Elementos de lógica formalizados en Isabelle/HOL

Facultad de Matemáticas
Departamento de Ciencias de la Computación e Inteligencia Artificial
Trabajo Fin de Grado

Sofía Santiago Fernández

El presente Trabajo Fin de Grado se ha realizado en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla y ha sido supervisado por José Antonio Alonso Jiménez y María José Hidalgo Doblado.

Índice

Sumario

El objetivo de la Lógica es la formalización del conocimiento y su razonamiento. En este trabajo, estudiaremos elementos de la lógica proposicional desde la perspectiva teórica de *First−Order Logic and Automated Theorem Proving* [4] de Melvin Fitting. En particular, nos centraremos en la sintaxis y la semántica, concluyendo con la versión proposicional del lema de Hintikka sobre la satisfacibilidad de una clase determinada de conjuntos de fórmulas. Siguiendo la inspiración de *Propositional Proof Systems* [10] por Julius Michaelis y Tobias Nipkow, los resultados expuestos serán formalizados mediante Isabelle: un demostrador interactivo que incluye herramientas de razonamiento automático para guiar al usuario en el proceso de formalización, verificación y automatización de resultados. Concretamente, Isabelle/HOL es una especialización de Isabelle para la lógica de orden superior. Las demostraciones de los resultados en Isabelle/HOL se elaborarán siguiendo dos tácticas distintas a lo largo del trabajo. En primer lugar, cada lema será probado de manera detallada prescindiendo de toda herramienta de razonamiento automático, como resultado de una búsqueda inversa en cada paso de la prueba. En contraposición, elaboraremos una demostración automática alternativa de cada resultado que utilice todas las herramientas de razonamiento automático que proporciona el demostrador. De este modo, se evidenciará la capacidad de razonamiento automático de Isabelle.

Logic's purpose is about knowledge's formalisation and its reasoning. In this project, we will approach Propositional Logic's elements from the theoretical perspective of *First−Order Logic and Automated Theorem Proving* [4] by Melvin Fitting. We will focus on the study of Syntax and Semantics to reach propositional version of Hintikka's lemma, which determinate the satisfiability of a concrete type of formula set. Inspired by *Propositional Proof Systems* [10] by Julius Michaelis and Tobias Nipkow, these results will be formalised using Isabelle: a proof assistant including automatic reasoning tools to guide the user on formalising, verifying and automating results. In particular, Isabelle/HOL is the specialization of Isabelle for High-Order Logic. The processing of the results formalised in Isabelle/HOL follows two directions. In the first place, each lemma will be proved on detail without any automation, as the result of an inverse research on every step of the demonstration until it is only completed with deductions based on elemen-

tary rules and definitions. Conversely, we will alternatively prove the results using all the automatic reasoning tools that are provide by the proof assistant. In this way, Isabelle's power of automatic reasoning will be shown as the contrast between these two different proving tactics.

Introducción

El objetivo de la Lógica es la formalización del conocimiento y el razonamiento sobre el mismo. Tiene su origen en la Antigua Grecia con Aristóteles y su investigación acerca de los principios del razonamiento válido o correcto, recogidos fundamentalmente en su obra *Organon*. De este modo, dio lugar a la lógica silogística, que consistía en la deducción de conclusiones a partir de dos premisas iniciales.

Posteriormente, los estoicos (400-200 a.C) comenzaron a cuestionarse temas relacionados con la semántica, como la naturaleza de la verdad. Formularon la *paradoja del mentiroso*, que plantea una incongruencia acerca de la veracidad del siguiente predicado.

Esta oración es falsa.

Sin embargo, no fue hasta el siglo XVII que el matemático y filósofo Gottfried Wilhelm Leibniz (1646 – 1716) instaura un programa lógico que propone la búsqueda de un sistema simbólico del lenguaje natural junto con la matematización del concepto de validez. Estas ideas fueron la principal motivación del desarrollo de la lógica moderna del siglo XIX de la mano de matemáticos y filósofos como Bernard Bolzano (1781 – 1848), George Boole (1815 – 1864), Charles Saunders Pierce (1839 – 1914) y Gottlob Frege (1848 – 1925). Fue este último quien introdujo el primer tratamiento sistemático de la lógica proposicional. Frege basó su tesis en el desarrollo de una sintaxis completa que combina el razonamiento de deducción de la silogística aristotélica con la noción estoica de conectivas para relacionar ideas. Paralelamente desarrolló una semántica asociada a dicha sintaxis que permitiese verificar la validez de los procesos deductivos. La lógica proposicional de Frege formó parte de la escuela denominada logicismo. Su objetivo consistía en investigar los fundamentos de las matemáticas con el fin de formalizarlos lógicamente, para así realizar deducciones y razonamientos válidos.

En las últimas décadas, el desarrollo de la computación y la inteligencia artificial ha permitido la formalización de las matemáticas y la lógica mediante el lenguaje computacional. Concretamente, el razonamiento automático es un área que investiga los distintos aspectos del razonamiento con el fin de crear programas y algoritmos para razonar de manera prácticamente automática. Se fundamenta en el programa lógico desarrollado por Leibniz, estructurado en base a dos principios: la formalización rigurosa

de resultados y el desarrollo de algoritmos que permitan manipular y razonar a partir de dichas formalizaciones. Entre las principales aplicaciones de este áres se encuentra la verificación y síntesis automáticas de programas. De este modo, podemos validar distintos razonamientos, así como crear herramientas de razonamiento automático que permitan el desarrollo de nuevos resultados.

En este contexto nace Isabelle en 1986, desarrollada por Larry Paulson de la Universidad de Cambridge y Tobias Nipkow del Techniche Universität München. Isabelle es un demostrador interactivo que, desde el razonamiento automático, facilita la formalización lógica de resultados y proporciona herramientas para realizar deducciones. En particular, Isabelle/HOL es la especialización de Isabelle para la lógica de orden superior. Junto con Coq, ACL2 y PVS, entre otros, constituye uno de los demostradores interactivos más influyentes.

Como demostrador interactivo, Isabelle permite automatizar razonamientos guiados por el usuario, verificando cada paso de una deducción de manera precisa. Además, incorpora herramientas de razonamiento automático para mejorar la productividad del proceso de demostración. Para ello, cuenta con una extensa librería de resultados lógicos y matemáticos que han sido formalizados y continúan en desarrollo por parte de proyectos como *The Alexandria Project*: *Large−Scale Formal Proof for the Working Mathematician*. Este proyecto comienza en 2017, dirigido por Lawrence Paulson desde la Universidad de Cambridge. Tiene como finalidad la formalización de distintas teorías para ampliar la librería de Isabelle, junto con la creación de herramientas interactivas que asistan a los matemáticos en el proceso de formalización, demostración y búsqueda de nuevos resultados.

El objetivo de este trabajo es la formalización de elementos y resultados destacados de la lógica proposicional en Isabelle/HOL. Está inspirado en la primera sección de la publicación *Propositional Proof Systems* [*10*] de Julius Michaelis y Tobias Nipkow. Del mismo modo, cabe citar los artículos *Constructive Formalization of Classical Modal Logic* [*3*] de Christian Doczkal y Gert Smolka, y *Propositional Calculus in Coq* [*13*] de Floris van Doorn, por la influencia ejercida en el desarrollo de este trabajo. El contenido teórico del mismo se fundamenta en el libro *First−Order Logic and Automated Theorem Proving* [*4*] de Melvin Fitting. Los tres capítulos tratados consisten en la sintaxis, semántica y, finalmente, la versión proposicional del lema de Hintikka. Este último fue desarrollado por el filósofo y lógico Jaakko Hintikka (1929- 2015) como herramienta para probar la completitud de la lógica de primer orden.

En el primer capítulo sobre sintaxis se establecen inicialmente las variables proposicionales que conforman los elementos básicos del alfabeto, junto con una serie de conectivas que actúan sobre ellas. De este modo, se define por recursión el conjunto de las fórmulas proposicionales como el menor conjunto de estructuras sintácticas con dicho alfabeto y conectivas que contiene a las fórmulas básicas (una constante $\bot$ y las propias

variables proposicionales, llamadas fórmulas atómicas) y es cerrado mediante procedimientos de formación de nuevas fórmulas a partir de otras, en los que intervienen las conectivas. Como es habitual, dada esta definición recursiva, se dispone de un esquema de inducción sobre fórmulas que nos permitirá probar los resultados expuestos. Del mismo modo, se define recursivamente el conjunto de subfórmulas de una fórmula, mostrando propiedades que describen la estructura de las mismas en relación con las propias fórmulas. Finalmente se presenta la fórmula $\top$ a partir de la constante $\bot$, y dos conectivas generalizadas que permiten extender conectivas binarias a una lista de fórmulas.

En el siguiente capítulo precisamos la semántica asociada a las estructuras sintácticas. Para ello, se define una interpretación como una aplicación que asocia un booleano a cada variable proposicional. Por recursión sobre la estructura de las fórmulas proposicionales, podemos definir el valor de una fórmula en una interpretación dada. De este modo, se prueba que dicho valor queda unívocamente determinado por la imagen que la interpretación asocia a cada variable proposicional que aparece en la fórmula. Las nociones semánticas se extienden análogamente a las fórmulas formadas con conectivas generalizadas.

Posteriormente se introducen dos definiciones semánticas fundamentales: modelo de una fórmula y fórmula satisfacible. La primera hace referencia a una interpretación en la que el valor de una fórmula dada es verdadero, mientras la segunda se trata de una fórmula para la que existe una interpretación que sea modelo suyo. Ambas nociones se extienden a conjuntos de fórmulas. Por otro lado, se define el concepto de tautología como aquella fórmula cuyo valor es verdadero en toda interpretación. Para concluir la sección, daremos una noción formal de consecuencia lógica.

El capítulo tercero, y último, tiene como objetivo probar el lema de Hintikka, que manifiesta la satisfacibilidad de ciertos conjuntos de fórmulas. Para ello define dicha clase de conjuntos, llamados conjuntos de Hintikka, de modo que para cada uno de ellos se determina paralelamente una interpretación asociada, garantizando que esta es modelo de cada fórmula del conjunto.

En lo referente a las demostraciones asistidas por Isabelle/HOL de los resultados formalizados a lo largo de las secciones, se elaborarán dos tipos de pruebas correspondientes a dos tácticas distintas. En primer lugar, se probará cada resultado siguiendo un esquema de demostración detallado. En él utilizaremos únicamente y de manera precisa las reglas de simplificación y definiciones incluidas en la librería de Isabelle, prescindiendo de las herramientas de razonamiento automático del demostrador. Para ello, se realiza una búsqueda inversa en cada paso de la demostración automática hasta llegar a un desarrollo de la prueba basado en deducciones a partir de resultados elementales que la completen de manera rigurosa. En contraposición, se evidenciará la capacidad de razonamiento automático de Isabelle/HOL mediante la realización de una

prueba alternativa siguiendo un esquema de demostración automático. Para ello se utilizarán las herramientas de razonamiento que han sido elaboradas en Isabelle/HOL con el objetivo de realizar deducciones de la manera más eficiente.

Este trabajo está disponible en la plataforma GitHub *mediante el siguiente enlace*:

https://github.com/sofsanfer/TFG

Capítulo 1

Sintaxis

1.1 Fórmulas

En esta sección presentaremos una formalización en Isabelle de la sintaxis de la lógica proposicional, junto con resultados y pruebas sobre la misma. En líneas generales, primero daremos las nociones de forma clásica y, a continuación, su correspondiente formalización.

En primer lugar, supondremos que disponemos de los siguientes elementos:

Alfabeto: Es una lista infinita de variables proposicionales. También pueden ser llamadas átomos o símbolos proposicionales.

Conectivas: Conjunto finito cuyos elementos interactúan con las variables. Pueden ser monarias que afectan a un único elemento o binarias que afectan a dos. En el primer grupo se encuentra la negación ($\neg$) y en el segundo la conjunción ($\wedge$), la disyunción ($\vee$) y la implicación ($\longrightarrow$).

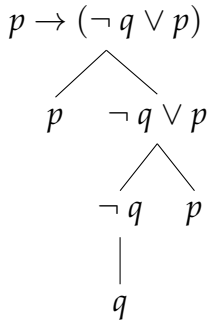
A continuación definiremos la estructura de fórmula sobre los elementos anteriores. Para ello daremos una definición recursiva basada en dos elementos: un conjunto de fórmulas básicas y una serie de procedimientos de definición de fórmulas a partir de otras. El conjunto de las fórmulas será el menor conjunto de estructuras sintácticas con dicho alfabeto y conectivas que contiene a las básicas y es cerrado mediante los procedimientos de definición que mostraremos a continuación.

Definición 1.1.1 *El conjunto de las fórmulas proposicionales está formado por las siguientes:*

- *Las fórmulas atómicas, constituidas únicamente por una variable del alfabeto.*
- *La constante $\bot$.*

- *Dada una fórmula F, la negación* $\neg F$ *es una fórmula.*
- *Dadas dos fórmulas F y G, la conjunción* $F \wedge G$ *es una fórmula.*
- *Dadas dos fórmulas F y G, la disyunción* $F \vee G$ *es una fórmula.*
- *Dadas dos fórmulas F y G, la implicación* $F \longrightarrow G$ *es una fórmula.*

Intuitivamente, las fórmulas proposicionales son entendidas como un tipo de árbol sintáctico cuyos nodos son las conectivas y sus hojas las fórmulas atómicas. Veamos, por ejemplo, el árbol sintáctico de la fórmula $p \to (\neg\, q \vee p)$.



A continuación, veamos la representación en Isabelle de la estructura de las fórmulas proposicionales.

datatype (*atoms*: *'a*) *formula* =
Atom 'a
| *Bot* (⊥)
| *Not 'a formula* (¬)
| *And 'a formula 'a formula* (**infix** ∧ *68*)
| *Or 'a formula 'a formula* (**infix** ∨ *68*)
| *Imp 'a formula 'a formula* (**infixr** → *68*)

Como podemos observar representamos las fórmulas proposicionales mediante un tipo de dato recursivo, *formula*, con los siguientes constructores sobre un tipo cualquiera:

Fórmulas básicas :

- *Atom* :: *'a* ⇒ *'a formula*
- ⊥ :: *'a formula*

Fórmulas compuestas :

- ¬ :: *'a formula* ⇒ *'a formula*
- (∧) :: *'a formula* ⇒ *'a formula* ⇒ *'a formula*

- $(\vee) :: 'a\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
- $(\rightarrow) :: 'a\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

Cabe señalar que los términos *infix* e *infixr* nos señalan que los constructores que representan a las conectivas se pueden usar de forma infija. En particular, *infixr* se trata de un infijo asociado a la derecha.

Por otro lado, la definición de *formula* genera automáticamente los siguientes lemas sobre la función $atoms :: 'a\ formula \Rightarrow 'a\ set$, que obtiene el conjunto de átomos de una fórmula.

$atoms\ (Atom\ x1.0) = \{x1.0\}$
$atoms\ \bot = \emptyset$
$atoms\ (\neg\ x3.0) = atoms\ x3.0$
$atoms\ (x41.0 \wedge x42.0) = atoms\ x41.0 \cup atoms\ x42.0$
$atoms\ (x51.0 \vee x52.0) = atoms\ x51.0 \cup atoms\ x52.0$
$atoms\ (x61.0 \rightarrow x62.0) = atoms\ x61.0 \cup atoms\ x62.0$

A continuación veremos varios ejemplos de fórmulas y el conjunto de sus variables proposicionales obtenido mediante *atoms*. Se observa que, por ser conjuntos, no contienen elementos repetidos.

```
notepad
begin
 fix p q r :: 'a

 have atoms (Atom p) = {p}
  by (simp only: formula.set)

 have atoms (¬ (Atom p)) = {p}
  by (simp only: formula.set)

 have atoms ((Atom p → Atom q) ∨ Atom r) = {p,q,r}
  by auto

 have atoms ((Atom p → Atom p) ∨ Atom r) = {p,r}
  by auto
end
```

En particular, el conjunto de símbolos proposicionales de la fórmula *Bot* es vacío. Además, para calcular esta constante es necesario especificar el tipo sobre el que se construye la fórmula.

```
notepad
begin
 fix p :: 'a

 have atoms ⊥ = ∅
  by (simp only: formula.set)

 have atoms (Atom p ∨ ⊥) = {p}
 proof −
  have atoms (Atom p ∨ ⊥) = atoms (Atom p) ∪ atoms Bot
   by (simp only: formula.set(5))
  also have ... = {p} ∪ atoms Bot
   by (simp only: formula.set(1))
  also have ... = {p} ∪ ∅
   by (simp only: formula.set(2))
  also have ... = {p}
   by (simp only: Un-empty-right)
  finally show atoms (Atom p ∨ ⊥) = {p}
   by this
 qed

 have atoms (Atom p ∨ ⊥) = {p}
  by (simp only: formula.set Un-empty-right)
end

value (Bot::nat formula)
```

Una vez definida la estructura de las fórmulas, vamos a introducir el método de demostración que seguirán los resultados que aquí presentaremos, tanto en la teoría clásica como en Isabelle.

Según la definición recursiva de las fórmulas, dispondremos de un esquema de inducción sobre las mismas:

Teorema 1.1.2 (Principio de inducción sobre fórmulas proposicionales) *Sea $\mathcal{P}$ una propiedad sobre fórmulas que verifica las siguientes condiciones:*

- *Las fórmulas atómicas la cumplen.*

- *La constante $\bot$ la cumple.*
- *Dada F fórmula que la cumple, entonces $\neg$ F la cumple.*
- *Dadas F y G fórmulas que la cumplen, entonces F $*$ G la cumple, donde $*$ simboliza cualquier conectiva binaria.*

Entonces, todas las fórmulas proposicionales tienen la propiedad $\mathcal{P}$.

Análogamente, como las fórmulas proposicionales están definidas mediante un tipo de datos recursivo, Isabelle genera de forma automática el esquema de inducción correspondiente. De este modo, en las pruebas formalizadas utilizaremos la táctica *induction*, que corresponde al siguiente esquema.

$\bigwedge x.\ P(Atom\ x)$

$P\ \bot$

$\bigwedge x.\ P\ x \Longrightarrow P(\neg\ x)$

$\bigwedge x1\ x2.\ P\ x1 \wedge P\ x2 \Longrightarrow P\ (x1 \wedge x2)$

$\bigwedge x1\ x2.\ P\ x1 \wedge P\ x2 \Longrightarrow P\ (x1 \vee x2)$

$\bigwedge x1\ x2.\ P\ x1 \wedge P\ x2 \Longrightarrow P\ (x1 \rightarrow x2)$

P formula

Como hemos señalado, el esquema inductivo genera así seis casos distintos como se muestra anteriormente. Además, todas las demostraciones sobre casos de conectivas binarias son equivalentes en esta sección, pues la construcción sintáctica de fórmulas es idéntica entre ellas. Estas se diferencian esencialmente en la connotación semántica que veremos más adelante.

A continuación el primer resultado de este apartado:

Lema 1.1.3 *El conjunto de los átomos de una fórmula proposicional es finito.*

Para proceder a la demostración, consideremos la siguiente definición inductiva de conjunto finito. Cabe añadir que la demostración seguirá el esquema inductivo relativo a la estructura de fórmula, y no el que induce esta última definición.

Definición 1.1.4 *Los conjuntos finitos son:*

- *El vacío.*

- *Dado un conjunto finito A y un elemento cualquiera a, entonces* $\{a\} \cup A$ *es finito.*

La formalización en Isabelle de la definición anterior es precisamente *finite* perteneciente a la teoría FiniteSet.thy. Dicha definición inductiva genera dos reglas análogas a las anteriores que definen a los conjuntos finitos y que emplearemos en la demostración del resultado.

$$\textit{fold-graph}\ f\ z\ \emptyset\ z \tag{\textit{emptyI}}$$

$$\frac{x \notin A \wedge \textit{fold-graph}\ f\ z\ A\ y}{\textit{fold-graph}\ f\ z\ (\{x\} \cup A)\ (f\ x\ y)} \tag{\textit{insertI}}$$

De este modo, en Isabelle podemos especificar el lema como sigue.

lemma *finite* (*atoms F*)
oops

A continuación, veamos la demostración clásica del lema.

Demostración: La prueba es por inducción sobre el tipo recursivo de las fórmulas. Veamos cada caso.

Consideremos una fórmula atómica p cualquiera. Entonces, su conjunto de variables proposicionales es $\{p\}$, finito.

Sea la fórmula $\bot$. Entonces, su conjunto de átomos es vacío y, por lo tanto, finito.

Sea F una fórmula cuyo conjunto de variables proposicionales sea finito. Entonces, por definición, $\neg F$ y F tienen igual conjunto de átomos y, por hipótesis de inducción, es finito.

Consideremos las fórmulas F y G cuyos conjuntos de átomos son finitos. Por construcción, el conjunto de variables de $F * G$ es la unión de sus respectivos conjuntos de átomos para cualquier $*$ conectiva binaria. Por lo tanto, usando la hipótesis de inducción, dicho conjunto es finito.

□

Veamos ahora la prueba detallada en Isabelle. Mostraremos con detalle todos los casos de conectivas binarias, aunque se puede observar que son completamente análogos. Para facilitar la lectura, primero demostraremos por separado cada uno de los casos según el esquema inductivo de fórmulas, y finalmente añadiremos la prueba para una fórmula cualquiera a partir de los anteriores.

lemma *atoms-finite-atom*:
finite (*atoms* (*Atom x*))

proof −
have *finite* ∅
by (*simp only*: *finite.emptyI*)
then have *finite* {*x*}
by (*simp only*: *finite-insert*)
then show *finite* (*atoms* (*Atom x*))
by (*simp only*: *formula.set*(*1*))
qed

lemma *atoms-finite-bot*:
finite (*atoms* ⊥)
proof −
have *finite* ∅
by (*simp only*: *finite.emptyI*)
then show *finite* (*atoms* ⊥)
by (*simp only*: *formula.set*(*2*))
qed

lemma *atoms-finite-not*:
assumes *finite* (*atoms F*)
shows *finite* (*atoms* (¬ *F*))
using *assms*
by (*simp only*: *formula.set*(*3*))

lemma *atoms-finite-and*:
assumes *finite* (*atoms F1*)
finite (*atoms F2*)
shows *finite* (*atoms* (*F1* ∧ *F2*))
proof −
have *finite* (*atoms F1* ∪ *atoms F2*)
using *assms*
by (*simp only*: *finite-UnI*)
then show *finite* (*atoms* (*F1* ∧ *F2*))
by (*simp only*: *formula.set*(*4*))
qed

lemma *atoms-finite-or*:
assumes *finite* (*atoms F1*)
finite (*atoms F2*)
shows *finite* (*atoms* (*F1* ∨ *F2*))

```
proof −
 have finite (atoms F1 ∪ atoms F2)
  using assms
  by (simp only: finite-UnI)
 then show finite (atoms (F1 ∨ F2))
  by (simp only: formula.set(5))
qed

lemma atoms-finite-imp:
 assumes finite (atoms F1)
      finite (atoms F2)
 shows  finite (atoms (F1 → F2))
proof −
 have finite (atoms F1 ∪ atoms F2)
  using assms
  by (simp only: finite-UnI)
 then show finite (atoms (F1 → F2))
  by (simp only: formula.set(6))
qed

lemma atoms-finite: finite (atoms F)
proof (induction F)
 case (Atom x)
 then show ?case by (simp only: atoms-finite-atom)
next
 case Bot
 then show ?case by (simp only: atoms-finite-bot)
next
 case (Not F)
 then show ?case by (simp only: atoms-finite-not)
next
 case (And F1 F2)
 then show ?case by (simp only: atoms-finite-and)
next
 case (Or F1 F2)
 then show ?case by (simp only: atoms-finite-or)
next
 case (Imp F1 F2)
 then show ?case by (simp only: atoms-finite-imp)
qed
```

Su demostración automática es la siguiente.

lemma *finite* (*atoms F*)
by (*induction F*) *simp-all*

1.2 Subfórmulas

Veamos la noción de subfórmulas.

Definición 1.2.1 *El conjunto de subfórmulas de una fórmula F, notada Subf*(F)*, se define recursivamente como:*

- $\{F\}$ *si F es una fórmula atómica.*
- $\{\bot\}$ *si F es* $\bot$*.*
- $\{F\} \cup$ *Subf*(G) *si F es* $\neg G$*.*
- $\{F\} \cup$ *Subf*$(G) \cup$ *Subf*(H) *si F es* $G * H$ *donde* $*$ *es cualquier conectiva binaria.*

Para proceder a la formalización de Isabelle, seguiremos dos etapas. En primer lugar, definimos la función primitiva recursiva *subformulae*. Esta nos devolverá la lista de todas las subfórmulas de una fórmula original obtenidas recursivamente.

primrec *subformulae* :: *'a formula* $\Rightarrow$ *'a formula list* **where**
subformulae (*Atom k*) $=$ [*Atom k*]
| *subformulae* $\bot$ $=$ [$\bot$]
| *subformulae* ($\neg$ *F*) $=$ ($\neg$ *F*) # *subformulae F*
| *subformulae* (*F* $\wedge$ *G*) $=$ (*F* $\wedge$ *G*) # *subformulae F* @ *subformulae G*
| *subformulae* (*F* $\vee$ *G*) $=$ (*F* $\vee$ *G*) # *subformulae F* @ *subformulae G*
| *subformulae* (*F* $\rightarrow$ *G*) $=$ (*F* $\rightarrow$ *G*) # *subformulae F* @ *subformulae G*

Observemos que, en la definición anterior, # es el operador que añade un elemento al comienzo de una lista y @ concatena varias listas.

Siguiendo con los ejemplos, apliquemos *subformulae* en las distintas fórmulas. En particular, al tratarse de una lista pueden aparecer elementos repetidos como se muestra a continuación.

notepad
begin
fix *p* :: *'a*

```
 have subformulae (Atom p) = [Atom p]
  by simp

 have subformulae (¬ (Atom p)) = [¬ (Atom p), Atom p]
  by simp

 have subformulae ((Atom p → Atom q) ∨ Atom r) =
    [(Atom p → Atom q) ∨ Atom r, Atom p → Atom q, Atom p,
     Atom q, Atom r]
  by simp

 have subformulae (Atom p ∧ ⊥) = [Atom p ∧ ⊥, Atom p, ⊥]
  by simp

 have subformulae (Atom p ∨ Atom p) =
    [Atom p ∨ Atom p, Atom p, Atom p]
  by simp
end
```

En la segunda etapa de formalización, definimos *setSubformulae*, que convierte al tipo conjunto la lista de subfórmulas anterior.

```
abbreviation setSubformulae :: 'a formula ⇒ 'a formula set where
 setSubformulae F ≡ set (subformulae F)
```

De este modo, la función *setSubformulae* es la formalización en Isabelle de $Subf(\cdot)$. En Isabelle, primero hemos definido la lista de subfórmulas pues, en algunos casos, es más sencilla la prueba de resultados sobre este tipo. Algunas de las ventajas del tipo conjuntos son la eliminación de elementos repetidos o las operaciones propias de teoría de conjuntos. Observemos los siguientes ejemplos con el tipo de conjuntos.

```
notepad
begin
 fix p q r :: 'a

 have setSubformulae (Atom p ∨ Atom p) = {Atom p ∨ Atom p, Atom p}
  by simp

 have setSubformulae ((Atom p → Atom q) ∨ Atom r) =
    {(Atom p → Atom q) ∨ Atom r, Atom p → Atom q, Atom p,
     Atom q, Atom r}
 by auto
end
```

Por otro lado, debemos señalar que el uso de *abbreviation* para definir *setSubformulae* no es arbitrario. No es una definición propiamente dicha, sino una forma de nombrar la composición de las funciones *set* y *subformulae*.

