msuc-def)
  hence suc S C f n ∪ {(¬.(∀.F))} ⊆ (⋃i. suc S C f i) by auto
  moreover
  have (⋃i. suc S C f i) ∈ C using hip4 by(simp only: Msuc-def)
  ultimately
  show ?thesis using hip1 by(simp add: subconj-cerrada-def)
 qed
 moreover
 have f n = (¬.(∀.F)) using hip2 by simp
 ultimately
 show ?thesis by auto
qed
```

```
lemma exten-hintikka14:
 assumes hip1: subconj-cerrada C
 and hip2: ∀F.∃n. (¬.(∀.F)) = f n
 and hip3: Msuc S C f ∈ C
 shows ∀F. (¬.(∀.F)) ∈ Msuc S C f ⟶
        (∃t. cerradot 0 t ∧ (¬.F[t\0]) ∈ Msuc S C f)
proof(rule allI impI)+
 fix F
 assume 1: (¬.(∀.F)) ∈ Msuc S C f
 show ∃t. cerradot 0 t ∧ (¬.F[t\0]) ∈ Msuc S C f
 proof −
  have ∃n. (¬.(∀.F)) = f n
   using hip2 by simp
  then obtain n where n: (¬.(∀.F)) = f n  by (rule exE)
  have
  cerradot 0 (Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G)) []) ∧
  (¬.F[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G)) []\0])∈
  (⋃n. suc S C f n)
  proof(rule conjI)
   show cerradot 0 (Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G)) []) by simp
   next
   show
   (¬.F[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G))
    []\0])∈(⋃n. suc S C f n)
   proof(rule-tac a=Suc n in UN-I[OF UNIV-I] )
    show
    (¬.F[Term (SOME k. k ∉(⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G))[]\0]) ∈
     suc S C f (Suc n)
     proof −
      have suc S C f (Suc n) =  suc S C f n ∪ {(¬.(∀.F))} ∪
          {¬.F[Term (SOME k. k ∉ (⋃G ∈ suc S C f n ∪ {(¬.(∀.F))}. simbfun G)) []\0]}
        using exten-hintikka14a[OF hip1 n 1 hip3] by auto
       thus ?thesis by auto
     qed
    qed
  qed
  thus ?thesis by (simp only: Msuc-def) blast
 qed
qed
```

La formalización de la prueba del teorema 9.6.1 es la siguiente.

theorem *MaximalHintikka*:
assumes *hip1*: *alt-consistencia C*
and *hip2*: *subconj-cerrada C*
and *hip3a*: $\forall F.\exists n.\ (\exists .F) = f\ n$
and *hip3b*: $\forall F.\exists n.\ (\neg.(\forall .F)) = f\ n$
and *hip4*: *maximal* $(Msuc\ S\ C\ f)\ C$
and *hip5*: $Msuc\ S\ C\ f \in C$
shows *hintikka* $(Msuc\ S\ C\ f)$
proof(*unfold hintikka-def*)
show $(\forall p\ ts.\ \neg\ (Atom\ p\ ts \in Msuc\ S\ C\ f \wedge (\neg.Atom\ p\ ts) \in Msuc\ S\ C\ f)) \wedge$
$FF' \notin Msuc\ S\ C\ f \wedge$
$(\neg.TT') \notin Msuc\ S\ C\ f \wedge$
$(\forall Z.\ (\neg.\neg.Z) \in Msuc\ S\ C\ f \longrightarrow Z \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (F \wedge.\ G) \in Msuc\ S\ C\ f \longrightarrow F \in Msuc\ S\ C\ f \wedge G \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (\neg.(F \vee.\ G)) \in Msuc\ S\ C\ f \longrightarrow$
$(\neg.F) \in Msuc\ S\ C\ f \wedge (\neg.G) \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (F \vee.\ G) \in Msuc\ S\ C\ f \longrightarrow F \in Msuc\ S\ C\ f \vee G \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (\neg.(F \wedge.\ G)) \in Msuc\ S\ C\ f \longrightarrow$
$(\neg.F) \in Msuc\ S\ C\ f \vee (\neg.G) \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (F \rightarrow.\ G) \in Msuc\ S\ C\ f \longrightarrow (\neg.F) \in Msuc\ S\ C\ f \vee G \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ G.\ (\neg.(F \rightarrow.\ G)) \in Msuc\ S\ C\ f \longrightarrow$
$F \in Msuc\ S\ C\ f \wedge (\neg.G) \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\forall .F) \in Msuc\ S\ C\ f \longrightarrow F[t\backslash 0] \in Msuc\ S\ C\ f) \wedge$
$(\forall F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists .F)) \in Msuc\ S\ C\ f \longrightarrow$
$(\neg.F[t\backslash 0]) \in Msuc\ S\ C\ f) \wedge$
$(\forall F.\ (\exists .F) \in Msuc\ S\ C\ f \longrightarrow (\exists t.\ cerradot\ 0\ t \wedge F[t\backslash 0] \in Msuc\ S\ C\ f)) \wedge$
$(\forall F.\ (\neg.(\forall .F)) \in Msuc\ S\ C\ f \longrightarrow$
$(\exists t.\ cerradot\ 0\ t \wedge (\neg.F[t\backslash 0]) \in Msuc\ S\ C\ f))$
using *exten-hintikka1*[*OF hip1 hip5*] *exten-hintikka2*[*OF hip1 hip5*]
exten-hintikka3[*OF hip1 hip5*] *exten-hintikka4*[*OF hip1 hip4 hip5*]
exten-hintikka5[*OF hip1 hip4 hip5*] *exten-hintikka6*[*OF hip1 hip4 hip5*]
exten-hintikka7[*OF hip1 hip4 hip5*] *exten-hintikka8*[*OF hip1 hip4 hip5*]
exten-hintikka9[*OF hip1 hip4 hip5*] *exten-hintikka10*[*OF hip1 hip4 hip5*]
exten-hintikka11[*OF hip1 hip4 hip5*] *exten-hintikka12*[*OF hip1 hip4 hip5*]
exten-hintikka13[*OF hip2 hip3a hip5*]
exten-hintikka14[*OF hip2 hip3b hip5*]
by *blast*
qed

9.7 Enumeración de fórmulas proposicionales

En esta sección mostramos una manera de enumerar las fórmulas de cualquier lenguaje de la lógica proposicional. Esta enumeración (definición 6.6.8) se basa en representar las fórmulas por medio de árboles binarios de números maturales. De esta forma, si se tiene una enumeración del conjunto de árboles binarios entonces se tiene una enumeración del conjunto de fórmulas del lenguaje dado.

Recordemos que una enumeración de un conjunto A es cualquier función sobreyectiva $f : \mathbb{N} \to A$.

Los siguientes lemas demuestran que, $f : \mathbb{N} \to A$ es una enumeración si existe una función g inversa por la derecha de f.

Lema 9.7.1 *Si f es una función sobreyectiva entonces, existe una función g inversa por la derecha de f.*

Su formalización es:

lemma *enum1*:
assumes $(\forall y.\exists x.\ y = (f\,x))$
shows $\exists g.\ \forall y.\ f\,(g\,y) = y$

Lema 9.7.2 *Si la función f tiene una inversa por la derecha entonces, f es sobreyectiva.*

Su formalización es:

lemma *enum2*:
assumes $\forall x.\ f\,(g\,x) = x$
shows $\forall y.\exists x.\ y = f\,x$

Así, en esta sección utilizaremos la siguiente forma equivalente de expresar el concepto de enumeración.

Lema 9.7.3 *$f : \mathbb{N} \to A$ es una enumeración si y solo si tiene una inversa por la derecha.*

Su formalización es:

lemma *enumeracion*: *enumeracion f* $= (\exists g.\ \forall y.\ f\,(g\,y) = y)$
using *enum1 enum2*
by(*unfold enumeracion-def*) *blast*

9.7.1 Enumeración de árboles binarios

En esta sección mostramos una manera de enumerar los árboles binarios. Consideraremos formalmente el tipo de dato *árbol binario* donde sus elementos son árboles binarios cuyas hojas son números naturales.

datatype *arbolb* = *Hoja nat* | *Arbol arbolb arbolb*

La enumeración del tipo de dato árbol binario está basada en una enumeración del producto cartesiano $\mathbb{N} \times \mathbb{N}$ de los números naturales.

Enumeración del producto cartesiano $\mathbb{N} \times \mathbb{N}$

Una forma de ordenar el conjunto de los pares de números naturales es la siguiente:
$(0,0)$,
$(0,1)$, $(1,0)$,
$(0,2)$, $(1,1)$, $(2,0)$,
$(0,3)$, $(1,2)$, $(2,1)$, $(3,0)$,
....

Es decir, se ordenan por la suma de sus componentes y los que tienen la misma suma se ordenan por su primera componente. La función *diag*, que le asigna a cada número natural n el par que ocupa la posición n en la ordenación anterior, se define como sigue:

1. $diag(0) = (0,0)$
2. $diag(n+1) = \begin{cases} (0, x+1), & \text{si } diag(n) = (x,0) \\ (x+1, y), & \text{si } diag(n) = (x, y+1) \end{cases}$

Su formalización es:

```
primrec diag :: nat ⇒ (nat × nat) where
  diag 0 = (0, 0)
| diag (Suc n) =
    (let (x, y) = diag n
     in case y of
        0 ⇒ (0, Suc x)
      | Suc y ⇒ (Suc x, y))
```

Para demostrar que *diag* es sobreyectiva, definimos la siguiente función *undiag* y demostraremos que es la inversa (por la derecha) de *diag*,

1. $undiag(0,0) = 0$

2. $undiag(0, y+1) = undiag(y, 0) + 1$

3. $undiag(x+1, y) = undiag(x, y+1) + 1$

Su formalización es:

```
function undiag :: nat × nat ⇒ nat where
  undiag (0, 0) = 0
| undiag (0, Suc y) = Suc (undiag (y, 0))
| undiag (Suc x, y) = Suc (undiag (x, Suc y))
  by pat-completeness auto
termination
  by (relation measure (λ(x, y). ((x + y) * (x + y + 1)) div 2 + x)) auto
```

Nótese que la función de medida de la función *undiag* está definida justamente por la expresión que define explícitamente a la función *undiag*.

El siguiente resultado demuestra que *undiag* es la inversa por la derecha de *diag* .

```
lemma diag-undiag [simp]: diag (undiag (x, y)) = (x, y)
by (rule undiag.induct) (simp add: Let-def)+
```

De esta forma, se tiene que la función *diag* es una enumeración de $\mathbb{N} \times \mathbb{N}$. Formalmente:

```
lemma enumeracion-natxnat: enumeracion (diag::nat ⇒ (nat × nat))
proof −
  have ∀ x y. diag (undiag (x, y)) = (x, y) using diag-undiag by auto
  hence ∃ undiag. ∀ x y. diag (undiag (x, y)) = (x, y) by blast
  thus ?thesis using enumeracion[of diag] by auto
qed
```

Enumeración del tipo de dato árbol binario

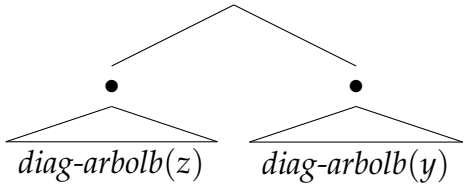
Con base a la enumeración *diag* del producto cartesiano $\mathbb{N} \times \mathbb{N}$, definimos la siguiente enumeración *diag-arbolb* : $\mathbb{N} \to$ *arbolb* del tipo de dato *arbolb*.

Dado $n \in \mathbb{N}$, supogamos que $diag(n) = (x, y)$.

1. Si $x = 0$ entonces *diag-arbolb*(n) es el árbol

$$\begin{array}{c} | \\ y \end{array}$$

2. Si $x = z + 1$ para algún $z \in \mathbb{N}$, entonces *diag-arbolb*(n) es el árbol




```
function diag-arbolb :: nat ⇒ arbolb where
diag-arbolb n = (case fst (diag n) of
  0 ⇒ Hoja (snd (diag n))
 | Suc z ⇒ Arbol (diag-arbolb z) (diag-arbolb (snd (diag n))))
by auto
```

La siguiente función *undiag-arbolb* corresponde a una inversa (por la derecha) de la función *diag-arbolb*.

```
primrec undiag-arbolb :: arbolb ⇒ nat where
 undiag-arbolb (Hoja n) = undiag (0, n)
| undiag-arbolb (Arbol t1 t2) =
  undiag (Suc (undiag-arbolb t1), undiag-arbolb t2)
```

El siguiente lema demuestra que *undiag-arbolb* es inversa por la derecha de *diag-arbolb*.

```
lemma diag-undiag-arbolb [simp]: diag-arbolb (undiag-arbolb t) = t
by (induct t) (simp-all add: Let-def)
```

Por consiguiente, la función *diag-arbolb* es una enumeración del tipo de dato *arbolb*:

```
lemma enumeracion-arbolb: enumeracion (diag-arbolb :: nat ⇒ arbolb)
proof −
 have ∀ x. diag-arbolb (undiag-arbolb x) = x
  using diag-undiag-arbolb by blast
 hence ∃ undiag-arbolb. ∀ x . diag-arbolb (undiag-arbolb x) = x by blast
 thus ?thesis using enumeracion[of diag-arbolb] by auto
qed
```

9.8 Enumeración de las fórmulas de un lenguaje de primer orden

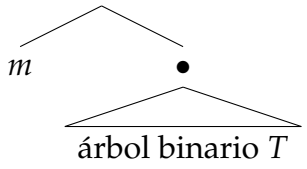
En esta sección mostramos una manera de enumerar las fórmulas de cualquier lenguaje de la lógica de primer orden. De la misma forma que en el caso proposicional (sección

9.8), esta enumeración se basa en representar las fórmulas por medio de árboles binarios de números naturales.

9.8.1 Enumeración de términos

Para enumerar los términos de un lenguaje **L** de primer orden inicialmente mostramos, dada una enumeración f de los símbolos de función, cómo asignarle un término a cada árbol binario que sea una hoja y a cada árbol cuyo árbol izquierdo este compuesto de una sola hoja.

1. Al árbol compuesto por la hoja m le corresponde la variable x_m.
2. Al árbol,

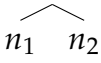


le corresponde el término $h(t_1, t_2, \ldots, t_n)$, en donde $f(m) = h$ y $(t_1, t_2, \ldots, t_n)$ es la lista de términos correspondiente al árbol T.

La correspondencia anterior se formaliza en Isabelle por medio de la siguiente función *parcial*.

```
fun term-de-arbolb :: (nat ⇒ 'a) ⇒ arbolb ⇒ 'a term and
term-list-de-arbolb :: (nat ⇒ 'a) ⇒ arbolb ⇒ 'a term list
where
  term-de-arbolb f (Hoja m) = Var m
| term-de-arbolb f (Arbol (Hoja m) T) =
   Term (f m) (term-list-de-arbolb f T)
| term-list-de-arbolb f (Hoja m) = []
| term-list-de-arbolb f (Arbol T1 T2) =
   term-de-arbolb f T1 # term-list-de-arbolb f T2
```

Por ejemplo, al árbol

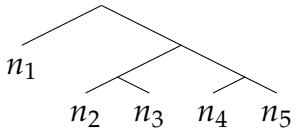


le corresponde el símbolo de constante c, donde $f(n_1) = c$.

Su formalización es:

lemma *term-de-arbolb f* (*Arbol* (*Hoja n1*) (*Hoja n2*)) = *Term* (*f n1*) [] **by** *auto*

Al árbol



le corresponde el término $g(c, x_{n_4})$, donde $f(n_1) = g$ y $f(n_2) = c$.

Su formalización es:

lemma *termino-arbol*:
term-de-arbolb f
(*Arbol* (*Hoja n1*) (*Arbol* (*Arbol* (*Hoja n2*) (*Hoja n3*))(*Arbol* (*Hoja n4*) (*Hoja n5*)))) =
Term (*f n1*) [*Term* (*f n2*) [], *Var n4*]
by *simp*

De manera recíproca, la siguiente función *arbolb-de-term* formaliza la inversa de la correspodencia anterior: dada una función f del conjunto de símbolos de función al conjunto de números naturales, a cada término le asigna un árbol binario.

primrec
arbolb-de-term :: ('*a* ⇒ *nat*) ⇒ '*a term* ⇒ *arbolb* **and**
arbolb-de-term-list :: ('*a* ⇒ *nat*) ⇒ '*a term list* ⇒ *arbolb*
where
arbolb-de-term g (*Var m*) = *Hoja m*
| *arbolb-de-term g* (*Term m ts*) = *Arbol* (*Hoja* (*g m*)) (*arbolb-de-term-list g ts*)
| *arbolb-de-term-list g* [] = *Hoja 0*
| *arbolb-de-term-list g* (*t* # *ts*) =
Arbol (*arbolb-de-term g t*) (*arbolb-de-term-list g ts*)

El siguiente lema demuestra que si f es una enumeración de los símbolos de función del lenguaje **L**, entonces la función *arbolb-de-term* es inversa por derecha de *term-de-arbolb*.

lemma *term-arbolb*:
assumes ∀ *x*. *f* (*g x*) = *x*
shows *term-de-arbolb f* (*arbolb-de-term g t*) = *t*
and *term-list-de-arbolb f* (*arbolb-de-term-list g ts*) = *ts*
using *assms*
by (*induct t* **and** *ts*) *auto*

Con base a lo anterior y usando la enumeración *diag-arbolb* del conjunto de árboles binarios (sección 9.7.1), por cada enumeración f del conjunto de símbolos de función de un lenguaje de primer orden construimos una enumeración *diag-term f* del correspondiente conjunto de términos:

```
definition diag-term :: (nat ⇒ 'a) ⇒ nat ⇒ 'a term where
  diag-term f n = term-de-arbolb f (diag-arbolb n)
```

La siguiente función *undiag-term* corresponde a una inversa (por derecha) de la función *diag-arbolb*.

```
definition undiag-term :: ('a ⇒ nat) ⇒ 'a term ⇒ nat where
  undiag-term f t = undiag-arbolb (arbolb-de-term f t)
```

El siguiente lema formaliza que *undiag-term* es inversa por derecha de *diag-term*.

```
lemma diag-undiag-term [simp]:
  assumes ∀ x. f (g x) = x
  shows diag-term f (undiag-term g t) = t
using assms
by(simp add: diag-term-def undiag-term-def term-arbolb)
```

Por consiguiente, si f es una enumeración del conjunto de símbolos de función, entonces la función *diag-term f* es una enumeración del correspondiente conjunto de términos:

```
lemma enumeracion-terminos:
  fixes f :: nat ⇒ 'a
  assumes enumeracion f
  shows enumeracion (diag-term f :: nat ⇒ 'a term)
proof −
  have ∃ g. ∀ y. f (g y) = y using assms enumeracion by auto
  then obtain g where  ∀ y. f (g y) = y by auto
  hence ∀ t. diag-term f (undiag-term g t) = t
    using  diag-undiag-term by auto
  hence ∃ undiag-term. ∀ t . diag-term f (undiag-term t) = t by blast
  thus ?thesis using enumeracion[of diag-term f] by auto
qed
```

Enumeración de listas

Siguiendo el mismo procedimiento que hemos usado para enumerar los términos que se pueden formar a partir de un conjunto enumerable de símbolos de función, se enumera el tipo de dato lista, asumiendo que el tipo de dato de los elementos de las listas es enumerable.

Las siguientes dos funciones formalizan una correspondencia entre árboles y listas.

```
fun list-de-arbolb :: (nat ⇒ 'd) ⇒ arbolb ⇒ 'd list where
  list-de-arbolb f (Hoja x) = []
| list-de-arbolb f (Arbol (Hoja n) t) = f n # list-de-arbolb f t
```

primrec *arbolb-de-list* :: $('d \Rightarrow nat) \Rightarrow 'd\ list \Rightarrow arbolb$ **where**
arbolb-de-list f [] = *Hoja 0*
| *arbolb-de-list f* (*x* # *xs*) = *Arbol* (*Hoja* (*f x*)) (*arbolb-de-list f xs*)

Usando la enumeración *diag-arbolb* del tipo de dato árboles binarios, por cada enumeración f del tipo de dato $'d$, definimos una enumeración *diag-list f* del tipo de dato $'d$ *list*.

definition *diag-list* :: $(nat \Rightarrow 'd) \Rightarrow nat \Rightarrow 'd\ list$ **where**
diag-list f n = *list-de-arbolb f* (*diag-arbolb n*)

Su inversa por derecha está definida por:

definition *undiag-list* :: $('d \Rightarrow nat) \Rightarrow 'd\ list \Rightarrow nat$ **where**
undiag-list g xs = *undiag-arbolb* (*arbolb-de-list g xs*)

El siguiente lema formaliza que *undiag-lista* es inversa por derecha de *diag-list*.

lemma *diag-undiag-list* [*simp*]:
assumes $\forall x. f(g\ x) = x$
shows *diag-list f* (*undiag-list g xs*) = *xs*

El siguiente lema formaliza que si f es una enumeración del tipo de dato $'d$ entonces la funcion *diag-list f* es una enumeracion del tipo de dato $'d$ *list*.

lemma *enumeracion-listas*:
fixes *f*:: $nat \Rightarrow 'd$
assumes *enumeracion f*
shows *enumeracion* (*diag-list f* :: $nat \Rightarrow 'd\ list$)

Obsérvese que en el caso particular en el que $'d$ es el tipo de dato $'a\ term$, tenemos que si f es una enumeración de $'a$ entonces *diag-term f* :: $nat \Rightarrow 'a\ term$ es una enumeración de $'a\ term$, y así *diag-list* (*diag-term f*) :: $nat \Rightarrow 'a\ term\ list$ es una enumeración del tipo de dato $'a\ term\ list$, su formalización es:

corollary *enumeracion-listas1*:
fixes *f*:: $nat \Rightarrow 'a$
assumes *enumeracion f*
shows *enumeracion* (*diag-list* (*diag-term f*) :: $nat \Rightarrow 'a\ term\ list$)
proof −
have *enumeracion* (*diag-term f* :: $nat \Rightarrow 'a\ term$)
using *assms enumeracion-terminos* **by** *auto*
thus *?thesis* **using** *enumeracion-listas* **by** *auto*
qed

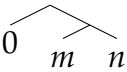
Con base a esta enumeración, en la próxima sección construimos una enumeración del conjunto de fórmulas de un lenguaje de primer orden.

9.8.2 Enumeración de fórmulas

Para enumerar las fórmulas de un lenguaje **L** de primer orden inicialmente mostramos, para cada enumeración f de los símbolos de función y cada enumeración g de los símbolos de relación, cómo asignarle una fórmula a cada árbol binario que sea una hoja con valor 0 o 1 y en forma recursiva a cada árbol cuyo árbol izquierdo esté compuesto de una sola hoja con valor entre 0 y 6, del cual dependerá el tipo de fórmula que se le asocie al árbol correspondiente:

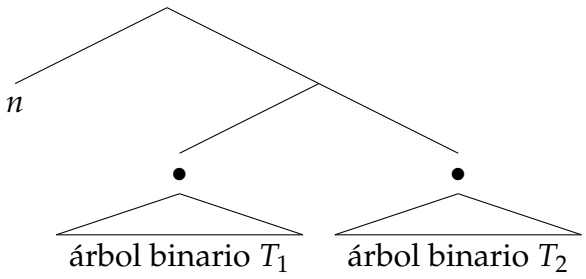
1. (a) Al árbol $\underset{0}{|}$ le corresponde la fórmula $\bot$.

   (b) Al árbol $\underset{1}{|}$ le corresponde la fórmula $\top$.

   (c) Al árbol 

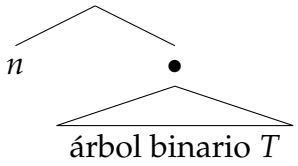
   le corresponde la fórmula atómica $P(t_1, t_2, \ldots, t_n)$, donde $g(m) = P$ y *diag-list* $(\textit{diag-term}\, f)\ n = (t_1, t_2, \ldots, t_n)$.

2. (a) Al árbol,



   le corresponde una de las fórmulas $F \wedge G$, $F \vee G$, $F \rightarrow G$ dependiendo si $n = 1$, $n = 2$ ó $n = 3$ respectivamente. En donde F, G son las fórmulas correspondientes a los árboles T_1, T_2.

   (b) Al árbol,



   le corresponde una de las fórmulas $\neg F, \forall x F, \exists x F$ dependiendo si $n = 4$, $n = 5$ ó $n = 6$ respectivamente. Donde F es la fórmula correspondiente al árbol T.

La siguiente función *parcial* formaliza la correspondencia anterior.

fun *form-de-arbolb* :: $(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'b) \Rightarrow arbolb \Rightarrow ('a, 'b)\ form$

where
form-de-arbolb f g (Hoja 0) = FF′
| *form-de-arbolb f g (Hoja (Suc 0)) = TT′*
| *form-de-arbolb f g (Arbol (Hoja 0) (Arbol (Hoja m) (Hoja n))) =*
Atom (g m) (diag-list (diag-term f) n)
| *form-de-arbolb f g (Arbol (Hoja (Suc 0)) (Arbol T1 T2)) =*
((form-de-arbolb f g T1) ∧. (form-de-arbolb f g T2))
| *form-de-arbolb f g (Arbol (Hoja (Suc (Suc 0))) (Arbol T1 T2)) =*
((form-de-arbolb f g T1) ∨. (form-de-arbolb f g T2))
| *form-de-arbolb f g (Arbol (Hoja (Suc (Suc (Suc 0)))) (Arbol T1 T2)) =*
((form-de-arbolb f g T1) →. (form-de-arbolb f g T2))
| *form-de-arbolb f g (Arbol (Hoja (Suc (Suc (Suc (Suc 0))))) T) =*
(¬. (form-de-arbolb f g T))
| *form-de-arbolb f g (Arbol (Hoja (Suc (Suc (Suc (Suc (Suc 0)))))) T) =*
(∀ .(form-de-arbolb f g T))
| *form-de-arbolb f g (Arbol (Hoja (Suc (Suc (Suc (Suc (Suc (Suc 0))))))) T) =*
(∃ .(form-de-arbolb f g T))

De manera recíproca, la siguiente función formaliza la inversa de la correspodencia anterior: dadas las funciónes f del conjunto de símbolos de función al conjunto de números naturales y g del conjunto de símbolos de relación al conjunto de números naturales, a cada fórmula se le asigna un árbol binario.

primrec *arbolb-de-form* :: *('a ⇒ nat) ⇒ ('b ⇒ nat) ⇒ ('a, 'b) form ⇒ arbolb*
where
arbolb-de-form f g FF′ = Hoja 0
| *arbolb-de-form f g TT′ = Hoja (Suc 0)*
| *arbolb-de-form f g (Atom P ts) = Arbol (Hoja 0)*
(Arbol (Hoja (g P)) (Hoja (undiag-list (undiag-term f) ts)))
| *arbolb-de-form f g (F ∧. G) = Arbol (Hoja (Suc 0))*
(Arbol (arbolb-de-form f g F) (arbolb-de-form f g G))
| *arbolb-de-form f g (F ∨. G) = Arbol (Hoja (Suc (Suc 0)))*
(Arbol (arbolb-de-form f g F) (arbolb-de-form f g G))
| *arbolb-de-form f g (F →. G) = Arbol (Hoja (Suc (Suc (Suc 0))))*
(Arbol (arbolb-de-form f g F) (arbolb-de-form f g G))
| *arbolb-de-form f g (¬. F) = Arbol (Hoja (Suc (Suc (Suc (Suc 0)))))*
(arbolb-de-form f g F)
| *arbolb-de-form f g (∀ . F) = Arbol (Hoja (Suc (Suc (Suc (Suc (Suc 0))))))*
(arbolb-de-form f g F)
| *arbolb-de-form f g (∃ . F) = Arbol*
(Hoja (Suc (Suc (Suc (Suc (Suc (Suc 0)))))))
(arbolb-de-form f g F)

Con base a lo anterior y usando la enumeración *diag-arbolb* del conjunto de árboles

binarios, en cualquier lenguaje de primer orden, por cada enumeración f del conjunto de símbolos de función y cada enumeración g del conjuntode símbolos de relación, construimos una enumeración $\Delta_{f,g}$ del correspondiente conjunto de fórmulas $\mathcal{F}$:

Definición 9.8.1 *La función $\Delta_{f,g} : \mathbb{N} \to \mathcal{F}$ está definida por,*

$$\Delta_{f,g}(n) = (form\text{-}de\text{-}arbolb\ f\ g)(diag\text{-}arbolb\ n).$$

Teorema 9.8.2 *Sea $\mathbf{L}(\mathbf{R}, \mathbf{F})$ un lenguaje de primer orden. Sean $f : \mathbb{N} \to \mathbf{F}$ una enumeración de los símbolos de función y $g : \mathbb{N} \to \mathbf{R}$ una enumeración de los símbolos de relación. Entonces, $\Delta_{f,g}$ es una enumeración del conjunto de fórmulas de $\mathbf{L}$.*

Demostración: Por hipótesis y el lema 9.7.3, existen funciones $f' : \mathbf{F} \to \mathbb{N}$, $g' : \mathbf{R} \to \mathbb{N}$ inversas a derecha de f y g repectivamente. Entonces, la función $\Delta'_{f',g'} : \mathcal{F} \to \mathbb{N}$ definida por,

$$\Delta'_{f',g'}(F) = undiag\text{-}arbolb\ (arbolb\text{-}de\text{-}form\ f'\ g'\ F),$$

es inversa por derecha de $\Delta_{f,g}$,. Es decir, $\Delta_{f,g}(\Delta'_{f',g'}(F)) = F$ para toda fórmula F.

□

Para la formalización del teorema anterior, definimos en Isabelle la funcion $\Delta_{f,g}$ y su inversa $\Delta'_{f,g}$ de la siguiente manera.