En primer lugar, veamos que *setSubformulae* es una formalización de *Subf* en Isabelle. Para ello utilizaremos el siguiente resultado sobre listas, probado como sigue.

lemma *set-insert*: *set* $(x \# ys) = \{x\} \cup$ *set ys*
by (*simp only*: *list.set*(2) *Un-insert-left sup-bot.left-neutral*)

Por tanto, obtenemos la equivalencia como resultado de los siguientes lemas, que aparecen demostrados de manera detallada.

lemma *setSubformulae-atom*:
setSubformulae (*Atom p*) = {*Atom p*}
by (*simp only*: *subformulae.simps*(1) *list.set*)

lemma *setSubformulae-bot*:
setSubformulae ($\bot$) = {$\bot$}
by (*simp only*: *subformulae.simps*(2) *list.set*)

lemma *setSubformulae-not*:
shows *setSubformulae* ($\neg$ *F*) = {$\neg$ *F*} $\cup$ *setSubformulae F*
proof −
have *setSubformulae* ($\neg$ *F*) = *set* ($\neg$ *F* # *subformulae F*)
by (*simp only*: *subformulae.simps*(3))
also have ... = {$\neg$ *F*} $\cup$ *set* (*subformulae F*)
by (*simp only*: *set-insert*)
finally show *?thesis*
by *this*
qed

lemma *setSubformulae-and*:
setSubformulae (*F1* $\wedge$ *F2*)
= {*F1* $\wedge$ *F2*} $\cup$ (*setSubformulae F1* $\cup$ *setSubformulae F2*)
proof −
have *setSubformulae* (*F1* $\wedge$ *F2*)
= *set* ((*F1* $\wedge$ *F2*) # (*subformulae F1* @ *subformulae F2*))
by (*simp only*: *subformulae.simps*(4))
also have ... = {*F1* $\wedge$ *F2*} $\cup$ (*set* (*subformulae F1* @ *subformulae F2*))
by (*simp only*: *set-insert*)
also have ... = {*F1* $\wedge$ *F2*} $\cup$ (*setSubformulae F1* $\cup$ *setSubformulae F2*)
by (*simp only*: *set-append*)

```
 finally show ?thesis
  by this
qed

lemma setSubformulae-or:
 setSubformulae (F1 ∨ F2)
  = {F1 ∨ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
proof −
 have setSubformulae (F1 ∨ F2)
     = set ((F1 ∨ F2) # (subformulae F1 @ subformulae F2))
  by (simp only: subformulae.simps(5))
 also have ... = {F1 ∨ F2} ∪ (set (subformulae F1 @ subformulae F2))
  by (simp only: set-insert)
 also have ... = {F1 ∨ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
  by (simp only: set-append)
 finally show ?thesis
  by this
qed

lemma setSubformulae-imp:
 setSubformulae (F1 → F2)
  = {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
proof −
 have setSubformulae (F1 → F2)
     = set ((F1 → F2) # (subformulae F1 @ subformulae F2))
  by (simp only: subformulae.simps(6))
 also have ... = {F1 → F2} ∪ (set (subformulae F1 @ subformulae F2))
  by (simp only: set-insert)
 also have ... = {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
  by (simp only: set-append)
 finally show ?thesis
  by this
qed
```

Una vez probada la equivalencia, comencemos con los resultados correspondientes a las subfórmulas. En primer lugar, tenemos la siguiente propiedad como consecuencia directa de la equivalencia de funciones anterior.

Lema 1.2.2 *Toda fórmula es subfórmula de ella misma.*

Demostración: La demostración se hace en cada caso de la estructura de las fórmulas.

Sea p fórmula atómica cualquiera. Por definición, tenemos que su conjunto de subfórmulas es $\{p\}$, luego se tiene la propiedad.

Sea la fórmula $\bot$. Por definición, su conjunto de subfórmulas es $\{\bot\}$, luego se verifica el resultado.

Sea la fórmula $\neg\ F$. Veamos que pertenece a su conjunto de subfórmulas. Por definición, tenemos que el conjunto de subfórmulas de $\neg\ F$ es $\{\neg\ F\} \cup \mathit{Subf}(F)$. Por tanto, $\neg\ F$ pertence a su propio conjunto de subfórmulas como queríamos demostrar.

Sea $*$ una conectiva binaria cualquiera y las fórmulas F y G Veamos que $F{*}G$ pertenece a su conjunto de subfórmulas. Por definición, tenemos que el conjunto de subfórmulas de $F{*}G$ es $\{F{*}G\} \cup \mathit{Subf}(F) \cup \mathit{Subf}(G)$. Por tanto, $F{*}G$ pertence a su propio conjunto de subfórmulas como queríamos demostrar.

□

Formalicemos ahora el lema con su correspondiente demostración detallada.

```
lemma subformulae-self: F ∈ setSubformulae F
proof (cases F)
 case (Atom x1)
 then show ?thesis
  by (simp only: singletonI setSubformulae-atom)
next
 case Bot
 then show ?thesis
  by (simp only: singletonI setSubformulae-bot)
next
 case (Not F)
 then show ?thesis
  by (simp only: singletonI UnI1 setSubformulae-not)
next
 case (And F1 F2)
 then show ?thesis
  by (simp only: singletonI UnI1 setSubformulae-and)
next
 case (Or F1 F2)
 then show ?thesis
  by (simp only: singletonI UnI1 setSubformulae-or)
next
 case (Imp F1 F2)
 then show ?thesis
  by (simp only: singletonI UnI1 setSubformulae-imp)
qed
```

La demostración automática es la siguiente.

lemma $F \in setSubformulae\ F$
by (*cases F*) *simp-all*

Procedamos con los demás resultados de la sección. Como hemos señalado con anterioridad, utilizaremos varias propiedades de conjuntos pertenecientes a la teoría Set.thy de Isabelle, que apareceran en el glosario final.

Además, definiremos dos reglas adicionales que utilizaremos con frecuencia.

lemma *subContUnionRev1*:
assumes $A \cup B \subseteq C$
shows $A \subseteq C$
proof −
have $A \subseteq C \wedge B \subseteq C$
using *assms*
by (*simp only*: *sup.bounded-iff*)
then show $A \subseteq C$
by (*rule conjunct1*)
qed

lemma *subContUnionRev2*:
assumes $A \cup B \subseteq C$
shows $B \subseteq C$
proof −
have $A \subseteq C \wedge B \subseteq C$
using *assms*
by (*simp only*: *sup.bounded-iff*)
then show $B \subseteq C$
by (*rule conjunct2*)
qed

Sus correspondientes demostraciones automáticas se muestran a continuación.

lemma $A \cup B \subseteq C \Longrightarrow A \subseteq C$
by *simp*

lemma $A \cup B \subseteq C \Longrightarrow B \subseteq C$
by *simp*

Veamos ahora los distintos resultados sobre subfórmulas.

Lema 1.2.3 *Todas las fórmulas atómicas de una fórmula son subfórmulas.*

Demostración: Aclaremos que el conjunto de las fórmulas atómicas de una fórmula cualquiera está formado a partir de cada elemento de su conjunto de variables proposicionales. Queremos demostrar que este conjunto está contenido en el conjunto de subfórmulas de dicha fórmula. De este modo, la prueba seguirá el esquema inductivo para la estructura de fórmulas. Veamos cada caso:

Consideremos la fórmula atómica p cualquiera. Como su conjunto de átomos es $\{p\}$, el conjunto de sus fórmulas atómicas correspondiente será $\{p\}$. Por otro lado, su conjunto de subfórmulas es también $\{p\}$, luego el conjunto de sus fórmulas atómicas está contenido en el conjunto de sus subfórmulas como queríamos demostrar.

Sea la fórmula $\bot$. Como su conjunto de átomos es vacío, es claro que el conjunto de sus fórmulas atómicas es también el vacío y, por tanto, está contenido en el conjunto de sus subfórmulas.

Sea la fórmula F tal que el conjunto de sus fórmulas atómicas está contenido en el conjunto de sus subfórmulas. Probemos el resultado para $\neg F$. En primer lugar, sabemos que los conjuntos de variables proposicionales de F y $\neg F$ coinciden, luego tendrán igual conjunto de fórmulas atómicas. Por lo tanto, por hipótesis de inducción tenemos que el conjunto de fórmulas atómicas de F está contenido en el conjunto de subfórmulas de F. Por otro lado, como el conjunto de subfórmulas de $\neg F$ está definido como $Subf(\neg F) = \{\neg F\} \cup Subf(F)$, tenemos que el el conjunto de subfórmulas de F está contenido en el de $\neg F$. Por tanto, por propiedades de contención, tenemos que el conjunto de fórmulas atómicas de $\neg F$ está contenido en el conjunto de subfórmulas de $\neg F$ como queríamos demostrar.

Sean las fórmulas F y G tales que sus conjuntos de fórmulas atómicas están contenidos en sus conjuntos de subfórmulas respectivamente. Probemos ahora el resultado para $F*G$, donde $*$ simboliza una conectiva binaria cualquiera. En primer lugar, sabemos que el conjunto de átomos de $F*G$ es la unión de sus correspondientes conjuntos de átomos. De este modo, el conjunto de fórmulas atómicas de $F*G$ será la unión del conjunto de fórmulas atómicas de F y el correspondiente de G. Por tanto, por hipótesis de inducción tenemos que el conjunto de fórmulas atómicas de $F*G$ está contenido en la unión del conjunto de subfórmulas de F y el conjunto de subfórmulas de G. Como el conjunto de subfórmulas de $F*G$ se define como $Subf(F*G) = \{F*G\} \cup Subf(F) \cup Subf(G)$, tenemos que la unión de los conjuntos de subfórmulas de F y G está contenida en el conjunto de subfórmulas de $F*G$. Por tanto, por propiedades de la contención, tenemos que le conjunto de fórmulas atómicas de $F*G$ está contenido en el conjunto de subfórmulas de $F*G$ como queríamos demostrar. □

En Isabelle, se especifica como sigue.

lemma *Atom ' atoms F ⊆ setSubformulae F*

oops

Debemos observar que *Atom ‘ atoms F* construye las fórmulas atómicas a partir de cada uno de los elementos de *atoms F*, creando un conjunto de fórmulas atómicas. Para ello emplea el infijo ‘ definido como notación abreviada de (‘) que calcula la imagen de un conjunto en la teoría Set.thy.

$$f\text{`}A = \{y \mid \exists x{\in}A.\ y = f\,x\} \qquad (\textit{image-def})$$

Para aclarar su funcionamiento, veamos ejemplos para distintos casos de fórmulas.

notepad
begin
fix $p\ q\ r :: 'a$

have *Atom ‘atoms* $(Atom\ p \vee \bot) = \{Atom\ p\}$
by *simp*

have *Atom ‘atoms* $((Atom\ p \longrightarrow Atom\ q) \vee Atom\ r) =$
$\{Atom\ p, Atom\ q, Atom\ r\}$
by *auto*

have *Atom ‘atoms* $((Atom\ p \longrightarrow Atom\ p) \vee Atom\ r) =$
$\{Atom\ p, Atom\ r\}$
by *auto*
end

Además, esta función tiene las siguientes propiedades sobre conjuntos que utilizaremos en la demostración.

$$f\text{`}(A \cup B) = f\text{`}A \cup f\text{`}B \qquad (\textit{image-Un})$$

$$f\text{`}(\{a\} \cup B) = \{f\,a\} \cup f\text{`}B \qquad (\textit{image-insert})$$

$$f\text{`}\emptyset = \emptyset \qquad (\textit{image-empty})$$

Una vez hechas las aclaraciones necesarias, comencemos la demostración estructurada. Esta seguirá el esquema inductivo señalado con anterioridad.

lemma *atoms-are-subformulae-atom*:
Atom ‘ atoms $(Atom\ x) \subseteq$ *setSubformulae* $(Atom\ x)$
proof −
have *Atom ‘ atoms* $(Atom\ x) =$ *Atom ‘* $\{x\}$

```
  by (simp only: formula.set(1))
 also have ... = {Atom x}
  by (simp only: image-insert image-empty)
 also have ... = set [Atom x]
  by (simp only: list.set(1) list.set(2))
 also have ... = set (subformulae (Atom x))
  by (simp only: subformulae.simps(1))
 finally have Atom ‘ atoms (Atom x) = set (subformulae (Atom x))
  by this
 then show ?thesis
  by (simp only: subset-refl)
qed

lemma atoms-are-subformulae-bot:
 Atom ‘ atoms ⊥ ⊆ setSubformulae ⊥
proof −
 have Atom ‘ atoms ⊥ = Atom ‘ ∅
  by (simp only: formula.set(2))
 also have ... = ∅
  by (simp only: image-empty)
 also have ... ⊆ setSubformulae ⊥
  by (simp only: empty-subsetI)
 finally show ?thesis
  by this
qed

lemma atoms-are-subformulae-not:
 assumes Atom ‘ atoms F ⊆ setSubformulae F
 shows  Atom ‘ atoms (¬ F) ⊆ setSubformulae (¬ F)
proof −
 have Atom ‘ atoms (¬ F) = Atom ‘ atoms F
  by (simp only: formula.set(3))
 also have ... ⊆ setSubformulae F
  by (simp only: assms)
 also have ... ⊆ {¬ F} ∪ setSubformulae F
  by (simp only: Un-upper2)
 also have ... = setSubformulae (¬ F)
  by (simp only: setSubformulae-not)
 finally show ?thesis
  by this
```

qed

lemma *atoms-are-subformulae-and*:
assumes *Atom ' atoms F1* $\subseteq$ *setSubformulae F1*
Atom ' atoms F2 $\subseteq$ *setSubformulae F2*
shows *Atom ' atoms* (*F1* $\wedge$ *F2*) $\subseteq$ *setSubformulae* (*F1* $\wedge$ *F2*)
proof −
have *Atom ' atoms* (*F1* $\wedge$ *F2*) = *Atom '* (*atoms F1* $\cup$ *atoms F2*)
by (*simp only*: *formula.set*(*4*))
also have ... = *Atom ' atoms F1* $\cup$ *Atom ' atoms F2*
by (*rule image-Un*)
also have ... $\subseteq$ *setSubformulae F1* $\cup$ *setSubformulae F2*
using *assms*
by (*rule Un-mono*)
also have ... $\subseteq$ {*F1* $\wedge$ *F2*} $\cup$ (*setSubformulae F1* $\cup$ *setSubformulae F2*)
by (*simp only*: *Un-upper2*)
also have ... = *setSubformulae* (*F1* $\wedge$ *F2*)
by (*simp only*: *setSubformulae-and*)
finally show *?thesis*
by *this*
qed

lemma *atoms-are-subformulae-or*:
assumes *Atom ' atoms F1* $\subseteq$ *setSubformulae F1*
Atom ' atoms F2 $\subseteq$ *setSubformulae F2*
shows *Atom ' atoms* (*F1* $\vee$ *F2*) $\subseteq$ *setSubformulae* (*F1* $\vee$ *F2*)
proof −
have *Atom ' atoms* (*F1* $\vee$ *F2*) = *Atom '* (*atoms F1* $\cup$ *atoms F2*)
by (*simp only*: *formula.set*(*5*))
also have ... = *Atom ' atoms F1* $\cup$ *Atom ' atoms F2*
by (*rule image-Un*)
also have ... $\subseteq$ *setSubformulae F1* $\cup$ *setSubformulae F2*
using *assms*
by (*rule Un-mono*)
also have ... $\subseteq$ {*F1* $\vee$ *F2*} $\cup$ (*setSubformulae F1* $\cup$ *setSubformulae F2*)
by (*simp only*: *Un-upper2*)
also have ... = *setSubformulae* (*F1* $\vee$ *F2*)
by (*simp only*: *setSubformulae-or*)
finally show *?thesis*
by *this*

qed

lemma *atoms-are-subformulae-imp*:
assumes *Atom ' atoms F1* ⊆ *setSubformulae F1*
Atom ' atoms F2 ⊆ *setSubformulae F2*
shows *Atom ' atoms* (*F1* ⟶ *F2*) ⊆ *setSubformulae* (*F1* ⟶ *F2*)
proof −
have *Atom ' atoms* (*F1* ⟶ *F2*) = *Atom '* (*atoms F1* ∪ *atoms F2*)
by (*simp only*: *formula.set*(6))
also have . . . = *Atom ' atoms F1* ∪ *Atom ' atoms F2*
by (*rule image-Un*)
also have . . . ⊆ *setSubformulae F1* ∪ *setSubformulae F2*
using *assms*
by (*rule Un-mono*)
also have . . . ⊆ {*F1* ⟶ *F2*} ∪ (*setSubformulae F1* ∪ *setSubformulae F2*)
by (*simp only*: *Un-upper2*)
also have . . . = *setSubformulae* (*F1* ⟶ *F2*)
by (*simp only*: *setSubformulae-imp*)
finally show *?thesis*
by *this*
qed

lemma *atoms-are-subformulae*:
Atom ' atoms F ⊆ *setSubformulae F*
proof (*induction F*)
case (*Atom x*)
then show *?case* **by** (*simp only*: *atoms-are-subformulae-atom*)
next
case *Bot*
then show *?case* **by** (*simp only*: *atoms-are-subformulae-bot*)
next
case (*Not F*)
then show *?case* **by** (*simp only*: *atoms-are-subformulae-not*)
next
case (*And F1 F2*)
then show *?case* **by** (*simp only*: *atoms-are-subformulae-and*)
next
case (*Or F1 F2*)
then show *?case* **by** (*simp only*: *atoms-are-subformulae-or*)
next

case (*Imp F1 F2*)
then show *?case* **by** (*simp only*: *atoms-are-subformulae-imp*)
qed

La demostración automática queda igualmente expuesta a continuación.

lemma *Atom ' atoms F* $\subseteq$ *setSubformulae F*
by (*induction F*) *auto*

La siguiente propiedad declara que el conjunto de átomos de una subfórmula está contenido en el conjunto de átomos de la propia fórmula.

Lema 1.2.4 *Dada una fórmula, los átomos de sus subfórmulas son átomos de ella misma.*

Demostración: Procedemos mediante inducción en la estructura de las fórmulas según los distintos casos:

Sea p una fórmula atómica cualquiera. Por definición de su conjunto de subfórmulas, su única subfórmula es ella misma, luego se verifica el resultado.

Sea la fórmula $\bot$. Por definición de su conjunto de subfórmulas, su única subfórmula es ella misma, luego se verifica análogamente la propiedad en este caso.

Sea la fórmula F tal que para cualquier subfórmula suya se verifica que el conjunto de sus átomos está contenido en el conjunto de átomos de F. Supongamos G subfórmula cualquiera de $\neg F$. Vamos a probar que el conjunto de átomos de G está contenido en el de $\neg F$. Por definición, tenemos que el conjunto de subfórmulas de $\neg F$ es de la forma $Subf(\neg F) = \{\neg F\} \cup Subf(F)$. De este modo, tenemos dos opciones posibles:
$G \in \{\neg F\}$ o $G \in Subf(F)$. Del primer caso se deduce $G = \neg F$ y, por tanto, tienen igual conjunto de átomos. Observando el segundo caso, por hipótesis de inducción, se tiene que el conjunto de átomos de G está contenido en el de F. Además, como el conjunto de átomos de F y $\neg F$ coinciden, se verifica el resultado.

Sea $F1$ una fórmula proposicional tal que el conjunto de los átomos de cualquier subfórmula suya está contenido en el conjunto de átomos de $F1$. Sea también $F2$ cumpliendo dicha hipótesis de inducción para sus correspondientes subfórmulas. Supongamos además que G es subfórmula de $F1{*}F2$, donde $*$ simboliza una conectiva binaria cualquiera. Vamos a probar que el conjunto de átomos de G está contenido en el conjunto de átomos de $F1{*}F2$. En primer lugar, por definición tenemos que el conjunto de subfórmulas de $F1{*}F2$ es de la forma
$Subf(F1{*}F2) = \{F1{*}F2\} \cup (Subf(F1) \cup Subf(F2))$. De este modo, tenemos dos posibles opciones: $G \in \{F1{*}F2\}$ o $G \in Subf(F1) \cup Subf(F2)$. Si $G \in \{F1{*}F2\}$, entonces $G = F1{*}F2$ y tienen igual conjunto de átomos. Por otro lado, si $G \in Subf(F1) \cup Subf(F2)$ tenemos dos nuevas posibilidades: G es subfórmula de $F1$ o G es subfórmula de $F2$. Suponiendo

que fuese subfórmula de *F1*, aplicando hipótesis de inducción tendríamos que el conjunto de átomos de *G* está contenido en el de *F1*. De este modo, como el conjunto de átomos de *F1∗F2* se define como la unión de los conjuntos de átomos de *F1* y *F2*, por propiedades de la contención se verifica que el conjunto de átomos de *G* está contenido en el de *F1∗F2*. Observemos que si *G* es subfórmula de *F2*, se demuestra análogamente cambiando los subíndices correspondientes. Por tanto, se tiene el resultado.

□

Formalizado en Isabelle:

```
lemma G ∈ setSubformulae F ⟹ atoms G ⊆ atoms F
 oops
```

Veamos su demostración estructurada.

```
lemma subformulas-atoms-atom:
 assumes G ∈ setSubformulae (Atom x)
 shows   atoms G ⊆ atoms (Atom x)
proof −
 have G ∈ {Atom x}
  using assms
  by (simp only: setSubformulae-atom)
 then have G = Atom x
  by (simp only: singletonD)
 then show ?thesis
  by (simp only: subset-refl)
qed

lemma subformulas-atoms-bot:
 assumes G ∈ setSubformulae ⊥
 shows   atoms G ⊆ atoms ⊥
proof −
 have G ∈ {⊥}
  using assms
  by (simp only: setSubformulae-bot)
 then have G = ⊥
  by (simp only: singletonD)
 then show ?thesis
  by (simp only: subset-refl)
qed

lemma subformulas-atoms-not:
 assumes G ∈ setSubformulae F ⟹ atoms G ⊆ atoms F
```

```
      G ∈ setSubformulae (¬ F)
 shows  atoms G ⊆ atoms (¬ F)
proof −
 have G ∈ {¬ F} ∪ setSubformulae F
  using assms(2)
  by (simp only: setSubformulae-not)
 then have G ∈ {¬ F} ∨ G ∈ setSubformulae F
  by (simp only: Un-iff)
 then show atoms G ⊆ atoms (¬ F)
 proof (rule disjE)
  assume G ∈ {¬ F}
  then have G = ¬ F
   by (simp only: singletonD)
  then show ?thesis
   by (simp only: subset-refl)
 next
  assume G ∈ setSubformulae F
  then have atoms G ⊆ atoms F
   by (simp only: assms(1))
  also have ... = atoms (¬ F)
   by (simp only: formula.set(3))
  finally show ?thesis
   by this
 qed
qed

lemma subformulas-atoms-and:
 assumes G ∈ setSubformulae F1 ⟹ atoms G ⊆ atoms F1
      G ∈ setSubformulae F2 ⟹ atoms G ⊆ atoms F2
      G ∈ setSubformulae (F1 ∧ F2)
 shows  atoms G ⊆ atoms (F1 ∧ F2)
proof −
 have G ∈ {F1 ∧ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
  using assms(3)
  by (simp only: setSubformulae-and)
 then have G ∈ {F1 ∧ F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
  by (simp only: Un-iff)
 then show ?thesis
 proof (rule disjE)
  assume G ∈ {F1 ∧ F2}
```

```
    then have G = F1 ∧ F2
      by (simp only: singletonD)
    then show ?thesis
      by (simp only: subset-refl)
  next
    assume G ∈ setSubformulae F1 ∪ setSubformulae F2
    then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
      by (simp only: Un-iff)
    then show ?thesis
    proof (rule disjE)
      assume G ∈ setSubformulae F1
      then have atoms G ⊆ atoms F1
        by (rule assms(1))
      also have ... ⊆ atoms F1 ∪ atoms F2
        by (simp only: Un-upper1)
      also have ... = atoms (F1 ∧ F2)
        by (simp only: formula.set(4))
      finally show ?thesis
        by this
    next
      assume G ∈ setSubformulae F2
      then have atoms G ⊆ atoms F2
        by (rule assms(2))
      also have ... ⊆ atoms F1 ∪ atoms F2
        by (simp only: Un-upper2)
      also have ... = atoms (F1 ∧ F2)
        by (simp only: formula.set(4))
      finally show ?thesis
        by this
    qed
  qed
qed

lemma subformulas-atoms-or:
  assumes G ∈ setSubformulae F1 ⟹ atoms G ⊆ atoms F1
          G ∈ setSubformulae F2 ⟹ atoms G ⊆ atoms F2
          G ∈ setSubformulae (F1 ∨ F2)
  shows   atoms G ⊆ atoms (F1 ∨ F2)
proof −
  have G ∈ {F1 ∨ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
```

```
    using assms(3)
    by (simp only: setSubformulae-or)
  then have G ∈ {F1 ∨ F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
    by (simp only: Un-iff)
  then show ?thesis
  proof (rule disjE)
    assume G ∈ {F1 ∨ F2}
    then have G = F1 ∨ F2
      by (simp only: singletonD)
    then show ?thesis
      by (simp only: subset-refl)
  next
    assume G ∈ setSubformulae F1 ∪ setSubformulae F2
    then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
      by (simp only: Un-iff)
    then show ?thesis
    proof (rule disjE)
      assume G ∈ setSubformulae F1
      then have atoms G ⊆ atoms F1
        by (rule assms(1))
      also have ... ⊆ atoms F1 ∪ atoms F2
        by (simp only: Un-upper1)
      also have ... = atoms (F1 ∨ F2)
        by (simp only: formula.set(5))
      finally show ?thesis
        by this
    next
      assume G ∈ setSubformulae F2
      then have atoms G ⊆ atoms F2
        by (rule assms(2))
      also have ... ⊆ atoms F1 ∪ atoms F2
        by (simp only: Un-upper2)
      also have ... = atoms (F1 ∨ F2)
        by (simp only: formula.set(5))
      finally show ?thesis
        by this
    qed
  qed
qed
```

```
lemma subformulas-atoms-imp:
 assumes G ∈ setSubformulae F1 ⟹ atoms G ⊆ atoms F1
       G ∈ setSubformulae F2 ⟹ atoms G ⊆ atoms F2
       G ∈ setSubformulae (F1 → F2)
 shows  atoms G ⊆ atoms (F1 → F2)
proof −
 have G ∈ {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
  using assms(3)
  by (simp only: setSubformulae-imp)
 then have G ∈ {F1 → F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
  by (simp only: Un-iff)
 then show ?thesis
 proof (rule disjE)
  assume G ∈ {F1 → F2}
  then have G = F1 → F2
   by (simp only: singletonD)
  then show ?thesis
   by (simp only: subset-refl)
 next
  assume G ∈ setSubformulae F1 ∪ setSubformulae F2
  then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
   by (simp only: Un-iff)
  then show ?thesis
  proof (rule disjE)
   assume G ∈ setSubformulae F1
   then have atoms G ⊆ atoms F1
    by (rule assms(1))
   also have ... ⊆ atoms F1 ∪ atoms F2
    by (simp only: Un-upper1)
   also have ... = atoms (F1 → F2)
    by (simp only: formula.set(6))
   finally show ?thesis
    by this
  next
   assume G ∈ setSubformulae F2
   then have atoms G ⊆ atoms F2
    by (rule assms(2))
   also have ... ⊆ atoms F1 ∪ atoms F2
    by (simp only: Un-upper2)
   also have ... = atoms (F1 → F2)
```

```
      by (simp only: formula.set(6))
    finally show ?thesis
      by this
  qed
 qed
qed

lemma subformulae-atoms:
 G ∈ setSubformulae F ⟹ atoms G ⊆ atoms F
proof (induction F)
 case (Atom x)
 then show ?case by (simp only: subformulas-atoms-atom)
next
 case Bot
 then show ?case by (simp only: subformulas-atoms-bot)
next
 case (Not F)
 then show ?case by (simp only: subformulas-atoms-not)
next
 case (And F1 F2)
 then show ?case by (simp only: subformulas-atoms-and)
next
 case (Or F1 F2)
 then show ?case by (simp only: subformulas-atoms-or)
next
 case (Imp F1 F2)
 then show ?case by (simp only: subformulas-atoms-imp)
qed
```

Por último, su demostración automática.

```
lemma G ∈ setSubformulae F ⟹ atoms G ⊆ atoms F
 by (induction F) auto
```

A continuación vamos a introducir un lema para facilitar las siguientes demostraciones detalladas mediante contenciones en cadena.

Lema 1.2.5 *Sea G una subfórmula de F, entonces el conjunto de subfórmulas de G está contenido en el de F.*

Demostración: La prueba es por inducción en la estructura de fórmula.

Sea p una fórmula atómica cualquiera. Por definición, el conjunto de sus subfórmulas es $\{p\}$, luego su única subfórmula es ella misma y, por tanto, tienen igual conjunto de subfórmulas.

Sea la fórmula $\bot$. Por definición, el conjunto de sus subfórmulas es $\{\bot\}$, luego su única subfórmula es ella misma y, por tanto, tienen igual conjunto de subfórmulas.

Sea una fórmula F tal que para toda subfórmula suya se tiene que el conjunto de sus subfórmulas está contenido en el conjunto de subfórmulas de F. Supongamos G subfórmula de $\neg\, F$. Vamos a probar que el conjunto de subfórmulas de G está contenido en el de $\neg\, F$. En primer lugar, por definición se cumple que el conjunto de subfórmulas de $\neg\, F$ es de la forma $Subf(\neg\, F) = \{\neg\, F\} \cup Subf(F)$. Como hemos supuesto G subfórmula de $\neg\, F$, hay dos opciones posibles: $G \in \{\neg\, F\}$ o $G \in Subf(F)$. Del primer caso se obtiene que $G = \neg\, F$ y, por tanto, tienen igual conjunto de subfórmulas. Por otro lado si suponemos que G es subfórmula de F, por hipótesis de inducción tenemos que el conjunto de subfórmulas de G está contenido en el de F. Como, a su vez, el conjunto de subfórmulas de F está contenido en el de $\neg\, F$ según la definición anterior, por propiedades de la contención de verifica que el conjunto de subfórmulas de G está contenido en el de $\neg\, F$, como queríamos demostrar.

Sean las fórmulas *F1* y *F2* tales que para cualquier subfórmula de *F1* el conjunto de sus subfórmulas está contenido en el conjunto de subfórmulas de *F1*, y para cualquier subfórmula de *F2* el conjunto de sus subfórmulas está contenido en el conjunto de subfórmulas de *F2*. Supongamos G subfórmula de $F1 * F2$ donde $*$ simboliza una conectiva binaria cualquiera. Vamos a probar que el conjunto de subfórmulas de G está contenido en el de $F1 * F2$. En primer lugar, por definición se cumple que el conjunto de subfórmulas de $F1 * F2$ es de la forma $\{F1 * F2\} \cup (Subf(F1) \cup Subf(F2))$. De este modo, tenemos dos opciones: $G \in \{F1 * F2\}$ o $G \in Subf(F1) \cup Subf(F2)$. De la primera opción se deduce $G = F1 * F2$ y, por tanto, tienen igual conjunto de subfórmulas. Por otro lado, si $G \in Subf(F1) \cup Subf(F2)$, tenemos a su vez dos opciones: G es subfórmula de *F1* o G es subfórmula de *F2*. Supongamos que fuese subfórmula de *F1*. En este caso, por hipótesis de inducción se tiene que el conjunto de subfórmulas de G está contenido en el de *F1*. Por la definición anterior del conjunto de subfórmulas de $F1 * F2$, se verifica que el conjunto de subfórmulas de *F1* está contenido en el de $F1 * F2$. Por tanto, por propiedades de contención se tiene que el conjunto de subfórmulas de G está contenido en el conjunto de subfórmulas de $F1 * F2$. El caso de G subfórmula de *F2* se demuestra análogamente cambiando el índice de la fórmula correspondiente. Por tanto, se verifica el resultado en este caso.

□

Veamos su formalización en Isabelle junto con su demostración estructurada.

lemma *subContsubformulae-atom*:
assumes $G \in setSubformulae\ (Atom\ x)$

```
 shows setSubformulae G ⊆ setSubformulae (Atom x)
proof −
 have G ∈ {Atom x} using assms
  by (simp only: setSubformulae-atom)
 then have G = Atom x
  by (simp only: singletonD)
 then show ?thesis
  by (simp only: subset-refl)
qed

lemma subContsubformulae-bot:
 assumes G ∈ setSubformulae ⊥
 shows  setSubformulae G ⊆ setSubformulae ⊥
proof −
 have G ∈ {⊥}
  using assms
  by (simp only: setSubformulae-bot)
 then have G = ⊥
  by (simp only: singletonD)
 then show ?thesis
  by (simp only: subset-refl)
qed

lemma subContsubformulae-not:
 assumes G ∈ setSubformulae F ⟹ setSubformulae G ⊆ setSubformulae F
      G ∈ setSubformulae (¬ F)
 shows  setSubformulae G ⊆ setSubformulae (¬ F)
proof −
 have G ∈ {¬ F} ∪ setSubformulae F
  using assms(2)
  by (simp only: setSubformulae-not)
 then have G ∈ {¬ F} ∨ G ∈ setSubformulae F
  by (simp only: Un-iff)
 then show setSubformulae G ⊆ setSubformulae (¬ F)
 proof
  assume G ∈ {¬ F}
  then have G = ¬ F
   by (simp only: singletonD)
  then show ?thesis
   by (simp only: subset-refl)
```

```
  next
    assume G ∈ setSubformulae F
    then have setSubformulae G ⊆ setSubformulae F
      by (simp only: assms(1))
    also have setSubformulae F ⊆ setSubformulae (¬ F)
      by (simp only: setSubformulae-not Un-upper2)
    finally show ?thesis
      by this
  qed
qed

lemma subContsubformulae-and:
  assumes G ∈ setSubformulae F1
          ⟹ setSubformulae G ⊆ setSubformulae F1
        G ∈ setSubformulae F2
          ⟹ setSubformulae G ⊆ setSubformulae F2
        G ∈ setSubformulae (F1 ∧ F2)
  shows   setSubformulae G ⊆ setSubformulae (F1 ∧ F2)
proof −
  have G ∈ {F1 ∧ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
    using assms(3)
    by (simp only: setSubformulae-and)
  then have G ∈ {F1 ∧ F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
    by (simp only: Un-iff)
  then show ?thesis
  proof (rule disjE)
    assume G ∈ {F1 ∧ F2}
    then have G = F1 ∧ F2
      by (simp only: singletonD)
    then show ?thesis
      by (simp only: subset-refl)
  next
    assume G ∈ setSubformulae F1 ∪ setSubformulae F2
    then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
      by (simp only: Un-iff)
    then show ?thesis
    proof
      assume G ∈ setSubformulae F1
      then have setSubformulae G ⊆ setSubformulae F1
        by (simp only: assms(1))
```

```
    also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
     by (simp only: Un-upper1)
    also have ... ⊆ setSubformulae (F1 ∧ F2)
     by (simp only: setSubformulae-and Un-upper2)
    finally show ?thesis
     by this
   next
    assume G ∈ setSubformulae F2
    then have setSubformulae G ⊆ setSubformulae F2
     by (rule assms(2))
    also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
     by (simp only: Un-upper2)
    also have ... ⊆ setSubformulae (F1 ∧ F2)
     by (simp only: setSubformulae-and Un-upper2)
    finally show ?thesis
     by this
   qed
  qed
qed

lemma subContsubformulae-or:
  assumes G ∈ setSubformulae F1
          ⟹ setSubformulae G ⊆ setSubformulae F1
        G ∈ setSubformulae F2
          ⟹ setSubformulae G ⊆ setSubformulae F2
        G ∈ setSubformulae (F1 ∨ F2)
  shows  setSubformulae G ⊆ setSubformulae (F1 ∨ F2)
proof −
  have G ∈ {F1 ∨ F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
   using assms(3)
   by (simp only: setSubformulae-or)
  then have G ∈ {F1 ∨ F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
   by (simp only: Un-iff)
  then show ?thesis
  proof (rule disjE)
   assume G ∈ {F1 ∨ F2}
   then have G = F1 ∨ F2
    by (simp only: singletonD)
   then show ?thesis
    by (simp only: subset-refl)
```

```
  next
   assume G ∈ setSubformulae F1 ∪ setSubformulae F2
   then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
    by (simp only: Un-iff)
   then show ?thesis
   proof (rule disjE)
    assume G ∈ setSubformulae F1
    then have setSubformulae G ⊆ setSubformulae F1
     by (simp only: assms(1))
    also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
     by (simp only: Un-upper1)
    also have ... ⊆ setSubformulae (F1 ∨ F2)
     by (simp only: setSubformulae-or Un-upper2)
    finally show ?thesis
     by this
   next
    assume G ∈ setSubformulae F2
    then have setSubformulae G ⊆ setSubformulae F2
     by (rule assms(2))
    also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
     by (simp only: Un-upper2)
    also have ... ⊆ setSubformulae (F1 ∨ F2)
     by (simp only: setSubformulae-or Un-upper2)
    finally show ?thesis
     by this
   qed
  qed
qed

lemma subContsubformulae-imp:
  assumes G ∈ setSubformulae F1
          ⟹ setSubformulae G ⊆ setSubformulae F1
        G ∈ setSubformulae F2
          ⟹ setSubformulae G ⊆ setSubformulae F2
        G ∈ setSubformulae (F1 → F2)
  shows  setSubformulae G ⊆ setSubformulae (F1 → F2)
proof −
  have G ∈ {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
   using assms(3)
   by (simp only: setSubformulae-imp)
```

```
 then have G ∈ {F1 → F2} ∨ G ∈ setSubformulae F1 ∪ setSubformulae F2
  by (simp only: Un-iff)
 then show ?thesis
 proof (rule disjE)
  assume G ∈ {F1 → F2}
  then have G = F1 → F2
   by (simp only: singletonD)
  then show ?thesis
   by (simp only: subset-refl)
 next
  assume G ∈ setSubformulae F1 ∪ setSubformulae F2
  then have G ∈ setSubformulae F1 ∨ G ∈ setSubformulae F2
   by (simp only: Un-iff)
  then show ?thesis
  proof (rule disjE)
   assume G ∈ setSubformulae F1
   then have setSubformulae G ⊆ setSubformulae F1
    by (simp only: assms(1))
   also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
    by (simp only: Un-upper1)
   also have ... ⊆ setSubformulae (F1 → F2)
    by (simp only: setSubformulae-imp Un-upper2)
   finally show ?thesis
    by this
  next
   assume G ∈ setSubformulae F2
   then have setSubformulae G ⊆ setSubformulae F2
    by (rule assms(2))
   also have ... ⊆ setSubformulae F1 ∪ setSubformulae F2
    by (simp only: Un-upper2)
   also have ... ⊆ setSubformulae (F1 → F2)
    by (simp only: setSubformulae-imp Un-upper2)
   finally show ?thesis
    by this
  qed
 qed
qed

lemma
 G ∈ setSubformulae F ⟹ setSubformulae G ⊆ setSubformulae F
```

```
proof (induction F)
 case (Atom x)
 then show ?case by (rule subContsubformulae-atom)
next
 case Bot
 then show ?case by (rule subContsubformulae-bot)
next
 case (Not F)
 then show ?case by (rule subContsubformulae-not)
next
 case (And F1 F2)
 then show ?case by (rule subContsubformulae-and)
next
 case (Or F1 F2)
 then show ?case by (rule subContsubformulae-or)
next
 case (Imp F1 F2)
 then show ?case by (rule subContsubformulae-imp)
qed
```

Finalmente, su demostración automática se muestra a continuación.

```
lemma subContsubformulae:
 G ∈ setSubformulae F ⟹ setSubformulae G ⊆ setSubformulae F
 by (induction F) auto
```

El siguiente lema nos da la noción de transitividad de contención en cadena de las subfórmulas, de modo que la subfórmula de una subfórmula es del mismo modo subfórmula de la mayor.

Lema 1.2.6 *Sea H una subfórmula de G que es a su vez subfórmula de F, entonces H es subfórmula de F.*

Demostración: La prueba está basada en el lema anterior. Hemos demostrado que si H es subfórmula de G, entonces el conjunto de subfórmulas de H está contenido en el conjunto de subfórmulas de G. Del mismo modo, como G es subfórmula de F, su conjunto de subfórmulas está contenido en el conjunto de subfórmulas de F. Por la transitividad de la contención, tenemos que el conjunto de subfórmulas de H está contenido en el de F. Por otro lema anterior, como H es subfórmula de ella misma, es decir, pertenece a su conjunto de subfórmulas, por la contención anterior se verifica que pertenece al conjunto de subfórmulas de F como queríamos demostrar.

□

Veamos su formalización y prueba estructurada en Isabelle.

```
lemma
 assumes G ∈ setSubformulae F
     H ∈ setSubformulae G
 shows  H ∈ setSubformulae F
proof −
 have 1: setSubformulae G ⊆ setSubformulae F
  using assms(1)
  by (rule subContsubformulae)
 have setSubformulae H ⊆ setSubformulae G
  using assms(2)
  by (rule subContsubformulae)
 then have 2: setSubformulae H ⊆ setSubformulae F
  using 1
  by (rule subset-trans)
 have H ∈ setSubformulae H
  by (simp only: subformulae-self)
 then show H ∈ setSubformulae F
  using 2
  by (rule rev-subsetD)
qed
```

A continuación su demostración automática.

```
lemma subsubformulae:
 G ∈ setSubformulae F
 ⟹ H ∈ setSubformulae G
 ⟹ H ∈ setSubformulae F
 by (drule subContsubformulae, erule subsetD)
```

Presentemos ahora otro resultado que relaciona las conectivas con los conjuntos de subfórmulas.

Para la demostración en Isabelle, probaremos cada caso de forma independiente.

```
lemma subformulas-in-subformulas-not:
 assumes ¬ G ∈ setSubformulae F
 shows G ∈ setSubformulae F
proof −
 have G ∈ setSubformulae G
  by (simp only: subformulae-self)
 then have G ∈ {¬ G} ∪ setSubformulae G
  by (simp only: UnI2)
```

```
  then have 1:G ∈ setSubformulae (¬ G)
    by (simp only: setSubformulae-not)
  show G ∈ setSubformulae F using assms 1
    by (rule subsubformulae)
qed

lemma subformulas-in-subformulas-and:
  assumes G ∧ H ∈ setSubformulae F
  shows G ∈ setSubformulae F ∧ H ∈ setSubformulae F
proof (rule conjI)
  have G ∈ setSubformulae (G ∧ H)
    by (simp only: subformulae-self UnI2 UnI1 setSubformulae-and)
  with assms show G ∈ setSubformulae F
    by (rule subsubformulae)
next
  have H ∈ setSubformulae (G ∧ H)
    by (simp only: subformulae-self UnI2 UnI1 setSubformulae-and)
  with assms show H ∈ setSubformulae F
    by (rule subsubformulae)
qed

lemma subformulas-in-subformulas-or:
  assumes G ∨ H ∈ setSubformulae F
  shows G ∈ setSubformulae F ∧ H ∈ setSubformulae F
proof (rule conjI)
  have G ∈ setSubformulae (G ∨ H)
    by (simp only: subformulae-self UnI2 UnI1 setSubformulae-or)
  with assms show G ∈ setSubformulae F
    by (rule subsubformulae)
next
  have H ∈ setSubformulae (G ∨ H)
    by (simp only: subformulae-self UnI2 UnI1 setSubformulae-or)
  with assms show H ∈ setSubformulae F
    by (rule subsubformulae)
qed

lemma subformulas-in-subformulas-imp:
  assumes G → H ∈ setSubformulae F
  shows G ∈ setSubformulae F ∧ H ∈ setSubformulae F
proof (rule conjI)
```

```
 have G ∈ setSubformulae (G → H)
  by (simp only: subformulae-self UnI2 UnI1 setSubformulae-imp)
 with assms show G ∈ setSubformulae F
  by (rule subsubformulae)
next
 have H ∈ setSubformulae (G → H)
  by (simp only: subformulae-self UnI2 UnI1 setSubformulae-imp)
 with assms show H ∈ setSubformulae F
  by (rule subsubformulae)
qed

lemmas subformulas-in-subformulas =
 subformulas-in-subformulas-and
 subformulas-in-subformulas-or
 subformulas-in-subformulas-imp
 subformulas-in-subformulas-not
```

1.3 Conectivas generalizadas

En esta sección definiremos nuevas conectivas y fórmulas a partir de las ya definidas en el apartado anterior, junto con varios resultados sobre las mismas. Veamos el primero.

Definición 1.3.1 *Se define la fórmula* $\top$ *como la implicación* $\bot \longrightarrow \bot$.

Se formaliza del siguiente modo.

```
definition Top (⊤) where
 ⊤ ≡ ⊥ → ⊥
```

Como podemos observar, se define mediante una relación de equivalencia con otra fórmula ya conocida. El uso de dicha equivalencia justifica el tipo *definition* empleado en este caso.

Por la propia definición, es claro que $\top$ no contiene ninguna variable proposicional, como se verifica a continuación en Isabelle.

```
lemma atoms ⊤ = ∅
 by (simp only: Top-def formula.set Un-absorb)
```

A continuación vamos a definir dos conectivas que generalizan la conjunción y la disyunción para una lista finita de fórmulas.

En Isabelle está predefinido el tipo listas de la siguiente manera:

Definición 1.3.2 *Las listas de un tipo de elemento cualquiera se definen recursivamente como sigue.*

La lista vacía es una lista.

Sea x un elemento, y xs una lista de elementos de su mismo tipo. Entonces, x#xs es una lista.

La conjunción y disyunción generalizadas se definen sobre listas de fórmulas de manera recursiva:

Definición 1.3.3 *La conjunción generalizada de una lista de fórmulas se define recursivamente como:*

- *La conjunción generalizada de la lista vacía es* $\neg\bot$.
- *Sea F una fórmula y Fs una lista de fórmulas. Entonces, la conjunción generalizada de F#Fs es la conjunción de F con la conjunción generalizada de Fs.*

Definición 1.3.4 *La disyunción generalizada de una lista de fórmulas se define recursivamente como:*

- *La disyunción generalizada de la lista vacía es* $\bot$.
- *Sea F una fórmula y Fs una lista de fórmulas. Entonces, la disyunción generalizada de F#Fs es la disyunción de F con la disyunción generalizada de Fs.*

Notemos que al referirnos simplemente a disyunción o conjunción en las siguientes definiciones nos referiremos a la de dos elementos.

Su formalización en Isabelle es la siguiente:

primrec *BigAnd* :: *'a formula list* ⇒ *'a formula* ($\bigwedge$-) **where**
$\bigwedge[] = (\neg\bot)$
| $\bigwedge(F\#Fs) = F \wedge \bigwedge Fs$

primrec *BigOr* :: *'a formula list* ⇒ *'a formula* ($\bigvee$-) **where**
$\bigvee[] = \bot$
| $\bigvee(F\#Fs) = F \vee \bigvee Fs$

Ambas nuevas conectivas se definen con el tipo funciones primitivas recursivas. Estas se basan en los dos casos descritos anteriormente según la definición recursiva de listas que se genera en Isabelle: la lista vacía representada como [] y la lista construida

añadiendo una fórmula a una lista de fórmulas. Además, se observa en cada definición el nuevo símbolo de notación que aparece entre paréntesis.

Por otro lado, como es habitual, de acuerdo a la definición recursiva de listas, Isabelle genera automáticamente un esquema inductivo que emplearemos más adelante.

Vamos a mostrar una propiedad sobre la conjunción plural.

Lema 1.3.5 *El conjunto de átomos de la conjunción generalizada de una lista de fórmulas es la unión de los conjuntos de átomos de cada fórmula de la lista.*

Demostración: La prueba se hace por inducción sobre listas, en particular, listas de fórmulas. Para ello, demostremos el resultado en los casos siguientes.

En primer lugar lo probaremos para la lista vacía de fórmulas. Es claro por definición que la conjunción generalizada de la lista vacía es $\neg \bot$. De este modo, su conjunto de átomos coincide con los de $\bot$, luego es el vacío. Por tanto, queda demostrado el resultado, pues el vacío es igual a la unión del conjunto de átomos de cada elemento de la lista vacía de fórmulas.

Supongamos ahora una lista de fórmulas *Fs* verificando el enunciado. Sea la fórmula *F*, vamos a probar que *F#Fs* cumple la propiedad. Por definición de la nueva conectiva, el conjunto de átomos de la conjunción generalizada de *F#Fs* es igual al conjunto de átomos de la conjunción de *F* con la conjunción generalizada de *Fs*. De este modo, por propiedades del conjunto de átomos de la conjunción, tenemos que dicho conjunto es la unión del conjunto de átomos de *F* y el conjunto de átomos de la conjunción generalizada de *Fs*. Aplicando ahora la hipótesis de inducción sobre *Fs*, tenemos que lo anterior es igual a la unión del conjunto de átomos de *F* con la unión (generalizada) de los conjuntos de átomos de cada fórmula de *Fs*. Luego, por propiedades de la unión, es equivalente a la unión de los conjuntos de átomos de cada elemento de *F#Fs* como queríamos demostrar.

□

En Isabelle se formaliza como sigue.

lemma *atoms-BigAnd*:
atoms $(\bigwedge Fs) = \bigcup$ (*atoms ' set Fs*)
oops

Observemos el lado izquierdo de la igualdad. *Fs* es una lista de fórmulas, luego la conjunción generalizada de dicha lista se trata de una fórmula. Al aplicarle *atoms* a dicha fórmula, obtenemos finalmente el conjunto de sus átomos. Por otro lado, en el lado derecho de la igualdad tenemos el conjunto *set Fs* cuyos elementos son las fórmulas de la lista *Fs*. De este modo, al aplicar *atoms '* a dicho conjunto obtenemos la imagen por *atoms* de cada uno de sus elementos, es decir, un conjunto cuyos elementos son los

conjuntos de átomos de cada fórmula de *Fs*. Por último, mediante la unión se obtiene el conjunto de los átomos de cada fórmula de la lista inicial.

Veamos ahora la demostración detallada. Esta seguirá el esquema de inducción sobre listas. Previamente vamos a probar cada caso por separado.

lemma *atoms-BigAnd-nil*:
atoms $(\bigwedge[]) = \bigcup$ (*atoms* ' *set Nil*)
proof −
have *atoms* $(\bigwedge[]) =$ *atoms* $(\neg \bot)$
by (*simp only*: *BigAnd.simps*(*1*))
also have ... $=$ *atoms* $\bot$
by (*simp only*: *formula.set*(*3*))
also have ... $= \emptyset$
by (*simp only*: *formula.set*(*2*))
also have ... $= \bigcup \emptyset$
by (*simp only*: *Union-empty*)
also have ... $= \bigcup$ (*atoms* ' $\emptyset$)
by (*simp only*: *image-empty*)
also have ... $= \bigcup$ (*atoms* ' *set* [])
by (*simp only*: *list.set*)
finally show *?thesis*
by *this*
qed

Mostramos el siguiente lema auxiliar que utilizaremos en la demostración del último caso de inducción.

lemma *union-imagen*: $f\,a \cup \bigcup$ (f ' B) $= \bigcup$ (f ' ($\{a\} \cup B$))
by (*simp only*: *Union-image-insert*
insert-is-Un[*THEN sym*])

Se trata de una modificación del lema *Union-image-insert* en Isabelle para adaptarlo al caso particular.

$$\bigcup (f \text{ ' } (\{a\} \cup B)) = f\,a \cup \bigcup (f \text{ ' } B) \qquad (\textit{Union-image-insert})$$

Para ello empleamos el lema *insert-is-Un*.

$$\textit{insert}\ a\ A = \{a\} \cup A \qquad (\textit{insert-is-Un})$$

De esta manera, la unión de un conjunto de un solo elemento y otro conjunto cualquiera es equivalente a insertar dicho elemento en el conjunto. Además, aplicamos

el lema seguido de [*THEN sym*] para mostrar la equivalencia en el sentido en el que acaba de ser enunciada por simetría, pues en Isabelle aparece en sentido opuesto. Por tanto, el lema auxiliar *union-imagen* es fundamentalmente el lema de Isabelle *Union-image-insert* teniendo en cuenta las equivalencias anteriores.

Procedamos a la demostración del último caso de inducción.

lemma *atoms-BigAnd-cons*:
assumes *atoms* $(\bigwedge Fs) = \bigcup$ $(atoms\ `\ set\ Fs)$
shows *atoms* $(\bigwedge(F\#Fs)) = \bigcup$ $(atoms\ `\ set\ (F\#Fs))$
proof −
have *atoms* $(\bigwedge(F\#Fs)) = atoms\ (F \wedge \bigwedge Fs)$
by (*simp only*: *BigAnd.simps*(2))
also have ... $= atoms\ F \cup atoms\ (\bigwedge Fs)$
by (*simp only*: *formula.set*(4))
also have ... $= atoms\ F \cup \bigcup(atoms\ `\ set\ Fs)$
by (*simp only*: *assms*)
also have ... $= \bigcup(atoms\ `\ (\{F\} \cup set\ Fs))$
by (*simp only*: *union-imagen*)
also have ... $= \bigcup(atoms\ `\ set\ (F\#Fs))$
by (*simp only*: *set-insert*)
finally show $atoms\ (\bigwedge(F\#Fs)) = \bigcup(atoms\ `\ set\ (F\#Fs))$
by *this*
qed

Por tanto, la demostración detallada completa es la siguiente.

lemma *atoms* $(\bigwedge Fs) = \bigcup$ $(atoms\ `\ set\ Fs)$
proof (*induction Fs*)
case *Nil*
then show *?case* **by** (*rule atoms-BigAnd-nil*)
next
case (*Cons a Fs*)
assume *atoms* $(\bigwedge Fs) = \bigcup(atoms\ `\ set\ Fs)$
then show *?case*
by (*rule atoms-BigAnd-cons*)
qed

Por último, su demostración automática.

lemma *atoms-BigAnd*:
atoms $(\bigwedge Fs) = \bigcup(atoms\ `\ set\ Fs)$
by (*induction Fs*) *simp-all*

Capítulo 2

Semántica

2.1 Semántica

En esta sección presentaremos la semántica de las fórmulas proposicionales y su formalización en Isabelle/HOL.

Definición 2.1.1 *Una interpretación es una aplicación del conjunto de variables proposicionales en el conjunto $\mathfrak{B}$ de los booleanos.*

De este modo, las interpretaciones asignan valores de verdad a las variables proposicionales.

En Isabelle, se formaliza como sigue.

type-synonym *'a valuation* $=$ *'a* $\Rightarrow$ *bool*

Como podemos observar, *'a valuation* representa una función entre elementos de tipo *'a* cualquiera que conforman los átomos de una fórmula proposicional a los que les asigna un booleano. Se define mediante el tipo *type-synonym*, pues consiste en renombrar una construcción ya existente en Isabelle.

Dada una interpretación, vamos a definir el valor de verdad de una fórmula proposicional en dicha interpretación.

Definición 2.1.2 *Para cada interpretación $\mathcal{A}$ existe una única aplicación $\mathcal{I}_{\mathcal{A}}$ desde el conjunto de fórmulas al conjunto $\mathfrak{B}$ de los booleanos definida recursivamente sobre la estructura de las fórmulas como sigue:*
Sea F una fórmula cualquiera,

- *Si F es una fórmula atómica de la forma p, entonces*
  $\mathcal{I}_{\mathcal{A}}(F) = \mathcal{A}(p)$.

- *Si F es la fórmula* $\bot$*, entonces* $\mathcal{I}_{\mathcal{A}}(F) = False$.
- *Si F es de la forma* $\neg$ *G para cierta fórmula G, entonces* $\mathcal{I}_{\mathcal{A}}(F) = \neg\, \mathcal{I}_{\mathcal{A}}(G)$.
- *Si F es de la forma* $G \wedge H$ *para ciertas fórmulas G y H, entonces* $\mathcal{I}_{\mathcal{A}}(F) = \mathcal{I}_{\mathcal{A}}(G) \wedge \mathcal{I}_{\mathcal{A}}(H)$.
- *Si F es de la forma* $G \vee H$ *para ciertas fórmulas G y H, entonces* $\mathcal{I}_{\mathcal{A}}(F) = \mathcal{I}_{\mathcal{A}}(G) \vee \mathcal{I}_{\mathcal{A}}(H)$.
- *Si F es de la forma* $G \longrightarrow H$ *para ciertas fórmulas G y H, entonces* $\mathcal{I}_{\mathcal{A}}(F) = \mathcal{I}_{\mathcal{A}}(G) \longrightarrow \mathcal{I}_{\mathcal{A}}(H)$.

En estas condiciones se dice que $\mathcal{I}_{\mathcal{A}}(F)$ *es el valor de la fórmula F en la interpretación* $\mathcal{A}$.

En Isabelle, dada una interpretación $\mathcal{A}$ y una fórmula F, vamos a definir $\mathcal{I}_{\mathcal{A}}(F)$ mediante la función *formula-semantics* $\mathcal{A}$ F, notado como $\mathcal{A} \models F$.

```
primrec formula-semantics ::
 'a valuation ⇒ 'a formula ⇒ bool (infix ⊨ 51) where
 A ⊨ Atom k = A k
| A ⊨ ⊥ = False
| A ⊨ Not F = (¬ A ⊨ F)
| A ⊨ And F G = (A ⊨ F ∧ A ⊨ G)
| A ⊨ Or F G = (A ⊨ F ∨ A ⊨ G)
| A ⊨ Imp F G = (A ⊨ F ⟶ A ⊨ G)
```

Como podemos observar, *formula-semantics* es una función primitiva recursiva, como indica el tipo *primrec*, notada con el símbolo infijo $\models$. De este modo, dada una interpretación $\mathcal{A}$ sobre variables proposicionales de un tipo *'a* cualquiera y una fórmula, se define el valor de la fórmula en la interpretación $\mathcal{A}$ como se muestra. Veamos algunos ejemplos.

```
notepad
begin
 fix A :: nat valuation

have (A (1 := True, 2 := False, 3 := True)
       ⊨ (¬ ((Atom 1 ∨ Atom 2)) → Atom 3)) = True
 by simp

 have (A (1 := True) ⊨ Atom 1) = True
  by simp
```

have $(\mathcal{A}\ (1 := True) \models \neg\ (Atom\ 1)) = False$
by *simp*

have $(\mathcal{A}\ (1 := True, 2 := False) \models \neg\ (Atom\ 1) \wedge (Atom\ 2)) = False$
by *simp*

have $(\mathcal{A}\ (1 := True, 2 := False, 3 := False)$
$\models (\neg\ ((Atom\ 1 \wedge Atom\ 2)) \rightarrow Atom\ 3)) = False$
by *simp*

end

En los ejemplos anteriores se ha usado la notación para funciones $f\ (a := b)$. Dicha notación abreviada se corresponde con la defínción de *fun-upd f a b*.

$$f(a := b) = (\lambda x.\ \textsf{if}\ x = a\ \textsf{then}\ b\ \textsf{else}\ f\ x) \qquad (\textit{fun-upd-def})$$

Es decir, $f\ (a{:=}b)$ es la función que para cualquier valor x del dominio, si $x = a$, entonces devuelve b. En caso contrario, devuelve el valor $f\ x$.