```
definition Δ :: (nat ⇒ 'a) ⇒ (nat ⇒ 'b) ⇒ nat ⇒ ('a, 'b) form where
 Δ f g n = form-de-arbolb f g (diag-arbolb n)

definition Δ' :: ('a ⇒ nat) ⇒ ('b ⇒ nat) ⇒ ('a, 'b) form ⇒ nat where
 Δ' f g F = undiag-arbolb (arbolb-de-form f g F)
```

La formalización del teorema anterior es la siguiente.

```
theorem enumeracionformulas[simp]:
 assumes (∀ x. f(f' x) = x) and (∀ x. g(g' x) = x)
 shows Δ f g (Δ' f' g' F) = F
using assms
by (induct F)(simp-all add: Δ-def Δ'-def)
```

Los siguiente dos corolarios muestran otras dos formas de expresar la anterior propiedad de enumerabilidad.

Corolario 9.8.3 *Sean $\mathbf{L}(\mathbf{R}, \mathbf{F})$ un lenguaje de primer orden, $f : \mathbb{N} \to \mathbf{F}$ tal que para todo símbolo de función y, $\exists n(y = f(n))$, y $g : \mathbb{N} \to \mathbf{R}$ tal que para todo símbolo de relación y, $\exists n\,(y = g(n))$. Entonces, $\forall y \exists n\,(y = \Delta_{f,g}(n))$.*

Demostración: Sea $f' : \mathbf{F} \to \mathbb{N}$ definida de la siguiente manera: dado $y \in \mathbf{F}$ definimos $f'(y) = n$, donde n es tal que $y = f(n)$. Sea $g' : \mathbf{R} \to \mathbb{N}$ definida de la siguiente manera: dado $y \in \mathbf{R}$ definimos $g'(y) = n$, donde n es tal que $y = g(n)$.

De las definiciones anteriores, se tiene que $f(f'(y)) = y$ para todo $y \in \mathbf{F}$ y $g(g'(y)) = y$ para todo $y \in \mathbf{R}$. Luego, por el teorema anterior, $\Delta_{f,g}(\Delta'_{f',g'}(y)) = y$ para toda formula y. De esta forma, para toda fórmula y existe n tal que, $y = \Delta_{f,g}(n)$, en donde $n = \Delta'_{f',g'}(y)$.

□

Su formalización es:

```
corollary EnumeracionFormulas:
 assumes h1: ∀ y. ∃ n. y = f n and h2: ∀ y. ∃ n. y = g n
 shows ∀ y. ∃ n. y = Δ f g n
proof(rule allI)
 fix y
 { have ∀ y. y = f (SOME n. y = (f n))
  proof(rule allI)
   fix y
   obtain n where n: y=f(n) using h1 by auto
   thus y = f (SOME n. y = (f n)) by (rule someI)
  qed}
 hence 1: ∀ y. f((λy. SOME n. y = (f n)) y) = y by simp
 { have ∀ y. y = g (SOME n. y = (g n))
  proof(rule allI)
   fix y
   obtain n where n: y=g(n) using h2 by auto
   thus y = g (SOME n. y = (g n)) by (rule someI)
  qed}
 hence 2: ∀ y. g((λy. SOME n. y = (g n)) y) = y by simp
 have y = Δ f g (Δ' (λy. SOME n. y = (f n)) (λy. SOME n. y = (g n)) y)
  using 1 and 2 and enumeracionformulas by simp
 thus ∃ n. y = Δ f g n
  by (rule-tac x= (Δ' (λy. SOME n. y = (f n)) (λy. SOME n. y = (g n)) y) in exI)
qed
```

Corolario 9.8.4 *Sean $f : \mathbb{N} \to \mathbf{F}$ y $g : \mathbb{N} \to \mathbf{R}$ enumeraciones de los símbolos de función y de relacion respectivamente de un lenguaje $\mathbf{L}$. Entonces, la función $\Delta_{f,g} : \mathbb{N} \to \mathcal{F}$ (definición 9.8.1) es una enumeración del conjunto $\mathcal{F}$ de fórmulas.*

Demostración: Es consecuecia del corolario anterior y la definición de enumeración.

□

corollary *EnumeracionFormulas1*:
fixes f:: $nat \Rightarrow 'a$ **and** g:: $nat \Rightarrow 'b$
assumes *enumeracion* f **and** *enumeracion* g
shows *enumeracion* $((\Delta\, f\, g)$:: $nat \Rightarrow ('a, 'b)\ form)$
proof −
have $\forall\, y.\ \exists\ n.\ y = f\, n$ **and** $\forall\, y.\ \exists\ n.\ y = g\, n$
using *assms* **by**(*unfold enumeracion-def*)
hence $\forall\, y.\ \exists\, n.\ y = \Delta\, f\, g\, n$ **using** *EnumeracionFormulas* **by** *blast*
thus *?thesis* **by**(*unfold enumeracion-def*)
qed

9.9 Formalización de la existencia de modelos de primer orden

En esta sección formalizamos la demostración del teorema de existencias de modelos en la lógica de primer orden. Es decir, demostramos en Isabelle que el subconjunto de *sentencias* de un conjunto *consistente* S de fórmulas de primer orden es *satisfacible*.

Teniendo en cuenta que el corolario (9.5.11) garantiza que el subconjunto de sentencias de un conjunto de Hintikka es satisfacible, la demostración del teorema está basada en que todo conjunto consistente S puede extenderse a un conjunto de Hintikka. Para la demostración de esto último se utilizará el teorema (9.6.1) que afirma que en una propiedad de consistencia $\mathcal{C}$ cerrada por subconjuntos, si el límite de la sucesión $S^n_{S,\mathcal{C},f}$ (definicion 9.4.1), $S_{S,\mathcal{C},f} = \bigcup_n S^n_{S,\mathcal{C},f}$ pertenece a $\mathcal{C}$ y es un conjunto maximal de $\mathcal{C}$ entonces es un conjunto de Hintikka. Además, el corolario (9.4.13) afirma que estas dos últimas condiciones se tienen en una propiedad de consistencia alternativa $\mathcal{C}$ de carácter finito tal que $S \in \mathcal{C}$.

De acuerdo a lo anterior, necesitamos poder extender una propiedad de consistencia a una propiedad de consistencia alternativa de carácter finito (y por lo tanto cerrada por subconjuntos). Los teoremas, (6.3.2) partes (a) y (b), (9.2.5) partes (a), (b) y (c), (6.5.1) partes (a) y (b), (9.1.4) y (9.3.1). permiten hacer esta extensión:

Teorema 9.9.1 *Si $\mathcal{C}$ es una colección de conjuntos entonces se tiene lo siguiente.*

(a) $\mathcal{C} \subseteq \mathcal{C}^{+0-}$.

(b) $\mathcal{C}^{+0-}$ *es de carácter finito.*

(c) Si $\mathcal{C}$ es una propiedad de consistencia entonces, $\mathcal{C}^{+0-}$ es una propiedad de consistencia alternativa.

Demostración:

(a) Tenemos que $\mathcal{C} \subseteq \mathcal{C}^+$ por el teorema (6.3.2) parte (a). Además, $\mathcal{C}^+ \subseteq \mathcal{C}^{+0}$ por el teorema (9.2.5) parte (a). Por último, puesto que $\mathcal{C}^+$ es cerrada por subconjuntos por el teorema (6.3.2) parte (b), y por lo tanto $\mathcal{C}^{+0}$ es cerrada por subconjuntos por el teorema (9.2.5) parte (b), se tiene que $\mathcal{C}^{+0} \subseteq \mathcal{C}^{+0-}$ por el teorema (6.5.1) parte (a). De esta forma, $\mathcal{C} \subseteq \mathcal{C}^{+0-}$.

(b) $\mathcal{C}^{+0-}$ es de carácter finito por el teorema (6.5.1) parte (b).

(c) Supongamos que $\mathcal{C}$ es una propiedad de consistencia. Entonces, $\mathcal{C}^+$ es una propiedad de consistencia por el teorema (9.1.4). Luego, $\mathcal{C}^{+0}$ es una propiedad de consistencia alternativa por el teorema (9.2.5) parte (c). Además, puesto que $\mathcal{C}^+$ es cerrada por subconjuntos por el teorema (6.3.2) parte (b) entonces, $\mathcal{C}^{+0}$ es cerrada por subconjuntos por el teorema (9.2.5) parte (b). De esto último se tiene que $\mathcal{C}^{+0-}$ es una propiedad de consistencia alternativa por el teorema (9.3.1).

□

Su formalización es:

```
theorem ExtensionCaracterFinito:
 shows C ⊆ C^+0-
 and caracter-finito (C^+0-)
 and consistencia C ⟶ alt-consistencia (C^+0-)
proof −
 show C ⊆ C^+0-
 proof −
  have C ⊆ C^+ using cerrado-subset by auto
  also
  have ...⊆ C^+0 using ext-consistencia-alt-subset by auto
  also
  have ...⊆ C^+0-
  proof −
   have subconj-cerrada (C^+) using cerrado-cerrado by auto
   hence subconj-cerrada (C^+0) using ext-consistencia-alt-cerrado by auto
   thus ?thesis using caracter-finito-subset by auto
  qed
  finally show ?thesis by simp
 qed
next
 show caracter-finito (C^+0-)
 using caracter-finito by auto
next
 show consistencia C ⟶ alt-consistencia (C^+0-)
 proof(rule impI)
```

```
  assume consistencia C
  hence  consistencia (C⁺) using cerrado-consistencia by auto
  hence alt-consistencia (C^+0) using alt-consistencia by auto
  moreover
  have subconj-cerrada (C^+0)
   using cerrado-cerrado  ext-consistencia-alt-cerrado by auto
  ultimately
  show alt-consistencia (C^+0-)
   using alt-consistencia-cfinita by auto
 qed
qed
```

Usando el teorema anterior se demuestra que $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ pertenece a $\mathcal{C}^{+0-}$ y es un elemento maximal de $\mathcal{C}^{+0-}$ que contiene a S.

Lema 9.9.2 *Sean* **L** *un lenguaje de primer orden, f una enumeración del conjunto de símbolos de función, g una enumeración del conjunto de símbolos de relación y $\Delta_{f,g}$ la correspondiente enumeración de las fórmulas de* **L** *(definición 9.8.3). Si $\mathcal{C}$ es una propiedad de consistencia tal que $S \in \mathcal{C}$ y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas de S es infinito, entonces*

(a) $S \subseteq S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$.

(b) $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ *es un elemento maximal de* $\mathcal{C}^{+0-}$.

(c) $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}} \in \mathcal{C}^{+0-}$.

Demostración: A partir de las hipótesis, tenemos que:

$\mathcal{C}^{+0-}$ es una propiedad de consistencia alternativa, por el teorema (9.9.1) parte (c).

$\mathcal{C}^{+0-}$ es de carácter finito, por el teorema (9.9.1) parte (b).

$S \in \mathcal{C}^{+0-}$, por el teorema (9.9.1) parte (a).

De lo anterior, por el corolario (9.4.13) se tienen (a), (b) y (c).

□

Su formalización es la siguiente:

```
lemma ExtensionConsistente1:
 assumes h0: ∀ y. ∃ n. y = f n
 and h'0: ∀ y. ∃ n. y = g n
```

```
 and h1: consistencia C
 and h2: S ∈ C
 and h3: infinite (− (⋃F ∈ S. simbfun F))
 shows S ⊆ Msuc S (C^{+0-}) (Δ f g)
 and maximal (Msuc S (C^{+0-}) (Δ f g))(C^{+0-})
 and Msuc S (C^{+0-}) (Δ f g) ∈ C^{+0-}
proof −
 have alt-consistencia (C^{+0-})
  using h1 and ExtensionCaracterFinito by auto
 moreover
 have caracter-finito (C^{+0-})
  using ExtensionCaracterFinito by auto
 moreover
 have S ∈ C^{+0-}
  using h2 and ExtensionCaracterFinito by auto
 moreover
 have infinite (− (⋃F∈S. simbfun F))
  using h3 by simp
 moreover
 have ∀ y. ∃ n. y = (Δ f g) n using EnumeracionFormulas[OF h0 h'0] by simp
 ultimately
 show S ⊆ Msuc S (C^{+0-}) (Δ f g)
  and maximal (Msuc S (C^{+0-}) (Δ f g)) (C^{+0-})
  and Msuc S (C^{+0-}) (Δ f g) ∈ C^{+0-}
  using ExtensionConsistente[of clausura-cfinito (ext-consistencia-alt (C^+ ))]
  by auto
qed
```

El siguiente resultado demuestra que si S es un conjunto consistente tal que el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas de S es infinito entonces $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ es un conjunto de Hintikka.

Teorema 9.9.3 *Sean* **L** *un lenguaje de primer orden, f una enumeración del conjunto de símbolos de función, g una enumeración del conjunto de símbolos de relación y $\Delta_{f,g}$ la correspondiente enumeración de las fórmulas de* **L** *(definición 9.8.3). Si $\mathcal{C}$ es una propiedad de consistencia tal que $S \in \mathcal{C}$ y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas de S es infinito, entonces $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ es un conjunto de Hintikka.*

Demostración: A partir de las hipótesis, se tiene lo siguiente:

$\mathcal{C}^{+0-}$ es una propiedad de consistencia alternativa, por el teorema (9.9.1) parte (c).

Puesto que $\mathcal{C}^{+0-}$ es de carácter finito por el teorema (9.9.1) parte (b), entonces se tiene que también es subconjunto cerrada por el teorema (6.4.2).

Para toda fórmula F, existe n, m tales que, $\exists x F(x) = \Delta_{f,g}(n)$ y $\neg\forall x F(x) = \Delta_{f,g}(m)$, por el corolario (9.8.1).

$S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ es un elemento maximal de $\mathcal{C}^{+0-}$ y $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}} \in \mathcal{C}^{+0-}$, por el teorema (9.9.2).

De lo anterior, se tiene que $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ es un conjunto de Hintikka por el teorema (9.6.1). □

Su formalización es la siguiente:

```
theorem Hintikka:
 assumes h0: ∀ y. ∃ n. y = f n
 and h'0: ∀ y. ∃ n. y = g n
 and h1: consistencia C
 and h2: S ∈ C
 and h3: infinite (− (⋃F ∈ S. simbfun F))
 shows hintikka (Msuc S (C^{+0-}) (Δ f g))
proof −
 have 1: alt-consistencia (C^{+0-})
  using h1 ExtensionCaracterFinito by auto
 have 2: subconj-cerrada (C^{+0-})
 proof −
  have caracter-finito (C^{+0-})
   using ExtensionCaracterFinito by auto
  thus subconj-cerrada (C^{+0-})
   by (rule caracter-finito-cerrado)
 qed
 have a3: ∀F. ∃n. (∃.F) = (Δ f g) n
  and b3: ∀F.∃n. (¬.(∀.F)) = (Δ f g) n
  using EnumeracionFormulas[OF h0 h'0] by auto
 have 4: maximal (Msuc S (C^{+0-}) (Δ f g)) (C^{+0-})
  and 5: Msuc S (C^{+0-}) (Δ f g) ∈ C^{+0-}
  using ExtensionConsistente1[OF h0 h'0 h1 h2 h3] by auto
 show ?thesis
  using 1 and 2 and a3 and b3 and 4 and 5 and MaximalHintikka[of C^{+0-}]
  by simp
qed
```

Por último, el siguiente teorema garantiza la existencia de modelos de Herbrand.

Teorema 9.9.4 *Sean* **L** *un lenguaje de primer orden, f una enumeración del conjunto de símbolos de función, g una enumeración del conjunto de símbolos de relación y $\Delta_{f,g}$ la correspondiente enumeración de las fórmulas de* **L** *(definición 9.8.1). Supongamos que,*

- *(hip1) $\mathcal{C}$ es una propiedad de consistencia de fórmulas de* **L**.
- *(hip2) $S \in \mathcal{C}$.*
- *(hip3) El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito.*
- *(hip4) $F \in S$.*
- *(hip5) F es una sentencia.*

Sea $H = S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ y $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ la correspondiente estructura de Herbrand (definición 9.5.4). Entonces, $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$.

Demostración: Tenemos lo siguiente:

Se tiene que $S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ es de Hintikka, por ser $\Delta_{f,g}$ una enumeración de las fórmulas de **L**, las hipótesis $(hip1)$, $(hip2)$, $(hip3)$ y el lema (9.9.3).

$F \in S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ por la hipótesis $(hip4)$, ya que $S \subseteq S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$ por el teorema (9.4.2).

F es una sentencia por la hipótesis $(hip5)$.

De lo anterior, por el corolario (9.5.11) queda demostrado el teorema.

□

Su formalización es la siguiente:

```
theorem ExistenciaModelo:
 assumes h0: ∀ y. ∃ n. y = f n
 and h'0: ∀ y. ∃ n. y = g n
 and h1: consistencia C
 and h2: S ∈ C
 and h3: infinite (− (⋃G ∈ S. simbfun G))
 and h4: F ∈ S
 and h5: sentencia 0 F
 shows eval (Rh (Msuc S (C^+0-) (Δ f g))) Fh I_A F
proof(rule ModeloHintikka)
 show hintikka (Msuc S (C^+0-) (Δ f g))
```

```
  using h0 and h'0 and h1 and h2 and h3 by (rule Hintikka)
next
 show F ∈ Msuc S (C^{+0-}) (Δ f g)
  using h4  Max-subconj by auto
next
 show sentencia 0 F using h5 by simp
qed
```

En el caso del lenguaje **L** de primer orden en el que identificamos los símbolos de función y de relación con los números naturales, tenemos que la función identidad $i(n) = n$, es una enumeración del conjunto de símbolos de función y del conjunto de símbolos de relación. En este caso la función $\Delta_{i,i}$ es una enumeración de las fórmulas de este lenguaje y se tiene el siguiente corolario.

Corolario 9.9.5 *Sean* **L** *el lenguaje con símbolos de función y de relación los números naturales e i la función identidad en* $\mathbb{N}$*. Supongamos que,*

- *(hip1)* $\mathcal{C}$ *es una propiedad de consistencia de fórmulas de* **L***.*
- *(hip2)* $S \in \mathcal{C}$*.*
- *(hip3) El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito.*
- *(hip4)* $F \in S$*.*
- *(hip5) F es una sentencia.*

Sea $H = S_{S,\mathcal{C}^{+0-},\Delta_{i,i}}$ *y* $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ *la correspondiente estructura de Herbrand (definición 9.5.4). Entonces,* $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$*.*

Demostración: Por hipótesis, la función identidad *i* es una enumeración de los símbolos de función y de relación de **L**. Así, por el teorema 9.9.4 se tiene que $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$.

□

Su formalización es:

```
corollary CasoExistenciaModelo:
 assumes h1: consistencia C
 and h2: S ∈ C
 and h3: infinite (− (⋃G ∈ S. simbfun G))
 and h4: F ∈ S
 and h5: sentencia 0 F
 shows eval (Rh (Msuc S (C^{+0-}) (Δ (λn. n) (λn. n)))) Fh I_A F
```

proof−
have *h0*:∀ *y*. ∃ *n*. *y* = (λ*n*. *n*) *n* **by** *simp*
have *h'0*: ∀ *y*. ∃ *n*. *y* = (λ*n*. *n*) *n* **by** *simp*
show *?thesis* **using** *ExistenciaModelo*[*OF h0 h'0 h1 h2 h3 h4 h5*] **by** *simp*
qed

Por último, como una consecuencia del teorema 9.9.4 tenemos que todo conjunto consistente es satisfacible (en una estructura de Herbrand).

Corolario 9.9.6 *Sea* **L** *un lenguaje de primer orden tal que existe una enumeración de los símbolos de función y existe una enumeración de los símbolos de relación. Supongamos que,*

- *(hip1) $\mathcal{C}$ es una propiedad de consistencia de fórmulas de* **L**.
- *(hip2) S es un conjunto de sentencias y $S \in \mathcal{C}$.*
- *(hip3) El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito.*

Entonces S es satisfacible en una estructura de Herbrand.

Demostración: Es consecuencia del teorema 9.9.4 y de la definición de satisfacibilidad (definición 8.2.7)

□

Su formalización es:

corollary *Conjuntosatisfacible*:
assumes *h1*: ∃*f*. *enumeracion* (*f*:: *nat* ⇒ *'a*)
and *h2*: ∃*g*. *enumeracion* (*g*:: *nat* ⇒ *'b*)
and *h3*: *consistencia* $\mathcal{C}$
and *h4*: (*S*:: (*'a*, *'b*) *form set*) ∈ $\mathcal{C}$
and *h5*: *infinite* (− (⋃ *G* ∈ *S*. *simbfun G*))
and *h6*: ∀ *F*∈ *S*. *sentencia 0 F*
shows ∃ (*I*$_R$ :: *'b* ⇒ *'a hterm list* ⇒ *bool*)
(*I*$_F$ ::*'a* ⇒ *'a hterm list* ⇒ *'a hterm*). *Satisfacible S I*$_R$ *I*$_F$
proof −
obtain *f g* **where** *f*: ∀ *y*. ∃ *n*. *y* = (*f*::*nat* ⇒ *'a*) *n*
and *g*: ∀ *y*. ∃ *n*. *y* = (*g*::*nat* ⇒ *'b*) *n*
using *h1 h2* **by**(*unfold enumeracion-def*) *auto*
let *?I*$_R$ = *Rh* (*Msuc S* ($\mathcal{C}^{+0-}$) (Δ *f g*))
let *?I*$_F$ = *Fh*
{ **fix** *F*
assume *hip*: *F* ∈ *S*

have *eval ?I_R ?I_F I_A F*
using *f g h3 h4 hip h5 h6 ExistenciaModelo* **by** *blast* }
hence $\forall F{\in}S$. *eval ?I_R ?I_F I_A F* **by**(*rule ballI*)
hence $\exists I_A. \forall F{\in}S$. *eval ?$I_R$?I_F I_A F* **by** *auto*
hence *Satisfacible S ?I_R ?I_F* **by**(*unfold Satisfacible-def*)
thus *?thesis* **by** *blast*
qed

En el caso del lenguaje de primer orden en el que identificamos los símbolos de función y de relación con los números naturales, tenemos el siguiente corolario.

Corolario 9.9.7 *Sean* **L** *el lenguaje de primer orden con símbolos de función y de relación los números naturales. Supongamos que,*

- *(hip1) $\mathcal{C}$ es una propiedad de consistencia de fórmulas de* **L**.
- *(hip2) S es un conjunto de sentencias y $S \in \mathcal{C}$.*
- *(hip3) El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito.*

Entonces S es satisfacible en una estructura de Herbrand.

Demostración: Puesto que existe una enumeración de $\mathbb{N}$, entonces por el coralario anterior se tiene el resultado.

□

Su formalización es:

corollary *Conjuntosatisfacible1*:
assumes *h1*: *consistencia $\mathcal{C}$*
and *h2*: (*S*:: (*nat*, *nat*) *form set*) $\in \mathcal{C}$
and *h3*: *infinite* $(-(\bigcup G \in S.$ *simbfun G*$))$
and *h4*: $\forall F{\in}$ *S*. *sentencia 0 F*
shows $\exists$ (I_R :: *nat* $\Rightarrow$ *nat hterm list* $\Rightarrow$ *bool*)
(I_F :: *nat* $\Rightarrow$ *nat hterm list* $\Rightarrow$ *nat hterm*) . *Satisfacible S I_R I_F*
using *enum-nat assms Conjuntosatisfacible*
by *blast*

9.10 Teorema de Löwenheim-Skolem

En esta sección utilizamos el teorema de existencia de modelos para formalizar la demostración del teorema de Löwenheim-Skolem. Este teorema afirma que un conjunto de fórmulas que es satisfacible (definición 8.2.7) en una *estructura arbitraria* también es satisfacible en un *modelo de Herbrand*.

La demostración está basada en probar que la colección de conjuntos satisfacibles son consistentes, es decir, los conjuntos satisfacibles forman una propiedad de consistencia y por lo tanto, por el teorema de existencia de modelos, son satisfacibles en un modelo de Herbrand.

Teorema 9.10.1 *Sea* **L** *un lenguaje de primer orden, $\mathcal{R}$ una interpretación de los símbolos de relación y A una asignación de valores a variables. Entonces, la colección de conjuntos de fórmulas*

$$\mathcal{C} = \{S | -(\bigcup_{G \in S} simbfun\, G) \text{ es infinito} \wedge \exists \mathcal{F}\, \forall G \in S (G^{\mathcal{R},\mathcal{F},A} = \mathsf{V})\}$$

es una propiedad de consistencia.

Demostracion: Sea $S \in \mathcal{C}$ entonces, $-(\bigcup_{G \in S} simbfun\, G)$ es infinito y existe $\mathcal{F}$ tal que para todo $G \in S$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. De esto último mostramos que se cumplen las condiciones para que $\mathcal{C}$ sea una propiedad de consistencia.

1. Sea P una fórmula atómica, hay que demostrar que $P \notin S$ o $\neg P \notin S$. La demostración es por contradicción. Supongamos que $P \in S$ y $\neg P \in S$ entonces, por hipótesis $P^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ y $(\neg P)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ lo cual es una contradicción.

2. Por definición, $\bot^{\mathcal{R},\mathcal{F},A} = \mathsf{F}$, luego por hipótesis $\bot \notin S$. De la misma forma, $\neg \top \notin S$.

3. Supongamos que $\neg\neg F \in S$. Entonces, por hipótesis $(\neg\neg F)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, luego $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Además, $-(\bigcup_{G \in S \cup \{F\}} simbfun\, G)$ es infinito ya que por hipótesis $-(\bigcup_{G \in S} simbfun\, G)$ es infinito y el conjunto de símbolos de función de F es finito. Así, $-(\bigcup_{G \in S \cup \{F\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. De esta forma, $S \cup \{F\} \in \mathcal{C}$.

4. Supongamos que $F_1 \wedge F_2 \in S$. Entonces, por hipótesis $(F_1 \wedge F_2)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, luego $F_1^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ y $F_2^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Además, $-(\bigcup_{G \in S \cup \{F_1,F_2\}} simbfun\, G)$ es infinito ya que por hipótesis $-(\bigcup_{G \in S} simbfun\, G)$ es infinito y los conjuntos de símbolos de función de F_1 y F_2 son finitos. Así, $-(\bigcup_{G \in S \cup \{F_1,F_2\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F_1, F_2\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. De esta forma, $S \cup \{F_1, F_2\} \in \mathcal{C}$.

Las demostraciones de las propiedades (5) y (9) son similares a la anterior.

6. Supongamos que $F_1 \vee F_2 \in S$. Entonces, por hipótesis $(F_1 \vee F_2)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, luego $F_1^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ o $F_2^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Además, $-(\bigcup_{G\in S\cup\{F_1\}} simbfun\, G)$ es infinito y $-(\bigcup_{G\in S\cup\{F_2\}} simbfun\, G)$ es infinito ya que por hipótesis $-(\bigcup_{G\in S} simbfun\, G)$ es infinito y los conjuntos de símbolos de función de F_1 y F_2 son finitos. Así, $-(\bigcup_{G\in S\cup\{F_1\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F_1\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, o $-(\bigcup_{G\in S\cup\{F_2\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F_2\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. De esta forma, $S \cup \{F_1\} \in \mathcal{C}$ o $S \cup \{F_2\} \in \mathcal{C}$.

Las demostraciones de las propiedades (7) y (8) son análogas a la anterior.

9. Supongamos que $\neg(F_1 \to F_2) \in S$. Por hipótesis, $\neg(F_1 \to F_2)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, luego $F_1^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ y $\neg F_2^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Además, $-(\bigcup_{G\in S\cup\{F_1,\neg F_2\}} simbfun\, G)$ es infinito ya que por hipótesis $-(\bigcup_{G\in S} simbfun\, G)$ es infinito y los conjuntos de símbolos de función de F_1 y $\neg F_2$ son finitos. Así, $-(\bigcup_{G\in S\cup\{F_1,\neg F_2\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F_1, \neg F_2\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. De esta forma, $S \cup \{F_1, \neg F_2\} \in \mathcal{C}$.

10. Supongamos que $\forall x F(x) \in S$. Sea t un término cerrado, hay que demostrar que $S \cup \{F(t)\} \in \mathcal{C}$. Por hipótesis $\forall x F(x)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ luego, $F(t)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Además, $-(\bigcup_{G\in S\cup\{F(t)\}} simbfun\, G)$ es infinito ya que por hipótesis $-(\bigcup_{G\in S} simbfun\, G)$ es infinito y el conjunto de símbolos de función de $F(t)$ es finito. De esta forma, $-(\bigcup_{G\in S\cup\{F(t)\}} simbfun\, G)$ es infinito y para todo $G \in S \cup \{F(t)\}$, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Por tanto, $S \cup \{F(t)\} \in \mathcal{C}$.

La demostración de la propiedad (11) es análoga a la anterior.

12. Supongamos que $\exists x F(x) \in S$ hay que demostrar que $S \cup \{F(c)\} \in \mathcal{C}$ para algún símbolo de constante c. Por hipótesis $(\exists x F(x))^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$, así $F(x)^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$ para alguna x-variante A' de A. También, por hipótesis $-(\bigcup_{G\in S} simbfun\, G)$ es infinito y $\exists x F(x) \in S$, entonces existe un símbolo de constante c que no pertenece a los símbolos de función de $\exists x F(x)$ y por tanto de F. Definimos la interpretación de símbolos de función $\mathcal{F}(c : A'(x))$ por, $\mathcal{F}(c : A'(x))(f) = F(f)$ si $f \neq c$ y $\mathcal{F}(c : A'(x))(c) = A'(x)$. De esta forma, $F(c)^{\mathcal{R},\mathcal{F}(\lfloor:\mathcal{A}'(\S)),A} = \mathsf{V}$ y $G^{\mathcal{R},\mathcal{F}(c:A'(x)),A} = \mathsf{V}$ para toda $G \in S$. Luego, $G^{\mathcal{R},\mathcal{F}(c:A'(x)),A} = \mathsf{V}$ para toda $G \in S \cup \{F(c)\}\}$ y $-(\bigcup_{G\in S\cup\{F(c)\}} simbfun\, G)$ es infinito. Así, $S \cup \{F(c)\} \in \mathcal{C}$.

La demostración de la propiedad (13) es análoga a la anterior.

Los siguientes lemas corresponden a la formalización de las anteriores propiedades.

lemma *Lowenheim1*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $(\forall P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S))$

lemma *Lowenheim2*:

assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $FF' \notin S$

lemma *Lowenheim3*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $(\neg.TT') \notin S$

lemma *Lowenheim4*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall Z.\ (\neg.\neg.Z) \in S \longrightarrow$
$S \cup \{Z\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim5*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall F1\ F2.\ (F1 \wedge.\ F2) \in S \longrightarrow$
$S \cup \{F1, F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim6*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall F1\ F2.\ (\neg.(F1 \vee.\ F2)) \in S \longrightarrow$
$S \cup \{\neg.F1, \neg.F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim7*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall F1\ F2.\ (F1 \vee.\ F2) \in S \longrightarrow$
$S \cup \{F1\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\} \vee$
$S \cup \{F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim8*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall F1\ F2.\ (\neg.(F1 \wedge.\ F2)) \in S \longrightarrow$
$S \cup \{\neg.F1\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\} \vee$

$S \cup \{\neg.F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim9*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall\, F1\ F2.\ (F1 \rightarrow.\ F2) \in S \longrightarrow$
$S \cup \{\neg.F1\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\} \vee$
$S \cup \{F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim10*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall\, F1\ F2.(\neg.(F1 \rightarrow.\ F2)) \in S \longrightarrow$
$S \cup \{F1, \neg.F2\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim11*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall\, G\ t.\ closedt\ 0\ t \longrightarrow$
$(\forall\, .G) \in S \longrightarrow$
$S \cup \{G[t\backslash 0]\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim12*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall\, G\ t.\ closedt\ 0\ t \longrightarrow$
$(\neg.(\exists\, .G)) \in S \longrightarrow$
$S \cup \{\neg.G[t\backslash 0]\} \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$

lemma *Lowenheim13*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\}$
shows $\forall\, G.\ (\exists\, .G) \in S \longrightarrow$
$(\exists\, c.\ S \cup \{G[Term\ c\ []\backslash 0]\} \in$
$\{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge (\exists\, I_F.\ \forall\, G{\in}S.\ (eval\ I_R\ I_F\ I_A)\ G)\})$

lemma *Lowenheim14*:
assumes $S \in \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, x{\in}S.\ (eval\ I_R\ I_F\ I_A)\ x)\}$

shows $\forall\, G.(\neg.(\forall\,.G)) \in S \longrightarrow$
$(\exists\, x.\ S \cup \{\neg.G[Term\ x\ []\backslash 0]\} \in$
$\{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, x{\in}S.\ (eval\ I_R\ I_F\ I_A)\ x)\})$

La formalización de la prueba del teorema 9.10.1 es la siguiente.

theorem *sat-consistencia*:
consistencia $\{S{::}('a, 'b)\ form\ set.$
$infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, (G{::}('a, 'b)form) \in S.\ eval\ I_R\ I_F\ I_A\ G)\}$
proof $(unfold\ consistencia\text{-}def, (rule\ allI\ impI)+)$
let $?C = \{S{::}('a, 'b)\ form\ set.$
$infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists\, I_F.\ \forall\, (G{::}('a, 'b)form) \in S.\ eval\ I_R\ I_F\ I_A\ G)\}$
fix $S{::}\ ('a, 'b)\ form\ set$
assume *h1*: $S \in ?C$
show $(\forall\, P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S)) \wedge$
$FF' \notin S \wedge$
$(\neg.TT') \notin S \wedge$
$(\forall\, F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in ?C) \wedge$
$(\forall\, F\ G.\ (F \wedge.\ G) \in S \longrightarrow\ S \cup \{F, G\} \in ?C) \wedge$
$(\forall\, F\ G.\ (\neg.(F \vee.\ G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in ?C) \wedge$
$(\forall\, F\ G.\ (F \vee.\ G) \in S \longrightarrow S \cup \{F\} \in ?C \vee S \cup \{G\} \in ?C) \wedge$
$(\forall\, F\ G.\ (\neg.(F \wedge.\ G)) \in S \longrightarrow S \cup \{\neg.F\} \in ?C \vee S \cup \{\neg.G\} \in ?C) \wedge$
$(\forall\, F\ G.\ (F \rightarrow.\ G) \in S \longrightarrow S \cup \{\neg.F\} \in ?C \vee S \cup \{G\} \in ?C) \wedge$
$(\forall\, F\ G.\ (\neg.(F \rightarrow.\ G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in ?C) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\forall\,.F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in ?C) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\neg.(\exists\,.F)) \in S \longrightarrow S \cup \{\neg.F[t\backslash 0]\} \in ?C) \wedge$
$(\forall\, F.\ (\exists\,.F) \in S \longrightarrow (\exists\, c.\ S \cup \{F[Term\ c\ []\backslash 0]\} \in ?C)) \wedge$
$(\forall\, F.\ (\neg.(\forall\,.F)) \in S \longrightarrow (\exists\, c.\ S \cup \{\neg.F[Term\ c\ []\backslash 0]\} \in ?C))$
proof $-$
have $(\forall\, P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S))$
using $Lowenheim1[OF\ h1\ ]$ **by** *simp*
moreover
have $FF' \notin S \wedge (\neg.TT') \notin S$
using $Lowenheim2[OF\ h1]\ \ Lowenheim3[OF\ h1]$ **by** *simp*
moreover
have $\forall\, F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in ?C$
using $Lowenheim4[OF\ h1]$ **by** *simp*
moreover
have $\forall\, F\ G.\ (F \wedge.\ G) \in S \longrightarrow S \cup \{F, G\} \in ?C$
using $Lowenheim5[OF\ h1]$ **by** *simp*
moreover

```
  have ∀ F G. (¬.(F ∨. G)) ∈ S ⟶ S ∪ {¬.F, ¬.G} ∈ ?C
   using Lowenheim6[OF h1] by simp
  moreover
  have ∀ F G. (F ∨. G) ∈ S ⟶ S ∪ {F} ∈ ?C ∨ S ∪ {G} ∈ ?C
   using Lowenheim7[OF h1] by simp
  moreover
  have ∀ F G. (¬.(F ∧. G)) ∈ S ⟶ S ∪ {¬.F} ∈ ?C ∨ S ∪ {¬.G} ∈ ?C
   using Lowenheim8[OF h1] by simp
  moreover
  have ∀ F G. (F →. G) ∈ S ⟶ S ∪ {¬.F} ∈ ?C ∨ S ∪ {G} ∈ ?C
   using Lowenheim9[OF h1] by simp
  moreover
  have ∀ F G. (¬.(F →. G)) ∈ S ⟶ S ∪ {F, ¬.G} ∈ ?C
   using Lowenheim10[OF h1] by simp
  moreover
  have ∀ F t. cerradot 0 t ⟶ (∀ .F) ∈ S ⟶ S ∪ {F[t\0]} ∈ ?C
   using Lowenheim11[OF h1] by simp
  moreover
  have ∀ F t. cerradot 0 t ⟶ (¬.(∃ .F)) ∈ S ⟶ S ∪ {¬.F[t\0]} ∈ ?C
   using Lowenheim12[OF h1] by simp
  moreover
  have ∀ F. (∃ .F) ∈ S ⟶ (∃ c. S ∪ {F[Term c []\0]} ∈ ?C)
   using Lowenheim13[OF h1] by simp
  moreover
  have ∀ F. (¬.(∀ .F)) ∈ S ⟶ (∃ c. S ∪ {¬.F[Term c []\0]} ∈ ?C)
   using Lowenheim14[OF h1] by simp
  ultimately
  show ?thesis by auto
 qed
qed
```

A partir del teorema anterior, demostramos el teorema de *Löwenheim-Skolem*.

Teorema 9.10.2 (Löwenheim-Skolem) *Sean* **L** *un lenguaje de primer orden, f una enumeración del conjunto de símbolos de función, g una enumeración del conjunto de símbolos de relación y $\Delta_{f,g}$ la correspondiente enumeración de las fórmulas de* **L** *(definición 9.8.1).*

Supongamos que S es un conjunto satisfacible en una **L***-estructura $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ y una asignación A, es decir, $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $F \in S$, y el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S es infinito. Entonces, existe un modelo de Herbrand para toda sentencia $F \in S$, es decir, existe $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ tal que $F^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$ para toda sentencia $F \in S$.*

Demostración: Por el teorema 9.10.1 se tiene que,

$$\mathcal{C} = \{S| - (\bigcup_{G \in S} simbfun\, G) \text{ es infinito} \wedge \exists \mathcal{F}\, \forall G \in S(G^{\mathcal{R},\mathcal{F},A} = \mathsf{V})\}$$

es una propiedad de consistencia. Además, por hipótesis $S \in \mathcal{C}$. Así, por el teorema 9.9.4 se tiene que para toda sentencia $F \in S$, $F^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$, donde $H = S_{S,\mathcal{C}^{+0-},\Delta_{f,g}}$. □

Su formalización es:

theorem *loewenheim-skolem*:
assumes *h0*: $\forall\ y.\ \exists\ n.\ y = f\ n$
and *h'0*: $\forall\ y.\ \exists\ n.\ y = g\ n$
and *h1*: $\forall G{\in}S.\ eval\ I_R\ I_F\ I_A\ G$
and *h2*: $infinite\ (-\ (\bigcup G \in S.\ simbfun\ G))$
and *h3*: $F \in S$
and *h4*: *sentencia 0 F*
and *h5*: $\mathcal{C} = \{S{::}('a, 'b)\ form\ set.$
$infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
$(\exists I_F.\ \forall\ (G{::}('a, 'b)form) \in S.\ eval\ I_R\ I_F\ I_A\ G)\}$
shows $eval\ (Rh\ (Msuc\ S\ (\mathcal{C}^{+0-})\ (\Delta\ f\ g)))\ Fh\ I_A{}'\ F$
proof −
have *H2*: $S{\in}\ \mathcal{C}$ **using** *h1 h2 h5* **by** *blast*
moreover
have *H1*: *consistencia* $\mathcal{C}$ **using** *h5 sat-consistencia* **by** *simp*
thus *?thesis* **using** *ExistenciaModelo*[*OF h0 h'0 H1 H2 h2 h3 h4*] **by** *simp*
qed

En lo que sigue demostramos el teorema anterior para el lenguaje con símbolos de relación y de función los números naturales. En este caso particular, para aplicar el teorema de existencia de modelos, mostramos cómo garantizar que el conjunto de símbolos de función que no ocurren en ninguna de las fórmulas de cualquier conjunto $S \in \mathcal{C}$ es infinito.

En el caso en el que identificamos los símbolos de función con los números naturales podemos garantizar que hay un número infinito de símbolos de función que no ocurren en las fórmulas de un conjunto S si identificamos convenientemente los símbolos de función que ocurren en S con elementos de un subconjunto propio de $\mathbb{N}$; por ejemplo, con números pares.

Lema 9.10.3 *Sea S un conjunto de fórmulas tal que sus símbolos de función son números pares, entonces el conjunto de símbolos de función que no ocurren en las fórmulas de S es infinito.*

Su formalización es:

lemma *InfinitosSimbFuncion*:
infinite $(-(\bigcup F \in psust\ (\lambda n{::}nat.\ 2 * n)\ `\ S.\ simbfun\ F))$

Lema 9.10.4 *Sean F una fórmula tal que sus símbolos de función son números naturales y G la fórmula que resulta de sustituir en la fórmula F cada símbolo de función n por $2n$. Entonces $G^{\mathcal{R},\mathcal{F},A} = F^{\mathcal{R},\mathcal{G},A}$, donde $\mathcal{G}$ es la interpretación de símbolos de función definida por $\mathcal{G}(n) = \mathcal{F}(2n)$.*

Su formalización es:

lemma *evalpares*:
$\bigwedge I_A.\ eval\ I_R\ I_F\ I_A\ (psust\ (\lambda n{::}nat.\ 2 * n)\ G) =\ eval\ I_R\ (\lambda n.\ I_F\ (2 * n))\ I_A\ G$

Lema 9.10.5 *Sea S un conjunto satisfacible de fórmulas tal que sus símbolos de función son números naturales. Entonces el conjunto de fórmulas S' que resulta de reemplazar en las fórmulas de S cada símbolo de función n por $2n$, también es satisfacible. Más precisamente, si $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $F \in S$, entonces $F^{\mathcal{R},\mathcal{G},A} = \mathsf{V}$ para toda $F \in S'$; donde $\mathcal{G}$ es la interpretación de símbolos de función definida por $\mathcal{G}(n) = \mathcal{F}(n\ div\ 2)$.*

Su formalización es:

lemma *doble-satisf*:
assumes $\forall\ (F{::}(nat, 'b)form) \in S.\ eval\ I_R\ I_F\ I_A\ F$
shows $\forall F{\in}(psust\ (\lambda n{::}nat.\ 2 * n)\ `\ S).\ (eval\ I_R\ (\lambda n.\ I_F\ (n\ div\ 2))\ I_A\ )\ F$
proof(*rule ballI*)
fix F
assume *h3*: $F \in psust\ (\lambda n.\ 2 * n)\ `\ S$
show $eval\ I_R\ (\lambda n.\ I_F\ (n\ div\ 2))\ I_A\ F$
proof(*rule imageE*)
show $F \in psust\ (\lambda n.\ 2 * n)\ `\ S$ **using** *h3* **by** *simp*
next
fix *Ha*
assume *h4*: $F = psust\ (\lambda n.\ 2 * n)\ Ha$ **and** *h5*: $Ha \in S$
show *?thesis* **using** *assms h4 h5* **by**(*simp add:evalpares*)
qed
qed

Lema 9.10.6 *Sea S un conjunto satisfacible de fórmulas tal que sus símbolos de función son números naturales. Entonces el conjunto de fórmulas S' que resulta de reemplazar en las fórmulas de S cada símbolo de función n por $2n$ pertenenece a la propiedad de consistencia definida por $\mathcal{C} = \{S| - (\bigcup_{G \in S} simbfun\ G)$ es infinito $\wedge\ \exists \mathcal{F}\ \forall G \in S(G^{\mathcal{R},\mathcal{F},A} = \mathsf{V})\}$.*

Demostración: Es consecuencia de los lemas 9.10.3 y 9.10.5.

□

La formalización de la prueba del lema anterior es la siguiente:

lemma *doble-consistencia*:
assumes $\forall\,(F::(nat, 'b)form) \in S.\ eval\ I_R\ I_F\ I_A\ F$
shows *psust* $(\lambda n.\ 2 * n)$ ' $S \in$
$\{S.\ infinite\ (-\ (\bigcup H{\in}S.\ simbfun\ H)) \wedge (\exists\, I_F.\ \forall\, H{\in}S.\ eval\ I_R\ I_F\ I_A\ H)\}$
proof −
have *infinite* $(-\ (\bigcup H{\in}psust\ (\lambda n.\ 2 * n)\ '\ S.\ simbfun\ H))$
using *InfinitosSimbFuncion* **by** *simp*
moreover
have $\exists\, I_F.\ \forall\, F{\in}psust\ (\lambda n.\ 2 * n)\ '\ S\ .\ eval\ I_R\ I_F\ I_A\ F$
using *assms doble-satisf* **by** *blast*
ultimately show *?thesis* **by** *auto*
qed

Teorema 9.10.7 (Löwenheim-Skolem1) *Sean* **L** *el lenguaje con símbolos de función y de relación los números naturales Supongamos que S es un conjunto satisfacible en una* **L***-estructura* $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ *y una asignación A, es decir,* $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ *para toda* $F \in S$*. Entonces, existe un modelo de Herbrand para toda sentencia* $F \in S$*, es decir, existe* $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ *tal que* $F^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$ *para toda sentencia* $F \in S$*.*

Demostración: Sean G la fórmula que resulta de reemplazar en la fórmula $F \in S$ cada símbolo de función n por $2n$, S' el conjunto que resulta de reemplazar en las fórmulas de S cada símbolo de función n por $2n$.

Sea $\mathcal{C} = \{S| - (\bigcup_{G\in S} simbfun\ G)$ es infinito $\wedge\ \exists\mathcal{F}\,\forall G \in S(G^{\mathcal{R},\mathcal{F},A} = \mathsf{V})\}$. Se tiene lo siguiente:

$\mathcal{C}$ es una propiedad de consistencia de fórmulas de **L**, por el teorema 9.10.1.

$S' \in \mathcal{C}$, por el lema 9.10.6.

El conjunto de símbolos de función que no ocurren en ninguna de las fórmulas del conjunto S' es infinito, por el lema 9.10.3.

$G \in S'$ por definición de S' y $F \in S$.

G es una sentencia, por ser F sentencia.

Sea $H = S_{S',\mathcal{C}^{+0-},\Delta_{i,i}}$ y $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ la correspondiente estructura de Herbrand (definición 9.5.4). Entonces, por el lema 9.9.5, $G^{\mathcal{R}_H,\mathcal{F}h,A'} = \mathsf{V}$. Por tanto, por

el lema 9.10.4, $F^{\mathcal{R}_H,\mathcal{G}h,A'} = \mathsf{V}$ donde $\mathcal{G}h$ es la interpretación de símbolos de función definida por $\mathcal{G}h(n) = \mathcal{F}h(2n)$. De esta forma, ${\mathcal{M}'}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{G}h)$ es un modelo de Herbrand para toda $F \in S$.

□

Su formalización es:

theorem *loewenheim-skolem1*:
 assumes *h1*: $\forall\, G{\in}S.\ eval\ I_R\ I_F\ I_A\ G$
 and *h2*: $F{\in}S$
 and *h3*: *sentencia 0 F*
 and *h4*: $\mathcal{C} = \{S.\ infinite\ (-\ (\bigcup G{\in}S.\ simbfun\ G)) \wedge$
 $(\exists\, I_F.\ \forall\, H{\in}S.\ eval\ I_R\ I_F\ I_A\ H)\}$
 shows *eval* (*Rh* (*Msuc* (*psust* ($\lambda n{::}nat.\ 2 * n$) ‘ *S*)
 ($\mathcal{C}^{+0\text{-}}$)
 (Δ ($\lambda n.\ n$) ($\lambda n.\ n$))))
 ($\lambda n{::}nat.\ Fh\ (2{*}n)$) $I_A{}'\ F$
proof(*simp only*: *evalpares*[*symmetric*])
 show *eval* (*Rh* (*Msuc* (*psust* ($\lambda n.\ 2 * n$) ‘ *S*)
 ($\mathcal{C}^{+0\text{-}}$)
 (Δ ($\lambda n.\ n$) ($\lambda n.\ n$))))
 Fh $I_A{}'$ (*psust* ($\lambda n.\ 2 * n$) *F*)
 proof −
 have *1*: *consistencia* $\{S.\ infinite\ (-\ (\bigcup H{\in}S.\ simbfun\ H)) \wedge$
 $(\exists\, I_F.\ \forall\, H{\in}S.\ eval\ I_R\ I_F\ I_A\ H)\}$
 using *sat-consistencia* **by** *simp*
 have *2*: *psust* ($\lambda n.\ 2 * n$) ‘ $S \in$
 $\{S.\ infinite\ (-\ (\bigcup H{\in}S.\ simbfun\ H)) \wedge$
 $(\exists\, I_F.\ \forall\, H{\in}S.\ eval\ I_R\ I_F\ I_A\ H)\}$
 using *h1 doble-consistencia* **by** *blast*
 moreover
 have *3*: *infinite* $(-\ (\bigcup H{\in}psust\ (\lambda n.\ 2 * n)$ ‘ $S.\ simbfun\ H))$
 by (*rule InfinitosSimbFuncion*)
 moreover
 have *4*: *psust* ($\lambda n.\ 2 * n$) $F \in$ *psust* ($\lambda n.\ 2 * n$) ‘ *S* **using** *h2* **by** *simp*
 moreover
 have *5*: *sentencia 0* (*psust* ($\lambda n.\ 2 * n$) *F*) **using** *h3* **by** *simp*
 ultimately
 show *?thesis* **using** *h4 CasoExistenciaModelo*[*OF 1 2 3 4 5*] **by** *auto*
 qed
qed

Capítulo 10

Deducción natural para la lógica de primer orden

En este capítulo formalizamos la completitud y corrección de un *sistema de deducción natural* para la lógica de primer orden. Para la formalización de la completitud utilizamos el teorema de existencia de modelos en la lógica de primer orden.

10.1 Un sistema de deducción natural para la lógica de primer orden

El sistema de deducción natural para la lógica de primer orden que estudiaremos en este capítulo está definido de la siguiente manera.

Definición 10.1.1 *Sea S un conjunto de fórmulas. Una fórmula F es **deducible** a partir de S, y se representa por $S \vdash F$, si se obtiene aplicando las siguientes reglas de inferencia:*

$$(Hip)\ \frac{F \in S}{S \vdash F} \qquad (TTI)\ \frac{}{S \vdash \top} \qquad (FFE)\ \frac{S \vdash \bot}{S \vdash F}$$

$$(NegI)\ \frac{F, S \vdash \bot}{S \vdash \neg F} \qquad (NegE)\ \frac{S \vdash \neg F \quad S \vdash F}{S \vdash \bot}$$

$$(Deriv)\ \frac{S \cup \{\neg F\} \vdash \bot}{S \vdash F}$$

$$(ConjI)\ \frac{S \vdash F \quad S \vdash G}{S \vdash F \wedge G} \qquad (ConjE1)\ \frac{S \vdash F \wedge G}{S \vdash F} \qquad (ConjE2)\ \frac{S \vdash F \wedge G}{S \vdash G}$$

$$(DisyI1)\ \frac{S \vdash F}{S \vdash F \vee G} \qquad (DisyI2)\ \frac{S \vdash G}{S \vdash F \vee G}$$

$$(DisyE)\ \frac{S \vdash F \vee G \quad S \cup \{F\} \vdash H \quad S \cup \{G\} \vdash H}{S \vdash H}$$

$$(ImplI)\ \frac{S \cup \{F\} \vdash G}{S \vdash F \to G} \qquad (ImplE)\ \frac{S \vdash F \to G \quad S \vdash F}{S \vdash G}$$

$$(TodoI)\ \frac{S \vdash F(a)}{S \vdash \forall x\, F(x)}$$

a es un símbolo de constante que no ocurre en ninguna de las fórmulas de S, ni en F.

$$(TodoE)\ \frac{S \vdash \forall x\, F(x)}{S \vdash F(t)}$$

t es un término libre para x en $F(x)$.

$$(ExisteI)\ \frac{S \vdash F(t)}{S \vdash \exists x\, F(x)}$$

t es un término libre para x en $F(x)$.

$$(ExisteE)\ \frac{S \vdash \exists x\, F(x), \quad S \cup \{F(a)\} \vdash G}{S \vdash G}$$

a es un símbolo de constante que no ocurre en ninguna de las fórmulas de S, ni en $\exists x\, F(x)$, ni en G.

La noción de derivabilidad, $S \vdash F$, se formaliza por medio del siguiente predicado inductivo.