A continuación veamos una serie de definiciones sobre fórmulas e interpretaciones. En primer lugar, la noción de modelo de una fórmula.

Definición 2.1.3 *Una interpretación es modelo de una fórmula si el valor de la fórmula dada dicha interpretación es Verdadero.*

En Isabelle se formaliza de la siguiente manera.

definition *isModel* $\mathcal{A}\ F \equiv \mathcal{A} \models F$

Veamos cuáles de las interpretaciones de los ejemplos anteriores son modelos de las fórmulas dadas.

notepad
begin
fix $\mathcal{A}$ :: *nat valuation*

have *isModel* $(\mathcal{A}\ (1 := True))\ (Atom\ 1)$
by (*simp add*: *isModel-def*)

have $\neg$ *isModel* $(\mathcal{A}\ (1 := True))\ (\neg\ (Atom\ 1))$
by (*simp add*: *isModel-def*)

have ¬ *isModel* (*A* (1 := *True*, 2 := *False*)) (¬ (*Atom* 1) ∧ (*Atom* 2))
by (*simp add*: *isModel-def*)

have ¬ *isModel* (*A* (1 := *True*, 2 := *False*, 3 := *False*))
(¬ ((*Atom* 1 ∧ *Atom* 2)) → *Atom* 3)
by (*simp add*: *isModel-def*)

have *isModel* (*A* (1 := *True*, 2 := *False*, 3 := *True*))
(¬ ((*Atom* 1 ∨ *Atom* 2)) → *Atom* 3)
by (*simp add*: *isModel-def*)

end

Demos ahora la noción de fórmula satisfacible.

Definición 2.1.4 *Una fórmula es satisfacible si tiene algún modelo.*

Se concreta en Isabelle como sigue.

definition *satF*(*F*) ≡ ∃ *A*. *A* ⊨ *F*

Mostremos ejemplos de fórmulas satisfacibles y no satisfacibles. Estas últimas son también llamadas contradicciones, pues son falsas para cualquier interpretación.

notepad
begin
fix *A* :: *nat valuation*

have *satF* (*Atom* 1)
by (*meson formula-semantics.simps*(1) *satF-def*)

have *satF* (*Atom* 1 ∧ *Atom* 3)
using *satF-def* **by** *force*

have ¬ *satF* (*Atom* 2 ∧ (¬ (*Atom* 2)))
using *satF-def* **by** *force*

end

Como podemos observar, *isModel* y *satF* se han formalizado usando el tipo *definition* pues, en ambos casos, hemos renombrado una construcción no recursiva ya existente en Isabelle/HOL.

Continuemos con la noción de fórmula válida o tautología.

Definición 2.1.5 *F es una fórmula válida o tautología ($\models F$) si toda interpretación es modelo de F.*

Es decir, una tautología es una fórmula que es verdadera para cualquier interpretación. En otras palabras, toda interpretación es modelo de dicha fórmula. En Isabelle se formaliza de la siguiente manera.

abbreviation *valid* $(\models$ - *51*$)$ **where**
$\models F \equiv \forall\, \mathcal{A}.\ \mathcal{A} \models F$

Por otro lado, podemos observar que se ha definido mediante el tipo *abbreviation*, introduciendo una nueva notación para la construcción formada por un cuantificador universal aplicado al primer argumento de *formula-semantics*.

Veamos un ejemplo clásico de tautología: el principio del tercio excluso.

notepad
begin
fix $\mathcal{A}$ :: *nat valuation*

have $\models$ (*Atom 5* $\vee$ ($\neg$ (*Atom 5*)))
by *simp*

end

Otro ejemplo de tautología se muestra con el siguiente lema.

Lema 2.1.6 *La fórmula $\top$ es una tautología.*

Veamos su prueba.

Demostración: Sea una interpretación cualquiera $\mathcal{A}$. Es obvio que, aplicando la propiedad reflexiva de la implicación, tenemos que *Verdadero* es equivalente a suponer que valor de $\bot$ en $\mathcal{A}$ se implica a sí mismo. Por definición, se tiene que la implicación anterior es, a su vez, equivalente al valor de la fórmula $\bot \to \bot$ en la interpretación $\mathcal{A}$. Según la definición de $\top$, tenemos que esto es igual al valor de la fórmula $\top$ en la interpretación $\mathcal{A}$. Finalmente, mediante esta cadena de equivalencias se observa que el valor de $\top$ en una interpretación $\mathcal{A}$ cualquiera es *Verdadero* como queríamos probar. □

En Isabelle se enuncia y demuestra de manera detallada como sigue.

lemma $\mathcal{A} \models \top$

proof −
have $\mathcal{A} \models \bot \longrightarrow \mathcal{A} \models \bot$
by (*rule imp-refl*)
then have $\mathcal{A} \models (\bot \rightarrow \bot)$
by (*simp only*: *formula-semantics.simps*(6))
thus $\mathcal{A} \models \top$ **unfolding** *Top-def* **by** *this*
qed

Se demuestra automáticamente como sigue.

lemma *top-semantics*: $\mathcal{A} \models \top$
unfolding *Top-def* **by** *simp*

Una vez presentados los conceptos y ejemplos anteriores, continuemos con el siguiente resultado de la sección.

Lema 2.1.7 *Sea una fórmula F de modo que p es una variable proposicional que no pertenece a su conjunto de átomos. Entonces, el valor de F en una interpretación no depende del valor de la variable p en dicha interpretación.*

En Isabelle se formaliza de la siguiente manera empleando la notación de *fun-upd*.

lemma $p \notin atoms\ F \Longrightarrow (\mathcal{A}(p := V)) \models F \longleftrightarrow \mathcal{A} \models F$
oops

Veamos ahora la prueba del lema.

Demostración: Vamos a probar el resultado por inducción en la estructura recursiva de las fórmulas. Para ello, dada una interpretación cualquier $\mathcal{A}$ y una variable p que no pertenece al conjunto de átomos de una fórmula, definimos la interpretación $\mathcal{A}'$ como aquella que devuelve $\mathcal{A}(q)$ para cualquier variable q distinta de p, y un valor cualquiera V en caso contrario. Para demostrar que el valor de una fórmula en una interpretación no depende del valor de p en dicha interpretación, basta probar que el valor de la fórmula en $\mathcal{A}$ coincide con su valor en $\mathcal{A}'$ según la definición dada anteriormente. Demostremos los siguientes casos.

Sea q una fórmula atómica cualquiera tal que p no pertenece al conjunto de sus átomos $\{q\}$. De este modo, se tiene $q \neq p$. Por definición, el valor de la fórmula atómica q en la interpretación $\mathcal{A}'$, es $\mathcal{A}'(q)$. Como hemos visto que $q \neq p$, tenemos a su vez $\mathcal{A}'(q) = \mathcal{A}(q)$ según la definición de $\mathcal{A}'$. A su vez, $\mathcal{A}(q)$ es el valor de la fórmula atómica q en la interpretación $\mathcal{A}$, luego se tiene finalmente que ambos valores coinciden.

Sea la fórmula $\bot$. Por definición, el valor de dicha fórmula es *Falso* en cualquier interpretación, luego se verifica el resultado en particular.

Sea F una fórmula tal que para cualquier variable que no pertenezca al conjunto de sus átomos, entonces el valor de F en la interpretación $\mathcal{A}$ coincide con su valor en la interpretación $\mathcal{A}'$ construida como se indica en el enunciado. Vamos a demostrar el resultado para la fórmula $\neg F$ considerando una variable p cualquiera que no pertenezca al conjunto de átomos de $\neg F$. Como los conjuntos de átomos de F y $\neg F$ son el mismo, entonces p tampoco pertenece al conjunto de átomos de F. De este modo, por hipótesis de inducción, el valor de la fórmula F en la interpretación $\mathcal{A}$ coincide con su valor en la interpretación $\mathcal{A}'$. Por otro lado, por definición tenemos que el valor de la fórmula $\neg F$ en $\mathcal{A}$ es la negación del valor de F en $\mathcal{A}$. Por lo visto anteriormente según la hipótesis de inducción, esto es igual a la negación del valor de F en $\mathcal{A}'$. Por último, por definición, esto es igual al valor de $\neg F$ en $\mathcal{A}'$, como quería demostrar.

Sean ahora las fórmulas G y H tales que, para cada una, su valor en la interpretación $\mathcal{A}$ coincide con su valor en la interpretación $\mathcal{A}'$ construida como se indica en el enunciado a partir de una variable que no pertenezca al conjunto de sus átomos. Veamos que se verifica el resultado para la fórmula $G \wedge H$. Sea p una variable proposicional que no pertenece al conjunto de átomos de $G \wedge H$. Por definición, dicho conjunto es igual a la unión del conjunto de átomos de G y el conjunto de átomos de H. Por tanto, en particular p no pertenece al conjunto de átomos de G y, por hipótesis de inducción, el valor de G en la interpretación $\mathcal{A}$ coincide con su valor en la interpretación $\mathcal{A}'$. Por el mismo motivo, p no pertenece al conjunto de átomos de H y, por hipótesis de inducción, el valor de H es el mismo en las interpretaciones $\mathcal{A}$ y $\mathcal{A}'$. Aclaradas estas observaciones, se tiene por definición que el valor de la fórmula $G \wedge H$ en la interpretación $\mathcal{A}'$ es la conjunción del valor de G en $\mathcal{A}'$ y el valor de H en $\mathcal{A}'$. Por lo demostrado anteriormente según las hipótesis de inducción, esto es igual a la conjunción del valor de G en $\mathcal{A}$ y el valor de H en $\mathcal{A}$. Aplicando la definición, esto es el valor de $G \wedge H$ en la interpretación $\mathcal{A}$. Por tanto, queda probada la equivalencia en este caso.

Sean las fórmulas G y H cumpliendo las hipótesis supuestas para el caso anterior. Veamos que el resultado se verifica para la fórmula $G \vee H$. Sea p una variable proposicional que no pertenece al conjunto de átomos de $G \vee H$. Por definición, dicho conjunto es la unión de los conjuntos de átomos de G y H. Por tanto, como se ha probado en el caso anterior, tenemos que p no pertenece al conjunto de átomos de G. Por tanto, aplicando la hipótesis de inducción se tiene que el valor de G en la interpretación $\mathcal{A}$ coincide con su valor en la interpretación $\mathcal{A}'$. Análogamente ocurre para la fórmula H como vimos en el caso anterior, de modo que p no pertenece al conjunto de átomos de H. Por tanto, por hipótesis de inducción, el valor de H es el mismo en $\mathcal{A}$ y $\mathcal{A}'$. Veamos la equivalencia. Por definición tenemos que el valor de la fórmula $G \vee H$ en la interpretación $\mathcal{A}'$ es la disyunción entre el valor de G en $\mathcal{A}'$ y el valor de H en $\mathcal{A}'$. Por lo probado anteriormente según las hipótesis de inducicón, esto es igual a la disyunción entre el valor de G en $\mathcal{A}$ y el valor de H en $\mathcal{A}$. Por definición, se verifica que es igual al valor de $G \vee H$ en la interpretación $\mathcal{A}$, como queríamos demostrar.

Probemos finalmente el último caso considerando las fórmulas G y H bajo las condiciones de los dos casos anteriores. Sea p una variable proposicional que no pertenece al conjunto de átomos de $G \to H$. Como dicho conjunto es la unión del conjunto de átomos de G y el de H, p no pertenece ni al conjunto de átomos de G ni al de H. Por lo tanto, por hipótesis de inducción, el valor de G es el mismo en las interpretaciones $\mathcal{A}$ y $\mathcal{A}'$, y lo mismo ocurre para H. Veamos ahora la cadena de equivalencias. Por definición tenemos que el valor de la fórmula $G \to H$ en la interpretación $\mathcal{A}'$ es la implicación del valor de G en $\mathcal{A}'$ y el valor de H en $\mathcal{A}'$. Análogamente a los casos anteriores por las hipótesis de inducción, esto es igual a la implicación del valor de G en $\mathcal{A}$ y el valor de H en $\mathcal{A}$. Por definición, esto es igual al valor de $G \to H$ en la interpretación $\mathcal{A}$, probando así el resultado.

□

Veamos a continuación la demostración detallada del lema en Isabelle/HOL. Para facilitar la lectura, inicialmente se ha probado el resultado para cada caso de la estructura de las fórmulas como es habitual. Además, se han empleado los lemas auxiliares *irrelevant-atom-atomic-l1*, *irrelevant-atom-not-l1*, *irrelevant-atom-and-l1*, *irrelevant-atom-or-l1* e *irrelevant-atom-imp-l1* para mostrar resultados sobre la no pertenencia a los conjuntos de átomos en cada caso. Es fácil observar que no ha sido necesario el uso de lemas auxiliares en el caso de la fórmula $\bot$, pues su conjunto de átomos es el vacío.

```
lemma irrelevant-atom-atomic-l1:
  assumes p ∉ atoms (Atom x)
  shows x ≠ p
proof (rule notI)
  assume x = p
  then have p ∈ {x}
    by (simp only: singleton-iff)
  also have ... = atoms (Atom x)
    by (simp only: formula.set(1))
  finally have p ∈ atoms (Atom x)
    by this
  with assms show False
    by (rule notE)
qed

lemma irrelevant-atom-atomic:
  assumes p ∉ atoms (Atom x)
  shows (A(p := V)) ⊨ (Atom x) ⟷ A ⊨ (Atom x)
proof −
  have x ≠ p
```

```
   using assms
   by (rule irrelevant-atom-atomic-l1)
  have (𝒜(p := V)) ⊨ (Atom x) = (𝒜(p := V)) x
   by (simp only: formula-semantics.simps(1))
  also have ... = 𝒜 x
   using ‹x ≠ p›
   by (rule fun-upd-other)
  also have ... = 𝒜 ⊨ (Atom x)
   by (simp only: formula-semantics.simps(1))
  finally show ?thesis
   by this
qed

lemma irrelevant-atom-bot:
  assumes p ∉ atoms ⊥
  shows (𝒜(p := V)) ⊨ ⊥ ⟷ 𝒜 ⊨ ⊥
  by (simp only: formula-semantics.simps(2))

lemma irrelevant-atom-not-l1:
  assumes p ∉ atoms (¬ F)
  shows   p ∉ atoms F
proof
  assume p ∈ atoms F
  then have p ∈ atoms (¬ F)
   by (simp only: formula.set(3))
  with assms show False
   by (rule notE)
qed

lemma irrelevant-atom-not:
  assumes p ∉ atoms F ⟹ 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
        p ∉ atoms (¬ F)
 shows 𝒜(p := V) ⊨ ¬ F ⟷ 𝒜 ⊨ ¬ F
proof −
  have p ∉ atoms F
   using assms(2) by (rule irrelevant-atom-not-l1)
  then have 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
   by (rule assms(1))
  have 𝒜(p := V) ⊨ ¬ F = (¬ 𝒜(p := V) ⊨ F)
   by (simp only: formula-semantics.simps(3))
```

```
  also have ... = (¬ 𝒜 ⊨ F)
    by (simp only: ‹𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F›)
  also have ... = 𝒜 ⊨ ¬ F
    by (simp only: formula-semantics.simps(3))
  finally show 𝒜(p := V) ⊨ ¬ F ⟷ 𝒜 ⊨ ¬ F
    by this
qed

lemma irrelevant-atom-and-l1:
  assumes p ∉ atoms (F ∧ G)
  shows   p ∉ atoms F
proof
  assume p ∈ atoms F
  then have p ∈ atoms F ∪ atoms G
    by (rule UnI1)
  then have p ∈ atoms (F ∧ G)
    by (simp only: formula.set(4))
  with assms show False
    by (rule notE)
qed

lemma irrelevant-atom-and-l2:
  assumes p ∉ atoms (F ∧ G)
  shows   p ∉ atoms G
proof
  assume p ∈ atoms G
  then have p ∈ atoms F ∪ atoms G
    by (rule UnI2)
  then have p ∈ atoms (F ∧ G)
    by (simp only: formula.set(4))
  with assms show False
    by (rule notE)
qed

lemma irrelevant-atom-and:
  assumes p ∉ atoms F ⟹ 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
          p ∉ atoms G ⟹ 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
          p ∉ atoms (F ∧ G)
  shows 𝒜(p := V) ⊨ (F ∧ G) ⟷ 𝒜 ⊨ (F ∧ G)
proof −
```

```
  have p ∉ atoms F
    using assms(3)
    by (rule irrelevant-atom-and-l1)
  then have HF: 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
    by (rule assms(1))
  have p ∉ atoms G
    using assms(3)
    by (rule irrelevant-atom-and-l2)
  then have HG: 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
    by (rule assms(2))
  have 𝒜(p := V) ⊨ (F ∧ G) = (𝒜(p := V) ⊨ F ∧ 𝒜(p := V) ⊨ G)
    by (simp only: formula-semantics.simps(4))
  also have ... = (𝒜 ⊨ F ∧ 𝒜 ⊨ G)
    by (simp only: HF HG)
  also have ... = 𝒜 ⊨ (F ∧ G)
    by (simp only: formula-semantics.simps(4))
  finally show 𝒜(p := V) ⊨ (F ∧ G) ⟷ 𝒜 ⊨ (F ∧ G)
    by this
qed

lemma irrelevant-atom-or-l1:
  assumes p ∉ atoms (F ∨ G)
  shows   p ∉ atoms F
proof
  assume p ∈ atoms F
  then have p ∈ atoms F ∪ atoms G
    by (rule UnI1)
  then have p ∈ atoms (F ∨ G)
    by (simp only: formula.set(5))
  with assms show False
    by (rule notE)
qed

lemma irrelevant-atom-or-l2:
  assumes p ∉ atoms (F ∨ G)
  shows   p ∉ atoms G
proof
  assume p ∈ atoms G
  then have p ∈ atoms F ∪ atoms G
    by (rule UnI2)
```

```
 then have p ∈ atoms (F ∨ G)
  by (simp only: formula.set(5))
 with assms show False
  by (rule notE)
qed

lemma irrelevant-atom-or:
 assumes p ∉ atoms F ⟹ 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
       p ∉ atoms G ⟹ 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
       p ∉ atoms (F ∨ G)
 shows  𝒜(p := V) ⊨ (F ∨ G) ⟷ 𝒜 ⊨ (F ∨ G)
proof −
 have p ∉ atoms F
  using assms(3)
  by (rule irrelevant-atom-or-l1)
 then have HF: 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
  by (rule assms(1))
 have p ∉ atoms G
  using assms(3)
  by (rule irrelevant-atom-or-l2)
 then have HG: 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
  by (rule assms(2))
 have 𝒜(p := V) ⊨ (F ∨ G) = (𝒜(p := V) ⊨ F ∨ 𝒜(p := V) ⊨ G)
  by (simp only: formula-semantics.simps(5))
 also have ... = (𝒜 ⊨ F ∨ 𝒜 ⊨ G)
  by (simp only: HF HG)
 also have ... = 𝒜 ⊨ (F ∨ G)
  by (simp only: formula-semantics.simps(5))
 finally show 𝒜(p := V) ⊨ (F ∨ G) ⟷ 𝒜 ⊨ (F ∨ G)
  by this
qed

lemma irrelevant-atom-imp-l1:
 assumes p ∉ atoms (F → G)
 shows  p ∉ atoms F
proof
 assume p ∈ atoms F
 then have p ∈ atoms F ∪ atoms G
  by (rule UnI1)
 then have p ∈ atoms (F → G)
```

```
  by (simp only: formula.set(6))
 with assms show False
  by (rule notE)
qed

lemma irrelevant-atom-imp-l2:
 assumes p ∉ atoms (F → G)
 shows   p ∉ atoms G
proof
 assume p ∈ atoms G
 then have p ∈ atoms F ∪ atoms G
  by (rule UnI2)
 then have p ∈ atoms (F → G)
  by (simp only: formula.set(6))
 with assms show False
  by (rule notE)
qed

lemma irrelevant-atom-imp:
 assumes p ∉ atoms F ⟹ 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
       p ∉ atoms G ⟹ 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
       p ∉ atoms (F → G)
 shows 𝒜(p := V) ⊨ (F → G) ⟷ 𝒜 ⊨ (F → G)
proof −
 have p ∉ atoms F
  using assms(3)
  by (rule irrelevant-atom-imp-l1)
 then have HF: 𝒜(p := V) ⊨ F ⟷ 𝒜 ⊨ F
  by (rule assms(1))
 have p ∉ atoms G
  using assms(3)
  by (rule irrelevant-atom-imp-l2)
 then have HG: 𝒜(p := V) ⊨ G ⟷ 𝒜 ⊨ G
  by (rule assms(2))
 have 𝒜(p := V) ⊨ (F → G) = (𝒜(p := V) ⊨ F ⟶ 𝒜(p := V) ⊨ G)
  by (simp only: formula-semantics.simps(6))
 also have ... = (𝒜 ⊨ F ⟶ 𝒜 ⊨ G)
  by (simp only: HF HG)
 also have ... = 𝒜 ⊨ (F → G)
  by (simp only: formula-semantics.simps(6))
```

```
 finally show A(p := V) ⊨ (F → G) ⟷ A ⊨ (F → G)
  by this
qed

lemma p ∉ atoms F ⟹ (A(p := V)) ⊨ F ⟷ A ⊨ F
proof (induction F)
 case (Atom x)
 then show ?case by (rule irrelevant-atom-atomic)
next
 case Bot
 then show ?case by (rule irrelevant-atom-bot)
next
 case (Not F)
 then show ?case by (rule irrelevant-atom-not)
next
 case (And F1 F2)
 then show ?case by (rule irrelevant-atom-and)
next
 case (Or F1 F2)
 then show ?case by (rule irrelevant-atom-or)
next
 case (Imp F1 F2)
 then show ?case by (rule irrelevant-atom-imp)
qed
```

Por último, su demostración automática.

```
lemma irrelevant-atom:
 p ∉ atoms F ⟹ (A(p := V)) ⊨ F ⟷ A ⊨ F
 by (induction F) simp-all
```

Procedamos con el siguiente lema de la sección.

Lema 2.1.8 *Si dos interpretaciones coinciden sobre el conjunto de átomos de una fórmula, entonces dicha fórmula tiene el mismo valor en ambas interpretaciones.*

Su formalización en Isabelle es la siguiente.

```
lemma ∀ k ∈ atoms F. A₁ k = A₂ k ⟹ A₁ ⊨ F ⟷ A₂ ⊨ F
 oops
```

Vamos a probar el resultado.

Demostración: La prueba sigue el esquema de inducción sobre la estructura de las fórmulas. De este modo, procedamos con la demostración de cada caso.

En primer lugar sea una fórmula atómica p, donde p es una variable proposicional cualquiera. Sean las interpretaciones $\mathcal{A}_1$ y $\mathcal{A}_2$ tales que toman los mismos valores sobre el conjunto de átomos de p. Veamos que el valor de p en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Por definición, el valor de p en la interpretación $\mathcal{A}_1$ es $\mathcal{A}_1(p)$. Como el conjunto de átomos de p es $\{p\}$, se tiene por hipótesis que $\mathcal{A}_1(p) = \mathcal{A}_2(p)$. Finalmente, aplicando la definición, esto es igual al valor de la fórmula p en la interpretación $\mathcal{A}_2$, como queríamos probar.

Consideremos ahora la fórmula $\bot$ y dos interpretaciones en las condiciones del enunciado. Es fácil observar que, como el valor de $\bot$ es *Falso* en cualquier interpretación, se tiene en particular el resultado.

Sea una fórmula F tal que, si dos interpretaciones coinciden sobre el conjunto de átomos de F, entonces el valor de F es el mismo en ambas interpretaciones. Sean dos interpretaciones cualesquiera $\mathcal{A}_1$ y $\mathcal{A}_2$ que toman los mismos valores sobre el conjunto de átomos de $\neg F$. Vamos a probar que el valor de $\neg F$ en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Observemos que, como el conjunto de átomos de F y $\neg F$ coinciden, se tiene por hipótesis de inducción que el valor de F en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Por otro lado, por definición, el valor de $\neg F$ en $\mathcal{A}_1$ es la negación del valor de F en $\mathcal{A}_1$. Por la observación anterior, esto es igual a la negación del valor de F en $\mathcal{A}_2$ que, por definición, es el valor de $\neg F$ en $\mathcal{A}_2$, probando así el resultado.

Consideremos las fórmulas F y G con las mismas hipótesis que la fórmula del caso anterior. Sean las interpretaciones $\mathcal{A}_1$ y $\mathcal{A}_2$ tales que coinciden sobre el conjunto de átomos de $F \wedge G$. Vamos a probar que el valor de $F \wedge G$ en $\mathcal{A}_1$ es el mismo que en $\mathcal{A}_2$. Como el conjunto de átomos de $F \wedge G$ es la unión del conjunto de átomos de F y el conjunto de átomos de G, tenemos que $\mathcal{A}_1$ y $\mathcal{A}_2$ coinciden sobre los elementos de dicha unión. En particular, coinciden sobre los elementos del conjunto de átomos de F y, por hipótesis de inducción, tenemos que el valor de F en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Del mismo modo, las interpretaciones anteriores coinciden también sobre los elementos del conjunto de átomos de G luego, aplicando análogamente la hipótesis de inducción, tenemos que el valor de G es el mismo en las interpretaciones $\mathcal{A}_1$ y $\mathcal{A}_2$. Veamos ahora que el valor de $F \wedge G$ también coincide en dichas interpretaciones. Por definición, el valor de $F \wedge G$ en $\mathcal{A}_1$ es la conjunción del valor de F en $\mathcal{A}_1$ y el valor de G en $\mathcal{A}_1$. Por lo obtenido anteriormente por las hipótesis de inducción, tenemos que esto es igual a la conjunción del valor de F en $\mathcal{A}_2$ y el valor de G en $\mathcal{A}_2$. Por último se tiene que esto es igual al valor de $F \wedge G$ en $\mathcal{A}_2$ tras aplicar la definición.

Volvamos a considerar F y G en las condiciones anteriores y dos interpretaciones $\mathcal{A}_1$ y $\mathcal{A}_2$ que coinciden sobre el conjunto de átomos de $F \vee G$. Vamos a probar que el valor de dicha fórmula es el mismo en ambas interpretaciones. De manera análoga al

caso anterior, como el conjunto de átomos de $F \vee G$ es la unión del conjunto de átomos de F y el conjunto de átomos de G, tenemos que las interpretaciones coinciden sobre los elementos de esta unión. En particular, coinciden sobre el conjunto de átomos de F. Por tanto, por hipótesis de inducción, el valor de F en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Igualmente obtenemos que las interpretaciones coinciden sobre el conjunto de átomos de G y, aplicando de nuevo hipótesis de inducción, el valor de G es el mismo en ambas interpretaciones. Por otra parte, por definición tenemos que le valor de $F \vee G$ en la interpretación $\mathcal{A}_1$ es la disyunción entre el valor de F en $\mathcal{A}_1$ y el valor de G en $\mathcal{A}_1$. Por las observaciones anteriores derivadas de las hipótesis de inducción, tenemos que esto es igual a la disyunción entre el valor de F en $\mathcal{A}_2$ y el valor de G en $\mathcal{A}_2$. Por definición, esto es el valor de $F \vee G$ en $\mathcal{A}_2$, como queríamos demostrar.

Veamos el último caso de las fórmulas. Sean F y G fórmulas en las condiciones de los casos anteriores. Consideremos las interpretaciones $\mathcal{A}_1$ y $\mathcal{A}_2$ que coinciden sobre los elementos del conjunto de átomos de $F \to G$. Probemos que el valor de $F \to G$ es el mismo en ambas interpretaciones. Por definición, el conjunto de átomos de $F \to G$ es la unión de los conjuntos de átomos de F y G. Por tanto, dichas interpretaciones coinciden sobre los elementos de dicha unión. Como hemos visto en casos anteriores, en particular coinciden sobre los átomos de F luego, por hipótesis de inducción, el valor de F en $\mathcal{A}_1$ coincide con su valor en $\mathcal{A}_2$. Análogamente, las interpretaciones coinciden sobre los átomos de G y, por hipótesis de inducción, el valor de G es el mismo en ambas interpretaciones. Probemos que también coincide el valor de $F \to G$ en $\mathcal{A}_1$ y $\mathcal{A}_2$. Por definición, el valor de $F \to G$ en $\mathcal{A}_1$ es la implicación entre el valor de F en $\mathcal{A}_1$ y el valor de G en $\mathcal{A}_1$. De esta manera, por las observaciones anteriores tenemos que esto es igual a la implicación entre el valor de F en $\mathcal{A}_2$ y el valor de G en $\mathcal{A}_2$. Finalmente, por definición, esto es el valor de $F \to G$ en la interpretación $\mathcal{A}_2$, probando así el resultado. □

Probemos ahora el lema de forma detallada en Isabelle, haciendo cada caso de la estructura de las fórmulas por separado y empleando lemas auxiliares sobre la pertenencia a los conjuntos de átomos cuando sea necesario.

```
lemma relevant-atoms-same-semantics-atomic-l1:
 x ∈ atoms (Atom x)
proof
 show x ∈ {x}
  by (simp only: singleton-iff)
next
 show {x} ⊆ atoms (Atom x)
  by (simp only: formula.set(1))
qed
```

lemma *relevant-atoms-same-semantics-atomic*:
assumes $\forall k \in atoms\ (Atom\ x).\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
shows $\mathcal{A}_1 \models Atom\ x \longleftrightarrow \mathcal{A}_2 \models Atom\ x$
proof −
have $\mathcal{A}_1 \models Atom\ x = \mathcal{A}_1\ x$
by (*simp only*: *formula-semantics.simps*(*1*))
also have $\ldots = \mathcal{A}_2\ x$
using *assms*(*1*)
by (*simp only*: *relevant-atoms-same-semantics-atomic-l1*)
also have $\ldots = \mathcal{A}_2 \models Atom\ x$
by (*simp only*: *formula-semantics.simps*(*1*))
finally show *?thesis*
by *this*
qed

Para las fórmulas atómicas, se observa el uso del lema auxiliar *relevant-atoms-same-semantics-atomic-l*. Sigamos con los siguientes casos.

lemma *relevant-atoms-same-semantics-bot*:
assumes $\forall k \in atoms\ \bot.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
shows $\mathcal{A}_1 \models \bot \longleftrightarrow \mathcal{A}_2 \models \bot$
by (*simp only*: *formula-semantics.simps*(*2*))

lemma *relevant-atoms-same-semantics-not*:
assumes $\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
$\forall k \in atoms\ (\neg F).\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
shows $\mathcal{A}_1 \models (\neg F) \longleftrightarrow \mathcal{A}_2 \models (\neg F)$
proof −
have $\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$ **using** *assms*(*2*)
by (*simp only*: *formula.set*(*3*))
then have *H*:$\mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
by (*simp only*: *assms*(*1*))
have $\mathcal{A}_1 \models (\neg F) = (\neg\ \mathcal{A}_1 \models F)$
by (*simp only*: *formula-semantics.simps*(*3*))
also have $\ldots = (\neg\ \mathcal{A}_2 \models F)$
using *H* **by** (*rule arg-cong*)
also have $\ldots = \mathcal{A}_2 \models (\neg F)$
by (*simp only*: *formula-semantics.simps*(*3*))
finally show *?thesis*
by *this*
qed

Para los casos de la fórmula $\bot$ y la negación $\neg\, F$ no han sido necesarios los lemas auxiliares pues, en el primer caso, el conjunto de átomos es el vacío y, en el segundo caso, el conjunto de átomos de $\neg\, F$ coincide con el de F. Finalmente, introducimos los siguientes lemas auxiliares para facilitar las demostraciones detalladas en Isabelle de los casos correspondientes a las conectivas binarias.

lemma *forall-union1*:
 assumes $\forall\, x \in A \cup B.\ P\ x$
 shows $\forall\, x \in A.\ P\ x$
proof (*rule ballI*)
 fix x
 assume $x \in A$
 then have $x \in A \cup B$
  by (*simp only*: *UnI1*)
 then show $P\ x$
  by (*simp only*: *assms*)
qed

lemma *forall-union2*:
 assumes $\forall\, x \in A \cup B.\ P\ x$
 shows $\forall\, x \in B.\ P\ x$
proof (*rule ballI*)
 fix x
 assume $x \in B$
 then have $x \in A \cup B$
  by (*simp only*: *UnI2*)
 then show $P\ x$
  by (*simp only*: *assms*)
qed

Empleando dichos resultados, veamos las demostraciones detalladas de los tres últimos casos. Después se mostrará la demostración detallada del lema completo en Isabelle.

lemma *relevant-atoms-same-semantics-and*:
 assumes $\forall\, k \in \mathit{atoms}\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
  $\forall\, k \in \mathit{atoms}\ G.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models G \longleftrightarrow \mathcal{A}_2 \models G$
  $\forall\, k \in \mathit{atoms}\ (F \wedge G).\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
 shows $\mathcal{A}_1 \models (F \wedge G) \longleftrightarrow \mathcal{A}_2 \models (F \wedge G)$
proof $-$
 have H:$\forall\, k \in \mathit{atoms}\ F \cup \mathit{atoms}\ G.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$ **using** *assms*(3)
  by (*simp only*: *formula.set*(4))

```
  then have ∀k ∈ atoms F. A₁ k = A₂ k
    by (rule forall-union1)
  then have H1:A₁ ⊨ F ⟷ A₂ ⊨ F
    by (simp only: assms(1))
  have ∀k ∈ atoms G. A₁ k = A₂ k
    using H by (rule forall-union2)
  then have H2:A₁ ⊨ G ⟷ A₂ ⊨ G
    by (simp only: assms(2))
  have A₁ ⊨ F ∧ G = (A₁ ⊨ F ∧ A₁ ⊨ G)
    by (simp only: formula-semantics.simps(4))
  also have ... = (A₂ ⊨ F ∧ A₂ ⊨ G)
    using H1 H2 by (rule arg-cong2)
  also have ... = A₂ ⊨ F ∧ G
    by (simp only: formula-semantics.simps(4))
  finally show ?thesis
    by this
qed

lemma relevant-atoms-same-semantics-or:
  assumes ∀k ∈ atoms F. A₁ k = A₂ k ⟹ A₁ ⊨ F ⟷ A₂ ⊨ F
          ∀k ∈ atoms G. A₁ k = A₂ k ⟹ A₁ ⊨ G ⟷ A₂ ⊨ G
          ∀k ∈ atoms (F ∨ G). A₁ k = A₂ k
    shows A₁ ⊨ (F ∨ G) ⟷ A₂ ⊨ (F ∨ G)
proof −
  have H:∀k ∈ atoms F ∪ atoms G. A₁ k = A₂ k using assms(3)
    by (simp only: formula.set(5))
  then have ∀k ∈ atoms F. A₁ k = A₂ k
    by (rule forall-union1)
  then have H1:A₁ ⊨ F ⟷ A₂ ⊨ F
    by (simp only: assms(1))
  have ∀k ∈ atoms G. A₁ k = A₂ k
    using H by (rule forall-union2)
  then have H2:A₁ ⊨ G ⟷ A₂ ⊨ G
    by (simp only: assms(2))
  have A₁ ⊨ F ∨ G = (A₁ ⊨ F ∨ A₁ ⊨ G)
    by (simp only: formula-semantics.simps(5))
  also have ... = (A₂ ⊨ F ∨ A₂ ⊨ G)
    using H1 H2 by (rule arg-cong2)
  also have ... = A₂ ⊨ F ∨ G
    by (simp only: formula-semantics.simps(5))
```

finally show *?thesis*
by *this*
qed

lemma *relevant-atoms-same-semantics-imp*:
assumes $\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
$\forall k \in atoms\ G.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models G \longleftrightarrow \mathcal{A}_2 \models G$
$\forall k \in atoms\ (F \rightarrow G).\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
shows $\mathcal{A}_1 \models (F \rightarrow G) \longleftrightarrow \mathcal{A}_2 \models (F \rightarrow G)$
proof −
have *H*:$\forall k \in atoms\ F \cup atoms\ G.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$ **using** *assms*(3)
by (*simp only*: *formula.set*(6))
then have $\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
by (*rule forall-union1*)
then have *H1*:$\mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
by (*simp only*: *assms*(1))
have $\forall k \in atoms\ G.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k$
using *H* **by** (*rule forall-union2*)
then have *H2*:$\mathcal{A}_1 \models G \longleftrightarrow \mathcal{A}_2 \models G$
by (*simp only*: *assms*(2))
have $\mathcal{A}_1 \models F \rightarrow G = (\mathcal{A}_1 \models F \longrightarrow \mathcal{A}_1 \models G)$
by (*simp only*: *formula-semantics.simps*(6))
also have $\ldots = (\mathcal{A}_2 \models F \longrightarrow \mathcal{A}_2 \models G)$
using *H1 H2* **by** (*rule arg-cong2*)
also have $\ldots = \mathcal{A}_2 \models F \rightarrow G$
by (*simp only*: *formula-semantics.simps*(6))
finally show *?thesis*
by *this*
qed

lemma $\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
proof (*induction F*)
case (*Atom x*)
then show *?case* **by** (*rule relevant-atoms-same-semantics-atomic*)
next
case *Bot*
then show *?case* **by** (*rule relevant-atoms-same-semantics-bot*)
next
case (*Not F*)
then show *?case* **by** (*rule relevant-atoms-same-semantics-not*)

next
case (*And F1 F2*)
then show *?case* **by** (*rule relevant-atoms-same-semantics-and*)
next
case (*Or F1 F2*)
then show *?case* **by** (*rule relevant-atoms-same-semantics-or*)
next
case (*Imp F1 F2*)
then show *?case* **by** (*rule relevant-atoms-same-semantics-imp*)
qed

Su demostración automática es la siguiente.

lemma *relevant-atoms-same-semantics*:
$\forall k \in atoms\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
by (*induction F*) *simp-all*

2.2 Semántica de fórmulas con conectivas generalizadas

En este apartado mostraremos varios resultados relativos a la semántica de las fórmulas construidas con conectivas generalizadas.

Lema 2.2.1 *Una interpretación es modelo de la conjunción generalizada de una lista de fórmulas si y solo si es modelo de cada fórmula de la lista.*

Su formalización en Isabelle es la siguiente.

lemma $(\mathcal{A} \models \bigwedge Fs) \longleftrightarrow (\forall f \in set\ Fs.\ \mathcal{A} \models f)$
oops

Como podemos observar, en el enunciado de la derecha hemos empleado *set* para cambiar al tipo de los conjuntos la lista de fórmulas, pues esto permite emplear el cuantificador universal.

Procedamos con la prueba del lema.

Demostración: La prueba se basa en el esquema de inducción sobre listas. Para ello, demostraremos el resultado mediante cadenas de equivalencias en los siguientes casos.

En primer lugar, lo probamos para la lista vacía de fórmulas. Sea la interpretación $\mathcal{A}$ tal que es modelo de la conjunción generalizada de la lista vacía. Por definición de la conjunción generalizada, $\mathcal{A}$ es modelo de $\neg\ \bot$. Aplicando la definición del valor de una fórmula en una interpretación para el caso de la negación, tenemos que esto es

equivalente a que $\mathcal{A}$ no es modelo de $\bot$. Análogamente, como sabemos que el valor de $\bot$ es *Falso* en cualquier interpretación, se tiene que lo anterior es equivalente a $\neg$ *Falso*, es decir, *Verdadero*. Por otro lado, por propiedades del conjunto vacío, se tiene que toda propiedad sobre sus elementos es verdadera. Por tanto, lo anterior es equivalente a decir que $\mathcal{A}$ es modelo de todos los elementos del conjunto vacío. Es decir, $\mathcal{A}$ es modelo de todos los elementos de la lista vacía, como queríamos demostrar.

Consideramos ahora la interpretación $\mathcal{A}$ y la lista de fórmulas *Fs* de modo que $\mathcal{A}$ es modelo de la conjunción generalizada de *Fs* si y solo si es modelo de cada fórmula de *Fs*. Veamos ahora que se verifica la propiedad para la lista *F#Fs* formada al añadir la fórmula *F*. En primer lugar, si $\mathcal{A}$ es modelo de la conjunción generalizada de *F#Fs*, por definición de dicha conjunción, esto es equivalente a que $\mathcal{A}$ es modelo de la conjunción de *F* y la conjunción generalizada de *Fs*. Según el valor de una fórmula en una interpretación, esto es a su vez equivalente a la conjunción de "$\mathcal{A}$ es modelo de *F*" y "$\mathcal{A}$ es modelo de la conjunción generalizada de *Fs*". Aplicando la hipótesis de inducción sobre el segundo término de la conjunción, es equivalente a la conjunción de "$\mathcal{A}$ es modelo de *F*" y "$\mathcal{A}$ es modelo de toda fórmula del conjunto formado por los elementos de *Fs*". Equivalentemente, $\mathcal{A}$ es modelo de toda fórmula del conjunto formado por la unión de $\{F\}$ y el conjunto de los elementos de *Fs*. Es decir, $\mathcal{A}$ es modelo de toda fórmula del conjunto formado por los elementos de *F#Fs*, probando así el resultado.

□

Veamos ahora su demostración detallada en Isabelle. Primero se probarán los dos casos correspondientes a la estructura de listas por separado.

lemma *BigAnd-semantics-nil*: $(\mathcal{A} \models \bigwedge[]) \longleftrightarrow (\forall f \in set\ []. \mathcal{A} \models f)$
proof−
 have $(\mathcal{A} \models \bigwedge[]) = \mathcal{A} \models (\neg\bot)$
  by (*simp only*: *BigAnd.simps*(*1*))
 also have $\ldots = (\neg\, \mathcal{A} \models \bot)$
  by (*simp only*: *formula-semantics.simps*(*3*))
 also have $\ldots = (\neg\, False)$
  by (*simp only*: *formula-semantics.simps*(*2*))
 also have $\ldots = True$
  by (*simp only*: *not-False-eq-True*)
 also have $\ldots = (\forall f \in \emptyset. \mathcal{A} \models f)$
  by (*simp only*: *ball-empty*)
 also have $\ldots = (\forall f \in set\ []. \mathcal{A} \models f)$
  by (*simp only*: *list.set*)
 finally show *?thesis*
  by *this*
qed

Para el caso de la lista formada añadiendo un elemento, vamos a emplear el siguiente lema auxiliar.

lemma *Bigprop1*: $(\forall\, x{\in}(\{a\} \cup B).\ P\ x) = (P\ a \wedge (\forall\, x{\in}B.\ P\ x))$
by (*simp only*: *ball-simps*(7)
insert-is-Un[*THEN sym*])

Se trata de una adaptación del séptimo apartado del lema *ball-simps* en Isabelle, para adecuarlo a nuestro caso particular.

$$(\forall\, x{\in}\mathit{insert}\ a\ B.\ P\ x) \longleftrightarrow (P\ a \wedge (\forall\, x{\in}B.\ P\ x)) \qquad (\mathit{ball\text{-}simps}(7))$$

Para ello, empleamos el lema *insert-is-Un* seguido de [*THEN sym*] como hemos hecho anteriormente.

Procedamos, así, con la demostración del caso del lema correspondiente. Por último, mostraremos también su demostración detallada completa.

lemma *BigAnd-semantics-cons*:
assumes $(\mathcal{A} \models \bigwedge Fs) \longleftrightarrow (\forall f \in \mathit{set}\ Fs.\ \mathcal{A} \models f)$
shows $(\mathcal{A} \models \bigwedge(F\#Fs)) \longleftrightarrow (\forall f \in \mathit{set}\ (F\#Fs).\ \mathcal{A} \models f)$
proof −
have $(\mathcal{A} \models \bigwedge(F\#Fs)) = \mathcal{A} \models F \wedge \bigwedge Fs$
by (*simp only*: *BigAnd.simps*(2))
also have $\ldots = (\mathcal{A} \models F \wedge \mathcal{A} \models \bigwedge Fs)$
by (*simp only*: *formula-semantics.simps*(4))
also have $\ldots = (\mathcal{A} \models F \wedge (\forall f \in \mathit{set}\ Fs.\ \mathcal{A} \models f))$
by (*simp only*: *assms*)
also have $\ldots = (\forall f \in (\{F\} \cup \mathit{set}\ Fs).\ \mathcal{A} \models f)$
by (*simp only*: *Bigprop1*)
also have $\ldots = (\forall f \in \mathit{set}\ (F\#Fs).\ \mathcal{A} \models f)$
by (*simp only*: *set-insert*)
finally show *?thesis*
by *this*
qed

lemma $(\mathcal{A} \models \bigwedge Fs) \longleftrightarrow (\forall f \in \mathit{set}\ Fs.\ \mathcal{A} \models f)$
proof (*induction Fs*)
case *Nil*
then show *?case* **by** (*rule BigAnd-semantics-nil*)
next
case (*Cons a Fs*)
then show *?case* **by** (*rule BigAnd-semantics-cons*)

qed

Su demostración automática es la siguiente.

lemma *BigAnd-semantics*:
$(\mathcal{A} \models \bigwedge Fs) \longleftrightarrow (\forall f \in set\ Fs.\ \mathcal{A} \models f)$
by (*induction Fs*) *simp-all*

Finalmente, un resultado sobre la disyunción generalizada.

Lema 2.2.2 *Una interpretación es modelo de la disyunción generalizada de una lista de fórmulas si y solo si existe una fórmula en la lista de la cual sea modelo.*

Su formalización en Isabelle es la siguiente.

lemma $(\mathcal{A} \models \bigvee Fs) \longleftrightarrow (\exists f \in set\ Fs.\ \mathcal{A} \models f)$
oops

Procedamos con la demostración del resultado.

Demostración: La prueba sigue el esquema de inducción sobre la estructura de listas. Vamos a probar los dos casos mediante cadenas de equivalencias.

Sea la lista vacía de fórmulas y una interpretación cualquiera $\mathcal{A}$. Por definición de la disyunción generalizada, si $\mathcal{A}$ es modelo de la disyunción generalizada de la lista vacía, equivalentemente tenemos que es modelo de $\bot$, es decir, *Falso*. En particular, esto es equivalente a suponer que existe una fórmula en el conjunto vacío tal que $\mathcal{A}$ es modelo suyo.

Consideremos ahora la lista de fórmulas *Fs* y una interpretación $\mathcal{A}$ tal que es modelo de *Fs* si y solo si es modelo de cada fórmula del conjunto formado por los elementos de *Fs*. Vamos a probar el resultado para la lista *F#Fs* dada cualquier fórmula *F*. Si $\mathcal{A}$ es modelo de la disyunción generalizada de *F#Fs*, por definición, es equivalente a la disyunción entre "$\mathcal{A}$ es modelo de *F*" y "$\mathcal{A}$ es modelo de la disyunción generalizada de *Fs*". Aplicando la hipótesis de inducción, tenemos equivalentemente la disyunción entre "$\mathcal{A}$ es modelo de *F*" y "existe una fórmula perteneciente al conjunto de elementos de *Fs* tal que $\mathcal{A}$ es modelo suyo". Por tanto, por propiedades de la disyunción, esto es equivalente a que exista una fórmula perteneciente a la unión de $\{F\}$ y el conjunto de los elementos de *Fs* que tiene a $\mathcal{A}$ como modelo. Finalmente, tenemos que esto ocurre si y solo si existe una fórmula en el conjunto de los elementos de *F#Fs* de la cual $\mathcal{A}$ sea modelo, como queríamos demostrar.

□

A continuación lo probamos de manera detallada con Isabelle/HOL, haciendo previamente cada paso por separado.

lemma *BigOr-semantics-nil*: $(\mathcal{A} \models \bigvee[]) \longleftrightarrow (\exists f \in set\ []. \mathcal{A} \models f)$
proof −
have $(\mathcal{A} \models \bigvee[]) = \mathcal{A} \models \bot$
by (*simp only*: *BigOr.simps*(1))
also have $\ldots = False$
by (*simp only*: *formula-semantics.simps*(2))
also have $\ldots = (\exists f \in \emptyset. \mathcal{A} \models f)$
by (*simp only*: *bex-empty*)
also have $\ldots = (\exists f \in set\ []. \mathcal{A} \models f)$
by (*simp only*: *list.set*)
finally show *?thesis*
by *this*
qed

Para el segundo caso de las listas emplearemos el siguiente lema auxiliar.

lemma *Bigprop2*: $(\exists x{\in}\{a\} \cup B.\ P\ x) = (P\ a \vee (\exists x{\in}B.\ P\ x))$
by (*simp only*: *bex-simps*(5)
insert-is-Un[*THEN sym*])

De forma similar al resultado sobre conjunción generalizada, se trata de una modificación del quinto apartado del lema *bex-simps* en Isabelle para adaptarlo al caso particular.

$$(\exists x{\in}insert\ a\ B.\ P\ x) \longleftrightarrow (P\ a \vee (\exists x{\in}B.\ P\ x)) \qquad (bex\text{-}simps(5))$$

Para modificarlo, empleamos análogamente el lema *insert-is-Un* seguido de [*THEN sym*], procediendo de manera similar a los casos en los que se ha usado con anterioridad.

Seguimos, así, con la demostración detallada del lema.

lemma *BigOr-semantics-cons*:
assumes $(\mathcal{A} \models \bigvee Fs) \longleftrightarrow (\exists f \in set\ Fs. \mathcal{A} \models f)$
shows $(\mathcal{A} \models \bigvee(F\#Fs)) \longleftrightarrow (\exists f \in set\ (F\#Fs). \mathcal{A} \models f)$
proof −
have $(\mathcal{A} \models \bigvee(F\#Fs)) = \mathcal{A} \models F \vee \bigvee Fs$
by (*simp only*: *BigOr.simps*(2))
also have $\ldots = (\mathcal{A} \models F \vee \mathcal{A} \models \bigvee Fs)$
by (*simp only*: *formula-semantics.simps*(5))
also have $\ldots = (\mathcal{A} \models F \vee (\exists f \in set\ Fs. \mathcal{A} \models f))$
by (*simp only*: *assms*)
also have $\ldots = (\exists f \in (\{F\} \cup set\ Fs). \mathcal{A} \models f)$
by (*simp only*: *Bigprop2*)

```
 also have ... = (∃f ∈ set (F#Fs). 𝒜 ⊨ f)
  by (simp only: set-insert)
 finally show ?thesis
  by this
qed

lemma (𝒜 ⊨ ⋁Fs) ⟷ (∃f ∈ set Fs. 𝒜 ⊨ f)
proof (induction Fs)
case Nil
 then show ?case by (rule BigOr-semantics-nil)
next
 case (Cons a Fs)
then show ?case by (rule BigOr-semantics-cons)
qed

lemma BigOr-semantics:
 (𝒜 ⊨ ⋁Fs) ⟷ (∃f ∈ set Fs. 𝒜 ⊨ f)
 by (induction Fs) simp-all
```

2.3 Semántica de conjuntos de fórmulas

Veamos definiciones y resultados relativos a la semántica de un conjunto de fórmulas.

En primer lugar, extendamos la noción de modelo a un conjunto de fórmulas.

Definición 2.3.1 *Una interpretación es modelo de un conjunto de fórmulas si es modelo de todas las fórmulas del conjunto.*

Su formalización en Isabelle es la siguiente.

```
definition isModelSet 𝒜 S ≡ ∀F. (F ∈ S ⟶ 𝒜 ⊨ F)
```

Continuando con los ejemplos anteriores, veamos una interpretación que es modelo de un conjunto de fórmulas.

```
notepad
begin
 fix 𝒜 :: nat valuation

 have isModelSet (𝒜 (4 := True))
   {Atom 4, (Atom 4 ∧ Atom 4) → Atom 4}
  by (simp add: isModelSet-def)
```

end

El siguiente resultado relaciona los conceptos de modelo de una fórmula y modelo de un conjunto de fórmulas en Isabelle. La equivalencia se demostrará fácilmente mediante las definiciones de *isModel* e *isModelSet*.

lemma *modelSet*:
isModelSet $\mathcal{A}$ $S \equiv \forall F.\ (F \in S \longrightarrow$ *isModel* $\mathcal{A}$ $F)$
by (*simp only*: *isModelSet-def isModel-def*)

Veamos la noción de satisfacibilidad para un conjunto de fórmulas.

Definición 2.3.2 *Un conjunto de fórmulas es satisfacible si tiene algún modelo.*

En Isabelle se formaliza de la siguiente manera.

definition *sat* $S \equiv \exists \mathcal{A}.\ \forall F \in S.\ \mathcal{A} \models F$

En otras palabras, la satisfacibilidad de un conjunto de fórmulas depende de la existencia de una interpretación que sea modelo de dicho conjunto, es decir, que sea modelo de todas las fórmulas del conjunto.

El siguiente lema muestra una forma alternativa de definir un conjunto de fórmulas satisfacible en Isabelle empleando *isModelSet* y *sat*, según la observación anterior.

lemma *sat* $S \equiv \exists \mathcal{A}.$ *isModelSet* $\mathcal{A}$ S
oops

Veamos sus demostraciones detallada y automática. Para ello, introducimos inicialmente el lema auxiliar *forall-set*.

lemma *forall-set*:
$(\forall x.\ (x \in A \longrightarrow P\ x)) = (\forall x \in A.\ P\ x)$
proof (*rule iffI*)
assume *H1*:$\forall x.\ (x \in A \longrightarrow P\ x)$
show $\forall x \in A.\ P\ x$
proof (*rule ballI*)
fix x
have $x \in A \longrightarrow P\ x$
using *H1* **by** (*rule allE*)
thus $x \in A \Longrightarrow P\ x$
by (*rule mp*)
qed
next

```
  assume H2: ∀ x ∈ A. P x
  show ∀ x. (x ∈ A ⟶ P x)
  proof (rule allI)
   fix x
   show x ∈ A ⟶ P x
   proof (rule impI)
    assume x ∈ A
    show P x
     using H2 ⟨x ∈ A⟩ by (rule bspec)
   qed
  qed
qed

lemma sat S ≡ ∃ 𝒜. isModelSet 𝒜 S
proof −
  have sat S = (∃ 𝒜. isModelSet 𝒜 S)
  proof (rule iffI)
   assume H1:∃ 𝒜. isModelSet 𝒜 S
   obtain 𝒜 where isModelSet 𝒜 S
    using H1 by (rule exE)
   then have ∀ F. (F ∈ S ⟶ 𝒜 ⊨ F)
    by (simp only: isModelSet-def)
   then have ∀ F ∈ S. 𝒜 ⊨ F
    by (simp only: forall-set)
   then have ∃ 𝒜. ∀ F ∈ S. 𝒜 ⊨ F
    by (simp only: exI)
   thus sat S
    by (simp only: sat-def)
  next
   assume sat S
   then have H2:∃ 𝒜. ∀ F ∈ S. 𝒜 ⊨ F
    by (simp only: sat-def)
   obtain 𝒜 where ∀ F ∈ S. 𝒜 ⊨ F
    using H2 by (rule exE)
   then have ∀ F. (F ∈ S ⟶ 𝒜 ⊨ F)
    by (simp only: forall-set)
   then have isModelSet 𝒜 S
    by (simp only: isModelSet-def)
   thus ∃ 𝒜. isModelSet 𝒜 S
    by (simp only: exI)
```

```
 qed
 thus sat S  ≡ ∃ 𝒜. isModelSet 𝒜 S
  by (simp only: atomize-eq)
qed

lemma satAlt:
 sat S ≡ ∃ 𝒜. isModelSet 𝒜 S
 by (smt isModelSet-def sat-def )
```

Mostremos algunos ejemplos de conjuncto satisfacible. Por definición, se observa que el conjunto de fórmulas utilizado en el ejemplo de *modelSet* es satisfacible. Por otro lado, un caso de conjunto de fórmulas no satisfacible es cualquiera que incluya una contradicción entre sus elementos.

Por otra parte, en particular, se puede definir un conjunto de fórmulas finitamente satisfacible.

Definición 2.3.3 *Un conjunto de fórmulas es finitamente satisfacible si todo subconjunto finito suyo es satisfacible.*

Su formalización en Isabelle se muestra a continuación.

```
definition fin-sat S ≡ (∀ s ⊆ S. finite s ⟶ sat s)
```

Continuemos con la noción de consecuencia lógica.

Definición 2.3.4 *Una fórmula es consecuencia lógica de un conjunto de fórmulas si todos los modelos del conjunto son modelos de la fórmula.*

Teniendo en cuenta la definición de modelo de una fórmula y modelo de un conjunto de fórmulas, su formalización en Isabelle es la siguiente.

```
definition entailment ::
 'a formula set ⇒ 'a formula ⇒ bool ((- ⊨/ -)
   [53,53] 53) where
 Γ ⊨ F ≡ (∀ 𝒜. ((∀ G ∈ Γ. 𝒜 ⊨ G) ⟶ (𝒜 ⊨ F)))
```

Hagamos varias observaciones sobre esta definición. En primer lugar, hemos usado el tipo *definition*. Por otro lado, no hemos empleado *isModel* ni *isModelSet* para su definición ya que, de este modo, no tenemos que desplegar dichas definiciones en las demostraciones detalladas y automáticas de los lemas posteriores. Finalmente se puede observar la notación ⊨. En la teoría clásica no se suele emplear una nueva notación, ya

que se diferencia por el contexto. En Isabelle/HOL es imprescindible aclarar la diferencia.