```
inductive
  deriv :: ('a, 'b) form list ⇒ ('a, 'b) form ⇒ bool (- ⊢ - [50,50] 50)
where
  Hip: F ∈ set S ⟹ S ⊢ F
| TTI: S ⊢ TT'
| FFE: S ⊢ FF' ⟹ S ⊢ F
| NegI: F # S ⊢ FF' ⟹ S ⊢ (¬.F)
| NegE: ⟦ S ⊢ (¬.F); S ⊢ F ⟧ ⟹ S ⊢ FF'
| Deriv: (¬.F) # S ⊢ FF' ⟹ S ⊢ F
| ConjI: ⟦S ⊢ F ; S ⊢ G ⟧ ⟹ S ⊢ (F ∧. G)
| ConjE1: S ⊢ (F ∧. G) ⟹ S ⊢ F
```

| *ConjE2*: $S \vdash (F \wedge. G) \Longrightarrow S \vdash G$
| *DisyI1*: $S \vdash F \Longrightarrow S \vdash (F \vee. G)$
| *DisyI2*: $S \vdash G \Longrightarrow S \vdash (F \vee. G)$
| *DisyE*: $[\![S \vdash (F \vee. G); F \# S \vdash H ; G \# S \vdash H]\!] \Longrightarrow S \vdash H$
| *ImplI*: $F \# S \vdash G \Longrightarrow S \vdash (F \rightarrow. G)$
| *ImplE*: $[\![S \vdash (F \rightarrow. G); S \vdash F]\!] \Longrightarrow S \vdash G$
| *TodoI*: $[\![S \vdash F[Term\ n\ []\backslash 0]; list\text{-}all\ (\lambda p.\ n \notin simbfun\ p)\ S;$
$n \notin simbfun\ F]\!] \Longrightarrow S \vdash (\forall .F)$
| *TodoE*: $S \vdash (\forall .F) \Longrightarrow S \vdash F[t\backslash 0]$
| *ExisteI*: $S \vdash F[t\backslash 0] \Longrightarrow S \vdash (\exists .F)$
| *ExisteE*: $[\![S \vdash (\exists .F);$
$F[Term\ n\ []\backslash 0] \# S \vdash G;$
$list\text{-}all\ (\lambda p.\ n \notin simbfun\ p)\ S; n \notin simbfun\ F;$
$n \notin simbfun\ G]\!]$
$\Longrightarrow S \vdash G$

Las siguientes son algunas reglas de inferencia derivadas de las anteriores.

Lema 10.1.2 (Regla del corte) *Supongamos que $S \vdash F$ y $F, S \vdash G$, entonces $S \vdash G$.*

Demostración: De la segunda hipótesis y la regla *ImplI*, tenemos que $S \vdash (F \rightarrow. G)$. De esto último y la primera hipótesis, usando la regla *ImplE*, tenemos que $S \vdash G$.

□

Su formalización es:

lemma *corte*:
assumes $S \vdash F$
and $F \# S \vdash G$
shows $S \vdash G$
proof −
have *1*: $S \vdash (F \rightarrow. G)$ **using** *assms*(*2*) **by** (*rule ImplI*)
show $S \vdash G$ **using** *1 assms*(*1*) **by** (*rule ImplE*)
qed

Lema 10.1.3 *Supongamos que $\neg F, S \vdash F$, entonces $S \vdash F$.*

Demostración:
1. $\neg F, S \vdash \neg F$ [Hip]
2. $\neg F, S \vdash \bot$ [NegE, 1, hipótesis]
3. $S \vdash F$ [Deriv, 1]

□

Su formalización es:

lemma *Deriv'*:
assumes $(\neg.F)\#S \vdash F$
shows $S \vdash F$
proof −
have *1*: $(\neg.F)\#S \vdash (\neg.F)$ **by** (*simp add*: *Hip*)
have *2*: $(\neg.F)\#S \vdash FF'$ **using** *1 assms* **by** (*rule NegE*)
show *3*: $S \vdash F$ **using** *2* **by** (*rule Deriv*)
qed

Lema 10.1.4 *Supongamos que* $S \vdash \forall\, x\, F(x)$ *y* $F(t)$, $S \vdash G$, *en donde t es un término libre para x en* $F(x)$. *Entonces* $S \vdash G$.

Demostración: De la primera hipótesis y la regla *TodoE*, tenemos que $S \vdash F(t)$. De esto último y la segunda hipótesis, usando la regla del corte, tenemos que $S \vdash G$.

□

Su formalización es:

lemma *TodoE'*:
assumes $S \vdash (\forall .F)$
and $F[t\backslash 0]$ # $S \vdash G$
shows $S \vdash G$
proof −
have $S \vdash F[t\backslash 0]$ **using** *assms*(*1*) **by** (*simp add*: *TodoE*)
thus $S \vdash G$ **using** *assms*(*2*) **by** (*rule corte*)
qed

10.1.1 Ejemplos

En la formalización de los siguientes ejemplos clásicos de los textos de lógica, ilustramos el uso de las diferentes reglas de inferencia.

Lema 10.1.5 (Principio del tercio excluso) $\vdash P \vee \neg\, P$

Demostración: En la siguiente demostración Q es $P \vee \neg P$.

1.	$[P, \neg Q] \vdash P$	[Hip]
2.	$[P, \neg Q] \vdash P \vee \neg P$	[DisyI1 1]
3.	$[P, \neg Q] \vdash \neg Q$	[Hip]
4.	$[P, \neg Q] \vdash \bot$	[NegE 3, 2]
5.	$[\neg Q] \vdash \neg P$	[NegI 4]
6.	$[\neg Q] \vdash P \vee \neg P$	[DisyI2, 5]
7.	$[\neg Q] \vdash \neg Q$	[Hip]
8.	$[\neg Q] \vdash \bot$	[NegE, 7,6]
9.	$[\,] \vdash Q$	[Deriv 8]

□

Su formalización es:

lemma *tercero-excluido*: $[] \vdash (Atom\ P\ [] \vee. \neg.Atom\ P\ [])$
proof −
let $?P = Atom\ P\ []$
let $?Q = ?P \vee. \neg. ?P$
have *1*: $[?P, \neg.?Q] \vdash ?P$ **by** (*simp add*: *Hip*)
have *2*: $[?P, \neg.?Q] \vdash (?P \vee. \neg.?P)$ **using** *1* **by** (*rule DisyI1*)
have *3*: $[?P, \neg.?Q] \vdash (\neg.?Q)$ **by** (*simp add*: *Hip*)
have *4*: $[?P, \neg.?Q] \vdash FF'$ **using** *3 2* **by** (*rule NegE*)
have *5*: $[\neg.?Q] \vdash (\neg.?P)$ **using** *4* **by** (*rule NegI*)
have *6*: $[\neg.?Q] \vdash (?P \vee. \neg.?P)$ **using** *5* **by** (*rule DisyI2*)
have *7*: $[\neg.?Q] \vdash (\neg.?Q)$ **by**(*simp add*: *Hip*)
have *8*: $[\neg.?Q] \vdash FF'$ **using** *7 6* **by** (*rule NegE*)
show *9*: $[] \vdash ?Q$ **using** *8* **by** (*rule Deriv*)
qed

Lema 10.1.6 (Ley de Peirce) $\vdash ((P \to Q) \to P) \to P$

Demostración: En la siguiente demostración R es $((P \to Q) \to P) \to P$.

1.	$[(P \to Q) \to P, P, (P \to Q) \to P, \neg R] \vdash P$	[Hip]
2.	$[P, (P \to Q) \to P, \neg R] \vdash ((P \to Q) \to P) \to P$	[ImplI, 1]
3.	$[P, (P \to Q) \to P, \neg R] \vdash \neg R$	[Hip]
4.	$[P, (P \to Q) \to P, \neg R] \vdash \bot$	[NegE, 3, 2]
5.	$[P, (P \to Q) \to P, \neg R] \vdash Q$	[FFE, 4]
6.	$[(P \to Q) \to P, \neg R] \vdash P \to Q$	[ImplI, 5]
7.	$[(P \to Q) \to P, \neg R] \vdash (P \to Q) \to P$	[Hip]
8.	$[(P \to Q) \to P, \neg R] \vdash P$	[ImplE 7, 6]
9.	$[\neg R] \vdash ((P \to Q) \to P) \to P$	[ImplI, 8]
10.	$[\,] \vdash ((P \to Q) \to P) \to P$	[Deriv', 9]

□

Su formalización es:

lemma *peirce*:
[] ⊢ (((*Atom P* [] →. *Atom Q* []) →. *Atom P* []) →. *Atom P* [])
proof −
let *?P*=*Atom P* []
let *?Q*=*Atom Q* []
let *?R*=((*?P* →. *?Q*) →. *?P*) →. *?P*
have *1*: [(*?P* →. *?Q*) →. *?P*, *?P*, (*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ *?P*
by (*simp add*:*Hip*)
have *2*: [*?P*, (*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ (((*?P* →. *?Q*) →. *?P*) →. *?P*)
using *1* **by** (*rule ImplI*)
have *3*: [*?P*, (*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ (¬.*?R*) **by** (*simp add*: *Hip*)
have *4*: [*?P*, (*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ *FF′*
using *3 2* **by** (*rule NegE*) — *F* = *R*
have *5*: [*?P*, (*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ *?Q* **using** *4* **by** (*rule FFE*)
have *6*: [(*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ (*?P* →. *?Q*) **using** *5* **by** (*rule ImplI*)
have *7*: [(*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ ((*?P* →. *?Q*) →. *?P*) **by** (*simp add*: *Hip*)
have *8*: [(*?P* →. *?Q*) →. *?P*, ¬.*?R*] ⊢ *?P*
using *7 6* **by** (*rule ImplE*) — *F* = *P* →. *Q*
have *9*: [¬.*?R*] ⊢ (((*?P* →. *?Q*) →. *?P*) →. *?P*) **using** *8* **by** (*rule ImplI*)
show *10*: [] ⊢ (((*?P* →. *?Q*) →. *?P*) →. *?P*) **using** *9* **by**(*rule Deriv′*)
qed

Lema 10.1.7 (Paradoja del bebedor) $\vdash \exists\, x\ (P[x] \to \forall\, x\ P[x])$

Demostración: En la siguiente demostración $R[X]$ es la fórmula $\exists\, x\ (P[x] \to \forall\, x\ P[x])$ y a es un símbolo de constante.

1.	$[P[a], \neg P[a], P[x], \neg R[X]] \vdash \neg\, P[a]$	[Hip]
2.	$[P[a], \neg P[a], P[x], \neg R[X]] \vdash P[a]$	[Hip]
3.	$[P[a], \neg P[a], P[x], \neg R[X]] \vdash \bot$	[NegE, 1, 2 $(F = P[a])$]
4.	$[P[a], \neg P[a], P[x], \neg R[X]] \vdash \forall\, x\, P[x]$	[FFE 3]
5.	$([\neg P[a], P[x], \neg R[X]] \vdash P[a] \rightarrow \forall\, x\, P[x]$	[ImplI, 4]
6.	$[\neg P[a], P[x], \neg R[X]] \vdash (P[x] \rightarrow \forall\, x\, P[x])[a/x]$	[simplificación 5]
7.	$[\neg P[a], P[x], \neg R[X]] \vdash \exists\, x\, (P[x] \rightarrow \forall\, x\, P[x])$	[ExisteI, 6 $(t{=}a)$]
8.	$[\neg P[a], P[x], \neg R[X]] \vdash \neg(\exists\, x\, (P[x] \rightarrow \forall\, x\, P[x]))$	[Hip]
9.	$[\neg P[a], P[x], \neg R[X]] \vdash \bot$	[NegE, 8, 7]
10.	$[P[x], \neg R[X]] \vdash P[a]$	[Deriv 9]
11.	$[P[x], \neg R[X]] \vdash P[x][a/x]$	[simplificación 10]
12.	$[P[x], \neg R[X]] \vdash \forall\, x\, P[x]$	[TodoI, 11]
13.	$[\neg R[X]] \vdash P[x] \rightarrow \forall\, x\, P[x]$	[ImplI, 12]
14.	$[\neg R[X]] \vdash (P[x] \rightarrow \forall\, x\, P[x]))[x/x]$	[simplificación 13]
15.	$[\neg R[X]] \vdash \exists\, x\, (P[x] \rightarrow \forall\, x\, P[x])$	[ExisteI, 14 $(t{=}x)$]
16.	$[\,] \vdash \exists\, x\, (P[x] \rightarrow \forall\, x\, P[x])$	[Deriv', 15]

□

Su formalización es:

lemma *ParadojaBebedor*:
$([]::(nat, 'b)\ form\ list) \vdash (\exists\ .Atom\ P\ [Var\ 0] \rightarrow .\ (\forall\ .Atom\ P\ [Var\ 0]))$
proof −
let *?x* = *Var 0*
let *?Px* = *Atom P* [*?x*]
let *?a* = *Term 0* []
let *?Pa* = *Atom P* [*?a*]
let *?Rx* = $\exists\ .?Px \rightarrow .\ (\forall\ .?Px)$
have *1*: $[?Pa, \neg.?Pa, ?Px, \neg.?Rx\ ] \vdash (\neg.?Pa)$ **by** (*simp add:Hip*)
have *2*: $[?Pa, \neg.?Pa, ?Px, \neg.?Rx\ ] \vdash ?Pa$ **by** (*simp add*: *Hip*)
have *3*: $[?Pa, \neg.?Pa, ?Px, \neg.?Rx\ ] \vdash FF'$
using *1 2* **by** (*rule NegE*) — $(F = Pa)$
have *4*: $[?Pa, \neg.?Pa, ?Px, \neg.?Rx\ ] \vdash (\forall\ .?Px)$ **using** *3* **by** (*rule FFE*)
have *5*: $[\neg.?Pa, ?Px, \neg.?Rx] \vdash (?Pa \rightarrow .\ (\forall\ .\ ?Px))$ **using** *4* **by** (*rule ImplI*)
have *6*: $[\neg.?Pa, ?Px, \neg.?Rx] \vdash (?Px \rightarrow .\ (\forall\ .?Px))[?a\backslash 0]$ **using** *5* **by** *simp*
have *7*: $[\neg.?Pa, ?Px, \neg.?Rx] \vdash (\exists\ .?Px \rightarrow .\ (\forall\ .?Px))$
using *6* **by** (*rule ExisteI*) — $(t{=}a)$
have *8*: $[\neg.?Pa, ?Px, \neg.?Rx] \vdash (\neg.(\exists\ .?Px \rightarrow .\ (\forall\ .?Px)))$ **by** (*simp add*: *Hip*)
have *9*: $[\neg.?Pa, ?Px, \neg.?Rx] \vdash FF'$ **using** *8 7* **by** (*rule NegE*)
have *10*: $[?Px, \neg.?Rx] \vdash ?Pa$ **using** *9* **by** (*rule Deriv*)
have *a11*: $[?Px, \neg.?Rx] \vdash ?Px[?a\backslash 0]$ **using** *10* **by** *simp*
have *b11*: *list-all* $(\lambda p.\ 0 \notin simbfun\ p)[?Px, \neg.?Rx]$ **by** *simp*
have *c11*: $0 \notin simbfun\ ?Px$ **by** *simp*

```
 have 12:  ([?Px, ¬.?Rx]::(nat, 'b) form list) ⊢ (∀ .?Px)
  using a11 b11 c11 by (rule TodoI)
 have 13:  ([¬.?Rx]::(nat, 'b) form list) ⊢ (?Px →. (∀ . ?Px))
  using 12 by (rule ImplI) — (t=x)
 have 14:  ([¬.?Rx]::(nat, 'b) form list) ⊢ (?Px →. (∀ .?Px))[?x\0]
  using 13 by simp
 have 15:  ([¬.?Rx]::(nat, 'b) form list) ⊢ (∃ .?Px →. (∀ .?Px))
  using 14 by (rule ExisteI) — (t=x)
 show 16:  ([]::(nat, 'b) form list) ⊢ (∃ .?Px →. (∀ .?Px))
  using 15 by (rule Deriv')
qed
```

Lema 10.1.8 $\vdash \exists\, y \forall\, x\, P[y, x] \to \forall\, y \exists\, x\, P[x, y]$

Demostración: En la siguiente demostración a, b son símbolos de constante.

1.	$[\forall\, x\, P[b, x], \exists\, y \forall\, x\, P[y, x]] \vdash \forall\, x\, P[b, x]$	[Hip]
2.	$[P[b, x][a/x], \forall\, x\, P[b,x], \exists\, y \forall\, x\, P[y, x]] \vdash P[b, a]$	[simplificación, Hip]
3.	$[\forall\, x\, P[b, x], \exists\, y \forall\, x\, P[y, x]] \vdash P[b, a]$	[TodoE' 1 2 $(t = a, F = P[b, x])$]
4.	$[\forall\, x\, P[b, x], \exists\, y \forall\, x\, P[y, x]] \vdash P[x, a][b/x]$	[simplificación, 3]
5.	$[\forall\, x\, P[b, x], \exists\, y \forall\, x\, P[y, x]] \vdash \exists\, x\, P[x, a]$	[ExisteI, 4 $(t = b)$]
6.	$[(\forall\, x\, P[y, x])[b/x], \exists\, y \forall\, x\, P[y, x]] \vdash \exists\, x\, P[x, a]$	[simplificación, 5]
7.	$[\exists\, y \forall\, x\, P[y, x]] \vdash \exists\, y \forall\, x\, P[y, x]$	[Hip]
8.	$[\exists\, y \forall\, x\, P[y, x]] \vdash \exists\, x\, P[x, a]$	[ExisteE, 7, 6]
9.	$[\exists\, y \forall\, x\, P[y, x]] \vdash (\exists\, x\, P[x, y])[a/y]$	[simplificación, 8]
10.	$[\exists\, y \forall\, x\, P[y, x]] \vdash \forall\, y \exists\, x\, P[x, y]$	[TodoI, 9]
11.	$[\,] \vdash \exists\, y \forall\, x\, P[y, x] \to \forall\, y \exists\, x\, P[x, y]$	[ImplI, 10]

□

Su formalización es:

```
lemma conmuta-existe-paratodo:
 ([]::(nat, 'b) form list) ⊢
  ((∃ .∀ .Atom p [Var 1, Var 0]) →. (∀ .∃ .Atom p [Var 0, Var 1]))
proof −
 let ?x = Var 0
 let ?y = Var 1
 let ?Pxy = Atom p [?x, ?y]
 let ?Pyx = Atom p [?y,?x]
 let ?a = Term 0 []
 let ?b = Term 1 []
 let ?Pxa = Atom p [?x, ?a]
 let ?Pbx = Atom p [?b,?x]
 let ?Pba = Atom p [?b, ?a]
```

have *1*: $[\forall .?Pbx, \exists .\forall .?Pyx] \vdash (\forall .?Pbx)$ **by** (*simp add*: *Hip*)
have *2*: $[?Pbx[?a\backslash 0], \forall .?Pbx, \exists .\forall .?Pyx] \vdash ?Pba$ **by** (*simp add*: *Hip*)
have *3*: $[\forall .?Pbx, \exists .\forall .?Pyx] \vdash ?Pba$
using *1 2* **by** (*rule TodoE'*) — (t = a, F = Pbx)
have *4*: $[\forall .?Pbx, \exists .\forall .?Pyx] \vdash ?Pxa[?b\backslash 0]$ **using** *3* **by** *simp*
have *5*: $[\forall .?Pbx, \exists .\forall .?Pyx] \vdash (\exists .?Pxa)$ **using** *4* **by** (*rule ExisteI*) — (t=b)
have *6*: $[(\forall .?Pyx)[?b\backslash 0], \exists .\forall .?Pyx] \vdash (\exists .?Pxa)$ **using** *5* **by** *simp*
have *a7*: $[\exists .\forall .?Pyx] \vdash (\exists .\forall .?Pyx)$ **by** (*simp add*: *Hip*)
have *b7*: *list-all* $(\lambda p.\ 1 \notin simbfun\ p)\ [\exists .\forall .\ ?Pyx]$ **by** *simp*
have *c7*: $1 \notin simbfun\ (\forall .?Pyx)$ **by** *simp*
have *d7*: $(1::nat) \notin simbfun\ (\exists .?Pxa)$ **by** *simp*
have *8*: $([\exists .\forall .?Pyx]::(nat, 'b)\ form\ list) \vdash (\exists .?Pxa)$
using *a7 6 b7 c7 d7*
by (*rule ExisteE*) — $(n = 1, F = \forall .Pyx)$
have *a9*: $([\exists .\forall .?Pyx]::(nat, 'b)\ form\ list) \vdash (\exists .?Pxy)[?a\backslash 0]$ **using** *8* **by** *simp*
have *b9*: *list-all* $(\lambda p.\ 0 \notin simbfun\ p)\ [\exists .\forall .?Pyx]$ **by** *simp*
have *c9*: $0 \notin simbfun\ (\exists .?Pxy)$ **by** *simp*
have *10*: $([\exists .\forall .?Pyx]::(nat, 'b)\ form\ list) \vdash (\forall .\exists .?Pxy)$
using *a9 b9 c9* **by** (*rule TodoI*) — (n=0)
show *11*: $([]::(nat, 'b)\ form\ list) \vdash ((\exists .\forall .?Pyx) \rightarrow . (\forall .\exists .?Pxy))$
using *10* **by** (*rule ImplI*)
qed

10.2 Teorema de corrección del sistema de deducción natural

En esta sección demostramos formalmente el teorema de correción del cálculo de prueba definido en la sección anterior: Si F es deducible a partir de S (definición 10.1.1), entonces F es consecuencia lógica de S (definición 8.2.7).

Teorema 10.2.1 (Corrección) *Si $S \vdash F$ entonces $S \models_{\mathcal{M},A} F$ para toda estructura $\mathcal{M}$ y asignación A.*

Demostración:

Sean $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una estructura y A una asignación. Supongamos que $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $G \in S$. Hay que demostrar que $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$.

La demostración es por inducción sobre la derivación de $S \vdash F$; la prueba de cada uno de los 18 casos (definición 10.1.1) se sigue directamente de la correspondiente hipótesis de inducción. A continuación probamos algunos de estos casos:

(DisyE) $\frac{S\vdash F\vee G \quad S\cup\{F\}\vdash H \quad S\cup\{G\}\vdash H}{S\vdash H}$

Hipótesis: $S \vDash_{\mathcal{M},A} F \vee G$, $S \cup \{F\} \vDash_{\mathcal{M},A} H$ y $S \cup \{G\} \vDash_{\mathcal{M},A} H$ para toda estructura $\mathcal{M}$ y asignación A. Hay que demostrar que $S \vDash_{\mathcal{M},A} H$ para toda estructura $\mathcal{M}$ y asignación A. Sean $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una estructura y A una asignación.

Supongamos que $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda fórmula $G \in S$. Entonces, por hipótesis $(F \vee G)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Luego, $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ o $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Por tanto, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $G \in S \cup \{F\}$ o $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $G \in S \cup \{G\}$. Así, por hipótesis, $H^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$.

((TodoI) $\frac{S\vdash F(a)}{S\vdash \forall x\, F(x)}$, donde a es un símbolo de constante que no ocurre en ninguna de las fórmulas de $S \cup \{F\}$.

Hipótesis: $S \vDash_{\mathcal{M},A} F(a)$ para toda estructura $\mathcal{M}$ y asignación A.

Hay que demostrar que $S \vDash_{\mathcal{M},A} \forall x\, F(x)$ para toda estructura $\mathcal{M}$ y asignación A. Sean $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una estructura y A una asignación.

Supongamos que $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $G \in S$. Sea A' una x-variante de A, hay que demostrar que $F^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$.

Sean $A'(x) = c$ y $\mathcal{F}(a : c)$ la interpretación de símbolos de función definida por, $\mathcal{F}(a : c)(f) = \mathcal{F}(f)$ si $f \neq a$ y $\mathcal{F}(a : c)(a) = c$

Por hipótesis a no curre en G para toda $G \in S$, luego $G^{\mathcal{R},\mathcal{F},A} = G^{\mathcal{R},\mathcal{F}(a:c),A}$ para toda $G \in S$. Por lo tanto, $G^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$. para toda $G \in S$.

Por otro lado, de la hipótesis $S \vDash F(a)$ tenemos que, si $G^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$ para toda $G \in S$, entonces $F(a)^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$. Luego, $F(a)^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$.

También tenemos que, $F(a)^{\mathcal{R},\mathcal{F}(a:c),A} = F^{\mathcal{R},\mathcal{F},A}(c)$ puesto que a no ocurre en F. Además $F^{\mathcal{R},\mathcal{F},A}(c) = F^{\mathcal{R},\mathcal{F},A'}(c) = F^{\mathcal{R},\mathcal{F},A'}(A'(x)) = F(x)^{\mathcal{R},\mathcal{F},A'}$. Por tanto, $F(x)^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$.

(ExisteE) $\frac{S\vdash \exists x\, F(x), \quad S\cup\{F(a)\}\vdash G}{S\vdash G}$, donde a es un símbolo de constante que no ocurre en ninguna de las fórmulas de $S \cup \{\exists x\, F(x), G\}$

Hipótesis: $S \vDash_{\mathcal{M},A} \exists x\, F(x)$ y $S \cup \{F(a)\} \vDash_{\mathcal{M},A} G$ para toda estructura $\mathcal{M}$ y asignación A. Hay que demostrar que $S \vDash_{\mathcal{M},A} G$ para toda estructura $\mathcal{M}$ y asignación A. Sean $\mathcal{M} = (\mathcal{D}, \mathcal{R}, \mathcal{F})$ una estructura y A una asignación.

Supongamos que $H^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda $H \in S$. Entonces, por hipótesis, se tiene que $\exists x\, F(x)^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$. Por tanto, $F(x)^{\mathcal{R},\mathcal{F},A'} = \mathsf{V}$ para alguna x-variante A' de A, y como $F(x)^{\mathcal{R},\mathcal{F},A'} = F^{\mathcal{R},\mathcal{F},A}(A'(x))$ por ser A' x-variante de A, tenemos que $F^{\mathcal{R},\mathcal{F},A}(A'(x)) = \mathsf{V}$.

Sean $A'(x) = c$ y $\mathcal{F}(a : c)$ la interpretación de símbolos de función definida por, $\mathcal{F}(a : c)(f) = \mathcal{F}(f)$ si $f \neq a$ y $\mathcal{F}(a : c)(a) = c$.

Por hipótesis a no curre en F, entonces $F(a)^{\mathcal{R},\mathcal{F}(a:c),A} = F^{\mathcal{R},\mathcal{F},A}(c) = F^{\mathcal{R},\mathcal{F},A}(A'(x))$ luego $F(a)^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$.

También, puesto que a no ocurre en ninguna de las fórmulas de S, se tiene que $H^{\mathcal{R},\mathcal{F}(a:c),A} = H^{\mathcal{R},\mathcal{F},A}$ para toda $H \in S$. De esta forma, $H^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$ para toda $H \in S \cup F(a)$.

Por otro lado, de la hipótesis $S \cup \{F(a)\} \models G$ tenemos que, si $H^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$ para toda $H \in S \cup \{F(a)\}$, entonces $G^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$.

Luego, $G^{\mathcal{R},\mathcal{F}(a:c),A} = \mathsf{V}$. Además, $G^{\mathcal{R},\mathcal{F}(a:c),A} = G^{\mathcal{R},\mathcal{F},A}$ puesto que a no ocurre en G. Por tanto, $G^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$.

La prueba de los demás casos son análogos a los anteriores.

□

Los siguientes lemas formalizan las pruebas de los casos estudiados anteriormente.