Mostremos algún ejemplo de fórmula que sea consecuencia lógica de un conjunto.

```
notepad
begin
 fix 𝒜 :: nat valuation

 have {Atom 1 ⟶ Atom 2, Atom 2 ⟶ Atom 3} ⟙= (Atom 1 ⟶ Atom 3)
  by (simp add: entailment-def)

end
```

Llegamos así al último lema de la sección.

Lema 2.3.5 *$\bot$ es consecuencia lógica de un conjunto si y solo si el conjunto es insatisfacible.*

Su formalización en Isabelle es la siguiente.

```
lemma Γ ⟙= ⊥ ⟷ ¬ sat Γ
 oops
```

Comencemos las demostraciones del resultado.

Demostración: Vamos a probar la doble implicación mediante una cadena de equivalencias.

Sea un conjunto de fórmulas Γ tal que la fórmula $\bot$ es consecuencia lógica de dicho conjunto. Por definición, esto es equivalente a que toda interpretación que sea modelo de Γ es, a su vez, modelo de $\bot$. Como el valor de $\bot$ es *Falso* en cualquier interpretación, ninguna interpretación es modelo suyo. Por tanto, por la hipótesis inicial, se verifica que ninguna interpretación es modelo del conjunto Γ. Es decir, no existe ninguna interpretación que sea modelo de dicho conjunto. Según la definición, esto es equivalente a que el conjunto Γ es insatisfacible, como queríamos demostrar.

□

Procedamos con las pruebas en Isabelle/HOL.

```
lemma Γ ⟙= ⊥ ⟷ ¬ sat Γ
proof −
 have Γ ⟙= ⊥ = (∀ 𝒜. ((∀ G ∈ Γ. 𝒜 ⟙= G) ⟶ 𝒜 ⟙= ⊥))
  by (simp only: entailment-def)
 also have ... = (∀ 𝒜. ((∀ G ∈ Γ. 𝒜 ⟙= G) ⟶ False))
  by (simp only: formula-semantics.simps(2))
```

```
  also have ... = (∀ 𝒜. ¬(∀ G ∈ Γ. 𝒜 ⊨ G))
    by (simp only: not-def)
  also have ... = (¬(∃ 𝒜. ∀ G ∈ Γ. 𝒜 ⊨ G))
    by (simp only: not-ex)
  also have ... = (¬ sat Γ)
    by (simp only: sat-def)
  finally show ?thesis
    by this
qed
```

Finalmente su demostración automática es la siguiente.

```
lemma entail-sat:
  Γ ⊨= ⊥ ⟷ ¬ sat Γ
  unfolding sat-def entailment-def
  by simp
```

Capítulo 3

Lema de Hintikka

3.1 Conjuntos de Hintikka y propiedades básicas

En esta sección presentaremos un tipo de conjuntos de fórmulas: los conjuntos de Hintikka. Probaremos finalmente que todo conjunto de Hintikka es satisfacible.

Definición 3.1.1 *Se llama conjunto de Hintikka a todo conjunto de fórmulas S que verifica las siguientes condiciones:*

1. $\bot \notin S$.
2. *Dada p una fórmula atómica cualquiera, no se tiene simultáneamente que* $p \in S$ *y* $\neg p \in S$.
3. *Si* $F \wedge G \in S$, *entonces* $F \in S$ *y* $G \in S$.
4. *Si* $F \vee G \in S$, *entonces* $F \in S$ *o* $G \in S$.
5. *Si* $F \to G \in S$, *entonces* $\neg F \in S$ *o* $G \in S$.
6. *Si* $\neg(\neg F) \in S$, *entonces* $F \in S$.
7. *Si* $\neg(F \wedge G) \in S$, *entonces* $\neg F \in S$ *o* $\neg G \in S$.
8. *Si* $\neg(F \vee G) \in S$, *entonces* $\neg F \in S$ *y* $\neg G \in S$.
9. *Si* $\neg(F \to G) \in S$, *entonces* $F \in S$ *y* $\neg G \in S$.

En Isabelle se formaliza mediante el tipo *definition* como sigue.

definition *Hintikka S* $\equiv$

$(\bot \notin S$
$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S))$

A continuación ilustramos con un ejemplo de conjunto de fórmulas de Hintikka.

notepad
begin

have *Hintikka* $\{Atom\ 0 \wedge ((\neg\ (Atom\ 1)) \rightarrow Atom\ 2),$
$((\neg\ (Atom\ 1)) \rightarrow Atom\ 2),$
$Atom\ 0, \neg(\neg\ (Atom\ 1)), Atom\ 1\}$
unfolding *Hintikka-def* **by** *simp*

end

En contraposición, el siguiente conjunto de fórmulas no es de Hintikka, pues no cumple la segunda condición de la definición anterior.

notepad
begin

have $\neg$ *Hintikka* $\{Atom\ 0 \wedge ((\neg\ (Atom\ 1)) \rightarrow Atom\ 2),$
$((\neg\ (Atom\ 1)) \rightarrow Atom\ 2),$
$Atom\ 0, \neg(\neg\ (Atom\ 1)), Atom\ 1, \neg(Atom\ 1)\}$
unfolding *Hintikka-def* **by** *auto*

end

En adelante presentaremos una serie de lemas auxiliares derivados de la definición de conjunto de Hintikka que nos facilitarán posteriormente las demostraciones en Isabelle/HOL.

El primer lema expresa que la conjunción de las nueve condiciones de la definición anterior es una condición necesaria para que un conjunto sea de Hintikka.

lemma *auxEq*:
assumes *Hintikka S*

```
 shows ⊥ ∉ S
 ∧ (∀k. Atom k ∈ S ⟶ ¬ (Atom k) ∈ S ⟶ False)
 ∧ (∀F G. F ∧ G ∈ S ⟶ F ∈ S ∧ G ∈ S)
 ∧ (∀F G. F ∨ G ∈ S ⟶ F ∈ S ∨ G ∈ S)
 ∧ (∀F G. F → G ∈ S ⟶ ¬F ∈ S ∨ G ∈ S)
 ∧ (∀F. ¬ (¬F) ∈ S ⟶ F ∈ S)
 ∧ (∀F G. ¬(F ∧ G) ∈ S ⟶ ¬ F ∈ S ∨ ¬ G ∈ S)
 ∧ (∀F G. ¬(F ∨ G) ∈ S ⟶ ¬ F ∈ S ∧ ¬ G ∈ S)
 ∧ (∀F G. ¬(F → G) ∈ S ⟶ F ∈ S ∧ ¬ G ∈ S)
proof −
 have Hintikka S = ( ⊥ ∉ S
 ∧ (∀k. Atom k ∈ S ⟶ ¬ (Atom k) ∈ S ⟶ False)
 ∧ (∀F G. F ∧ G ∈ S ⟶ F ∈ S ∧ G ∈ S)
 ∧ (∀F G. F ∨ G ∈ S ⟶ F ∈ S ∨ G ∈ S)
 ∧ (∀F G. F → G ∈ S ⟶ ¬F ∈ S ∨ G ∈ S)
 ∧ (∀F. ¬ (¬F) ∈ S ⟶ F ∈ S)
 ∧ (∀F G. ¬(F ∧ G) ∈ S ⟶ ¬ F ∈ S ∨ ¬ G ∈ S)
 ∧ (∀F G. ¬(F ∨ G) ∈ S ⟶ ¬ F ∈ S ∧ ¬ G ∈ S)
 ∧ (∀F G. ¬(F → G) ∈ S ⟶ F ∈ S ∧ ¬ G ∈ S))
  using Hintikka-def by (simp only: atomize-eq)
 then show ?thesis
  using assms by (rule iffD1)
qed
```

Asimismo presentaremos nueve lemas correspondientes a cada condición de la definición de conjunto de Hintikka.

Lema 3.1.2 *Si S es un conjunto de Hintikka, $\bot \notin S$.*

Lema 3.1.3 *Sea S un conjunto de Hintikka. Si p es una fórmula atómica tal que $p \in S$, entonces $\neg\, p \notin S$.*

Lema 3.1.4 *Sea S un conjunto de Hintikka. Si $F \wedge G \in S$, entonces $F \in S$ y $G \in S$.*

Lema 3.1.5 *Sea S un conjunto de Hintikka. Si $F \vee G \in S$, entonces $F \in S$ o $G \in S$.*

Lema 3.1.6 *Sea S un conjunto de Hintikka. Si $F \to G \in S$, entonces $\neg\, F \in S$ o $G \in S$.*

Lema 3.1.7 *Sea S un conjunto de Hintikka. Si $\neg(\neg\, F) \in S$, entonces $F \in S$.*

Lema 3.1.8 *Sea S un conjunto de Hintikka. Si $\neg(F \wedge G) \in S$, entonces $\neg\, F \in S$ o $\neg\, G \in S$.*

Lema 3.1.9 *Sea S un conjunto de Hintikka. Si* $\neg(F \vee G) \in S$*, entonces* $\neg\, F \in S$ *y* $\neg\, G \in S$*.*

Lema 3.1.10 *Sea S un conjunto de Hintikka. Si* $\neg(F \to G) \in S$*, entonces* $F \in S$ *y* $\neg\, G \in S$*.*

Como se puede observar, los lemas anteriores se corresponden con cada condición necesaria de la definición de conjunto de Hintikka. De este modo, la prueba de estos resultados se obtiene directamente de dicha definición al suponer que S es un conjunto de Hintikka.

Su formalización y demostración en Isabelle/HOL son las siguientes.

```
lemma
 assumes Hintikka S
 shows ⊥ ∉ S
proof −
 have ⊥ ∉ S
  ∧ (∀k. Atom k ∈ S ⟶ ¬ (Atom k) ∈ S ⟶ False)
  ∧ (∀F G. F ∧ G ∈ S ⟶ F ∈ S ∧ G ∈ S)
  ∧ (∀F G. F ∨ G ∈ S ⟶ F ∈ S ∨ G ∈ S)
  ∧ (∀F G. F → G ∈ S ⟶ ¬F ∈ S ∨ G ∈ S)
  ∧ (∀F. ¬ (¬F) ∈ S ⟶ F ∈ S)
  ∧ (∀F G. ¬(F ∧ G) ∈ S ⟶ ¬ F ∈ S ∨ ¬ G ∈ S)
  ∧ (∀F G. ¬(F ∨ G) ∈ S ⟶ ¬ F ∈ S ∧ ¬ G ∈ S)
  ∧ (∀F G. ¬(F → G) ∈ S ⟶ F ∈ S ∧ ¬ G ∈ S)
   using assms by (rule auxEq)
 thus ⊥ ∉ S by (rule conjunct1)
qed

lemma Hintikka-l1:
 Hintikka S ⟹ ⊥ ∉ S
 using Hintikka-def by blast

lemma
 assumes Hintikka S
 shows Atom k ∈ S ⟶ ¬ (Atom k) ∉ S
proof (rule impI)
 assume H:Atom k ∈ S
 have ⊥ ∉ S
  ∧ (∀k. Atom k ∈ S ⟶ ¬ (Atom k) ∈ S ⟶ False)
  ∧ (∀F G. F ∧ G ∈ S ⟶ F ∈ S ∧ G ∈ S)
  ∧ (∀F G. F ∨ G ∈ S ⟶ F ∈ S ∨ G ∈ S)
  ∧ (∀F G. F → G ∈ S ⟶ ¬F ∈ S ∨ G ∈ S)
```

$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $(\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \notin S$
by (*simp only*: *not-def*)
then have $Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \notin S$
by (*rule allE*)
thus $\neg\ (Atom\ k) \notin S$
using H **by** (*rule mp*)
qed

lemma *Hintikka-l2*:
$Hintikka\ S \Longrightarrow\ (Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \notin S)$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $F \wedge G \in S \longrightarrow F \in S \wedge G \in S$
proof (*rule impI*)
assume $F \wedge G \in S$
have $\bot \notin S$
$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $F \wedge G \in S \longrightarrow F \in S \wedge G \in S$
by (*iprover elim*: *allE*)
thus $F \in S \wedge G \in S$
using $\langle F \wedge G \in S \rangle$ **by** (*rule mp*)

qed

lemma *Hintikka-l3*:
Hintikka S $\Longrightarrow$ $(F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $F \vee G \in S \longrightarrow F \in S \vee G \in S$
proof (*rule impI*)
assume *H*:$F \vee G \in S$
have $\bot \notin S$
$\wedge$ $(\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge$ $(\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge$ $(\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge$ $(\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge$ $(\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge$ $(\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge$ $(\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge$ $(\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $F \vee G \in S \longrightarrow F \in S \vee G \in S$
by (*iprover elim*: *allE*)
thus $F \in S \vee G \in S$
using *H* **by** (*rule mp*)
qed

lemma *Hintikka-l4*:
Hintikka S $\Longrightarrow$ $(F \vee G \in S \longrightarrow F \in S \vee G \in S)$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S$
proof (*rule impI*)
assume *H*:$F \rightarrow G \in S$
have $\bot \notin S$
$\wedge$ $(\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$

$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S$
by (*iprover elim*: *allE*)
thus $\neg F \in S \vee G \in S$
using H **by** (*rule mp*)
qed

lemma *Hintikka-l5*:
$Hintikka\ S \Longrightarrow\ \ (F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $(\neg\ (\neg F) \in S \longrightarrow F \in S)$
proof (*rule impI*)
assume H:$\neg\ (\neg F) \in S$
have $\bot \notin S$
$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $\neg\ (\neg F) \in S \longrightarrow F \in S$
by (*rule allE*)
thus $F \in S$

using H **by** (*rule mp*)
qed

lemma *Hintikka-l6*:
Hintikka S $\Longrightarrow \neg\ (\neg F) \in S \longrightarrow F \in S$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $\neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S$
proof (*rule impI*)
assume H:$\neg(F \wedge G) \in S$
have $\bot \notin S$
$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $\neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S$
by (*iprover elim*: *allE*)
thus $\neg\ F \in S \vee \neg\ G \in S$
using H **by** (*rule mp*)
qed

lemma *Hintikka-l7*:
Hintikka S $\Longrightarrow \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $\neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S$
proof (*rule impI*)
assume H:$\neg(F \vee G) \in S$
have $\bot \notin S$

$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S$
by (*iprover elim*: *conjunct2 conjunct1*)
then have $\neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S$
by (*iprover elim*: *allE*)
thus $\neg\ F \in S \wedge \neg\ G \in S$
using H **by** (*rule mp*)
qed

lemma *Hintikka-l8*:
$Hintikka\ S \Longrightarrow (\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
by (*smt Hintikka-def*)

lemma
assumes *Hintikka S*
shows $\neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S$
proof (*rule impI*)
assume H:$\neg(F \rightarrow G) \in S$
have $\bot \notin S$
$\wedge\ (\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge\ (\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$
$\wedge\ (\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S)$
$\wedge\ (\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$
$\wedge\ (\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S)$
$\wedge\ (\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S)$
using *assms* **by** (*rule auxEq*)
then have $\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S$
by (*iprover elim*: *conjunct2*)
then have $\neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S$
by (*iprover elim*: *allE*)

thus $F \in S \wedge \neg\, G \in S$
using H **by** (*rule mp*)
qed

lemma *Hintikka-l9*:
Hintikka $S \Longrightarrow \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\, G \in S$
by (*smt Hintikka-def*)

Las pruebas anteriores siguen un esquema similar en Isabelle. En primer lugar, mediante el lema *auxEq*, al suponer inicialmente que el conjunto es de Hintikka, se verifica la conjunción de las condiciones de su definición. Posteriormente se emplean distintos métodos para disgregar estas condiciones en los distintos lemas. Para este propósito se utiliza, en particular, la táctica de demostración (*iprover elim*: ‹*rules*›). Con esta táctica aplicamos reiteradamente una o varias reglas y reducimos pasos en la prueba de Isabelle/HOL. Para ello, nos hemos servido del método de demostración *elim* que permite aplicar repetidamente reglas de eliminación especificadas. En nuestro caso, hemos utilizado las reglas de eliminación de la conjunción y la regla de eliminación del cuantificador existencial. Por otro lado, *iprover* es uno de los métodos automáticos de demostración en Isabelle/HOL que depende del contexto y de las reglas o métodos específicamente declarados a continuación del mismo.

Finalmente, veamos un resultado derivado de las condiciones exigidas a los conjuntos de Hintikka.

Lema 3.1.11 *Dado un conjunto de Hintikka, una fórmula no pertenece al conjunto si su negación sí pertenece al mismo. Es decir, si S es un conjunto de Hintikka y* $\neg\, F \in S$*, entonces* $F \notin S$*.*

Antes de pasar a la demostración del resultado, cabe añadir que para su prueba utilizaremos la regla lógica *modus tollens*.

Lema 3.1.12 (Modus tollens) *Si P implica Q y Q es falso, entonces P es también falso.*

Teniendo esto en cuenta, la demostración del lema es la siguiente.

Demostración: Sea S un conjunto de Hintikka y F un fórmula. Hay que probar que si $\neg F \in S$, entonces $F \notin S$. La prueba se realiza por inducción sobre la estructura de las fórmulas proposicionales. Veamos los distintos casos.

Caso 1: $F = p$, fórmula atómica.

Supongamos que $\neg\, p \in S$. Si $p \in S$ por definición de conjunto de Hintikka, $\neg\, p \notin S$, en contra de la hipótesis.

Caso 2: $F = \bot$

Supongamos que $\neg \bot \in S$. Como S es un conjunto de Hintikka, por definición se tiene que $\bot \notin S$, como queríamos demostrar.

Caso 3: $F = \neg G$, y G verifica la hipótesis de inducción.

Es decir, G verifica *HI*: $\neg G \in S \Longrightarrow G \notin S$.

Probemos que $\neg F \in S \Longrightarrow F \notin S$.

En efecto,

$$\begin{array}{rrl} & \neg F \in S & \\ \Longrightarrow & \neg\neg G \in S & \\ \Longrightarrow & G \in S & (S \text{ es conjunto de Hintikka}) \\ \Longrightarrow & \neg G \notin S & (HI \text{ y contraposición}) \\ \Longrightarrow & F \notin S & \end{array}$$

Caso 4: $F = G \wedge H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifica *HI*: $\neg G \in S \Longrightarrow G \notin S$ y $\neg H \in S \Longrightarrow H \notin S$.

Probemos que $\neg F \in S \Longrightarrow F \notin S$.

En efecto,

$$\begin{array}{rrl} & \neg F \in S & \\ \Longrightarrow & \neg (G \wedge H) \in S & \\ \Longrightarrow & \neg G \in S \vee \neg H \in S & (S \text{ es conjunto de Hintikka}) \\ \Longrightarrow & G \notin S \vee H \notin S & (HI) \end{array}$$

Por otra parte, si $F \in S$ se tiene que $G \in S \wedge H \in S$, por ser S conjunto de Hintikka, lo que contradice lo anterior. Por tanto, $F \notin S$.

Caso 5: $F = G \vee H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifica *HI*: $\neg G \in S \Longrightarrow G \notin S$ y $\neg H \in S \Longrightarrow H \notin S$.

Probemos que $\neg F \in S \Longrightarrow F \notin S$.

En efecto,

$$\begin{array}{rrl} & \neg F \in S & \\ \Longrightarrow & \neg (G \vee H) \in S & \\ \Longrightarrow & \neg G \in S \wedge \neg H \in S & (S \text{ es conjunto de Hintikka}) \\ \Longrightarrow & G \notin S \wedge H \notin S & (HI) \end{array}$$

Por otra parte, si $F \in S$ se tiene que $G \in S \vee H \in S$, por ser S conjunto de Hintikka, en contra de lo obtenido anteriormente. Por tanto, $F \notin S$.

Caso 6: $F = G \longrightarrow H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifica *HI*: $\neg\, G \in S \Longrightarrow G \notin S$ y $\neg\, H \in S \Longrightarrow H \notin S$.

Probemos que $\neg\, F \in S \Longrightarrow F \notin S$.

En efecto,

$$\begin{array}{rll} & \neg\, F \in S & \\ \Longrightarrow & \neg\, (G \longrightarrow H) \in S & \\ \Longrightarrow & G \in S \wedge \neg\, H \in S & (S \textit{ es conjunto de Hintikka}) \\ \Longrightarrow & \neg\, G \notin S \wedge \neg\, H \in S & (HI \textit{ y contraposición}) \\ \Longrightarrow & \neg\, G \notin S \wedge H \notin S & (HI) \end{array}$$

Por otra parte, si $F \in S$ se tiene que $\neg\, G \in S \vee H \in S$, por ser S conjunto de Hintikka, lo que contradice lo anterior. Por lo tanto, $F \notin S$.

Con lo que termina la demostración.

□

Por otra parte, su enunciado se formaliza en Isabelle de la siguiente forma.

lemma *Hintikka S* $\Longrightarrow$ $(\neg\, F \in S \longrightarrow F \notin S)$
oops

Antes de proceder con las distintas pruebas en Isabelle/HOL, vamos a formalizar la regla *modus tollens* usada en las demostraciones. Esta regla no está definida en Isabelle, de modo que se introducirá a continuación como lema auxiliar. Además, mostraremos su prueba detallada.

lemma *mt*:
assumes $F \longrightarrow G$
$\neg\, G$
shows $\neg\, F$
proof −
have $\neg\, G \longrightarrow \neg\, F$
using *assms*(1) **by** (*rule not-mono*)
thus $\neg\, F$
using *assms*(2) **by** (*rule mp*)
qed

Procedamos con la demostración del lema en Isabelle/HOL de manera detallada. Como es habitual para facilitar dicha prueba, se hará cada caso de la estructura de fórmulas por separado.

lemma *Hintikka-l10-atom*:

```
 assumes Hintikka S
 shows ¬ (Atom x) ∈ S ⟶ Atom x ∉ S
proof (rule impI)
 assume ¬ (Atom x) ∈ S
 then have ¬ (¬ (Atom x) ∉ S)
  by (rule contrapos-pn)
 have Atom x ∈ S ⟶ ¬ (Atom x) ∉ S
  using assms by (rule Hintikka-l2)
 thus Atom x ∉ S
  using ‹¬(¬ (Atom x) ∉ S)› by (rule mt)
qed

lemma Hintikka-l10-bot:
 assumes Hintikka S
 shows ¬ ⊥ ∈ S ⟶ ⊥ ∉ S
proof (rule impI)
 assume ¬ ⊥ ∈ S
 show ⊥ ∉ S
  using assms by (rule Hintikka-l1)
qed

lemma Hintikka-l10-not:
 assumes Hintikka S ⟹ ¬ F ∈ S ⟶ F ∉ S
      Hintikka S
     shows ¬ (¬ F) ∈ S ⟶ ¬ F ∉ S
proof (rule impI)
 assume ¬ (¬ F) ∈ S
 have ¬ (¬ F) ∈ S ⟶ F ∈ S
  using assms(2) by (rule Hintikka-l6)
 then have F ∈ S
  using ‹¬ (¬ F) ∈ S› by (rule mp)
 then have ¬ (F ∉ S)
  by (rule contrapos-pn)
 have ¬ F ∈ S ⟶ F ∉ S
  using assms by this
 thus ¬ F ∉ S
  using ‹¬ (F ∉ S)› by (rule mt)
qed
```

Para facilitar las pruebas de los casos correspondientes a las conectivas binarias emplearemos los siguientes lemas auxiliares. Estos nos permitirán, a partir de la negación

de un predicado, introducir la negación de una conjunción o disyunción de dicho predicado con otro cualquiera.

lemma *notConj1*:
assumes $\neg P$
shows $\neg (P \wedge Q)$
proof (*rule notI*)
assume $P \wedge Q$
then have P
by (*rule conjunct1*)
show *False*
using *assms* ⟨P⟩ **by** (*rule notE*)
qed

lemma *notConj2*:
assumes $\neg Q$
shows $\neg (P \wedge Q)$
proof (*rule notI*)
assume $P \wedge Q$
then have Q
by (*rule conjunct2*)
show *False*
using *assms* ⟨Q⟩ **by** (*rule notE*)
qed

lemma *notDisj*:
assumes $\neg P$
$\neg Q$
shows $\neg (P \vee Q)$
proof (*rule notI*)
assume $P \vee Q$
then show *False*
proof (*rule disjE*)
assume P
show *False*
using *assms*(1) ⟨P⟩ **by** (*rule notE*)
next
assume Q
show *False*
using *assms*(2) ⟨Q⟩ **by** (*rule notE*)
qed

qed

Comencemos las pruebas detalladas de los casos correspondientes a las conectivas binarias. Una vez terminadas, se mostrará la prueba detallada del lema completo.

lemma *Hintikka-l10-and*:
assumes *Hintikka* $S \Longrightarrow \neg\, G \in S \longrightarrow G \notin S$
Hintikka $S \Longrightarrow \neg\, H \in S \longrightarrow H \notin S$
Hintikka S
shows $\neg\, (G \wedge H) \in S \longrightarrow G \wedge H \notin S$
proof (*rule impI*)
assume $\neg\, (G \wedge H) \in S$
have $\neg\, (G \wedge H) \in S \longrightarrow \neg\, G \in S \vee \neg\, H \in S$
using *assms*(3) **by** (*rule Hintikka-l7*)
then have $\neg\, G \in S \vee \neg\, H \in S$
using ⟨$\neg\, (G \wedge H) \in S$⟩ **by** (*rule mp*)
then show $G \wedge H \notin S$
proof (*rule disjE*)
assume $\neg\, G \in S$
have $\neg\, G \in S \longrightarrow G \notin S$
using *assms*(1) *assms*(3) **by** *this*
then have $G \notin S$
using ⟨$\neg\, G \in S$⟩ **by** (*rule mp*)
then have $\neg\, (G \in S \wedge H \in S)$
by (*rule notConj1*)
have $G \wedge H \in S \longrightarrow G \in S \wedge H \in S$
using *assms*(3) **by** (*rule Hintikka-l3*)
thus $G \wedge H \notin S$
using ⟨$\neg\, (G \in S \wedge H \in S)$⟩ **by** (*rule mt*)
next
assume $\neg\, H \in S$
have $\neg\, H \in S \longrightarrow H \notin S$
using *assms*(2) *assms*(3) **by** *this*
then have $H \notin S$
using ⟨$\neg\, H \in S$⟩ **by** (*rule mp*)
then have $\neg\, (G \in S \wedge H \in S)$
by (*rule notConj2*)
have $G \wedge H \in S \longrightarrow G \in S \wedge H \in S$
using *assms*(3) **by** (*rule Hintikka-l3*)
thus $G \wedge H \notin S$
using ⟨$\neg\, (G \in S \wedge H \in S)$⟩ **by** (*rule mt*)
qed

qed

lemma *Hintikka-l10-or*:
assumes *Hintikka S* $\Longrightarrow \neg G \in S \longrightarrow G \notin S$
Hintikka S $\Longrightarrow \neg H \in S \longrightarrow H \notin S$
Hintikka S
shows $\neg (G \vee H) \in S \longrightarrow G \vee H \notin S$
proof (*rule impI*)
assume $\neg (G \vee H) \in S$
have $\neg (G \vee H) \in S \longrightarrow (\neg G \in S \wedge \neg H \in S)$
using *assms*(3) **by** (*rule Hintikka-l8*)
then have *C*:$\neg G \in S \wedge \neg H \in S$
using ⟨$\neg (G \vee H) \in S$⟩ **by** (*rule mp*)
then have $\neg G \in S$
by (*rule conjunct1*)
have $\neg G \in S \longrightarrow G \notin S$
using *assms*(1) *assms*(3) **by** *this*
then have $G \notin S$
using ⟨$\neg G \in S$⟩ **by** (*rule mp*)
have $\neg H \in S$
using *C* **by** (*rule conjunct2*)
have $\neg H \in S \longrightarrow H \notin S$
using *assms*(2) *assms*(3) **by** *this*
then have $H \notin S$
using ⟨$\neg H \in S$⟩ **by** (*rule mp*)
have $\neg (G \in S \vee H \in S)$
using ⟨$G \notin S$⟩ ⟨$H \notin S$⟩ **by** (*rule notDisj*)
have $(G \vee H \in S \longrightarrow G \in S \vee H \in S)$
using *assms*(3) **by** (*rule Hintikka-l4*)
thus $G \vee H \notin S$
using ⟨$\neg (G \in S \vee H \in S)$⟩ **by** (*rule mt*)
qed

lemma *Hintikka-l10-imp*:
assumes *Hintikka S* $\Longrightarrow \neg G \in S \longrightarrow G \notin S$
Hintikka S $\Longrightarrow \neg H \in S \longrightarrow H \notin S$
Hintikka S
shows $\neg (G \rightarrow H) \in S \longrightarrow G \rightarrow H \notin S$
proof (*rule impI*)
assume $\neg (G \rightarrow H) \in S$

```
 have ¬ (G ⟶ H) ∈ S ⟶ (G ∈ S ∧ ¬ H ∈ S)
  using assms(3) by (rule Hintikka-l9)
 then have C:G ∈ S ∧ ¬ H ∈ S
  using ⟨¬ (G ⟶ H) ∈ S⟩ by (rule mp)
 then have G ∈ S
  by (rule conjunct1)
 then have ¬ (G ∉ S)
  by (rule contrapos-pn)
 have ¬ G ∈ S ⟶ G ∉ S
  using assms(1) assms(3) by this
 then have ¬ G ∉ S
  using ⟨¬ (G ∉ S)⟩ by (rule mt)
 have ¬ H ∈ S
  using C by (rule conjunct2)
 have ¬ H ∈ S ⟶ H ∉ S
  using assms(2) assms(3) by this
 then have H ∉ S
  using ⟨¬ H ∈ S⟩ by (rule mp)
 have ¬ (¬ G ∈ S ∨ H ∈ S)
  using ⟨¬ G ∉ S⟩ ⟨H ∉ S⟩ by (rule notDisj)
 have G ⟶ H ∈ S ⟶ ¬ G ∈ S ∨ H ∈ S
  using assms(3) by (rule Hintikka-l5)
 thus G ⟶ H ∉ S
  using ⟨¬ (¬ G ∈ S ∨ H ∈ S)⟩ by (rule mt)
qed

lemma Hintikka S ⟹ ¬ F ∈ S ⟶ F ∉ S
proof (induct F)
case (Atom x)
 then show ?case by (rule Hintikka-l10-atom)
next
 case Bot
 then show ?case by (rule Hintikka-l10-bot)
next
 case (Not F)
 then show ?case by (rule Hintikka-l10-not)
next
 case (And F1 F2)
 then show ?case by (rule Hintikka-l10-and)
next
```

case (*Or F1 F2*)
then show *?case* **by** (*rule Hintikka-l10-or*)
next
case (*Imp F1 F2*)
then show *?case* **by** (*rule Hintikka-l10-imp*)
qed

Por último, su demostración automática es la que sigue.

lemma *Hintikka-l10*:
Hintikka S $\Longrightarrow \neg F \in S \longrightarrow F \notin S$
apply (*induct F*)
apply (*meson Hintikka-l2*)
apply (*simp add*: *Hintikka-l1*)
using *Hintikka-l6* **apply** *blast*
using *Hintikka-l3 Hintikka-l7* **apply** *blast*
apply (*smt Hintikka-def*)
using *Hintikka-l5 Hintikka-l9* **by** *blast*

3.2 Lema de Hintikka

Una vez definida la noción de conjunto de Hintikka y conocidas las propiedades que se deducen de ella, nuestro objetivo será demostrar que todo conjunto de Hintikka es satisfacible. Por definición, para probar que un conjunto es satisfacible basta hallar una interpretación que sea modelo suyo. De este modo, definimos el siguiente tipo de interpretaciones.

Definición 3.2.1 *Sea un conjunto de fórmulas cualquiera. Se define la interpretación asociada al conjunto como aquella que devuelve Verdadero sobre las variables proposicionales cuya correspondiente fórmula atómica pertence al conjunto, y Falso en caso contrario.*

En Isabelle se formalizará mediante el tipo *definition* como se expone a continuación.

definition *setValuation* ::
(*'a formula*) *set* $\Rightarrow$ *'a valuation* **where**
setValuation S $\equiv \lambda k.\ Atom\ k \in S$

Presentemos ahora ejemplos del valor de ciertas fórmulas en la interpretación asociada a los conjuntos siguientes.

notepad

begin

have $(setValuation\ \{Atom\ 0 \wedge ((\neg\ (Atom\ 1)) \rightarrow Atom\ 2),$
$((\neg\ (Atom\ 1)) \rightarrow Atom\ 2), Atom\ 0,$
$\neg(\neg\ (Atom\ 1)), Atom\ 1\}) \models Atom\ 1 \rightarrow Atom\ 0 = True$
unfolding *setValuation-def* **by** *simp*

have $(setValuation\ \{Atom\ 3 \vee (\neg\ (Atom\ 1)),$
$\neg\ (\neg\ (Atom\ 6))\}) \models Atom\ 2 \vee Atom\ 6 = False$
unfolding *setValuation-def* **by** *simp*

end

Previamente a probar que los conjuntos de Hintikka son satisfacibles y con el fin de facilitar dicha demostración, introducimos el siguiente resultado.

Lema 3.2.2 *La interpretación asociada a un conjunto de Hintikka es modelo de una fórmula si esta pertenece al conjunto. Además, dicha interpretación no es modelo de una fórmula si su negación pertenece al conjunto.*

Su formalización en Isabelle es la siguiente.

lemma $Hintikka\ S \Longrightarrow (F \in S \longrightarrow isModel\ (setValuation\ S)\ F)$
$\wedge\ (\neg\ F \in S \longrightarrow (\neg\ (isModel\ (setValuation\ S)\ F)))$
oops

Procedamos a la demostración del resultado.

Demostración: Sea S un conjunto de Hintikka y denotemos por $\mathcal{I}_S$ a la interpretación asociada a S. Sea F una fórmula, hay que probar lo siguiente:

$(F \in S \Longrightarrow \mathcal{I}_S \models F) \wedge (\neg\ F \in S \Longrightarrow\ \mathcal{I}_S \not\models F)$

La prueba se realiza por inducción sobre la estructura de las fórmulas proposicionales. Veamos los distintos casos.

Caso 1: $F = p$, fórmula atómica.

Si $p \in S$, entonces $\mathcal{I}_S(p) = True$ por definición de la interpretación asociada a S.

Por otro lado, si $\neg\ p \in S$, entonces $p \notin S$ por ser S de Hintikka. Por tanto, $\mathcal{I}_S(p) = False$ por definición de $\mathcal{I}_S$.

Caso 2: $F = \bot$

Si $\bot \in S$, como por definición de conjunto de Hintikka sabemos que $\bot \notin S$, se tendría una contradicción. Luego, en particular, tenemos el resultado.

Por otra parte, si $\neg \bot \in S$, $\mathcal{I}_S$ no es modelo de $\bot$ pues el valor de $\bot$ es *Falso* en cualquier interpretación.

Caso 3: $F = \neg G$, y G verifica la hipótesis de inducción.

Es decir, *HI*: $(G \in S \Longrightarrow \mathcal{I}_S \models G) \wedge (\neg G \in S \Longrightarrow \mathcal{I}_S \not\models G)$

Probemos que $(F \in S \Longrightarrow \mathcal{I}_S \models F) \wedge (\neg F \in S \Longrightarrow \mathcal{I}_S \not\models F)$

En efecto,

$$\begin{array}{rrl} & F \in S & \\ \Longrightarrow & \neg G \in S & \\ \Longrightarrow & \mathcal{I}_S(G) = \mathit{False} & (\mathit{HI}) \\ \Longrightarrow & \mathcal{I}_S(\neg G) = \mathit{True} & \\ \Longrightarrow & \mathcal{I}_S \models F & \end{array}$$

Análogamente,

$$\begin{array}{rrl} & \neg F \in S & \\ \Longrightarrow & \neg\neg G \in S & \\ \Longrightarrow & G \in S & (\mathit{S\ es\ conjunto\ de\ Hintikka}) \\ \Longrightarrow & \mathcal{I}_S(G) = \mathit{True} & (\mathit{HI}) \\ \Longrightarrow & \mathcal{I}_S(\neg G) = \mathit{False} & \\ \Longrightarrow & \mathcal{I}_S(F) = \mathit{False} & \\ \Longrightarrow & \mathcal{I}_S \not\models F & \end{array}$$

Caso 4: $F = G \wedge H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifican

HI1: $(G \in S \Longrightarrow \mathcal{I}_S \models G) \wedge (\neg G \in S \Longrightarrow \mathcal{I}_S \not\models G)$

HI2: $(H \in S \Longrightarrow \mathcal{I}_S \models H) \wedge (\neg H \in S \Longrightarrow \mathcal{I}_S \not\models H)$

Probemos que $(F \in S \Longrightarrow \mathcal{I}_S \models F) \wedge (\neg F \in S \Longrightarrow \mathcal{I}_S \not\models F)$

En efecto,

$$\begin{array}{rrl} & F \in S & \\ \Longrightarrow & G \wedge H \in S & \\ \Longrightarrow & G \in S \wedge H \in S & (\mathit{S\ es\ conjunto\ de\ Hintikka}) \\ \Longrightarrow & \mathcal{I}_S \models G \wedge \mathcal{I}_S \models H & (\mathit{HI1\ y\ HI2}) \\ \Longrightarrow & \mathcal{I}_S(G) \wedge \mathcal{I}_S(H) = \mathit{True} & \\ \Longrightarrow & \mathcal{I}_S(G \wedge H) = \mathit{True} & \\ \Longrightarrow & \mathcal{I}_S(F) = \mathit{True} & \\ \Longrightarrow & \mathcal{I}_S \models F & \end{array}$$

Por otra parte,

$$\begin{array}{rr} & \neg F \in S \\ \Longrightarrow & \neg (G \wedge H) \in S \\ \Longrightarrow & \neg G \in S \vee \neg H \in S \quad (S \textit{ es conjunto de Hintikka}) \\ \Longrightarrow & \mathcal{I}_S \not\models G \vee \mathcal{I}_S \not\models H \quad (\textit{HI1 y HI2}) \\ \Longrightarrow & \neg (\mathcal{I}_S \models G \wedge \mathcal{I}_S \models H) \\ \Longrightarrow & \mathcal{I}_S(G) \wedge \mathcal{I}_S(H) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S(G \wedge H) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S \not\models F \end{array}$$

Caso 5: $F = G \vee H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifican

HI1: $(G \in S \Longrightarrow \mathcal{I}_S \models G) \wedge (\neg G \in S \Longrightarrow \mathcal{I}_S \not\models G)$

HI2: $(H \in S \Longrightarrow \mathcal{I}_S \models H) \wedge (\neg H \in S \Longrightarrow \mathcal{I}_S \not\models H)$

Probemos que $(F \in S \Longrightarrow \mathcal{I}_S \models F) \wedge (\neg F \in S \Longrightarrow \mathcal{I}_S \not\models F)$

En efecto,

$$\begin{array}{rr} & F \in S \\ \Longrightarrow & G \vee H \in S \\ \Longrightarrow & G \in S \vee H \in S \quad (S \textit{ es conjunto de Hintikka}) \\ \Longrightarrow & \mathcal{I}_S \models G \vee \mathcal{I}_S \models H \quad (\textit{HI1 y HI2}) \\ \Longrightarrow & \mathcal{I}_S(G) \vee \mathcal{I}_S(H) = \textit{True} \\ \Longrightarrow & \mathcal{I}_S(G \vee H) = \textit{True} \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{True} \\ \Longrightarrow & \mathcal{I}_S \models F \end{array}$$

Por otra parte,

$$\begin{array}{rr} & \neg F \in S \\ \Longrightarrow & \neg (G \vee H) \in S \\ \Longrightarrow & \neg G \in S \wedge \neg H \in S \quad (S \textit{ es conjunto de Hintikka}) \\ \Longrightarrow & \mathcal{I}_S \not\models G \wedge \mathcal{I}_S \not\models H \quad (\textit{HI1 y HI2}) \\ \Longrightarrow & \neg (\mathcal{I}_S \models G \vee \mathcal{I}_S \models H) \\ \Longrightarrow & \mathcal{I}_S(G) \vee \mathcal{I}_S(H) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S(G \vee H) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{False} \\ \Longrightarrow & \mathcal{I}_S \not\models F \end{array}$$

Caso 6: $F = G \longrightarrow H$ y tanto G como H verifican la hipótesis de inducción.

Es decir, se verifican

HI1: $(G \in S \Longrightarrow \mathcal{I}_S \models G) \wedge (\neg\, G \in S \Longrightarrow\ \mathcal{I}_S \not\models G)$

HI2: $(H \in S \Longrightarrow \mathcal{I}_S \models H) \wedge (\neg\, H \in S \Longrightarrow\ \mathcal{I}_S \not\models H)$

Probemos que $(F \in S \Longrightarrow \mathcal{I}_S \models F) \wedge (\neg\, F \in S \Longrightarrow\ \mathcal{I}_S \not\models F)$

En efecto,

$$\begin{array}{rrl} & F \in S & \\ \Longrightarrow & G \longrightarrow H \in S & \\ \Longrightarrow & \neg\, G \in S \vee H \in S & (S \textit{ es conjunto de Hintikka}) \end{array}$$

Veamos que $\mathcal{I}_S \models F$ considerando ambos casos de la disyunción anterior.

$$\begin{array}{rrl} & \neg\, G \in S & \\ \Longrightarrow & \mathcal{I}_S \not\models G & (HI1) \\ \Longrightarrow & \mathcal{I}_S \models G \longrightarrow \mathcal{I}_S \models H & \\ \Longrightarrow & \mathcal{I}_S(G) \longrightarrow \mathcal{I}_S(H) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S(G \longrightarrow H) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S \models F & \end{array}$$

$$\begin{array}{rrl} & H \in S & \\ \Longrightarrow & \mathcal{I}_S \models H & (HI2) \\ \Longrightarrow & \mathcal{I}_S \models G \longrightarrow \mathcal{I}_S \models H & \\ \Longrightarrow & \mathcal{I}_S(G) \longrightarrow \mathcal{I}_S(H) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S(G \longrightarrow H) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{True} & \\ \Longrightarrow & \mathcal{I}_S \models F & \end{array}$$

Probando el resultado para este caso.

Por otra parte,

$$\begin{array}{rrl} & \neg\, F \in S & \\ \Longrightarrow & \neg\, (G \longrightarrow H) \in S & \\ \Longrightarrow & G \in S \wedge \neg\, H \in S & (S \textit{ es conjunto de Hintikka}) \\ \Longrightarrow & \mathcal{I}_S \models G \wedge \mathcal{I}_S \not\models H & (HI1 \textit{ y } HI2) \\ \Longrightarrow & \neg\, (\mathcal{I}_S \models G \longrightarrow \mathcal{I}_S \models H) & \\ \Longrightarrow & \mathcal{I}_S(G) \longrightarrow \mathcal{I}_S(H) = \textit{False} & \\ \Longrightarrow & \mathcal{I}_S(G \longrightarrow H) = \textit{False} & \\ \Longrightarrow & \mathcal{I}_S(F) = \textit{False} & \\ \Longrightarrow & \mathcal{I}_S \not\models F & \end{array}$$

Queda probada la segunda afirmación.

Con lo que termina la demostración.

□

Una vez terminada la prueba anterior, procedemos a las distintas demostraciones del lema mediante Isabelle/HOL. En primer lugar aparecerán las demostraciones detalladas de cada caso de la estructura de las fórmulas por separado. Posteriormente se mostrará la prueba detallada del lema completo.

```
lemma
 assumes Hintikka S
 shows ⋀x. (Atom x ∈ S ⟶ isModel (setValuation S) (Atom x)) ∧
      (¬ (Atom x) ∈ S ⟶ ¬ isModel (setValuation S) (Atom x))
proof (rule conjI)
 show ⋀x. Atom x ∈ S ⟶ isModel (setValuation S) (Atom x)
 proof
  fix x
  assume Atom x ∈ S
  hence (setValuation S) x
   by (simp only: setValuation-def)
  hence setValuation S ⊨ Atom x
   by (simp only: formula-semantics.simps(1))
  thus isModel (setValuation S) (Atom x)
   by (simp only: isModel-def)
 qed
next
 show
 ⋀x. ¬ (Atom x) ∈ S ⟶ ¬ isModel (setValuation S) (Atom x)
 proof
  fix x
  assume ¬ (Atom x) ∈ S
  have ¬ (Atom x) ∈ S ⟶ Atom x ∉ S
   using assms by (rule Hintikka-l10)
  then have Atom x ∉ S
   using ‹¬ (Atom x) ∈ S› by (rule mp)
  also have (¬ (Atom x ∈ S)) = (¬ (setValuation S) x)
   by (simp only: setValuation-def)
  also have ... = (¬ ((setValuation S) ⊨ (Atom x)))
   by (simp only: formula-semantics.simps(1))
  also have ... = (¬ isModel (setValuation S) (Atom x))
   by (simp only: isModel-def)
```

```
   finally show ¬ isModel (setValuation S) (Atom x)
     by this
  qed
qed

lemma Hl2-1:
  assumes Hintikka S
  shows ⋀x. (Atom x ∈ S ⟶ isModel (setValuation S) (Atom x)) ∧
        (¬ (Atom x) ∈ S ⟶ ¬ isModel (setValuation S) (Atom x))
  by (simp add: Hintikka-l10 assms isModel-def setValuation-def)

lemma
  assumes Hintikka S
  shows (⊥ ∈ S ⟶ isModel (setValuation S) ⊥)
        ∧ (¬ ⊥ ∈ S ⟶ (¬(isModel (setValuation S) ⊥)))
proof (rule conjI)
  show ⊥ ∈ S ⟶ isModel (setValuation S) ⊥
  proof (rule impI)
    assume ⊥ ∈ S
    have ⊥ ∉ S
      using assms by (rule Hintikka-l1)
    thus isModel (setValuation S) ⊥
      using ⟨⊥ ∈ S⟩ by (rule notE)
  qed
next
  show ¬ ⊥ ∈ S ⟶ ¬ isModel (setValuation S) ⊥
  proof (rule impI)
    assume ¬ ⊥ ∈ S
    have ¬ (setValuation S) ⊨ ⊥
    proof (rule notI)
     assume setValuation S ⊨ ⊥
      thus False
        by (simp only: formula-semantics.simps(2))
    qed
    also have (¬ (setValuation S) ⊨ ⊥) = (¬ isModel (setValuation S) ⊥)
      by (simp only: isModel-def)
    finally show ¬ isModel (setValuation S) ⊥
      by this
  qed
qed
```

lemma *Hl2-2*:
assumes *Hintikka S*
shows $(\bot \in S \longrightarrow isModel\ (setValuation\ S)\ \bot)$
$\wedge\ (\neg\ \bot \in S \longrightarrow (\neg\ (isModel\ (setValuation\ S)\ \bot)))$
by (*simp add*: *Hintikka-l1 assms isModel-def*)

lemma
assumes *Hintikka S*
$\bigwedge F.\ (F \in S \longrightarrow isModel\ (setValuation\ S)\ F) \wedge$
$(\neg\ F \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F)$
shows $\bigwedge F.\ (\neg\ F \in S \longrightarrow isModel\ (setValuation\ S)\ (\neg\ F)) \wedge$
$(\neg\ (\neg\ F) \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ (\neg\ F))$
proof (*rule conjI*)
show $\bigwedge F.\ \neg\ F \in S \longrightarrow isModel\ (setValuation\ S)\ (\neg\ F)$
proof
fix *F*
assume $\neg\ F \in S$
have $\neg\ F \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F$
using *assms*(2) **by** (*rule conjunct2*)
then have $\neg\ isModel\ (setValuation\ S)\ F$
using ‹$\neg\ F \in S$› **by** (*rule mp*)
also have $(\neg\ isModel\ (setValuation\ S)\ F) = (\neg\ (setValuation\ S) \models F)$
by (*simp only*: *isModel-def*)
also have $\ldots = setValuation\ S \models (\neg\ F)$
by (*simp only*: *formula-semantics.simps*(3))
also have $\ldots = isModel\ (setValuation\ S)\ (\neg\ F)$
by (*simp only*: *isModel-def*)
finally show $isModel\ (setValuation\ S)\ (\neg\ F)$
by *this*
qed
next
show $\bigwedge F.\ \neg\ (\neg\ F) \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ (\neg\ F)$
proof
fix *F*
assume $\neg\ (\neg\ F) \in S$
have $\neg\ (\neg\ F) \in S \longrightarrow F \in S$
using *assms*(1) **by** (*rule Hintikka-l6*)
then have $F \in S$
using ‹$\neg\ (\neg\ F) \in S$› **by** (*rule mp*)

```
  have F ∈ S ⟶ isModel (setValuation S) F
    using assms(2) by (rule conjunct1)
  then have isModel (setValuation S) F
    using ⟨F ∈ S⟩ by (rule mp)
  then have (¬ (¬ isModel (setValuation S) F))
    by (rule contrapos-pn)
  also have (¬ (¬ isModel (setValuation S) F)) =
      (¬ (¬ (setValuation S) ⊨ F))
    by (simp only: isModel-def)
  also have ... = (¬ (setValuation S) ⊨ (¬ F))
    by (simp only: formula-semantics.simps(3))
  also have ... = (¬ isModel (setValuation S) (¬ F))
    by (simp only: isModel-def)
  finally show ¬ isModel (setValuation S) (¬ F)
    by this
 qed
qed

lemma Hl2-3:
 assumes Hintikka S
 shows ⋀F. (F ∈ S ⟶ isModel (setValuation S) F) ∧
       (¬ F ∈ S ⟶ ¬ isModel (setValuation S) F) ⟹
       (¬ F ∈ S ⟶ isModel (setValuation S) (¬ F)) ∧
       (¬ (¬ F) ∈ S ⟶ ¬ isModel (setValuation S) (¬ F))
 using Hintikka-l6 assms isModel-def formula-semantics.simps(3) by blast

lemma
 assumes Hintikka S
       ⋀F1. (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
        (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1)
       ⋀F2. (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
        (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2)
 shows ⋀F1 F2. (F1 ∧ F2 ∈ S ⟶ isModel (setValuation S) (F1 ∧ F2)) ∧
    (¬ (F1 ∧ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∧ F2))
proof (rule conjI)
 show ⋀F1 F2. F1 ∧ F2 ∈ S ⟶ isModel (setValuation S) (F1 ∧ F2)
 proof
  fix F1 F2
  assume F1 ∧ F2 ∈ S
  have F1 ∧ F2 ∈ S ⟶ F1 ∈ S ∧ F2 ∈ S
```

```
    using assms(1) by (rule Hintikka-l3)
   then have C:F1 ∈ S ∧ F2 ∈ S
    using ⟨F1 ∧ F2 ∈ S⟩ by (rule mp)
   then have F1 ∈ S
    by (rule conjunct1)
   have F1 ∈ S ⟶ isModel (setValuation S) F1
    using assms(2) by (rule conjunct1)
   then have isModel (setValuation S) F1
    using ⟨F1 ∈ S⟩ by (rule mp)
   then have I1:(setValuation S) ⊨ F1
    by (simp only: isModel-def)
   have F2 ∈ S
    using C by (rule conjunct2)
   have F2 ∈ S ⟶ isModel (setValuation S) F2
    using assms(3) by (rule conjunct1)
   then have isModel (setValuation S) F2
    using ⟨F2 ∈ S⟩ by (rule mp)
   then have I2:(setValuation S) ⊨ F2
    by (simp only: isModel-def)
   have ((setValuation S) ⊨ F1) ∧ ((setValuation S) ⊨ F2)
    using I1 I2 by (rule conjI)
   then have (setValuation S) ⊨ (F1 ∧ F2)
    by (simp only: formula-semantics.simps(4))
   thus isModel (setValuation S) (F1 ∧ F2)
    by (simp only: isModel-def)
  qed
next
  show ⋀F1 F2. ¬ (F1 ∧ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∧ F2)
  proof
   fix F1 F2
   assume ¬ (F1 ∧ F2) ∈ S
   have ¬ (F1 ∧ F2) ∈ S ⟶ ¬ F1 ∈ S ∨ ¬ F2 ∈ S
    using assms(1) by (rule Hintikka-l7)
   then have ¬ F1 ∈ S ∨ ¬ F2 ∈ S
    using ⟨¬ (F1 ∧ F2) ∈ S⟩ by (rule mp)
   then show ¬ isModel (setValuation S) (F1 ∧ F2)
   proof (rule disjE)
    assume ¬ F1 ∈ S
    have ¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1
     using assms(2) by (rule conjunct2)
```

```
    then have ¬ isModel (setValuation S) F1
      using ⟨¬ F1 ∈ S⟩ by (rule mp)
    also have (¬ isModel (setValuation S) F1) = (¬ (setValuation S) ⊨ F1)
      by (simp only: isModel-def)
    finally have ¬ (setValuation S) ⊨ F1
      by this
    then have ¬ ((setValuation S) ⊨ F1 ∧ (setValuation S) ⊨ F2)
      by (rule notConj1)
    also have (¬ ((setValuation S) ⊨ F1 ∧ (setValuation S) ⊨ F2)) =
        (¬ ((setValuation S) ⊨ F1 ∧ F2))
      by (simp only: formula-semantics.simps(4))
    also have ... = (¬ isModel (setValuation S) (F1 ∧ F2))
      by (simp only: isModel-def)
    finally show (¬ isModel (setValuation S) (F1 ∧ F2))
      by this
  next
    assume ¬ F2 ∈ S
    have ¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2
      using assms(3) by (rule conjunct2)
    then have ¬ isModel (setValuation S) F2
      using ⟨¬ F2 ∈ S⟩ by (rule mp)
    also have (¬ isModel (setValuation S) F2) = (¬ (setValuation S) ⊨ F2)
      by (simp only: isModel-def)
    finally have ¬ (setValuation S) ⊨ F2
      by this
    then have ¬ ((setValuation S) ⊨ F1 ∧ (setValuation S) ⊨ F2)
      by (rule notConj2)
    also have (¬ ((setValuation S) ⊨ F1 ∧ (setValuation S) ⊨ F2)) =
        (¬ ((setValuation S) ⊨ (F1 ∧ F2)))
      by (simp only: formula-semantics.simps(4))
    also have ... = (¬ isModel (setValuation S) (F1 ∧ F2))
      by (simp only: isModel-def)
    finally show ¬ isModel (setValuation S) (F1 ∧ F2)
      by this
  qed
 qed
qed

lemma Hl2-4:
 assumes Hintikka S
```

shows $\bigwedge F1\ F2.$
$(F1 \in S \longrightarrow isModel\ (setValuation\ S)\ F1) \wedge$
$(\neg\ F1 \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F1) \Longrightarrow$
$(F2 \in S \longrightarrow isModel\ (setValuation\ S)\ F2) \wedge$
$(\neg\ F2 \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F2) \Longrightarrow$
$(F1 \wedge F2 \in S \longrightarrow isModel\ (setValuation\ S)\ (F1 \wedge F2)) \wedge$
$(\neg\ (F1 \wedge F2) \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ (F1 \wedge F2))$
by (*meson Hintikka-l3 Hintikka-l7 assms isModel-def formula-semantics.simps*(4))