```
lemma disyE:
 assumes ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ (F ∨. G)
 and ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, (F # S) ⊨ H
 and ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, (G # S) ⊨ H
 shows ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ H
proof(unfold Consecuencia-def)
 show  ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A H
 proof(rule allI)+
  fix I_A::nat ⇒ 'c
  fix I_R I_F
  show list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A H
  proof(rule impI)
   assume hip: list-all (eval I_R I_F I_A) S
   hence eval I_R I_F I_A (F ∨. G)
    using assms(1) by(unfold Consecuencia-def) simp
   hence eval I_R I_F I_A F  ∨ eval I_R I_F I_A G by simp
   thus eval I_R I_F I_A H
   proof(rule disjE)
    assume eval I_R I_F I_A F
    hence list-all (eval I_R I_F I_A) (F # S) using hip by simp
    thus ?thesis using assms(2) by(unfold Consecuencia-def) simp
    next
    assume eval I_R I_F I_A G
    hence list-all (eval I_R I_F I_A) (G # S) using hip by simp
    thus ?thesis using assms(3) by(unfold Consecuencia-def) simp
   qed
  qed
```

```
 qed
qed

lemma todoI:
 assumes hip1: ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ F[Term a []\0]
 and hip2: list-all (λG. a ∉ simbfun G) S
 and hip3: a ∉ simbfun F
 shows ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ ∀ .F
proof(unfold Consecuencia-def)
 show ∀ (I_A::nat⇒ 'c) I_F I_R.
     list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∀ . F)
 proof(rule allI)+
  fix I_A::nat ⇒ 'c
  fix I_R I_F
  show list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∀ . F)
  proof(rule impI)
   assume hip: list-all (eval I_R I_F I_A) S
   show eval I_R I_F I_A (∀ . F)
   proof −
    have ∀ c. eval I_R I_F (I_A⟨0:c⟩) F
    proof(rule allI)
     fix c
     show eval I_R I_F (I_A⟨0:c⟩) F
     proof −
      have eval I_R I_F I_A (F[Term a []\0]) =
          eval I_R I_F (I_A⟨0:I_F a []⟩) F
       by simp
      hence ∀ (I_A::nat ⇒ 'c) I_F I_R.
           list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F (I_A⟨0:I_F a []⟩) F
       using hip1 by (unfold Consecuencia-def) simp
      hence ∀ I_F I_R.
           list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F (I_A⟨0:I_F a []⟩) F
       by (rule-tac x = (I_A::nat⇒ 'c) in allE)
      hence ∀ I_R. list-all (eval I_R (I_F(a:= λx. c )) I_A) S ⟶
              eval I_R (I_F (a:= λx. c))(I_A⟨0:(I_F(a:= λx. c)) a []⟩) F
       by(rule-tac x = I_F(a:= λx. c) in allE)
      moreover
      have list-all (eval I_R (I_F(a:= λx. c )) I_A) S =
          list-all (eval I_R I_F I_A) S
       using hip2
       by simp
      ultimately
      have list-all (eval I_R I_F I_A) S ⟶
```

```
          eval I_R (I_F(a:= λx. c)) (I_A⟨0:(I_F(a:= λx. c)) a []⟩) F
        by simp
      hence eval I_R (I_F(a:= λx. c)) (I_A⟨0:(I_F(a:= λx. c)) a []⟩) F
      using hip by simp
      moreover
      { have eval I_R (I_F(a:= λx. c)) (I_A⟨0:(I_F(a:= λx. c)) a []⟩) F =
          eval I_R I_F (I_A⟨0:(I_F(a:= λx. c)) a []⟩) F
        using hip3 by simp
       also have ... = eval I_R I_F (I_A⟨0:c⟩) F
        by(simp del: fun-upd-apply add : fun-upd-same)
       finally
       have eval I_R (I_F(a := λx. c)) (I_A⟨0:(I_F(a := λx. c)) a []⟩) F =
          eval I_R I_F (I_A⟨0:c⟩) F
        by simp}
      ultimately
      show eval I_R I_F (I_A⟨0:c⟩) F by simp
     qed
    qed
    thus ?thesis  by simp
   qed
  qed
 qed
qed

lemma existeE:
 assumes hip1: ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ ∃ .F
 and hip2: ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A,  (F[Term a []\0] # S) ⊨ G
 and hip3:list-all (λG. a ∉ simbfun G) S
 and hip4: a ∉ simbfun F
 and hip5: a ∉ simbfun G
 shows  ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ G
proof(unfold Consecuencia-def)
 show ∀ (I_A::nat⇒ 'c) I_F I_R . list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
 proof(rule allI)+
  fix I_A::nat⇒ 'c
  fix I_R I_F
  show list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
  proof(rule impI)
   assume hip: list-all (eval I_R I_F I_A) S
   show eval I_R I_F I_A G
   proof −
    have eval I_R I_F I_A (∃ . F)
     using hip1 hip by(unfold Consecuencia-def) auto
```

```
    hence ∃ c. eval I_R I_F (I_A⟨0:c⟩) F by simp
    then obtain c where c: eval I_R I_F (I_A⟨0:c⟩) F by auto
    hence 1: eval I_R (I_F (a:= λx. c)) (I_A⟨0:c⟩) F  using hip4 by simp
    have  eval I_R (I_F (a:= λx. c)) (I_A⟨0:c⟩) F =
        eval I_R (I_F (a:= λx. c)) I_A (F[Term a []\0])
     by simp
    hence a: eval I_R (I_F (a:= λx. c)) I_A (F[Term a []\0])
     using 1 by simp
    moreover
    have list-all (eval I_R (I_F(a:= λx. c )) I_A) S =
       list-all (eval I_R I_F I_A) S
     using hip3 by simp
    hence b: list-all (eval I_R (I_F (a:= λx. c)) I_A) S using hip by simp
    moreover
    have c: list-all (eval I_R (I_F (a:= λx. c)) I_A) (F[Term a []\0] # S)
          ⟶ eval I_R (I_F (a:= λx. c)) I_A G
     using hip2 by(unfold Consecuencia-def) auto
    ultimately
    have eval I_R (I_F (a:= λx. c)) I_A G  by auto
    thus eval I_R I_F I_A G using hip5 by simp
   qed
  qed
 qed
qed
```

Otros casos se formalizan por medio de los siguientes lemas.

Lema 10.2.2 *Supongamos que $S \cup \{F\} \models_{\mathcal{M},A} \bot$ para toda estructura $\mathcal{M}$ y asignación A. Entonces, $S \models_{\mathcal{M},A} \neg F$ para toda estructura $\mathcal{M}$ y asignación A.*

Su formalización es:

```
lemma negI:
 assumes ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, (F # S) ⊨ FF'
 shows ∀ (I_A::nat⇒'c) I_F I_R. I_R, I_F, I_A, S ⊨ (¬.F)
```

Lema 10.2.3 *Supongamos que $S \models_{\mathcal{M},A} \neg F$ y $S \models_{\mathcal{M},A} F$ para toda estructura $\mathcal{M}$ y asignación A. Entonces, $S \models_{\mathcal{M},A} \bot$ para toda estructura $\mathcal{M}$ y asignación A.*

Su formalización es:

lemma *negE*:
assumes *hip1*: $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A,\;S \models (\neg.F)$
and *hip2*: $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A,\;S \models F$
shows $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A,\;S \models FF'$

Lema 10.2.4 *Supongamos que* $S \cup \{\neg F\} \vDash_{\mathcal{M},A} \bot$ *para toda estructura* $\mathcal{M}$ *y asignación* A. *Entonces,* $S \vDash_{\mathcal{M},A} F$ *para toda estructura* $\mathcal{M}$ *y asignación* A.

Su formalización es:

lemma *Derivs*:
assumes *hip*: $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_R\;I_F.\;I_R, I_F, I_A, ((\neg.F)\;\#\;S) \models FF'$
shows $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A, S \models F$

Lema 10.2.5 *Supongamos que* $S \vDash_{\mathcal{M},A} F(t)$ *para toda estructura* $\mathcal{M}$ *y asignación* A, *donde* t *es un término libre para* x *en* $F(x)$. *Entonces,* $S \vDash_{\mathcal{M},A} \exists x\, F(x)$ *para toda estructura* $\mathcal{M}$ *y asignación* A.

Su formalización es:

lemma *existeI*:
fixes *t*::*'a term*
assumes $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A,\;S \models (F[t\backslash 0])$
shows $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A,\;S \models (\exists\,.\,F)$

Usando los lemas anteriores, la formalización de la prueba de la correción del sistema de deducción natural es la siguiente.

theorem *correccion*:
assumes $S \vdash F$
shows $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;I_R, I_F, I_A, S \models F$
proof(*unfold Consecuencia-def*)
show $S \vdash F \Longrightarrow \forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;list\text{-}all\;(eval\;I_R\;I_F\;I_A)\;S \longrightarrow eval\;I_R\;I_F\;I_A\;F$
proof (*induct rule*: *deriv.induct*)
case(*Hip F S*)
{ **fix** *F*
fix *S*::(*'a*,*'b*)*form list*
assume *hip1*: $F \in set\;S$
thus $\forall\,(I_A{::}nat{\Rightarrow}'c)\;I_F\;I_R.\;list\text{-}all\;(eval\;I_R\;I_F\;I_A)\;S \longrightarrow eval\;I_R\;I_F\;I_A\;F$
by(*simp add* : *list-all-iff set-eq-iff*)
}
next
case(*TTI S*)

```
  { fix S
    show ∀ I_A I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A TT'
      by simp
  }
 next
  case(FFE S F)
  { fix S F
    assume S ⊢ FF'
      and ∀ (I_A::nat ⇒ 'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A FF'
    thus ∀ (I_A::nat ⇒ 'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
      by simp
  }
 next
  case(NegI F S)
  { fix F S
    assume F # S ⊢ FF'
      and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) (F # S) ⟶ eval I_R I_F I_A FF'
   thus ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (¬.F)
    by (simp add: negI) auto
 }
next
 case(NegE S F)
 { fix S F
   assume S ⊢ (¬.F)
     and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (¬.F)
     and S ⊢ F
     and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
   thus  ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A FF'
    by (simp add: negE)
 }
next
 case(Deriv F S)
 { fix F S
   assume (¬.F) # S ⊢ FF'
     and hip: ∀ (I_A::nat⇒'c) I_F I_R.
           list-all (eval I_R I_F I_A) ((¬.F) # S) ⟶ eval I_R I_F I_A FF'
   thus ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
    by (auto simp add: Derivs)
 }
next
 case (ConjI S F G )
 { fix S F G
   assume S ⊢ F
```

```
    and ∀ (I_A::nat⇒'c)  I_F I_R.
        list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
    and S ⊢ G
    and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
  thus ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∧. G)
    by simp
 }
next
 case(ConjE1 S F G)
 { fix S F G
  assume S ⊢ (F ∧. G)
    and ∀ (I_A::nat⇒'c) I_F I_R.
        list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∧. G)
  thus ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F by simp
 }
next
 case(ConjE2 S F G )
 { fix S F G
  assume S ⊢ (F ∧. G)
    and ∀ (I_A::nat⇒'c) I_F I_R.
        list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∧. G)
  thus ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
    by simp
 }
next
 case(DisyI1 S F G)
 { fix S F G
  assume S ⊢ F
    and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
  thus ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∨. G)
    by simp
 }
next
 case(DisyI2 S F G)
 { fix S G F
  assume S ⊢ G
    and ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
  thus ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∨. G)
    by simp
```

```
 }
next
 case(DisyE S F G H)
 { fix S F G H
  assume S ⊢ (F ∨. G)
   and ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F ∨. G)
   and F # S ⊢ H
   and ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) (F # S) ⟶ eval I_R I_F I_A H
   and G # S ⊢ H
   and ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) (G # S) ⟶ eval I_R I_F I_A H
  thus ∀ (I_A::nat⇒'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A H
   using disyE by (unfold Consecuencia-def) simp
 }
next
 case(ImplI F S G)
 { fix F S G
  assume F # S ⊢ G
   and hip: ∀ (I_A::nat⇒'c) I_F I_R.
         list-all (eval I_R I_F I_A) (F # S) ⟶ eval I_R I_F I_A G
  show ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F →. G)
  proof(rule allI)+
   fix I_A::nat⇒'c
   fix I_R I_F
   show list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (Impl F G)
    using hip by simp
  qed
 }
next
 case(ImplE S F G)
 { fix S F G
  assume S ⊢ (F →. G)
   and hip1: ∀ (I_A::nat⇒'c) I_F I_R.
         list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F →. G)
   and S ⊢ F
   and hip2: ∀ (I_A::nat⇒'c) I_F I_R.
         list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A F
  show ∀ (I_A::nat⇒'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
  proof(rule allI)+
```

```
    fix I_A::nat⇒'c
    fix I_R I_F
    show list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
     using hip1 hip2 by simp
   qed
 }
next
 case(TodoI S F a)
 { fix S F a
  assume S ⊢ F[Term a []\0]
   and ∀ (I_A::nat⇒ 'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F[Term a []\0])
   and list-all (λH. a ∉ simbfun H) S
   and a ∉ simbfun F
  thus ∀ (I_A::nat⇒ 'c) I_F I_R. list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∀ .F)
   using todoI by (unfold Consecuencia-def) simp
 }
next
 { fix S F t
  assume S ⊢ (∀ . F)
   and ∀ (I_A::nat⇒'c)I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∀ .F)
  thus ∀ (I_A::nat⇒'c) I_F I_R.
      list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F[t\0])
   by simp
 }
next
 case(ExisteI S F t)
 { fix S F
  fix t::'a term
  assume S ⊢ F[t\0]
   and ∀ (I_A::nat⇒ 'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (F[t\0])
  thus ∀ (I_A::nat⇒ 'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∃ .F)
    using existeI by auto
 }
next
 case(ExisteE S F a G)
 { fix S F a G
  assume S ⊢ (∃ . F)
   and ∀ (I_A::nat⇒ 'c) I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A (∃ .F)
   and F[Term a []\0] # S ⊢ G
```

```
    and ∀ (I_A::nat⇒ 'c) I_F I_R.
       list-all (eval I_R I_F I_A) (F[Term a []\0] # S) ⟶ eval I_R I_F I_A G
    and list-all (λG. a ∉ simbfun G) S
    and a ∉ simbfun F
    and a ∉ simbfun G
  thus ∀ (I_A::nat⇒ 'c)  I_F I_R.
       list-all (eval I_R I_F I_A) S ⟶ eval I_R I_F I_A G
    using existeE by (simp add: Consecuencia-def)
 }
qed
thus S ⊢ F using assms by auto
qed
```

10.3 Completitud del sistema de deducción natural

En esta sección demostramos la completitud del cálculo de deducción natural. Primero demostramos que $\mathcal{C} = \{S | S \nvdash \bot\}$ es una propiedad de consistencia y, posteriormente, aplicando el teorema de existencia de modelos se demuestra la completitud del sistema.

Teorema 10.3.1 *Sea $\mathcal{C}$ la siguiente colección de conjuntos de fórmulas de un lenguaje* **L**,

$$\mathcal{C} = \{S | S \nvdash \bot\}$$

Supongamos que el conjunto de símbolos de función de **L** *es infinito, entonces $\mathcal{C}$ es una propiedad de consistencia.*

Demostración: Sea $S \in \mathcal{C}$. Entonces $S \nvdash \bot$, de este hecho mostramos que se cumplen las condiciones para que $\mathcal{C}$ sea una propiedad de consistencia definición (9.1.3). La demostración está basada en las reglas de inferencia que definen la relación de derivabilidad $\vdash$ (definición 10.1.1) y las reglas derivadas *corte* (lema 10.1.2) y *Deriv'* (lema 10.1.3).

En lo que sigue, a la prueba informal de cada condición le asociamos la correspondiente formalización en Isar.

1. Sea $P(t_1, t_2, \ldots, t_n)$ una fórmula atómica, demostramos por contradicción que $P(t_1, t_2, \ldots, t_n) \notin S$ o $\neg P(t_1, t_2, \ldots, t_n) \notin S$.

Supongamos que $P(t_1, t_2, \ldots, t_n) \in S$ y $\neg P(t_1, t_2, \ldots, t_n) \in S$, entonces por la regla de inferencia *Hip* tenemos que $S \vdash P(t_1, t_2, \ldots, t_n)$ y $S \vdash \neg P(t_1, t_2, \ldots, t_n) \in S$. Luego $S \vdash \bot$ por la regla de inferencia *NegE*. De esta forma, usando la hipótesis $S \nvdash \bot$, obtenemos la contradicción $S \nvdash \bot$ y $S \vdash \bot$. El siguiente árbol de prueba muestra la deducción de $S \vdash \bot$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P(t_1,t_2,\ldots,t_n)\in S \text{ y } \neg P(t_1,t_2,\ldots,t_n)\in S}\text{(hipótesis)}}{P(t_1,t_2,\ldots,t_n)\in S}}{S\vdash P(t_1,t_2,\ldots,t_n)}\text{(Hip)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{P(t_1,t_2,\ldots,t_n)\in S \text{ y } \neg P(t_1,t_2,\ldots,t_n)\in S}\text{(hipótesis)}}{\neg P(t_1,t_2,\ldots,t_n)\in S}}{S\vdash \neg P(t_1,t_2,\ldots,t_n)}\text{(Hip)}}{S\vdash \bot}\text{(NegE)}$$

Su formalización es:

```
lemma consis1:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows (∀ P ts. ¬ (Atom P ts ∈ S ∧ (¬.Atom P ts) ∈ S))
proof(rule allI notI)+
 fix P ts
 assume h1: Atom P ts ∈ S ∧ (¬.Atom P ts) ∈ S
 show False
 proof −
  have T ⊢ FF'
  proof(rule NegE)
   show T ⊢ (¬.Atom P ts)
   proof −
    have (¬.Atom P ts) ∈ set T using h1 assms by (simp add: set-eq-iff)
    thus  T ⊢ (¬.Atom P ts) by (rule Hip)
   qed
  next
   show T ⊢ Atom P ts
   proof −
    have Atom P ts ∈ set T using h1 assms by (simp add: set-eq-iff)
    thus T ⊢ Atom P ts by (rule Hip)
   qed
  qed
  thus False using assms by simp
 qed
qed
```

2. La demostración de $\bot \notin S$ es por contradicción. Supongamos que $\bot \in S$, entonces por la regla *Hip* tenemos que $S \vdash \bot$. De esta forma, usando la hipótesis $S \nvdash \bot$, obtenemos la contradicción $S \nvdash \bot$ y $S \vdash \bot$. El siguiente árbol de prueba muestra la deducción de $S \vdash \bot$.

$$\dfrac{\dfrac{}{\bot\in S}\text{(hipótesis)}}{S\vdash\bot}\text{(Hip)}$$

Su formalización es:

```
lemma consis2:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows  FF' ∉ S
```

```
proof(rule notI)
 assume h1: FF' ∈ S show False
 proof −
  have FF' ∈ set T using h1 assms by (simp add: set-eq-iff)
  hence T ⊢ FF' by (rule Hip)
  thus False using assms by simp
 qed
qed
```

3. La demostración de $\neg\top \notin S$ es por contradicción. Supongamos que $\neg\top \in S$ entonces por la regla de inferencia *Hip* tenemos que $S \vdash \neg\top$, luego $S \vdash \top$ por la regla de inferencia *TTI*. De esta forma, usando la hipótesis $S \nvdash \bot$, obtenemos la contradicción $S \nvdash \bot$ y $S \vdash \bot$. El siguiente árbol de prueba muestra la deducción de $S \vdash \bot$.

$$\dfrac{\dfrac{\dfrac{}{\neg\top \in S}\,\text{(hipótesis)}}{S \vdash \neg\top}\,\text{(Hip)} \qquad \dfrac{}{S \vdash \top}\,\text{(TTI)}}{S \vdash \bot}\,\text{(NegE)}$$

Su formalización es:

```
lemma consis3:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows (¬.TT') ∉ S
proof(rule notI)
 assume h1: (¬.TT') ∈ S
 show False
 proof −
  have T ⊢ FF'
  proof(rule NegE)
   show T ⊢ (¬.TT')
   proof −
    have (¬.TT') ∈ set T using h1 assms by (simp add: set-eq-iff)
    thus T ⊢ (¬.TT') by (rule Hip)
   qed
   next
   show T ⊢ TT' by (rule TTI)
  qed
  thus False using assms by simp
 qed
qed
```

Nota 10.3.2 *De igual forma que las pruebas anteriores, las demostraciones de las otras condiciones son por contradicción probando $S \vdash \bot$ y utilizando la hipótesis $S \nvdash \bot$. En lo que sigue, en la demostraciones informales, mostramos $S \vdash \bot$ utilizando árboles de prueba.*

4. Supongamos que $\neg\neg F \in S$. Demostramos por contradicción que $S \cup \{F\} \in \mathcal{C}$. Supongamos que $S \cup \{F\} \notin \mathcal{C}$, es decir, $S \cup \{F\} \vdash \bot$. De estas hipótesis, demostramos que $S \vdash \bot$:

$$\dfrac{\dfrac{}{S \cup \{F\} \notin \mathcal{C}}\,(\text{hipótesis})}{\dfrac{S \cup \{F\} \vdash \bot}{S \vdash \bot}\,(\text{corte})} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg F \in S}\,(\text{hipótesis})}{\neg\neg F \in S \cup \{\neg F\}}}{S \cup \{\neg F\} \vdash \neg\neg F}\,(\text{Hip}) \qquad \dfrac{\neg F \in S \cup \{\neg F\}}{S \cup \{\neg F\} \vdash \neg F}\,(\text{Deriv})}{\dfrac{S \cup \{\neg F\} \vdash \bot}{S \vdash F}\,(\text{Deriv})}\,(\text{NegE})$$

Su formalización es:

```
lemma consis4:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F. (¬.¬.F) ∈ S ⟶ S ∪ {F} ∈
       {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F
 assume h1: (¬.¬.F) ∈ S
 show S ∪ {F} ∈ ?C
 proof −
  have 1: S ∪ {F} = set (F # T) using assms by simp
  have ¬ F # T ⊢ FF'
  proof(rule notI)
   assume h2: F # T ⊢ FF' show False
   proof −
    have T ⊢ FF'
    proof(rule corte)
     show F # T ⊢ FF' using h2 by simp
    next
     show  T ⊢ F
     proof(rule Deriv)
      show (¬.F) # T ⊢ FF'
      proof(rule NegE)
       show  (¬.F) # T ⊢ (¬.¬.F)
       proof −
        have (¬.¬.F) ∈ set ((¬.F) # T)
         using assms h1 by(simp add: set-eq-iff)
        thus (¬.F) # T ⊢ (¬.¬.F) by(rule Hip)
       qed
      next
       show (¬.F) # T ⊢ (¬.F)
       proof −
```

```
        have (¬.F) ∈ set ((¬.F) # T) by simp
        thus (¬.F) # T ⊢ (¬.F) by (rule Hip)
       qed
      qed
     qed
    qed
    thus False using assms by simp
   qed
  qed
  with 1 have S ∪ {F} = set (F # T) ∧ ¬ F # T ⊢ FF' by simp
  hence ∃U. S ∪ {F} = set U ∧ ¬ U ⊢ FF' by (rule exI)
  thus S ∪ {F} ∈ ?C by simp
 qed
qed
```

5. Supongamos que $F \wedge G \in S$ entonces, demostramos por contradicción que

$$S \cup \{F, G\} \in \mathcal{C}$$

Supongamos que $S \cup \{F, G\} \notin \mathcal{C}$, es decir, $S \cup \{F, G\} \vdash \bot$. De estas hipótesis, demostramos que $S \vdash \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F \wedge G \in S}\text{(hipótesis)}}{S \vdash F \wedge G}\text{(Hip)}}{S \vdash G}\text{(ConjE2)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{F \wedge G \in S}\text{(hipótesis)}}{F \wedge G \in S \cup \{G\}}\text{(Hip)}}{S \cup \{G\} \vdash F \wedge G}\text{(ConjE1)}}{S \cup \{G\} \vdash F} \qquad \dfrac{}{S \cup \{F, G\} \notin \mathcal{C}}\text{(hipótesis)}}{S \cup \{F, G\} \vdash \bot}\text{(corte)}}{S \vdash \bot}\text{(corte)}$$

Su formalización es:

```
lemma consis5:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀F G. (F ∧. G) ∈ S ⟶ S ∪ {F, G} ∈
      {S::('a, 'b) form set. ∃T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃T. S = set T ∧ ¬ T ⊢ FF'}
 fix F G
 assume h1: (F ∧. G) ∈ S
 show S ∪ {F, G} ∈ ?C
 proof −
  have 1: S ∪ {F, G} = set (F #G # T) using assms by simp
  have ¬ (F #G # T) ⊢ FF'
  proof(rule notI)
   assume h2: (F #G # T)⊢ FF' show False
   proof −
```

```
      have T ⊢ FF'
      proof(rule corte)
       show T ⊢ G
       proof −
        have (F ∧. G) ∈ set T using assms h1 by(simp add: set-eq-iff)
        hence T ⊢ (F ∧. G)  by (rule Hip)
        thus  T ⊢ G by(rule ConjE2)
       qed
       show G # T ⊢ FF'
       proof(rule corte )
        show G # T ⊢ F
        proof −
         have (F ∧. G) ∈ set (G # T) using assms h1 by(simp add: set-eq-iff)
         hence G # T ⊢ (F ∧. G)  by (rule Hip)
         thus  G # T ⊢ F by(rule ConjE1)
        qed
        next
        show (F #G # T)⊢ FF' using h2 by simp
       qed
      qed
      thus False using assms by simp
     qed
    qed
    with 1 have  S ∪ {F,G} = set (F#G#T) ∧ ¬ (F#G#T) ⊢ FF' by simp
    hence ∃ U. S ∪ {F,G} = set U ∧ ¬ U ⊢ FF' by (rule exI)
    thus S ∪ {F, G} ∈ ?C by simp
   qed
qed
```

6. Supongamos que $\neg(F \vee G) \in S$ entonces, demostramos por contradicción que $S \cup \{\neg F, \neg G\} \in \mathcal{C}$. Supongamos que $S \cup \{\neg F, \neg G\} \notin \mathcal{C}$, es decir, $S \cup \{\neg F, \neg G\} \vdash \bot$. De estas hipótesis, demostramos que $S \vdash \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(F\vee G)\in S}\,\text{(hipótesis)}}{\neg(F\vee G)\in S\cup\{G\}}\,\text{(Hip)}}{S\cup\{G\}\vdash\neg(F\vee G)}\quad \dfrac{\dfrac{G\in S\cup\{G\}}{S\cup\{G\}\vdash G}\,\text{(Hip)}}{S\cup\{G\}\vdash F\vee G}\,\text{(DisyI2)}}{S\cup\{G\}\vdash\bot}\,\text{(NegE)}}{S\vdash\neg G}\,\text{(NegI)}\qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\neg(F\vee G)\in S}\,\text{(hipótesis)}}{\neg(F\vee G)\in S\cup\{F,\neg G\}}\,\text{(Hip)}}{S\cup\{F,\neg G\}\vdash\neg(F\vee G)}\quad \dfrac{\dfrac{F\in S\cup\{F,\neg G\}}{S\cup\{F,\neg G\}\vdash F}\,\text{(Hip)}}{S\cup\{F,\neg G\}\vdash F\vee G}\,\text{(DisyI1)}}{S\cup\{F\neg G\}\vdash\bot}\,\text{(NegE)}}{S\cup\{\neg G\}\vdash\neg F}\,\text{(NegI)}\qquad \dfrac{\dfrac{}{S\cup\{\neg F,\neg G\}\notin\mathcal{C}}\,\text{(hipótesis)}}{S\cup\{\neg F,\neg G\}\vdash\bot}$$

$$\dfrac{S\cup\{\neg G\}\vdash\bot}{S\vdash\bot}\,\text{(corte)}\qquad \text{(corte)}$$

Su formalización es:

```
lemma consis6:
 assumes S = set T ∧ ¬ T ⊢ FF'
```

```
 shows ∀ F G. (¬.(F ∨. G)) ∈ S ⟶
     S ∪ {¬.F, ¬.G} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F G
 assume h1: (¬.(F ∨. G)) ∈ S
 show S ∪ {¬.F, ¬.G} ∈ ?C
 proof −
  have 1: S ∪ {¬.F, ¬.G} = set ((¬.F)#(¬.G) # T) using assms by simp
  have ¬ ((¬.F)#(¬.G) # T) ⊢ FF'
  proof(rule notI)
   assume h2: ((¬.F)#(¬.G) # T) ⊢ FF' show False
   proof −
    have T ⊢ FF'
    proof(rule corte)
     show T ⊢ (¬.G)
     proof(rule NegI)
      show G # T ⊢ FF'
      proof(rule NegE)
       show G # T ⊢ (¬.(F ∨. G))
       proof −
        have (¬.(F ∨. G)) ∈ set (G # T)
         using h1 assms by (simp add: set-eq-iff)
        thus G # T ⊢ (¬.(F ∨. G)) by(rule Hip)
       qed
      next
       show G # T ⊢ (F ∨. G)
       proof −
        have G ∈ set (G # T) by simp
        hence G # T ⊢ G by (rule Hip)
        thus G # T ⊢ (F ∨. G) by(rule DisyI2)
       qed
      qed
     qed
    next
     show (¬.G) # T ⊢ FF'
     proof(rule corte)
      show (¬.G) # T ⊢ (¬.F)
      proof(rule NegI)
       show F # (¬.G) # T ⊢ FF'
       proof(rule NegE)
        show F # (¬.G) # T ⊢ (¬.(F ∨. G))
        proof −
```

```
          have (¬.(F ∨. G)) ∈ set (F # (¬.G) # T)
           using assms h1 by(simp add: set-eq-iff)
          thus  F # (¬.G) # T ⊢ (¬.(F ∨. G)) by(rule Hip)
         qed
        next
         show F # (¬.G) # T ⊢ (F ∨. G)
         proof −
          have F ∈ set (F # (¬.G) # T) by simp
          hence F # (¬.G) # T ⊢ F by (rule Hip)
          thus F # (¬.G) # T ⊢ (F ∨. G) by(rule DisyI1)
         qed
        qed
       qed
      next
       show ((¬.F)#(¬.G) # T) ⊢ FF' using h2 by simp
      qed
     qed
     thus False using assms by simp
    qed
   qed
   with 1 have  S ∪ {(¬.F),(¬.G)} = set ((¬.F)#(¬.G)#T) ∧
           ¬ ((¬.F)#(¬.G)#T) ⊢ FF'
    by simp
   hence ∃ U. S ∪  {(¬.F),(¬.G)} = set U ∧ ¬ U ⊢ FF' by (rule exI)
   thus S ∪ {¬.F, ¬.G} ∈ ?C by simp
  qed
qed
```

7. Supongamos que $F \vee G \in S$ entonces, demostramos por contradicción que $S \cup \{F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$ Supongamos que $S \cup \{F\} \notin \mathcal{C}$ y $S \cup \{G\} \notin \mathcal{C}$, es decir, $S \cup \{F\} \vdash \bot$ y $S \cup \{G\} \vdash \bot$. De las anteriores hipótesis, demostramos que $S \vdash \bot$:

$$\dfrac{\dfrac{}{F \vee G \in S}\,(\text{hipótesis})}{S \vdash F \vee G} \qquad \dfrac{\dfrac{}{S \cup \{F\} \notin \mathcal{C} \text{ y } S \cup \{G\} \notin \mathcal{C}}\,(\text{hipótesis})}{\dfrac{S \cup \{F\} \notin \mathcal{C}}{S \cup \{F\} \vdash \bot}} \qquad \dfrac{\dfrac{}{S \cup \{F\} \notin \mathcal{C} \text{ y } S \cup \{G\} \notin \mathcal{C}}\,(\text{hipótesis})}{\dfrac{S \cup \{G\} \notin \mathcal{C}}{S \cup \{G\} \vdash \bot}}$$
$$\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{S \vdash \bot}\,(\text{DisyE})$$

Su formalización es:

```
lemma consis7:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F G. (F ∨. G) ∈ S ⟶
        S ∪ {F} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'} ∨
        S ∪ {G} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
```

```
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F G
 assume h1: (F ∨. G) ∈ S
 show  S ∪ {F} ∈ ?C ∨ S ∪ {G} ∈ ?C
 proof(rule ccontr)
  assume h2: ¬ (S ∪ {F} ∈ ?C ∨ S ∪ {G} ∈ ?C)
  show False
  proof −
   have T ⊢ FF'
   proof(rule DisyE)
    show T ⊢ (F ∨. G)
    proof −
     have (F ∨. G) ∈ set T using h1 assms by (simp add: set-eq-iff)
     thus T ⊢ (F ∨. G) by (rule Hip)
    qed
   next
    show F # T ⊢ FF'
    proof −
     have  ¬ S ∪ {F} ∈ ?C using h2 by simp
     hence  ∀ Ta. S ∪ {F} = set Ta ⟶ Ta ⊢ FF' by simp
     moreover
     have S ∪ {F} = set T ∪ {F} using assms by simp
     ultimately
     show F # T ⊢ FF' by simp
    qed
   next
    show G # T ⊢ FF'
    proof −
     have  ¬ S ∪ {G} ∈ ?C using h2 by simp
     hence  ∀ U. S ∪ {G} = set U ⟶ U ⊢ FF' by simp
     moreover
     have S ∪ {G} = set T ∪ {G} using assms by simp
     ultimately
     show G # T ⊢ FF' by simp
    qed
   qed
   thus False using assms by simp
  qed
 qed
qed
```

8. Supongamos que $\neg(F \wedge G) \in S$ entonces, demostramos por contradicción que $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{\neg G\} \in \mathcal{C}$. Supongamos que $S \cup \{\neg F\} \notin \mathcal{C}$ y $S \cup \{\neg G\} \notin \mathcal{C}$, es

decir, $S \cup \{\neg F\} \vdash \bot$ y $S \cup \{\neg G\} \vdash \bot$. De las anteriores hipótesis, demostramos que $S \vdash \bot$. Se demuestra utilizando la regla de inferencia *DisyE* de la siguiente manera:

$$\dfrac{\begin{array}{ccc}\vdots & \vdots & \vdots\\ \dfrac{}{S \vdash \neg F \vee \neg G} & \dfrac{}{S \cup \{\neg F\} \vdash \bot} & \dfrac{}{S \cup \{\neg G\} \vdash \bot}\end{array}}{S \vdash \bot}\ (\text{DisyE})$$

La demostración de $S \vdash \neg F \vee \neg G$ es la siguiente.

$$\dfrac{\dfrac{\neg(\neg F \vee \neg G) \in S \cup \{\neg(\neg F \vee \neg G), F\}}{S \cup \{\neg(\neg F \vee \neg G), F\} \vdash \neg(\neg F \vee \neg G)}\ (\text{Hip}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(F \wedge G) \in S}\ (\text{hipótesis})}{\neg(F \wedge G) \in S \cup \{\neg(\neg F \vee \neg G), F, G\}}\ (\text{Hip})}{S \cup \{\neg(\neg F \vee \neg G), F, G\} \vdash \neg(F \wedge G)} \qquad \dfrac{F \in S \cup \{\neg(\neg F \vee \neg G), F, G\}}{S \cup \{\neg(\neg F \vee \neg G), F, G\} \vdash F}\ (\text{Hip}) \qquad \dfrac{G \in S \cup \{\neg(\neg F \vee \neg G), F, G\}}{S \cup \{\neg(\neg F \vee \neg G), F, G\} \vdash G}\ (\text{Hip})}{S \cup \{\neg(\neg F \vee \neg G), F, G\} \vdash \{F \wedge G\}}\ (\text{ConjI})}{S \cup \{\neg(\neg F \vee \neg G), F, G\} \vdash \bot}\ (\text{NegE})}{S \cup \{\neg(\neg F \vee \neg G), F\} \vdash \neg G}\ (\text{NegI})}{S \cup \{\neg(\neg F \vee \neg G), F\} \vdash \neg F \vee \neg G}\ (\text{DisyI2})}{S \cup \{\neg(\neg F \vee \neg G), F\} \vdash \bot}\ (\text{NegE})}{S \cup \{\neg(\neg F \vee \neg G)\} \vdash \neg F}\ (\text{NegI})}{S \cup \{\neg(\neg F \vee \neg G)\} \vdash \neg F \vee \neg G}\ (\text{DisyI1})}{S \vdash \neg F \vee \neg G}\ (\text{Deriv}')$$

La demostración de $S \cup \{\neg F\} \vdash \bot$ es la siguiente.

$$\dfrac{\dfrac{\dfrac{}{S \cup \{\neg F\} \notin \mathcal{C} \text{ y } S \cup \{\neg G\} \notin \mathcal{C}}\ (\text{hipótesis})}{S \cup \{\neg F\} \notin \mathcal{C}}}{S \cup \{\neg F\} \vdash \bot}$$

De la misma forma tenemos la demostración de $S \cup \{\neg G\} \vdash \bot$,

$$\dfrac{\dfrac{\dfrac{}{S \cup \{\neg F\} \notin \mathcal{C} \text{ y } S \cup \{\neg G\} \notin \mathcal{C}}\ (\text{hipótesis})}{S \cup \{\neg G\} \notin \mathcal{C}}}{S \cup \{\neg G\} \vdash \bot}$$

Su formalización es:

lemma *consis8*:
 assumes $S = set\ T \wedge \neg\ T \vdash FF'$
shows $\forall F\ G.\ (\neg.(F \wedge. G)) \in S \longrightarrow$
 $S \cup \{\neg.F\} \in \{S::('a, 'b)\ form\ set.\ \exists T.\ S = set\ T \wedge \neg\ T \vdash FF'\} \vee$
 $S \cup \{\neg.G\} \in \{S::('a, 'b)\ form\ set.\ \exists T.\ S = set\ T \wedge \neg\ T \vdash FF'\}$
proof(*rule allI impI*)+
 let *?C* $= \{S::('a, 'b)\ form\ set.\ \exists T.\ S = set\ T \wedge \neg\ T \vdash FF'\}$
 fix *F G*
 assume *h1*: $(\neg.(F \wedge. G)) \in S$
 show $S \cup \{\neg.F\} \in$ *?C* $\vee\ S \cup \{\neg.G\} \in$ *?C*
 proof(*rule ccontr*)
  assume *h2*: $\neg\ (S \cup \{\neg.F\} \in$ *?C* $\vee\ S \cup \{\neg.G\} \in$ *?C*$)$
  show *False*
  proof $-$

```
have T ⊢ FF'
proof(rule DisyE)
 show T ⊢ (¬.F ∨. ¬.G)
 proof(rule Deriv')
  show (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.F ∨. ¬.G)
  proof(rule DisyI1)
   show (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.F)
   proof(rule NegI)
    show  F # (¬.(¬.F ∨. ¬.G)) # T ⊢ FF'
    proof(rule NegE)
     show F # (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.(¬.F ∨. ¬.G))
     proof(rule Hip)
      show (¬.(¬.F ∨. ¬.G)) ∈ set (F # (¬.(¬.F ∨. ¬.G)) # T)
       by(simp add: set-eq-iff)
     qed
    next
     show F # (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.F ∨. ¬.G)
     proof(rule DisyI2)
      show F # (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.G)
      proof(rule NegI)
       show G # F # (¬.(¬.F ∨. ¬.G)) # T ⊢ FF'
       proof(rule NegE)
        show G # F # (¬.(¬.F ∨. ¬.G)) # T ⊢ (¬.(F ∧. G))
        proof(rule Hip)
         show (¬.(F ∧. G)) ∈ set (G # F # (¬.(¬.F ∨. ¬.G)) # T)
         using h1 assms  by(simp add: set-eq-iff)
        qed
       next
        show G # F # (¬.(¬.F ∨. ¬.G)) # T ⊢ (F ∧. G)
        proof(rule  ConjI)
         show G # F # (¬.(¬.F ∨. ¬.G)) # T ⊢ F
          by(rule Hip) simp
        next
         show G # F # (¬.(¬.F ∨. ¬.G)) # T ⊢ G
          by(rule Hip) simp
        qed
       qed
      qed
     qed
    qed
   qed
  qed
 qed
qed
```

```
    next
     show  (¬.F) # T ⊢ FF'
     proof −
      have  ¬ S ∪ {¬.F} ∈ ?C using h2 by simp
      hence  ∀ Ta. S ∪ {¬.F} = set Ta ⟶ Ta ⊢ FF' by simp
      moreover
      have S ∪ {¬.F} = set T ∪ {¬.F} using assms by simp
      ultimately
      show (¬.F) # T ⊢ FF' by simp
     qed
    next
     show (¬.G) # T ⊢ FF'
     proof −
      have  ¬ S ∪ {¬.G} ∈ ?C using h2 by simp
      hence  ∀ U. S ∪ {¬.G} = set U ⟶ U ⊢ FF' by simp
      moreover
      have S ∪ {¬.G} = set T ∪ {¬.G} using assms by simp
      ultimately
      show (¬.G) # T ⊢ FF' by simp
     qed
    qed
    thus False using assms by simp
   qed
  qed
qed
```

9. Supongamos que $F \rightarrow G \in S$ entonces, demostramos por contradicción que $S \cup \{\neg F\} \in \mathcal{C}$ o $S \cup \{G\} \in \mathcal{C}$. Supongamos que $S \cup \{\neg F\} \notin \mathcal{C}$ y $S \cup \{G\} \notin \mathcal{C}$, es decir, $S \cup \{\neg F\} \vdash \bot$ y $S \cup \{G\} \vdash \bot$. De las anteriores hipótesis, demostramos que $S \vdash \bot$. Se demuestra utilizando la regla de inferencia *DisyE* de la siguiente manera:

$$\dfrac{\begin{array}{ccc}\vdots & \vdots & \vdots\\ S \vdash \neg F \vee G & S \cup \{\neg F\} \vdash \bot & S \cup \{G\} \vdash \bot\end{array}}{S \vdash \bot}\ (\text{DisyE})$$

La demostración de $S \vdash \neg F \vee G$ es la siguiente:

$$\dfrac{\dfrac{\dfrac{\neg(\neg F \vee G) \in S \cup \{\neg(\neg F \vee G), F\}}{S \cup \{\neg(\neg F \vee G), F\} \vdash \neg(\neg F \vee G)}\ (\text{Hip}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{F \rightarrow G \in S}\ (\text{hipótesis})}{F \rightarrow G \in S \cup \{\neg(\neg F \vee G), F\}}\ (\text{Hip}) \qquad \dfrac{F \in S \cup \{\neg(\neg F \vee G), F\}}{S \cup \{\neg(\neg F \vee G), F\} \vdash F}\ (\text{Hip})}{S \cup \{\neg(\neg F \vee G), F\} \vdash F \rightarrow G}\ (\text{ImplE})}{S \cup \{\neg(\neg F \vee G), F\} \vdash \neg F \vee G}\ (\text{DisyI2})}{S \cup \{\neg(\neg F \vee G), F\} \vdash \bot}\ (\text{NegE})}{S \cup \{\neg(\neg F \vee G)\} \vdash \neg F}\ (\text{NegI})}{S \cup \{\neg(\neg F \vee G)\} \vdash \neg F \vee G}\ (\text{DisyI1})}{S \vdash \neg F \vee G}\ (\text{Deriv'})$$

La demostración de $S \cup \{\neg F\} \vdash \bot$ es la siguiente.

$$\dfrac{\dfrac{\dfrac{}{S \cup \{\neg F\} \notin \mathcal{C} \text{ y } S \cup \{G\} \notin \mathcal{C}}\ \text{(hipótesis)}}{S \cup \{\neg F\} \notin \mathcal{C}}}{S \cup \{\neg F\} \vdash \bot}$$

De la misma forma tenemos la demostración de $S \cup \{G\} \vdash \bot$,

$$\dfrac{\dfrac{\dfrac{}{S \cup \{\neg F\} \notin \mathcal{C} \text{ y } S \cup \{G\} \notin \mathcal{C}}\ \text{(hipótesis)}}{S \cup \{G\} \notin \mathcal{C}}}{S \cup \{\neg G\} \vdash \bot}$$

Su formalización es:

```
lemma consis9:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F G. (F →. G) ∈ S ⟶
      S ∪ {¬.F} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'} ∨
      S ∪ {G} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F G
 assume h1: (F →. G) ∈ S
 show S ∪ {¬.F} ∈ ?C ∨ S ∪ {G} ∈ ?C
 proof(rule ccontr)
  assume h2: ¬ (S ∪ {¬.F} ∈ ?C ∨ S ∪ {G} ∈ ?C)
  show False
  proof −
   have T ⊢ FF'
   proof(rule DisyE)
    show T ⊢ (¬.F ∨. G)
    proof(rule Deriv')
     show (¬.(¬.F ∨. G)) # T ⊢ (¬.F ∨. G)
     proof(rule DisyI1)
      show (¬.(¬.F ∨. G)) # T ⊢ (¬.F)
      proof(rule NegI)
       show F # (¬.(¬.F ∨. G)) # T ⊢ FF'
       proof(rule NegE)
        show F # (¬.(¬.F ∨. G)) # T ⊢ (¬.(¬.F ∨. G))
        proof(rule Hip)
         show (¬.(¬.F ∨. G)) ∈ set (F # (¬.(¬.F ∨. G)) # T)
          by(simp add: set-eq-iff)
        qed
       next
        show F # (¬.(¬.F ∨. G)) # T ⊢ (¬.F ∨. G)
        proof(rule DisyI2)
```

```
            show F # (¬.(¬.F ∨. G)) # T ⊢ G
            proof(rule ImplE)
             show  F # (¬.(¬.F ∨. G)) # T ⊢ (F →. G)
             proof(rule Hip)
              show (F →. G) ∈ set (F # (¬.(¬.F ∨. G)) # T)
               using h1 assms by(simp add: set-eq-iff)
             qed
            next
             show F # (¬.(¬.F ∨. G)) # T ⊢ F by(rule Hip, simp)
            qed
           qed
          qed
         qed
        qed
       qed
      next
       show (¬.F) # T ⊢ FF'
       proof −
        have  ¬ S ∪ {¬.F} ∈ ?C using h2 by simp
        hence  ∀ Ta. S ∪ {¬.F} = set Ta ⟶ Ta ⊢ FF' by simp
        moreover
        have S ∪ {¬.F} = set T ∪ {¬.F} using assms by simp
        ultimately
        show (¬.F) # T ⊢ FF' by simp
       qed
      next
       show G # T ⊢ FF'
       proof −
        have  ¬ S ∪ {G} ∈ ?C using h2 by simp
        hence  ∀ U. S ∪ {G} = set U ⟶ U ⊢ FF' by simp
        moreover
        have S ∪ {G} = set T ∪ {G} using assms by simp
        ultimately
        show G # T ⊢ FF' by simp
       qed
      qed
      thus False using assms by simp
     qed
    qed
qed
```

10. Supongamos que $\neg(F \to G) \in S$ entonces, demostramos por contradicción que $S \cup \{F, \neg G\} \in \mathcal{C}$. Supongamos que $S \cup \{F, \neg G\} \notin \mathcal{C}$, es decir, $S \cup \{F, \neg G\} \vdash \bot$. De estas

hipótesis, demostramos que $S \vdash \bot$. Se demuestra utilizando la regla derivada *corte* de la siguiente manera:

$$\dfrac{\dfrac{\vdots}{S \vdash \neg G} \qquad \dfrac{\vdots}{S \cup \{\neg G\} \vdash \bot}}{S \vdash \bot}\ (\text{corte})$$

La demostración de $S \vdash \neg G$ es la siguiente.

$$\dfrac{\dfrac{\dfrac{\dfrac{G \in S \cup \{F, G\}}{S \cup \{F, G\} \vdash G}\ (\text{Hip})}{S \cup \{G\} \vdash F \to G}\ (\text{ImplI}) \qquad \dfrac{\dfrac{}{\neg(F \to G) \in S}\ (\text{hipótesis})}{\dfrac{\neg(F \to G) \in S \cup \{G\}}{S \cup \{G\} \vdash \neg(F \to G)}\ (\text{Hip})}}{S \cup \{G\} \vdash \bot}\ (\text{NegE})}{S \vdash \neg G}\ (\text{NegI})$$

La demostración de $S \cup \{\neg G\} \vdash \bot$ es la siguiente.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(F \to G) \in S}\ (\text{hipótesis})}{\dfrac{\neg(F \to G) \in S \cup \{\neg F, \neg G\}}{S \cup \{\neg F, \neg G\} \vdash \neg(F \to G)}\ (\text{Hip})} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\neg F \in S \cup \{\neg F, \neg G, F\}}{S \cup \{\neg F, \neg G, F\} \vdash \neg F}\ (\text{Hip}) \qquad \dfrac{F \in S \cup \{\neg F, \neg G, F\}}{S \cup \{\neg F, \neg G, F\} \vdash F}\ (\text{Hip})}{S \cup \{\neg F, \neg G, F\} \vdash \bot}\ (\text{NegE})}{S \cup \{\neg F, \neg G, F\} \vdash G}\ (\text{FFE})}{S \cup \{\neg F, \neg G\} \vdash F \to G}\ (\text{ImplI})}{S \cup \{\neg F, \neg G\} \vdash \bot}\ (\text{NegE}) \qquad \dfrac{\dfrac{}{S \cup \{F, \neg G\} \notin \mathcal{C}}\ (\text{hipótesis})}{S \cup \{F, \neg G\} \vdash \bot}}{S \cup \{\neg G\} \vdash F}\ (\text{Deriv})}{S \cup \{\neg G\} \vdash \bot}\ (\text{corte})$$

Su formalización es:

```
lemma consis10:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F G. (¬.(F →. G)) ∈ S ⟶
       S ∪ {F, ¬.G} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F G
 assume h1: (¬.(F →. G)) ∈ S
 show S ∪ {F, ¬.G} ∈ ?C
 proof −
  have {F, ¬.G} ∪ (set T) = set(F # (¬.G) # T) ∧ ¬ F # (¬.G) # T ⊢ FF'
  proof −
   have ¬ F # (¬.G) # T ⊢ FF'
   proof(rule notI)
    assume h2: F # (¬.G) # T ⊢ FF'
    show False
    proof −
     have T ⊢ FF'
     proof(rule corte)
      show T ⊢ (¬.G)
```

```
 proof(rule NegI)
  show G # T ⊢ FF'
  proof(rule NegE)
   show G # T ⊢ (F →. G)
   proof(rule ImplI)
    show F # G # T ⊢ G by(rule Hip) simp
   qed
  next
   show G # T ⊢ (¬.(F →. G))
   proof(rule Hip)
    show (¬.(F →. G)) ∈ set (G # T)
     using h1 assms by(simp add: set-eq-iff)
   qed
  qed
 qed
next
 show (¬.G) # T ⊢ FF'
 proof(rule corte)
  show (¬.G) # T ⊢ F
  proof(rule Deriv)
   show  (¬.F) # (¬.G) # T ⊢ FF'
   proof(rule NegE)
    show (¬.F) # (¬.G) # T ⊢ (¬.(F →. G))
    proof(rule Hip)
     show (¬.(F →. G)) ∈ set ((¬.F) # (¬.G) # T)
      using h1 assms by(simp add: set-eq-iff)
    qed
   next
    show (¬.F) # (¬.G) # T ⊢ (F →. G)
    proof(rule ImplI)
     show F # (¬.F) # (¬.G) # T ⊢ G
     proof(rule FFE)
      show F # (¬.F) # (¬.G) # T ⊢ FF'
      proof(rule NegE)
       show F # (¬.F) # (¬.G) # T ⊢ (¬.F) by(rule Hip) simp
       next
       show F # (¬.F) # (¬.G) # T ⊢ F by(rule Hip) simp
      qed
     qed
    qed
   qed
  qed
 next
```

```
        show  F # (¬.G) # T ⊢ FF' using h2 by simp
       qed
      qed
      thus False using assms by simp
     qed
    qed
    thus {F, ¬.G} ∪ (set T) = set(F # (¬.G) # T) ∧ ¬ F # (¬.G) # T ⊢ FF'
     by simp
   qed
   thus S ∪ {F, ¬.G} ∈ ?C using assms by blast
  qed
qed
```

11. Sean t un término cerrado y F una fórmula. Supongamos que $\forall x F(x) \in S$ entonces, demostramos por contradicción que $S \cup \{F(t)\} \in \mathcal{C}$.

Supongamos que $S \cup \{F(t)\} \notin \mathcal{C}$, es decir, $S \cup \{F(t)\} \vdash \bot$. De estas hipótesis, demostramos que $S \vdash \bot$.

$$\dfrac{\dfrac{\dfrac{}{S \cup \{F(t)\} \notin \mathcal{C}}\,(\text{hipótesis})}{S \cup \{F(t)\} \vdash \bot} \qquad \dfrac{\dfrac{\dfrac{}{\forall x F(x) \in S}\,(\text{hipótesis})}{S \vdash \forall x F(x)}\,(\text{Hip})}{S \vdash F(t)}\,(\text{TodoE})}{S \vdash \bot}\,(\text{corte})$$

Su formalización es:

```
lemma consis11:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F t. cerradot 0 t ⟶ (∀ .F) ∈ S ⟶
        S ∪ {F[t\0]} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F
 fix t :: 'a term
 assume h1: cerradot 0 t  and  h2: (∀ .F) ∈ S
 show S ∪ {F[t\0]} ∈ ?C
 proof
  show  ∃ T1. S ∪ {F[t\0]} = set T1 ∧ ¬ T1 ⊢ FF'
  proof(rule-tac x = F[t\0] # T in exI)
   show S ∪ {F[t\0]} = set(F[t\0] # T) ∧ ¬ F[t\0] # T ⊢ FF'
   proof −
    have S ∪ {F[t\0]} = set(F[t\0] # T) using assms by simp
    moreover
    have  ¬ F[t\0] # T ⊢ FF'
    proof(rule notI)
```

```
      assume h3: F[t\0] # T ⊢ FF'
      show False
      proof −
       have  T ⊢ FF'
       proof(rule corte)
        show F[t\0] # T ⊢ FF' using h3 by simp
       next
        show T ⊢ F[t\0]
        proof(rule TodoE)
         show T ⊢ (∀ .F)
         proof(rule Hip)
          show (∀ .F) ∈ set T using h2 assms by (simp add: set-eq-iff)
         qed
        qed
       qed
       thus False using assms by simp
      qed
     qed
     ultimately
     show S ∪ {F[t\0]} = set(F[t\0] # T) ∧ ¬ F[t\0] # T ⊢ FF' by simp
    qed
   qed
  qed
qed
```

12. Sean t un término cerrado y F una fórmula tal que $\neg\exists xF(x) \in S$, entonces $S \cup \{\neg F(t)\} \in \mathcal{C}$. La prueba es por contradicción. Supongamos que $S \cup \{\neg F(t)\} \notin \mathcal{C}$, es decir, $S \cup \{\neg F(t)\} \vdash \bot$. De estas hipótesis, demostramos que $S \vdash \bot$.

$$\dfrac{\dfrac{}{S \cup \{\neg F(t)\} \notin \mathcal{C}}\ \text{(hipótesis)}}{S \cup \{\neg F(t)\} \vdash \bot} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\neg\exists xF(x) \in S}\ \text{(hipótesis)}}{\neg\exists xF(x) \in S \cup \{F(t)\}}\ \text{(Hip)}}{S \cup \{F(t)\} \vdash \neg\exists xF(x)} \qquad \dfrac{\dfrac{\dfrac{}{F(t) \in S \cup \{F(t)\}}\ \text{(Hip)}}{S \cup \{F(t)\} \vdash F(t)}\ \text{(ExisteI)}}{S \cup \{F(t)\} \vdash \exists xF(x)}\ \text{(NegE)}}{S \cup \{F(t)\} \vdash \bot}\ \text{(NegI)}}{S \vdash \neg F(t)} \qquad \text{(corte)}$$

$$S \vdash \bot$$

Su formalización es:

```
lemma consis12:
 assumes S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F t. cerradot 0 t ⟶
      (¬.(∃ .F)) ∈ S ⟶
      S ∪ {¬.F[t\0]} ∈ {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof(rule allI impI)+
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
```

```
fix F
fix t :: 'a term
assume h1: cerradot 0 t and h2: (¬.(∃ .F)) ∈ S
show S ∪ {¬.F[t\0]} ∈ ?C
proof
 show ∃ T1. S ∪ {¬.F[t\0]} = set T1 ∧ ¬ T1 ⊢ FF'
 proof(rule-tac x= (¬.F[t\0]) # T in exI)
  show S ∪ {¬.F[t\0]} = set((¬.F[t\0]) # T) ∧ ¬ (¬.F[t\0]) # T ⊢ FF'
  proof −
   have S ∪ {¬.F[t\0]} = set((¬.F[t\0]) # T) using assms by simp
   moreover
   have ¬ (¬.F[t\0]) # T ⊢ FF'
   proof(rule notI)
    assume h3: (¬.F[t\0]) # T ⊢ FF'
    show False
    proof −
     have T ⊢ FF'
     proof(rule corte)
      show (¬.F[t\0]) # T ⊢ FF' using h3 by simp
     next
      show T ⊢ (¬.F[t\0])
      proof(rule NegI)
       show F[t\0] # T ⊢ FF'
       proof(rule NegE)
        show F[t\0] # T ⊢ (¬.(∃ .F))
        proof(rule Hip)
         show (¬.(∃ .F)) ∈ set (F[t\0] # T) using h2 assms
         by(simp add: set-eq-iff)
        qed
       next
        show F[t\0] # T ⊢ (∃ .F)
        proof(rule ExisteI)
         show F[t\0] # T ⊢ F[t\0] by(rule Hip) simp
        qed
       qed
      qed
     qed
     thus False using assms by simp
    qed
   qed
   ultimately
   show S ∪ {¬.F[t\0]} = set((¬.F[t\0]) # T) ∧ ¬ (¬.F[t\0]) # T ⊢ FF'
    by simp
```

```
    qed
  qed
 qed
qed
```

Para la demostración de las últimas dos condiciones, que hacen referencia a la introducción de símbolos de constante por cada $S \in \mathcal{C}$, se utilizan las reglas de inferencia *ExistI* y *TodoI* que requieren que estos parámetros no ocurran en las fórmulas de S. Usando la hipótesis de que el conjunto de símbolos de función es infinito se demuestra el siguiente lema que es utilizado en la formalización de las condiciones mencionadas anteriormente.

Lema 10.3.3 *Supongamos que el conjunto* **F** *de símbolos de función de un lenguaje dado es infinito. Sea S un conjunto de fórmulas finito. Entonces, para cualquier fórmula F existe un símbolo de función c que no ocurre en ninguna de las fórmulas de S ni en F.*

Demostración: Sea A el conjunto de símbolos de función que ocurren en las fórmulas de S o en F, entonces A es finito ya que el conjunto de símbolos de función que ocurren en cualquier fórmula es finito y además por hipótesis S es finito. Luego, $\mathbf{F} - A$ es infinito, ya que por hipótesis **F** es infinito, por lo tanto es no vacío, luego existe $c \in \mathbf{F} - A$.

□

Para la formalización del lema anterior, primero formalizamos el siguiente resultado básico sobre conjuntos utilizado en la prueba: si A es infinito, entonces tiene algún elemento.

```
lemma infinito-novacio:
 assumes infinite A
 shows ∃ x. x ∈ A
```

y la propiedad acerca de la finitud del conjunto de símbolos de función de una fórmula, que está basada en la finitud del conjunto de símbolos de función de un término.

```
lemma finito-simbfunt [simp]:
 finite (simbfunt (t :: 'a term))
 finite (simbfunts (ts :: 'a term list))
lemma finito-simbfun [simp]: finite (simbfun F)
```

La formalización de la prueba del lema es:

```
lemma existe-parametro:
 assumes inf-param: infinite (UNIV::'a set) and finite S
 shows ∃ (c:: 'a). c ∈ UNIV − ((⋃ G ∈ S. simbfun G) ∪ simbfun F)
```

```
proof(rule infinito-novacio)
 show infinite (UNIV − ((⋃p ∈ S. simbfun p) ∪ simbfun F))
 proof −
  show  infinite (UNIV −((⋃p ∈ S. simbfun p) ∪ simbfun F))
  proof(rule Diff-infinite-finite)
   show finite ((⋃p ∈ S. simbfun p) ∪ simbfun F)  using assms by simp
  next
   show infinite (UNIV::'a set) by (rule inf-param)
  qed
 qed
qed
```

13. Supongamos que $\exists x F(x) \in S$ entonces, por la regla *Hip*, $S \vdash \exists x F(x)$. Hay que demostrar que existe un símbolo de constante c tal que $S \cup \{F(c)\} \in \mathcal{C}$, es decir, hay que demostrar que $S \cup \{F(c)\} \nvdash \bot$ para algún símbolo de constante c.

Puesto que por hipótesis el conjunto de símbolos de función es infinito, entonces por el teorema 10.3.3 existe un símbolo de constante $c \notin S \cup \{F\}$. Demostramos por contradicción que $S \cup \{F(c)\} \nvdash \bot$. Supongamos que $S \cup \{F(c)\} \vdash \bot$ entonces, como $S \vdash \exists x F(x)$ y $c \notin S \cup \{F\}$, por la regla de inferencia *ExistE* se tiene que $S \vdash \bot$. De esto último y de la hipótesis $S \nvdash \bot$ obtenemos una contradicción. La demostración de $S \vdash \bot$ se ilustra en el siguiente diagrama de prueba.

$$\dfrac{\dfrac{\dfrac{}{\exists x F(x) \in S}\,\text{(hipótesis)}}{S \vdash \exists x F(x)}\,\text{(Hip)} \qquad \dfrac{}{S \cup \{F(c)\} \vdash \bot}\,\text{(hipótesis)}}{S \vdash \bot}\,\text{(ExisteE)}$$

Su formalización es:

```
lemma consis13:
 assumes inf-param: infinite (UNIV::'a set)
 and hip2: S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F. (∃ .F) ∈ S ⟶
        (∃ c. S ∪ {F[Term c []\0]} ∈
          {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'})
proof(rule allI impI)
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F
 show (∃ .F) ∈ S ⟶ (∃ c. S ∪ {F[Term c []\0]} ∈ ?C)
 proof(rule impI)
  assume h1: (∃ .F) ∈ S
  show ∃ c. S ∪ {F[Term c []\0]} ∈ ?C
  proof −
   have ∃ c. c ∈ UNIV − ((⋃G ∈ set T. simbfun G) ∪ simbfun F)
    using inf-param and existe-parametro by auto
```

hence $\exists\, c.\ (\forall\, G \in \mathit{set}\ T.\ c \notin \mathit{simbfun}\ G) \wedge c \notin \mathit{simbfun}\ F$ **by** *simp*
then obtain *c* **where** *c*: $(\forall\, G \in \mathit{set}\ T.\ c \notin \mathit{simbfun}\ G) \wedge c \notin \mathit{simbfun}\ F$
by (*rule exE*)
have $\exists\, U.\ \{F[\mathit{Term}\ c\ []\backslash 0]\} \cup \mathit{set}\ T = \mathit{set}\ U \wedge \neg\ U \vdash FF'$
proof(*rule-tac* $x = F[\mathit{Term}\ c\ []\backslash 0] \#\ T$ **in** *exI*)
show $\{F[\mathit{Term}\ c\ []\backslash 0]\} \cup\ \mathit{set}\ T = \mathit{set}(F[\mathit{Term}\ c\ []\backslash 0] \#\ T) \wedge$
$\neg F[\mathit{Term}\ c\ []\backslash 0] \#\ T \vdash FF'$
proof −
have $\{F[\mathit{Term}\ c\ []\backslash 0]\} \cup \mathit{set}\ T = \mathit{set}(F[\mathit{Term}\ c\ []\backslash 0] \#\ T)$ **by** *simp*
moreover
have $\neg\ F[\mathit{Term}\ c\ []\backslash 0] \#\ T \vdash FF'$
proof(*rule notI*)
assume *h2*: $F[\mathit{Term}\ c\ []\backslash 0] \#\ T \vdash FF'$
show *False*
proof −
have $T \vdash FF'$
proof(*rule ExisteE*)
show $T \vdash (\exists\, .F)$
proof(*rule Hip*)
show $(\exists\, .F) \in \mathit{set}\ T$ **using** *h1 hip2* **by** (*simp add*: *set-eq-iff*)
qed
next
show $F[\mathit{Term}\ c\ []\backslash 0] \#\ T \vdash FF'$ **using** *h2* **by** *simp*
next
show *list-all* $(\lambda G.\ c \notin \mathit{simbfun}\ G)\ T$
using *c* **by**(*simp add*: *list-all-iff*)
next
show $c \notin \mathit{simbfun}\ F$ **using** *c* **by** *simp*
next
show $c \notin \mathit{simbfun}\ FF'$ **by** *simp*
qed
thus *False* **using** *assms* **by** *simp*
qed
qed
ultimately
show $\{F[\mathit{Term}\ c\ []\backslash 0]\} \cup \mathit{set}\ T = \mathit{set}(F[\mathit{Term}\ c\ []\backslash 0] \#\ T) \wedge$
$\neg\ F[\mathit{Term}\ c\ []\backslash 0] \#\ T \vdash FF'$
by *simp*
qed
qed
hence $\exists\, U.\ \{F[\mathit{Term}\ c\ []\backslash 0]\} \cup S = \mathit{set}\ U \wedge \neg\ U \vdash FF'$ **using** *hip2* **by** *simp*
hence $S \cup \{F[\mathit{Term}\ c\ []\backslash 0]\} \in ?\mathcal{C}$ **by** *simp*
thus $\exists\, c.\ S \cup \{F[\mathit{Term}\ c\ []\backslash 0]\} \in ?\mathcal{C}$ **by** *blast*

```
  qed
 qed
qed
```

14. Supongamos que $\neg\forall x F(x) \in S$. Hay que demostrar que existe un símbolo de constante c tal que $S \cup \{\neg F(c)\} \in \mathcal{C}$, es decir, hay que demostrar que $S \cup \{\neg F(c)\} \nvdash \bot$ para algún símbolo de constante c.

Puesto que por hipótesis el conjunto **C** de símbolos de función es infinito, entonces por el teorema 10.3.3 existe un símbolo de constante $c \notin S \cup \{F\}$. Demostramos por contradicción que $S \cup \{\neg F(c)\} \nvdash \bot$.

A partir de las hipótesis $S \cup \{\neg F(c)\} \vdash \bot$ y $\neg\forall x F(x) \in S$, utilizando las reglas de inferencia *ExisteI* y *TodoI*, en el siguiente árbol de prueba se demuestra $S \vdash FF'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\forall x F(x) \in S}\,(\text{hipótesis})}{\dfrac{\neg\forall x F(x) \in S \cup \{\neg(\exists x \neg F(x))\}}{S \cup \{\neg(\exists x \neg F(x))\} \vdash \neg\forall x F(x)}\,(\text{Hip})}}{\dfrac{S \cup \{\neg(\exists x \neg F(x))\} \vdash \bot}{S \vdash \exists x \neg F(x)}\,(\text{Deriv})} \qquad \dfrac{\dfrac{\dfrac{\neg(\exists x \neg F(x)) \in S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\}}{S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\} \vdash \neg(\exists x \neg F(x))}\,(\text{Hip}) \qquad \dfrac{\dfrac{\neg F(c) \in S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\}}{S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\} \vdash \neg F(c)}\,(\text{Hip})}{S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\} \vdash \exists x \neg F(x)}\,(\text{ExisteI})}{S \cup \{\neg(\exists x \neg F(x)), \neg F(c)\} \vdash \bot}\,(\text{NegE})}{\dfrac{S \cup \{\neg(\exists x \neg F(x))\} \vdash F(c)}{S \cup \{\neg(\exists x \neg F(x))\} \vdash \forall x F(x)}\,(\text{TodoI})}\,(\text{Deriv})}{S \cup \{\neg(\exists x \neg F(x))\} \vdash \bot}\,(\text{NegE}) \qquad \dfrac{}{S \cup \{\neg F(c)\} \vdash \bot}\,(\text{hipótesis})}{S \vdash \bot}\,(\text{ExisteE})$$

□

Su formalización es:

```
lemma consis14:
 assumes inf-param: infinite (UNIV::'a set)
 and hip2: S = set T ∧ ¬ T ⊢ FF'
 shows ∀ F. (¬.(∀ .F)) ∈ S ⟶
        (∃ c. S ∪ {¬.F[Term c []\0]} ∈
           {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'})
proof(rule allI impI)
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
 fix F
 show (¬.(∀ .F)) ∈ S ⟶ (∃ c. S ∪ {¬.F[Term c []\0]} ∈ ?C)
 proof(rule impI)
  assume h1: (¬.(∀ .F)) ∈ S
  show ∃ c. S ∪ {¬.F[Term c []\0]} ∈ ?C
  proof −
   have ∃ c. c ∈ UNIV − ((⋃ G ∈ set T. simbfun G) ∪ simbfun F)
   using inf-param existe-parametro by auto
   hence ∃ c. (∀ xa ∈ set T. c ∉ simbfun xa) ∧ c ∉ simbfun F by simp
   then obtain c where c: (∀ G ∈ set T. c ∉ simbfun G) ∧ c ∉ simbfun F
```

```
  by (rule exE)
have ∃ U. {¬.F[Term c []\0]} ∪ set T = set U ∧ ¬ U ⊢ FF'
proof(rule-tac x = Neg (sust F (Term c []) 0) # T in exI)
 show {¬.F[Term c []\0]} ∪ set T = set((¬.F[Term c []\0]) # T) ∧
     ¬ (¬.F[Term c []\0]) # T ⊢ FF'
 proof −
  have {¬.F[Term c []\0]} ∪ set T = set((¬.F[Term c []\0]) # T)
   by simp
  moreover
  have ¬(¬. F[Term c []\0]) # T ⊢ FF'
  proof(rule notI)
   assume h2: (¬.F[Term c []\0]) # T ⊢ FF'
   show False
   proof(rule notE)
    show ¬ T ⊢ FF' using hip2 by simp
   next
    show T ⊢ FF'
    proof(rule ExisteE)
     show T ⊢ (∃ .¬.F)
     proof(rule Deriv)
      show (¬.(∃ .¬.F)) # T ⊢ FF'
      proof(rule NegE)
       show (¬.(∃ .¬.F)) # T ⊢ (¬.(∀ .F))
       proof(rule Hip)
        show (¬.(∀ .F)) ∈ set ((¬.(∃ .¬.F)) # T)
         using h1 hip2 by(simp add: set-eq-iff)
       qed
      next
       show (¬.(∃ .¬.F)) # T ⊢ (∀ .F)
       proof(rule TodoI)
        show (¬.(∃ .¬.F)) # T ⊢ F[Term c []\0]
        proof(rule Deriv)
         show (¬.F[Term c []\0]) # (¬.(∃ .¬.F)) # T ⊢ FF'
         proof(rule NegE)
          show (¬.F[Term c []\0]) # (¬.(∃ .¬.F)) # T ⊢
              (¬.(∃ .¬.F))
           by (rule Hip) simp
         next
          show (¬.F[Term c []\0]) # (¬.(∃ .¬.F)) # T ⊢ (∃ .¬.F)
          proof(rule ExisteI)
           show (¬.F[Term c []\0]) # (¬.(∃ .¬.F)) # T ⊢
               (¬.F)[Term c []\0]
            by (rule Hip) simp
```

```
            qed
           qed
          qed
         next
          show list-all (λG. c ∉ simbfun G) ((¬.(∃ .¬.F)) # T)
           using c by(simp add: list-all-iff)
         next
          show c ∉ simbfun F using c by simp
         qed
        qed
       qed
      next
       show (¬.F)[Term c []\0] # T ⊢ FF' using h2 by simp
      next
       show  list-all (λG. c ∉ simbfun G) T
        using c by(simp add: list-all-iff)
      next
       show  c ∉ simbfun (¬.F) using c  by simp
      next
       show c ∉ simbfun FF' by simp
      qed
     qed
    qed
    ultimately
    show {¬.F[Term c []\0]} ∪ set T = set((¬.F[Term c []\0]) # T) ∧
        ¬(¬.F[Term c []\0]) # T ⊢ FF'
     by simp
   qed
  qed
  hence ∃ U. {¬.F[Term c []\0]} ∪ S = set U ∧ ¬ U ⊢ FF'
   using hip2 by simp
  hence S ∪ {¬.F[Term c []\0]} ∈ {set T |T. ¬ T ⊢ FF'} by simp
  thus ∃ c. S ∪ {¬.F[Term c []\0]} ∈ {set T |T. ¬ T ⊢ FF'} by blast
 qed
qed
qed
```

Usando los lemas anteriores la formalización del teorema 10.3.1 es la siguiente.

```
theorem deriv-consistencia:
 assumes inf-param: infinite (UNIV::'a set)
 shows consistencia {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
proof −
 let ?C = {S::('a, 'b) form set. ∃ T. S = set T ∧ ¬ T ⊢ FF'}
```

{ **fix** S::('a, 'b) *form set*
assume $S \in ?\mathcal{C}$
hence $\exists\, T.\ S = set\ T \wedge \neg\ T \vdash FF'$ **by** *simp*
then obtain T **where** T: $S = set\ T \wedge \neg\ T \vdash FF'$ **by** (*rule exE*)
have $(\forall\, P\ ts.\ \neg\ (Atom\ P\ ts \in S \wedge (\neg.Atom\ P\ ts) \in S)) \wedge$
$FF' \notin S \wedge$
$(\neg.TT') \notin S \wedge$
$(\forall\, F.\ (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (F \wedge. G) \in S \longrightarrow S \cup \{F, G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \vee. G)) \in S \longrightarrow S \cup \{\neg.F, \neg.G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (F \vee. G) \in S \longrightarrow$
$S \cup \{F\} \in \{set\ T\ |T.\ \neg\ T \vdash FF'\} \vee S \cup \{G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \wedge. G)) \in S \longrightarrow$
$S \cup \{\neg.F\} \in ?\mathcal{C} \vee S \cup \{\neg.G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (F \rightarrow. G) \in S \longrightarrow$
$S \cup \{\neg.F\} \in ?\mathcal{C} \vee S \cup \{G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ G.\ (\neg.(F \rightarrow. G)) \in S \longrightarrow S \cup \{F, \neg.G\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow (\forall\, .F) \in S \longrightarrow S \cup \{F[t\backslash 0]\} \in ?\mathcal{C}) \wedge$
$(\forall\, F\ t.\ cerradot\ 0\ t \longrightarrow$
$(\neg.(\exists\, .F)) \in S \longrightarrow S \cup \{\neg.F[t\backslash 0]\} \in ?\mathcal{C}) \wedge$
$(\forall\, F.\ (\exists\, .F) \in S \longrightarrow (\exists\, c.\ S \cup \{F[Term\ c\ []\backslash 0]\} \in ?\mathcal{C})) \wedge$
$(\forall\, F.\ (\neg.(\forall\, .F)) \in S \longrightarrow (\exists\, c.\ S \cup \{\neg.F[Term\ c\ []\backslash 0]\} \in ?\mathcal{C}))$
using
consis1[OF T] *consis2*[OF T] *consis3*[OF T]
consis4[OF T] *consis5*[OF T] *consis6*[OF T]
consis7[OF T] *consis8*[OF T] *consis9*[OF T]
consis10[OF T] *consis11*[OF T] *consis12*[OF T] *consis13*[OF *inf-param* T]
consis14[OF *inf-param* T]
by *blast*}
thus *?thesis* **by**(*simp add*: *consistencia-def*)
qed

lemma *Compl-UNIV-eq*: $-\ A = UNIV - A$

Teorema 10.3.4 (Completitud) *Sea* **L** *lenguaje de primer orden tal que existe una enumeración de los símbolos de función y existe una enumeración de los símbolos de relación. Supongamos que,*

- *(hip1) El conjunto de símbolos de función es infinito.*
- *(hip2)* $S \cup \{G\}$ *es un conjunto de sentencias.*

Si para toda **L***-estructura* $\mathcal{M} = (\mathcal{D}h, \mathcal{R}, \mathcal{F})$*, con dominio* $\mathcal{D}h$ *el conjunto de términos de Herbrand, y toda asignación* A *se tiene que* $S \models_{\mathcal{M},A} G$*, entonces* $S \vdash G$*.*

Demostración:

Para demostrar $S \vdash G$ utilizamos la regla de inferencia *Deriv*. Así, basta con demostrar que $S \cup \{\neg G\} \vdash \bot$. La demostración es por contradicción, supongamos que $S \cup \{\neg G\} \nvdash \bot$. A partir de esta hipótesis probamos que existe una estructura $\mathcal{M} = (\mathcal{D}h, \mathcal{R}, \mathcal{F})$ tal que, $F^{\mathcal{R},\mathcal{F},A} = \mathsf{V}$ para toda fórmula $F \in S \cup \{\neg G\}$. Sean f y g enumeraciones de los símbolos de función y de relación respectivamente.

Consideremos la estructura de Herbrand, $\mathcal{M}_H = (\mathcal{D}h, \mathcal{R}_H, \mathcal{F}h)$ donde $\mathcal{C} = \{S | S \nvdash \bot\}$ y $H = S_{S\cup\{G\},((\mathcal{C}^+)^0)^-,\Delta_{f,g}}$. Entonces $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$ para toda fórmula $F \in S \cup \{\neg G\}$.

La demostración es como sigue. Sea $F \in S \cup \{\neg G\}$.

1. Por hipótesis el conjunto de símbolos de función es infinito, luego por el teorema 10.3.1 se tiene que $\mathcal{C} = \{S | S \nvdash \bot\}$ es una propiedad de consistencia.

2. De la hipótesis $S \cup \{\neg G\} \nvdash \bot$ se tiene que $S \cup \{\neg G\} \in \mathcal{C}$.

3. El conjunto de símbolos de función que no ocurren en $S \cup \{\neg G\} \nvdash \bot$ es infinito, ya que por hipótesis el conjunto de símbolos de función de **L** es infinito.

4. F es una sentencia ya que $F \in S \cup \{\neg G\}$ y por hipótesis $S \cup \{G\}$ es un conjunto de sentencias.

Por (1), (2), (3) y (4) se tiene, por el teorema de existencia de modelos (teorema 9.9.4), que $F^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. En particular, $(\neg G)^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. Además por hipótesis $S \vDash_{\mathcal{M}_H,A} G$, luego $G^{\mathcal{R}_H,\mathcal{F}h,A} = \mathsf{V}$. De esta forma obtenemos una contradicción.

□