lemma
assumes *Hintikka S*
$\bigwedge F1.\ (F1 \in S \longrightarrow isModel\ (setValuation\ S)\ F1) \wedge$
$(\neg\ F1 \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F1)$
$\bigwedge F2.\ (F2 \in S \longrightarrow isModel\ (setValuation\ S)\ F2) \wedge$
$(\neg\ F2 \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ F2)$
shows $\bigwedge F1\ F2.\ (F1 \vee F2 \in S \longrightarrow isModel\ (setValuation\ S)\ (F1 \vee F2))$
$\wedge\ (\neg\ (F1 \vee F2) \in S \longrightarrow \neg\ isModel\ (setValuation\ S)\ (F1 \vee F2))$
proof (*rule conjI*)
show $\bigwedge F1\ F2.\ F1 \vee F2 \in S \longrightarrow isModel\ (setValuation\ S)\ (F1 \vee F2)$
proof
fix *F1 F2*
assume $F1 \vee F2 \in S$
have $F1 \vee F2 \in S \longrightarrow F1 \in S \vee F2 \in S$
using *assms*(1) **by** (*rule Hintikka-l4*)
then have $F1 \in S \vee F2 \in S$
using ‹$F1 \vee F2 \in S$› **by** (*rule mp*)
then show $isModel\ (setValuation\ S)\ (F1 \vee F2)$
proof (*rule disjE*)
assume $F1 \in S$
have $F1 \in S \longrightarrow isModel\ (setValuation\ S)\ F1$
using *assms*(2) **by** (*rule conjunct1*)
then have $isModel\ (setValuation\ S)\ F1$
using ‹$F1 \in S$› **by** (*rule mp*)
then have $(setValuation\ S) \models F1$
by (*simp only*: *isModel-def*)
then have $(setValuation\ S) \models F1 \vee (setValuation\ S) \models F2$
by (*rule disjI1*)
then have $(setValuation\ S) \models (F1 \vee F2)$
by (*simp only*: *formula-semantics.simps*(5))
thus $isModel\ (setValuation\ S)\ (F1 \vee F2)$

```
      by (simp only: isModel-def)
    next
     assume F2 ∈ S
     have F2 ∈ S ⟶ isModel (setValuation S) F2
      using assms(3) by (rule conjunct1)
     then have isModel (setValuation S) F2
      using ‹F2 ∈ S› by (rule mp)
     then have (setValuation S) ⊨ F2
      by (simp only: isModel-def)
     then have (setValuation S) ⊨ F1 ∨ (setValuation S) ⊨ F2
      by (rule disjI2)
     then have (setValuation S) ⊨ (F1 ∨ F2)
      by (simp only: formula-semantics.simps(5))
     thus isModel (setValuation S) (F1 ∨ F2)
      by (simp only: isModel-def)
    qed
   qed
next
  show ⋀F1 F2. ¬ (F1 ∨ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∨ F2)
  proof (rule impI)
   fix F1 F2
   assume ¬ (F1 ∨ F2) ∈ S
   have ¬ (F1 ∨ F2) ∈ S ⟶ ¬ F1 ∈ S ∧ ¬ F2 ∈ S
    using assms(1) by (rule Hintikka-l8)
   then have C:¬ F1 ∈ S ∧ ¬ F2 ∈ S
    using ‹¬ (F1 ∨ F2) ∈ S› by (rule mp)
   then have ¬ F1 ∈ S
    by (rule conjunct1)
   have ¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1
    using assms(2) by (rule conjunct2)
   then have ¬ isModel (setValuation S) F1
    using ‹¬ F1 ∈ S› by (rule mp)
   also have (¬ isModel (setValuation S) F1) = (¬ (setValuation S) ⊨ F1)
    by (simp only: isModel-def)
   finally have D1:¬ (setValuation S) ⊨ F1
    by this
   have ¬ F2 ∈ S
    using C by (rule conjunct2)
   have ¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2
    using assms(3) by (rule conjunct2)
```

```
  then have ¬ isModel (setValuation S) F2
    using ⟨¬ F2 ∈ S⟩ by (rule mp)
  also have (¬ isModel (setValuation S) F2) = (¬ (setValuation S) ⊨ F2)
    by (simp only: isModel-def)
  finally have D2:¬ (setValuation S) ⊨ F2
    by this
  have ¬ ((setValuation S) ⊨ F1 ∨ (setValuation S) ⊨ F2)
    using D1 D2 by (rule notDisj)
  also have (¬ ((setValuation S) ⊨ F1 ∨ (setValuation S) ⊨ F2)) =
      (¬ (setValuation S) ⊨ (F1 ∨ F2))
    by (simp only: formula-semantics.simps(5))
  also have ... = (¬ isModel (setValuation S) (F1 ∨ F2))
    by (simp only: isModel-def)
  finally show ¬ isModel (setValuation S) (F1 ∨ F2)
    by this
 qed
qed


lemma Hl2-5:
 assumes Hintikka S
 shows ⋀F1 F2.
    (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
    (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1) ⟹
    (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
    (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2) ⟹
    (F1 ∨ F2 ∈ S ⟶ isModel (setValuation S) (F1 ∨ F2)) ∧
    (¬ (F1 ∨ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∨ F2))
 by (smt Hintikka-def assms isModel-def formula-semantics.simps(5))

lemma
 assumes Hintikka S
      ⋀F1. (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
       (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1)
      ⋀F2. (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
       (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2)
 shows ⋀F1 F2. (F1 → F2 ∈ S ⟶ isModel (setValuation S) (F1 → F2))
   ∧ (¬ (F1 → F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 → F2))
proof (rule conjI)
 show ⋀F1 F2. F1 → F2 ∈ S ⟶ isModel (setValuation S) (F1 → F2)
```

proof (*rule impI*)
fix *F1 F2*
assume $F1 \longrightarrow F2 \in S$
have $F1 \longrightarrow F2 \in S \longrightarrow \neg\, F1 \in S \vee F2 \in S$
using *assms*(*1*) **by** (*rule Hintikka-l5*)
then have $\neg\, F1 \in S \vee F2 \in S$
using ‹$F1 \longrightarrow F2 \in S$› **by** (*rule mp*)
then show *isModel* (*setValuation S*) ($F1 \longrightarrow F2$)
proof (*rule disjE*)
assume $\neg\, F1 \in S$
have $\neg\, F1 \in S \longrightarrow \neg$ *isModel* (*setValuation S*) *F1*
using *assms*(*2*) **by** (*rule conjunct2*)
then have $\neg$ *isModel* (*setValuation S*) *F1*
using ‹$\neg\, F1 \in S$› **by** (*rule mp*)
also have ($\neg$ *isModel* (*setValuation S*) *F1*) = ($\neg$ (*setValuation S*) $\models$ *F1*)
by (*simp only*: *isModel-def*)
finally have $\neg$ (*setValuation S*) $\models$ *F1*
by *this*
have (*setValuation S*) $\models$ *F1* $\longrightarrow$ (*setValuation S*) $\models$ *F2*
proof (*rule impI*)
assume (*setValuation S*) $\models$ *F1*
show (*setValuation S*) $\models$ *F2*
using ‹$\neg$ (*setValuation S*) $\models$ *F1*› ‹(*setValuation S*) $\models$ *F1*› **by** (*rule notE*)
qed
then have (*setValuation S*) $\models$ (*F1* $\longrightarrow$ *F2*)
by (*simp only*: *formula-semantics.simps*(*6*))
thus *?thesis*
by (*simp only*: *isModel-def*)
next
assume *F2* $\in$ *S*
have *F2* $\in$ *S* $\longrightarrow$ *isModel* (*setValuation S*) *F2*
using *assms*(*3*) **by** (*rule conjunct1*)
then have *isModel* (*setValuation S*) *F2*
using ‹*F2* $\in$ *S*› **by** (*rule mp*)
then have (*setValuation S*) $\models$ *F2*
by (*simp only*: *isModel-def*)
have (*setValuation S*) $\models$ *F1* $\longrightarrow$ (*setValuation S*) $\models$ *F2*
proof (*rule impI*)
assume (*setValuation S*) $\models$ *F1*
show (*setValuation S*) $\models$ *F2*

```
      using ⟨(setValuation S) ⊨ F2⟩ by this
    qed
    then have (setValuation S) ⊨ (F1 → F2)
     by (simp only: formula-semantics.simps(6))
    thus ?thesis
     by (simp only: isModel-def)
   qed
  qed
next
  show ⋀F1 F2. ¬ (F1 → F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 → F2)
  proof (rule impI)
   fix F1 F2
   assume ¬ (F1 → F2) ∈ S
   have ¬ (F1 → F2) ∈ S ⟶ F1 ∈ S ∧ ¬ F2 ∈ S
    using assms(1) by (rule Hintikka-l9)
   then have C:F1 ∈ S ∧ ¬ F2 ∈ S
    using ⟨¬ (F1 → F2) ∈ S⟩ by (rule mp)
   then have F1 ∈ S
    by (rule conjunct1)
   have F1 ∈ S ⟶ isModel (setValuation S) F1
    using assms(2) by (rule conjunct1)
   then have isModel (setValuation S) F1
    using ⟨F1 ∈ S⟩ by (rule mp)
   then have (setValuation S) ⊨ F1
    by (simp only: isModel-def)
   have ¬ F2 ∈ S
    using C by (rule conjunct2)
   have ¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2
    using assms(3) by (rule conjunct2)
   then have ¬ isModel (setValuation S) F2
    using ⟨¬ F2 ∈ S⟩ by (rule mp)
   also have (¬ isModel (setValuation S) F2) = (¬ (setValuation S) ⊨ F2)
    by (simp only: isModel-def)
   finally have ¬ (setValuation S) ⊨ F2
    by this
   have ¬ ((setValuation S) ⊨ F1 ⟶ (setValuation S) ⊨ F2)
   proof (rule notI)
    assume (setValuation S) ⊨ F1 ⟶ (setValuation S) ⊨ F2
    then have (setValuation S) ⊨ F2
     using ⟨(setValuation S) ⊨ F1⟩ by (rule mp)
```

```
    show False
      using ⟨¬ (setValuation S) ⊨ F2⟩ ⟨(setValuation S) ⊨ F2⟩ by (rule notE)
   qed
   also have (¬ ((setValuation S) ⊨ F1 ⟶ (setValuation S) ⊨ F2)) =
     (¬ (setValuation S) ⊨ (F1 → F2))
    by (simp only: formula-semantics.simps(6))
   also have ... = (¬ isModel (setValuation S) (F1 → F2))
    by (simp only: isModel-def)
   finally show ¬ isModel (setValuation S) (F1 → F2)
    by this
  qed
qed

lemma Hl2-6:
  assumes Hintikka S
  shows ⋀F1 F2.
      (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
      (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1) ⟹
      (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
      (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2) ⟹
      (F1 → F2 ∈ S ⟶ isModel (setValuation S) (F1 → F2)) ∧
      (¬ (F1 → F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 → F2))
  by (meson Hintikka-l5 Hintikka-l9 assms isModel-def formula-semantics.simps(6))

lemma Hintikkas-lemma-l2:
  assumes Hintikka S
  shows (F ∈ S ⟶ isModel (setValuation S) F)
        ∧ (¬ F ∈ S ⟶ (¬(isModel (setValuation S) F)))
proof (induct F)
  fix x
  show (Atom x ∈ S ⟶ isModel (setValuation S) (Atom x)) ∧
       (¬ (Atom x) ∈ S ⟶ ¬ isModel (setValuation S) (Atom x))
    using assms by (rule Hl2-1)
next
  show (⊥ ∈ S ⟶ isModel (setValuation S) ⊥) ∧
    (¬ ⊥ ∈ S ⟶ ¬ isModel (setValuation S) ⊥)
    using assms by (rule Hl2-2)
next
  fix F
  show (F ∈ S ⟶ isModel (setValuation S) F) ∧
```

```
      (¬ F ∈ S ⟶ ¬ isModel (setValuation S) F) ⟹
      (¬ F ∈ S ⟶ isModel (setValuation S) (¬ F)) ∧
      (¬ (¬ F) ∈ S ⟶ ¬ isModel (setValuation S) (¬ F))
   using assms by (rule Hl2-3)
next
 fix F1 F2
 show (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
    (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1) ⟹
    (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
    (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2) ⟹
    (F1 ∧ F2 ∈ S ⟶ isModel (setValuation S) (F1 ∧ F2)) ∧
    (¬ (F1 ∧ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∧ F2))
   using assms by (rule Hl2-4)
next
 fix F1 F2
 show (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
    (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1) ⟹
    (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
    (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2) ⟹
    (F1 ∨ F2 ∈ S ⟶ isModel (setValuation S) (F1 ∨ F2)) ∧
    (¬ (F1 ∨ F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 ∨ F2))
   using assms by (rule Hl2-5)
next
 fix F1 F2
 show (F1 ∈ S ⟶ isModel (setValuation S) F1) ∧
    (¬ F1 ∈ S ⟶ ¬ isModel (setValuation S) F1) ⟹
    (F2 ∈ S ⟶ isModel (setValuation S) F2) ∧
    (¬ F2 ∈ S ⟶ ¬ isModel (setValuation S) F2) ⟹
    (F1 → F2 ∈ S ⟶ isModel (setValuation S) (F1 → F2)) ∧
    (¬ (F1 → F2) ∈ S ⟶ ¬ isModel (setValuation S) (F1 → F2))
   using assms by (rule Hl2-6)
qed
```

Para concluir, demostremos el *Lema de Hintikka* empleando el resultado anterior.

Teorema 3.2.3 (Lema de Hintikka) *Todo conjunto de Hintikka es satisfacible.*

Demostración: Consideremos un conjunto de fórmulas S tal que sea un conjunto de Hintikka. Queremos demostrar que S es satisfacible, es decir, que tiene algún modelo. En otras palabras, debemos hallar una interpretación que sea modelo de S.

En primer lugar, probemos que la interpretación asociada a S es modelo de S. Por definición de modelo de un conjunto, basta comprobar que es modelo de toda fórmula perteneciente al mismo. Fijada una fórmula cualquiera, hemos visto anteriormente que la interpretación asociada a S es modelo de la fórmula si esta pertenece al conjunto. Por tanto, dicha interpretación es, en efecto, modelo de todas las fórmulas que pertenecen a S. Luego la interpretación asociada a S es modelo de S.

En conclusión, hemos hallado una interpretación que es modelo del conjunto. Por lo tanto, S es satisfacible, como se quería probar.

□

Por su parte, la prueba detallada en Isabelle emplea el lema auxiliar *Hintikka-model*. Con él se demuestra la primera parte del lema de Hintikka: dado un conjunto de Hintikka, la interpretación asociada al conjunto es modelo del mismo.

```
lemma Hintikka-model:
 assumes Hintikka S
 shows isModelSet (setValuation S) S
proof −
 have ∀ F. (F ∈ S ⟶ isModel (setValuation S) F)
 proof (rule allI)
  fix F
  have (F ∈ S ⟶ isModel (setValuation S) F)
       ∧ (¬ F ∈ S ⟶ (¬(isModel (setValuation S) F)))
   using assms by (rule Hintikkas-lemma-l2)
  thus F ∈ S ⟶ isModel (setValuation S) F
   by (rule conjunct1)
 qed
 thus isModelSet (setValuation S) S
  by (simp only: modelSet)
qed
```

Finalmente, las pruebas detallada y automática del *Lema de Hintikka* en Isabelle/HOL.

```
theorem
 assumes Hintikka S
 shows sat S
proof −
 have isModelSet (setValuation S) S
  using assms by (rule Hintikka-model)
 then have ∃ A. isModelSet A S
  by (simp only: exI)
 thus sat S
  by (simp only: satAlt)
```

qed

theorem *Hintikkaslemma*:
assumes *Hintikka S*
shows *sat S*
using *Hintikka-model assms satAlt* **by** *blast*

Apéndice A

Lemas de HOL usados

En este glosario se recoge la lista de los lemas y reglas usadas indicando la página del libro de HOL donde se encuentran.

A.1 La base de lógica de primer orden (2)

En Isabelle corresponde a la teoría HOL.thy

A.1.1 Lógica primitiva (2.1)

A.1.1.1 Conectivas y cuantificadores definidos (2.1.2)

- (p.34) $\neg\, P \equiv P \longrightarrow \mathit{False}$ (*not-def*)

A.1.1.2 Axiomas y definiciones básicas (2.1.4)

- (p.36) $\dfrac{\dfrac{P}{Q}}{P \longrightarrow Q}$ (*impI*)
- (p.36) $\dfrac{(P \longrightarrow Q) \wedge P}{Q}$ (*mp*)

A.1.2 Reglas fundamentales (2.2)

A.1.2.1 Reglas de congruencia para aplicaciones (2.2.2)

- (p.37) $\dfrac{x = y}{f\,x = f\,y}$ (*arg-cong*)
- (p.37) $\dfrac{a = b \wedge c = d}{f\,a\,c = f\,b\,d}$ (*arg-cong2*)

A.1.2.2 Igualdad de booleanos - *iff* (2.2.3)

- (p.38) $\dfrac{Q = P \wedge Q}{P}$ (*iffD1*)

A.1.2.3 Cuantificador universal I (2.2.5)

- (p.38) $\dfrac{\forall x.\ P\,x \qquad \dfrac{P\,x}{R}}{R}$ (*allE*)

A.1.2.4 Negación (2.2.7)

- (p.39) $\dfrac{\dfrac{P}{False}}{\neg\, P}$ (*notI*)
- (p.39) $\dfrac{\neg\, P \wedge P}{R}$ (*notE*)

A.1.2.5 Implicación (2.2.8)

- (p.40) $\dfrac{Q \qquad \dfrac{P}{\neg\, Q}}{\neg\, P}$ (*contrapos-pn*)

A.1.2.6 Disyunción I (2.2.9)

- (p.40) $\dfrac{P \vee Q \qquad \dfrac{P}{R} \qquad \dfrac{Q}{R}}{R}$ (*disjE*)

A.1.2.7 Derivación de *iffI* (2.2.10)

- (p.40) $\dfrac{\dfrac{P}{Q} \qquad \dfrac{Q}{P}}{P = Q}$ (*iffI*)

A.1.2.8 Cuantificador universal II (2.2.12)

- (p.41) $\dfrac{\bigwedge x.\ P\ x}{\forall\, x.\ P\ x}$ (*allI*)

A.1.2.9 Cuantificador existencia (2.2.13)

- (p.41) $\dfrac{P\ x}{\exists\, x.\ P\ x}$ (*exI*)

- (p.41) $\dfrac{\exists\, x.\ P\ x \qquad \bigwedge x.\ \dfrac{P\ x}{Q}}{Q}$ (*exE*)

A.1.2.10 Conjunción (2.2.14)

- (p.41) $\dfrac{P \wedge Q}{P \wedge Q}$ (*conjI*)

- (p.41) $\dfrac{P \wedge Q}{P}$ (*conjunct1*)

- (p.41) $\dfrac{P \wedge Q}{Q}$ (*conjunct2*)

A.1.2.11 Disyunción II (2.2.15)

- (p.42) $\dfrac{P}{P \vee Q}$ (*disjI1*)
- (p.42) $\dfrac{Q}{P \vee Q}$ (*disjI2*)

A.1.2.12 Atomización de conectivas de nivel intermedio (2.2.20)

- (p.46) $(x \equiv y) \equiv x = y$ (*atomize-eq*)

A.1.3 Configuración del paquete (2.3)

A.1.3.1 Simplificadores (2.3.4)

- (p.50) $(\neg\ False) = True$ (*not-False-eq-True*)
- (p.53) $(\nexists x.\ P\ x) = (\forall x.\ \neg\ P\ x)$ (*not-ex*)

A.2 Grupos, también combinados con órdenes (5)

Los siguientes resultados pertenecen a la teoría de grupos Groups.thy.

A.2.1 Estructuras abstractas

- (p.109) $sup\ bot\ a = a$ (*sup-bot.left-neutral*)

A.3 Retículos abstractos (6)

Los resultados expuestos a continuación pertenecen a la teoría de retículos Lattices.thy.

- (p.139) $(sup\ b\ c \leq a) = (b \leq a \wedge c \leq a)$ (*sup.bounded-iff*)

A.4 Teoría de conjuntos para lógica de orden superior (7)

Los siguientes resultados corresponden a la teoría de conjuntos Set.thy.

A.4.1 Subconjuntos y cuantificadores acotados (7.2)

- (p.163) $\dfrac{\bigwedge x.\ \dfrac{x \in A}{P\ x}}{\forall\, x{\in}A.\ P\ x}$ (*ballI*)
- (p.163) $\dfrac{(\forall\, x{\in}A.\ P\ x) \wedge x \in A}{P\ x}$ (*bspec*)

A.4.2 Operaciones básicas (7.3)

A.4.2.1 Subconjuntos (7.3.1)

- (p.165) $\dfrac{c \in A \wedge A \subseteq B}{c \in B}$ (*rev-subsetD*)
- (p.166) $A \subseteq A$ (*subset-refl*)
- (p.166) $\dfrac{A \subseteq B \wedge B \subseteq C}{A \subseteq C}$ (*subset-trans*)

A.4.2.2 El conjunto vacío (7.3.3)

- (p.167) $\emptyset \subseteq A$ (*empty-subsetI*)
- (p.167) $Ball\ \emptyset\ P = True$ (*ball-empty*)
- (p.167) $Bex\ \emptyset\ P = False$ (*bex-empty*)

A.4.2.3 Unión binaria (7.3.8)

- (p.169) $(c \in A \cup B) = (c \in A \vee c \in B)$ (*Un-iff*)
- (p.169) $\dfrac{c \in A}{c \in A \cup B}$ (*UnI1*)
- (p.170) $\dfrac{c \in B}{c \in A \cup B}$ (*UnI2*)

A.4.2.4 Aumentar un conjunto - insertar (7.3.10)

- (p.171) *List.insert x xs* $= \{x\} \cup xs$ (*set-insert*)

A.4.2.5 Conjuntos unitarios, insertar (7.3.11)

- (p.172) $a \in \{a\}$ (*singletonI*)
- (p.172) $\dfrac{b \in \{a\}}{b = a}$ (*singletonD*)
- (p.172) $(b \in \{a\}) = (b = a)$ (*singleton-iff*)

A.4.2.6 Imagen de un conjunto por una función (7.3.12)

- (p.173) $f\,`\,A = \{y \mid \exists\, x{\in}A.\ y = f\,x\}$ (*image-def*)
- (p.173) $f\,`\,(A \cup B) = f\,`\,A \cup f\,`\,B$ (*image-Un*)
- (p.174) $f\,`\,\emptyset = \emptyset$ (*image-empty*)
- (p.174) $f\,`\,(\{a\} \cup B) = \{f\,a\} \cup f\,`\,B$ (*image-insert*)

A.4.3 Más operaciones y lemas (7.4)

A.4.3.1 Reglas derivadas sobre subconjuntos (7.4.2)

- (p.177) $A \subseteq A \cup B$ (*Un-upper1*)
- (p.177) $B \subseteq A \cup B$ (*Un-upper2*)

A.4.3.2 Igualdades sobre la union, intersección, inclusion, etc. (7.4.3)

- (p.179) $\{a\} \cup A = \{a\} \cup A$ (*insert-is-Un*)
- (p.181) $A \cup A = A$ (*Un-absorb*)
- (p.181) $A \cup \emptyset = A$ (*Un-empty-right*)
- (p.182) $\{a\} \cup B \cup C = \{a\} \cup (B \cup C)$ (*Un-insert-left*)
- (p.187) $(\forall x{\in}A.\ P\ x \vee Q) = ((\forall x{\in}A.\ P\ x) \vee Q)$
  $(\forall x{\in}A.\ P \vee Q\ x) = (P \vee (\forall x{\in}A.\ Q\ x))$
  $(\forall x{\in}A.\ P \longrightarrow Q\ x) = (P \longrightarrow (\forall x{\in}A.\ Q\ x))$
  $(\forall x{\in}A.\ P\ x \longrightarrow Q) = ((\exists x{\in}A.\ P\ x) \longrightarrow Q)$
  $(\forall x{\in}\emptyset.\ P\ x) = \mathit{True}$
  $(\forall x{\in}\mathit{UNIV}.\ P\ x) = (\forall x.\ P\ x)$
  $(\forall x{\in}\{a\} \cup B.\ P\ x) = (P\ a \wedge (\forall x{\in}B.\ P\ x))$
  $(\forall x{\in}\mathit{Collect}\ Q.\ P\ x) = (\forall x.\ Q\ x \longrightarrow P\ x)$
  $(\forall x{\in}f\ `\ A.\ P\ x) = (\forall x{\in}A.\ P\ (f\ x))$
  $(\neg\ (\forall x{\in}A.\ P\ x)) = (\exists x{\in}A.\ \neg\ P\ x)$ (*ball-simps*)
- (p.187) $(\exists x{\in}A.\ P\ x \wedge Q) = ((\exists x{\in}A.\ P\ x) \wedge Q)$
  $(\exists x{\in}A.\ P \wedge Q\ x) = (P \wedge (\exists x{\in}A.\ Q\ x))$
  $(\exists x{\in}\emptyset.\ P\ x) = \mathit{False}$
  $(\exists x{\in}\mathit{UNIV}.\ P\ x) = (\exists x.\ P\ x)$
  $(\exists x{\in}\{a\} \cup B.\ P\ x) = (P\ a \vee (\exists x{\in}B.\ P\ x))$
  $(\exists x{\in}\mathit{Collect}\ Q.\ P\ x) = (\exists x.\ Q\ x \wedge P\ x)$
  $(\exists x{\in}f\ `\ A.\ P\ x) = (\exists x{\in}A.\ P\ (f\ x))$
  $(\neg\ (\exists x{\in}A.\ P\ x)) = (\forall x{\in}A.\ \neg\ P\ x)$ (*bex-simps*)

A.4.3.3 Monotonía de varias operaciones (7.4.4)

- (p.188) $\dfrac{A \subseteq C \wedge B \subseteq D}{A \cup B \subseteq C \cup D}$ (*Un-mono*)
- (p.188) $P \longrightarrow P$ (*imp-refl*)

- (p.188) $\dfrac{Q \longrightarrow P}{\neg P \longrightarrow \neg Q}$ (*not-mono*)

A.5 Nociones sobre funciones (9)

En Isabelle, la teoría de funciones se corresponde con Fun.thy.

A.5.1 Actualización de funciones (9.6)

- (p.212) $f(a := b) = (\lambda x.\ \mathsf{if}\ x = a\ \mathsf{then}\ b\ \mathsf{else}\ f\ x)$ (*fun-upd-def*)
- (p.213) $\dfrac{z \neq x}{(f(x := y))\ z = f\ z}$ (*fun-upd-other*)

A.6 Retículos completos (10)

En Isabelle corresponde a la teoría Complete-Lattices.thy.

A.6.1 Retículos completos en conjuntos (10.6)

A.6.1.1 Unión (10.6.3)

- (p.238) $\bigcup \emptyset = \emptyset$ (*Union-empty*)

A.7 Conjuntos finitos (18)

A continuación se muestran resultados relativos a la teoría Finite-Set.thy.

A.7.1 Predicado de conjuntos finitos (18.1)

- (p.419) *finite A* (*finite*)

A.7.2 Finitud y operaciones de conjuntos comunes (18.2)

- (p.422) $\dfrac{\mathit{finite}\ F \wedge \mathit{finite}\ G}{\mathit{finite}\ (F \cup G)}$ (*finite-UnI*)
- (p.423) $\mathit{finite}\ (\{a\} \cup A) = \mathit{finite}\ A$ (*finite-insert*)

A.8 Composición de functores naturales acotados (33)

En esta sección se muestran resultados pertenecientes a la teoría de composición de functores naturales acotados de Isabelle BNFComposition.thy.

- (p.718) $\bigcup\ (f\ `\ (\{a\} \cup B)) = f\ a \cup \bigcup\ (f\ `\ B)$ (*Union-image-insert*)

A.9 El tipo de datos de la listas finitas (66)

En esta sección se muestran resultados sobre listas finitas dentro de la teoría de listas de Isabelle List.thy.

- (p.1169) $[] = \emptyset$
  $x21 \cdot x22 = \{x21\} \cup x22$ (*list.set*)

A.9.1 Funciones básicas de procesamiento de listas (66.1)

A.9.1.1 Función *set*

- (p.1195) $xs\ @\ ys = xs \cup ys$ (*set-append*)

Bibliografía

[1] José A. Alonso. Temas de "Lógica matemática y fundamentos (2018–19)". Technical report, Univ. de Sevilla, 2019. En https://www.cs.us.es/~jalonso/cursos/lmf-18/temas.php.

[2] Lawrence C. Paulson Computer Laboratory. Old Introduction to Isabelle. Technical report, University of Cambridge, 2019. En https://isabelle.in.tum.de/website-Isabelle2019/dist/Isabelle2019/doc/intro.pdf.

[3] Christian Doczkal and Gert Smolka. Constructive Formalization of Classical Modal Logic. Technical report, 2011. En http://www.cs.ru.nl/~spitters/coqw_files/paper_1.pdf.

[4] M. Fitting. *First-order Logic and Automated Theorem Proving*. Graduate texts in computer science. Springer, 1996.

[5] L.T.F Gamut. *Introducción a la lógica*. Editorial Universitaria de Buenos Aires, 2002.

[6] John Harrison. An overview of automated reasoning. Technical report, 2014. En https://www.cl.cam.ac.uk/~jrh13/slides/lyon-03feb14/slides.pdf.

[7] Angeliki Koutsoukou-Argyraki. Formalising Mathematics -in praxis, 2019. En https://www.researchgate.net/publication/334549483_formalising_mathematis_-in_praxis_a_mathematician's_first_experiences_with_isabellehol_and_the_why_and_how_of_getting_started.

[8] Dr. Kevin P. Lee. A Guide to Writing Mathematics. En http://cc.kangwon.ac.kr/~kimoon/me/me-132/math-writing.pdf.

[9] F. Félix Lara Martín. Temas 3 de "Ciencias de la computación (2018–19)": Funciones recursivas. Technical report, Univ. de Sevilla, 2019. En http://www.cs.us.es/cursos/cc-2018/Tema-03.pdf.

[10] Julius Michaelis and Tobias Nipkow. Propositional Proof Systems. Technical report, 2020. En https://www.isa-afp.org/browser_info/current/AFP/Propositional_Proof_Systems/document.pdf.

[11] Tobias Nipkow. What's in Main, 2019. En https://isabelle.in.tum.de/website-Isabelle2019/dist/Isabelle2019/doc/main.pdf.

[12] Lawrence C. Paulson Tobias Nipkow and Markus Wenzel. *Isabelle/HOL: A proof assistant for Higher–Order Logic*. Lecture Notes in Computer Science, Vol. 2283, Springer–Verlag, 2019. En https://isabelle.in.tum.de/website-Isabelle2019/dist/Isabelle2019/doc/tutorial.pdf.

[13] Floris van Doorn. Propositional Calculus in Coq. Technical report, 2015. En https://arxiv.org/abs/1503.08744.

[14] Makarius Wenzel. The Isabelle/Isar Implementation, 2019. En https://isabelle.in.tum.de/website-Isabelle2019/dist/Isabelle2019/doc/implementation.pdf.

[15] Makarius Wenzel. The Isabelle/Isar Reference Manual, 2019. En https://isabelle.in.tum.de/website-Isabelle2019/dist/Isabelle2019/doc/isar-ref.pdf.