```
theorem CompletitudDeduccionNatural:
 assumes hip0: ∃f. enumeracion (f:: nat ⇒ 'a)
 and hip0': ∃g. enumeracion (g:: nat ⇒ 'b)
 and hip1: infinite (UNIV::'a set)
 and hip2: list-all (sentencia 0) (G # S)
 shows ∀ (I_A::nat ⇒ 'a hterm)
       (I_F::'a ⇒ 'a hterm list ⇒ 'a hterm)
       (I_R::'b ⇒ 'a hterm list ⇒ bool).
       I_R, I_F, I_A, S ⊨ G ⟹ S ⊢ G
proof(rule Deriv)
 obtain f g where f: ∀ y. ∃ n. y = (f::nat ⇒ 'a) n
  and g: ∀ y. ∃ n. y = (g::nat ⇒ 'b) n
  using hip0 hip0' by(unfold enumeracion-def) auto
 assume h1: ∀ (I_A::nat ⇒ 'a hterm) I_F I_R. I_R,I_F,I_A,S ⊨ G
 show (¬.G) # S ⊢ FF'
```

```
proof(rule ccontr)
 assume h2: ¬ (¬.G) # S ⊢ FF'
 show False
 proof −
  have 1: ∃ (I_A::nat ⇒ 'a hterm) I_R I_F.
        list-all (eval I_R I_F I_A) ((¬.G) # S)
  proof (simp only: list-all-iff)
   show ∃ (I_A::nat ⇒ 'a hterm) I_R I_F.
       ∀ x ∈ set ((¬.G) # S). eval I_R I_F I_A x
   proof(rule-tac x = (cualquiera::nat ⇒ 'a hterm) in exI)
    show ∃ I_R I_F. ∀ x ∈ set((¬.G) # S).
        eval I_R I_F (cualquiera::nat ⇒ 'a hterm) x
    proof(rule exI ballI)+
     fix F
     assume hd: F ∈ set((¬.G) # S)
     show eval (Rh (Msuc (set ((¬.G) # S))
                   ({set S |S. ¬ S ⊢ FF'}^+0-)
                   (Δ f g)))
           Fh cualquiera F
     proof −
      have ha: consistencia
            {((set S)::('a, 'b) form set) |S. ¬ S ⊢ FF'}
       using hip1 by(rule deriv-consistencia)
      have hb: set ((¬.G) # S) ∈ {set S |S. ¬ S ⊢ FF'}
      proof(simp del: set.simps)
       show ∃ T. set((¬.G) # S) = set T ∧ ¬ T ⊢ FF'
       proof(rule-tac x=((¬.G) # S) in exI)
        show set ((¬.G) # S) = set ((¬.G) # S) ∧ ¬ (¬.G) # S ⊢ FF'
        proof −
         have (¬.G) # S = (¬.G) # S by simp
         moreover
         have ¬ (¬.G) # S ⊢ FF' using h2 by simp
         ultimately show ?thesis by simp
        qed
       qed
      qed
      have hc: infinite ( − (⋃ p ∈ set((¬.G) # S). simbfun p))
      proof −
       show infinite (− (⋃ p ∈ set((¬.G) # S). simbfun p))
        using hip1 by(simp add: Compl-UNIV-eq)
      qed
      have he: sentencia 0 F
      proof −
```

```
          have list-all (sentencia 0) ((¬.G) # S) using hip2 by simp
          thus ?thesis using hd by (simp only: list-all-iff)
        qed
        show eval (Rh (Msuc (set ((¬.G) # S))
                  ({set S |S. ¬ S ⊢ FF'}^+0-)
                  (Δ f g)))
             Fh cualquiera F
          using ExistenciaModelo[OF f g ha hb hc hd he] by auto
      qed
     qed
    qed
   qed
   then obtain I_R I_F I_A
    where eIi: list-all (eval I_R I_F (I_A::nat ⇒ 'a hterm)) ((¬.G) # S)
    by blast
   hence eval I_R I_F I_A (¬.G) by simp
   moreover
   have eval I_R I_F I_A G
    using eIi and h1 by(simp add: Consecuencia-def) blast
   ultimately
   show False by simp
  qed
 qed
qed
```

En el caso del lenguaje **L** de primer orden en el que identificamos los símbolos de función y de relación con los números naturales, tenemos el siguiente corolario.

Corolario 10.3.5 *Sean* **L** *el lenguaje de primer orden con símbolos de función y de relación los números naturales. Supongamos que* $S \cup \{G\}$ *es un conjunto de sentencias.*

Si para toda **L***-estructura* $\mathcal{M} = (\mathcal{D}h, \mathcal{R}, \mathcal{F})$*, con dominio* $\mathcal{D}h$ *el conjunto de términos de Herbrand, y toda asignación* A *se tiene que* $S \models_{\mathcal{M},A} G$*, entonces* $S \vdash G$*.*

Demostración: Es consecuencia del teorema anterior teniendo en cuenta que el conjunto de los números naturales es infinito y enumerable,

□

```
corollary CDN:
 assumes list-all (sentencia 0) (G # S)
 shows ∀ (I_A::nat ⇒ nat hterm)
       (I_F::nat ⇒ nat hterm list ⇒ nat hterm)
       (I_R::nat ⇒ nat hterm list ⇒ bool).
       I_R,I_F,I_A,S ⊨ G ⟹ S ⊢ G
```

proof −
have *h1*: *infinite* (*UNIV*:: *nat set*)
using *nat-infinite* **by**(*simp add*: *Compl-UNIV-eq*)
show ∀ (I_A::*nat* ⇒ *nat hterm*)
(I_F::*nat* ⇒ *nat hterm list* ⇒ *nat hterm*)
(I_R::*nat* ⇒ *nat hterm list* ⇒ *bool*).
I_R,I_F,I_A,*S* ⊨ *G* ⟹ *S* ⊢ *G*
using *CompletitudDeduccionNatural*[*OF enum-nat enum-nat h1 assms*] **by** *simp*
qed

Capítulo 11

Conclusiones y trabajo futuro

El tema central de esta tesis ha sido el estudio de la formalización del teorema de existencia de modelos en la lógica proposicional y de primer orden y sus aplicaciones en la formalización de metateoremas en estas lógicas, en correspondencia con el desarrollo de las pruebas naturales que aparecen en los textos de lógica y en particular en [14]; en el mismo sentido se estudió la formalización de la corrección y completitud de los sistemas deductivos de Hilbert y tableros semánticos para la lógica proposicional, y de deducción natural para la lógica de primer orden. Enseguida presentamos las conclusiones y líneas de trabajo futuro que se pueden desarrollar como continuación o complemento de este trabajo.

Se ha conseguido formalizar el desarrollo de las demostraciones estudiadas de manera aproximada a la forma como se proponen en los textos de lógica. Esta actividad, más allá de la "simple" verificación o certificación mecánica de resultados matemáticos, nos ha permitido tener una mayor aprehensión de los conceptos absractos involucrados en las pruebas de resultados de la metalógica clásica, y en particular de las propiedades utilizadas para la formalización constructiva del teorema de existencia de modelos. La legibilidad de sus presentaciones ha sido posible en gran parte a las características que tiene el lenguaje estructurado Isar para tal fin.

En el proceso de formalización de las demostraciones estructuradas, al interactuar con el sistema Isabelle, el método aplicativo resultó útil en algunos casos, para determinar qué resultados intermedios eran necesarios para la automatización de la pruebas respectivas; estos resultados por lo general correspondieron a propiedades de los tipos de datos que se definieron para representar en Isabelle los objetos matemáticos tratados. Por ejemplo, en el capítulo 4 para la formalización de la corrección del algoritmo de la forma normal conjuntiva se definieron y utilizaron un gran número de propiedades sobre listas relacionadas con la implementación del algoritmo. Este factor de implementación hace que la mecanización de teoremas sea una labor tediosa y poco atractiva para un matemático.

En general, se pudo comprobar que las pruebas formales resultan ser bastante extensas comparadas con las correspondientes pruebas informales. Esto se debe principalmente al número de propiedades intermedias que son necesarias utilizar en el proceso de la automatización. Algunas de estas corresponden, como se mencionó anteriormente, a las estructuras de datos empleadas en la representación de los objetos matemáticos que intervienen en las pruebas; otras tienen que ver con el número de casos análogos que hay que considerar y que no aportan conceptualmente nada nuevo en el desarrollo de la formalización de una prueba. Por ejemplo, en el capítulo 7 en la formalización del teorema de interpolación de Craig, a pesar de que la estructura de la demostración, basada en la aplicación del teorema de existencia de modelos proposicionales, es bastante sencilla, su formalización resultó muy extensa. Sin embargo, para no oscurecer las ideas principales de las pruebas, el sistema Isabelle permite, y así se hizo en este trabajo, "esconder" los resultados "técnicos" utilizados en la formalización de teorías. En general, La presentación de las demostraciones formalizadas en Isabelle/Isar se puede modificar de tal forma que se pueda elegir el nivel de detalle y los pasos esenciales de las pruebas.

Por último, la dificultad más importante que se presentó en el desarrollo del trabajo tuvo que ver con las formalización de las propiedades matemáticas "intuitivas" que se utilizan de manera implícita en la pruebas informales. En este sentido, el grado de informalidad y creatividad presentes en una prueba natural determina qué tan compleja resulta la formalización de la misma.

Se espera que las teorías desarrolladasn es esta tesis sirvan de apoyo desde el punto de vista del aprendizaje de la lógica para verificar la solución de problemas relacionados con los temas tratados, por ejemplo en la solución de los ejercicios correspondientes que se proponen en los primeros cinco capítulos del libro de Fitting [14] y el capítulo 1 del libro de Mendelson [36], de tal forma que permita al estudiante afianzar los conceptos correspondientes mediante la verificación de la solución informal. En forma más general, como se indica más adelante, modificar las teorías con el fin de formalizar otros teoremas importantes de la metalógica.

También se espera que las ideas presentadas en este trabajo contribuyan en la investigación actual (ver por ejemplo, [9], [40]) que se realiza sobre las características que se deben tener en cuenta, y en particular las que debe poseer un probador autómatico de teoremas, para la certificación de las demostraciones matemáticas de la misma forma como se realizan con lápiz y papel.

Algunos de los trabajos futuros que se pueden desarrollar en relación con los temas estudiados en esta tesis se describen a continuación.

En el capítulo 3 se definió un sistema axiomático de Hilbert para la lógica proposicional y se fomalizó, en particular, la completitud del sistema de manera directa: si una fórmula es tautología, entonces es derivable en el sistema. Para la demostración se uti-

lizó el método de eliminación de variables con el fin de formalizar la prueba clásica que aparece en los libros de lógica en la que se utiliza el *teorema de deducción*. Otra forma de demostrar esta propiedad es de manera indirecta: si una fórmula no es derivable en el sistema, entonces no es tautología. En este caso, para su formalización se puede utilizar el teorema de existencia de modelos en la lógica proposicional (teorema 6.11.4).

El procedimiento de prueba en la lógica proposicional basado en *resolución* [47] puede implementarse a partir de un algoritmo que permita, convertir una fórmula en forma normal conjuntiva y aplicar la *regla de resolución de Robinson*. Por lo tanto, con base a la formalización del algoritmo de la forma normal conjuntiva que se estudió en el capítulo 4, puede formalizarse este cálculo de prueba basado en resolución. De igual forma, este algoritmo sirve para formalizar el procedimiento de Davis–Putnam [62] el cual es un método de prueba para decidir satisfactibilidad de una fórmula proposicional en forma normal conjuntiva.

En el capítulo 6 se formalizó usando el *lema de Hintikka* la completitud del método de prueba (def. 5.2.1) basado en tableros *estrictos* (def. 5.1.2). En el caso de que no se exiga esta restricción, la completitud de este método de cálculo basado en tableros semánticos puede formalizarse usando el teorema de existencia de modelos y también para la formalización del cálculo de prueba basado en resolución [14] (páginas 65-67).

En este capítulo también se formalizó el teorema de existencia de modelos en la lógica proposicional con base a la noción abstracta de *propiedad de consistencia proposicional* y a los conceptos de *conjunto maximal consistente* y *conjunto de Hintikka*. La construcción de un conjunto maximal consistente dependió de una enumeración del conjunto de símbolos proposicionales. Sin embargo, el teorema de existencia de modelos sigue siendo verdadero en lenguajes que tienen un conjunto no enumerable de símbolos proposicionales. En este caso, para su formalización es necesario utilizar el axioma de elección o el Lema de Zorn.

En el capítulo 7, como una aplicación del teorema de existencia de modelos en la lógica proposicional, se formalizaron los teoremas de compacidad y de interpolación de Craig que tienen importantes consecuencias en la teoría de modelos de la lógica de primer orden. El teorema de compacidad también tiene aplicaciones directas en la solución de problemas de matemáticas discretas. Por ejemplo, a partir de la formalización del teorema de compacidad es posible formalizar el problema sobre coloración de grafos infinitos que se estudia en [26] (página 109).

En el capítulo 8 se formalizó la sintaxis de la lógica de primer orden y se mostró que el uso de la notación de de Bruijn facilita la mecanización del proceso de *sustitución* pero dificulta su manejo intuitivo. En este sentido, un tema de investigación es explorar la posibilidad de rehacer la formalización del teorema de existencia de modelos utilizando otro formalismo para la representación de las fórmulas, por ejemplo, como se propone en [24] (página 4). Adicionalmente, de forma similar a como se hizo para

el caso proposicional, las fórmulas de primer orden pueden clasificarse con base a la noción de fórmulas alfa, beta, delta y gama. Esta notación uniforme para las fórmulas permite reducir la longitud de las pruebas.

En el capítulo 9 se formalizó el teorema de existencia de modelos en la lógica de primer orden de manera similar a la formalización desarrollada en capítulo 6 para el caso proposicional. Sin embargo, como consecuencia de la adición de fórmulas cuantificadas fue necesario introducir la noción de *propiedad de consistencia alternativa*, formalizar de nuevo las propiedades sobre conjuntos maximales y, en general, aquellas que para su demostración requieren de un conjunto infinito de *parámetros* o símbolos de función. De igual forma, para enunciar y demostrar el lema de Hintikka para conjuntos de fórmulas de primer orden, se formalizó la noción de *estructuras de Herbrand*.

Como aplicaciones de la formalización del teorema de existencia de modelos en la lógica de primer orden se formalizó la demostración del teorema de Löwenheim-Skolem y la completitud del sistema de deducción natural estudiado en el capítulo 10. Otras aplicaciones son el teorema de compacidad de primer orden, que se formaliza igual que el caso proposicional que se presentó en capítulo 7, el teorema de interpolación de Craig, que se formaliza en forma muy similar al caso proposicional, y la completitud de los cálculos de prueba basados en tableros semánticos y resolución.

También se tiene una extensión directa de la formalización del teorema de existencia de modelos a la lógica de primer orden con *igualdad* [14] (página 284). El único trabajo adicional se presenta en la formalización de la nueva versión del lemma de Hintikka que en [14] (página 281) se formula en términos de los conceptos de *modelo canónico* y *modelo normal*. De igual forma, en esta versión del teorema se tienen las aplicaciones correspondientes descritas anteriormente.

El *teorema de Herbrand* [14] (páginas 216-220), que permite reducir el problema de la validez lógica en el cálculo de predicados al caso de fórmulas sin cuantificadores y al cálculo puramente proposicional, es otro importante teorema que puede ser formalizado a partir de la formalización del teorema se existencia de modelos en la lógica de primer orden (sin igualdad).

Bibliografía

[1] J. R. A. La ciencia cognitiva como disciplina unificada. *Informática Educativa Proyecto SIIE, Colombia*, 6(2):95–108, 1993. www.colombiaaprende.edu.co/html/.../articles-127655_archivo.pdf.

[2] C. S. C. Andrea Asperti, Wilmer Ricciotti and E. Tassi. Formal metatheory of programming languages in the Matita interactive theorem prover. *Kluwer Academic Publishers*, 2011. http://www.msr-inria.inria.fr/~gares/jar09.pdf.

[3] A. Asperti. A survey on Interactive Theorem Proving. Technical report, Department of Computer Science, University of Bologna, 2009. http://www.cs.unibo.it/~asperti/SLIDES/itp.pdf.

[4] J. Avigad. Notes on a formalization of the prime number theorem. Technical report, Carnegie Mellon University, 2004. http://www.andrew.cmu.edu/user/avigad/Papers/pntnotes.pdf.

[5] G. Bauer, D. v. O. Tobias Nipkow, L. C. Paulson, T. M. Rasmussen, C. Tabacznyj, and M. Wenzel. *The supplemental Isabelle/HOL library*, 2002. http://isabelle.in.tum.de/library/HOL/Library/document.pdf.

[6] S. Berghofer. *Meta-Theory of first-order predicate logic*, 2009. http://afp.sourceforge.net/release/afp-FOL-Fitting-2010-06-30.tar.gz.

[7] S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In: J. Cuellar, Z. Liu (eds.) Software Engineering and Formal Methods (SEFM 2004). *IEEE Computer Society*, 10:230–239, 2004.

[8] Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors. *Theorem Proving in Higher Order Logics: TPHOLs '99*, volume 1690 of *Lecture Notes in Computer Science*, 1999.

[9] M. Caminati. *A simplified framework for first -order languages and its formalization in Mizar*. PhD thesis, University of Rome, 2011. http://www.mat.uniroma1.it/~caminati/CaminatiTesi20111130_09.16.44.pdf.

[10] A. Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, pages 1–46, 2011. http://www.chargueraud.org/research/2009/ln/main.pdf.

[11] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic 5, no. 2*, pages 56–68, 1940.

[12] M. Davis. *The Early History of Automated Deduction, chapter 1 of Handbook of Automated Reasoning*. Elsevier Science Publishers B. V., 2001. http://http://bit.ly/ozd1O8.

[13] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 75:381–392, 1972.

[14] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition, 1996.

[15] G. K. y. C. P.-M. G. Huet. The coq proof assistant a tutorial. Technical report, Inria research center, 2007. http://coq.inria.fr/V8.1/files/doc/Tutorial.pdf.

[16] H. Geuvers. Proof assistants: History, ideas and future. *Sadhan*, 34:3–25, 2009. http://www.ias.ac.in/sadhana/Pdf2009Feb/3.pdf.

[17] G. Gonthier. A computer-checked proof of the Four Colour Theorem. Technical report, Microsoft Research, 2004. http://research.microsoft.com/en-us/people/gonthier/4colproof.pdf.

[18] M. Gordon. From lcf to hol: a short history. *Proof, Language, and Interaction*, pages 169–185, 2000.

[19] M. J. C. Gordon and A. M. Pitts. The HOL logic and system, towards verified systems (j. bowen, ed.), real-time safety critical systems. *Elsevier Science B.V.*, 2:49–70, 1994.

[20] J. Grundy and M. Newey, editors. *Theorem Proving in Higher Order Logics: TPHOLs '98*, volume 1479 of *Lecture Notes in Computer Science*, 1998.

[21] T. C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005. http://www.math.pitt.edu/~thales/papers/A%20Proof%20of%20the%20Kepler%20Conjecture%20(Annals).pdf.

[22] T. C. Hales. The Jordan Curve Theorem, Formally and Informally. Technical report, University of Pittsburgh, 2007. http://www.math.pitt.edu/~thales/papers/The%20Jordan%20Curve%20Theorem,%20Formally%20and%20Informally.pdf.

[23] J. Harrison. *Formalized Mathematics*, 1996. http://www.cl.cam.ac.uk/users/jrh/papers/reflect.html.

[24] J. Harrison. Formalizing Basic First Order Model Theory. *Theorem Proving in Higher Order Logics*, 1497:153–170, 2008. http://www.cl.cam.ac.uk/~jrh13/papers/model.ps.gz.

[25] J. Harrison. Formalizing an analytic proof of the Prime Number Theorem . *Journal of Automated Reasoning*, 43:243–261, 2009. http://www.cl.cam.ac.uk/~jrh13/papers/mikefest.pdf.

[26] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[27] J. Harrison. Hol light tutorial (for version 2.20). Technical report, Cambridge University, 2011. http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial_220.pdf.

[28] J. V. Heijenoort. *From Frege to Gödel: a source book in mathematical logic, 1879-1931*. Harvard University Press, 2002.

[29] D. Hendriks. *Metamathematics in Coq*. PhD thesis, Utrecht University, 2003. http://igitur-archive.library.uu.nl/dissertations/2003-1103-120942/full.pdf.

[30] P. Janičić. Automated reasoning: some successes and new challenges. Technical report, Faculty of Mathematics, University of Belgrade, Serbia, 2011. http://argo.matf.bg.ac.rs/publications/2011/CECIIS-AR-Janicic.pdf.

[31] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. http://isabelle.in.tum.de/doc/functions.pdf.

[32] A. J. M. M. J. Gordon and C. P. Wadsworth. *Edinburgh LCF*, volume 78. Springer-Verlag, 1979.

[33] D. Mackeanzie. Computers and the Sociology of Mathematical Proof. Technical report, Sociology, School of Social and Political Science, University of Edinburgh, 1998. http://goo.gl/vHnQJ.

[34] L. C. P. Markus Wenzel and T. Nipkow. *The Isabelle Framework*. http://www4.informatik.tu-muenchen.de/~nipkow/pubs/tphols08t.html.

[35] F. J. Martín. *Teoría Computacional (en ACL2) sobre cálculos proposicionales*. PhD thesis, Universidad de Sevilla, 2002. http://www.glc.us.es/~jalonso/trabajos_dirigidos/2002-tesis-FJMM.pdf.

[36] E. Mendelson. *Introduction to mathematical logic*. CRC Press, 2001.

[37] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs*. http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf.

[38] T. Nipkow. Linear quantifier elimination. *Journal of Automated Reasoning*, 45:189–212, 2010. http://www4.informatik.tu-muenchen.de/~nipkow/pubs/jar10.pdf.

[39] T. Nipkow. *Interactive Proof: Introduction to Isabelle Hol*, 2011. http://asimod.in.tum.de/2011/slides/nipkow-slides.pdf.

[40] S. Obua. A cloud-based interactive theorem proving system. Technical report, Cornell UNiversity, 2012. http://arxiv.org/abs/1201.0540.

[41] M. Ojeda. Lógica, Matemática, Deducción Automática. *Gaceta de la Real Sociedad Matemática Española*, 8(1):93–119, 2005. http://goo.gl/Updrb.

[42] R. O'Connor. *Incompleteness and Completeness: Formalizing Logic and Analysis in Type Theory*. PhD thesis, Faculty of Science, Mathematics and Computer Science, RU, 2009. http://webdoc.ubn.ru.nl/mono/o/oconnor_r/incoanco.pdf.

[43] L. C. Paulson. *Introduction to Isabelle*. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2011/doc/intro.pdf.

[44] L. C. Paulson. *Designing a Theorem Proving*, 1990. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-192.pdf.

[45] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Lecture Notes in Computer Science, vol 828, Springer-Verlag, 1994.

[46] L. C. Paulson. A generic tableau prover and its integration with isabelle. *Journal of Universal Computer Science, 5(3)*, 1999.

[47] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[48] N. Shankar. *Metamathematics, Machines, and Goedel's Proof*. Cambridge University Press, 1994.

[49] J. Siek. Practical Theorem Proving with Isabelle/Isar Lecture Notes. Technical report, University of Colorado, 2007. http://www.cs.colorado.edu/~siek/7000/spring07/isabelle-notes.pdf.

[50] R. M. Smullyan. *First-Order Logic*. Dover Publications Inc., 1994.

[51] L. C. P. Tobias Nipkow and M. Wenzel. *Isabelle's Logics: HOL*. http://isabelle.in.tum.de/doc/logics-HOL.pdf.

[52] L. C. P. Tobias Nipkow and M. Wenzel. *Isabelle/HOL: A proof assistant for higher-order logic*. Springer-Verlag, 2002. http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf.

[53] T. Weber. Bounded model generation for Isabelle/HOL. In: W. Ahrendt, P. Baumgartner, H de Nivelle, S. Ranise, C Tinelle (eds.) Workshops Disproving and Pragmatics of Decision Procedures (PDPAR 2004). *ENTCS Computer Society. Elsevier (2005)*, 125:103–116, 2004.

[54] M. Wenzel. *Isabelle/Isar — a generic framework for human-readable proof documents*. http://www4.in.tum.de/~wenzelm/papers/isar-framework.pdf.

[55] M. Wenzel. *The Isabelle/Isar Reference Manual*. http://isabelle.in.tum.de/dist/Isabelle/doc/isar-ref.pdf.

[56] M. Wenzel. *Miscellaneous Isabelle/Isar examples for Higher-Order Logic*. http://isabelle.in.tum.de/dist/library/HOL/Isar_examples/index.html.

[57] M. Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Bertot et al. [8]. http://www4.in.tum.de/~wenzelm/papers/Isar-TPHOLs99.pdf.

[58] M. Wenzel. *Isabelle/Isar - a versatile environment for human-readable formal proof documents*. PhD thesis, Institut für Informatik, Technische Universiät München, 2002. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html.

[59] F. Wiedijk. *Formalizing 100 theorems*. htpp://www.cs.ru.nl/~freek/100/.

[60] F. Wiedijk. Mizar: An impression. Technical report, Computing Science Department, Radboud University Nijmegen, 2011. http://www.cs.ru.nl/~freek/mizar/mizarintro.ps.gz.

[61] A. N. y H. A. Simon. *Human Problem Solving*. Prentice-Hall Englewood Cliffs, 1972.

[62] M. D. y H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[63] M. K. y J.S. Moore. Industrial proofs with acl2. Technical report, Department of Computer Sciences, University of Texas at Austin. http://www.cs.utexas.edu/~moore/publications/how-to-prove-thms/intro-to-acl2.pdf.

[64] M. H. y M. Ryan. *Logic in Computer Science*. Cambridge University Press, 2004.

[65] W. M. y R. Murawski. *Mechanization of Reasoning in a Historical Perspective*. Rodopi Bv Editions, 1995.

[66] M. K. y R. S. Boyer. The Boyer-Moore Theorem Prover and Its Interactive Enhancement. *Computers and Mathematics with Applications*, 29(2):27–62, 1995. http://www.cs.utexas.edu/users/boyer/wos.ps.Z.