

Lógica proposicional en Isabelle/HOL

Sofía Santiago Fernández

actualizado el 11 de diciembre de 2019

Comentario 1: Falta la introducción.

Resumen

Añadir introducción

Índice

1	Sintaxis	1
1.1	Fórmulas	1
1.2	Subfórmulas	9
2	Glosario de reglas	27
2.1	Teoría de conjuntos finitos	27
2.2	Teoría de listas	27
2.3	Teoría de conjuntos	28
2.4	Lógica de primer orden	29
	Lista de tareas pendientes	29

1 Sintaxis

1.1 Fórmulas

Comentario 2: Explicar la siguiente notación y recolocarla donde se use por primera vez.

notation *insert* (- ▷ - [56,55] 55)

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

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

Comentario 3: Incluir el árbol de formación.

A continuación, veamos su representación en Isabelle

```
datatype (atoms: 'a) formula =
  Atom 'a
| Bot                                     ( $\perp$ )
| Not 'a formula                       ( $\neg$ )
| And 'a formula 'a formula           (infix  $\wedge$  68)
| Or 'a formula 'a formula            (infix  $\vee$  68)
| Imp 'a formula 'a formula           (infixr  $\rightarrow$  68)
```

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

Fórmulas básicas :

- $Atom :: 'a \Rightarrow 'a\ formula$
- $\perp :: 'a\ formula$

Fórmulas compuestas :

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

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.

Además se define simultáneamente la función $atoms :: 'a\ formula \Rightarrow 'a\ set$, que obtiene el conjunto de variables proposicionales de una fórmula.

Por otro lado, la definición de *formula* genera automáticamente los siguientes lemas sobre la función de conjuntos *atoms* en Isabelle.

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

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

```

notepad
begin
  fix p q r :: 'a

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

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

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

  have atoms ((Atom p  $\rightarrow$  Atom p)  $\vee$  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  $\perp$  =  $\emptyset$ 
    by (simp only: formula.set)

  have atoms (Atom p  $\vee$   $\perp$ ) = {p}
  proof -
    have atoms (Atom p  $\vee$   $\perp$ ) = atoms (Atom p)  $\cup$  atoms Bot
      by (simp only: formula.set(5))
    also have ... = {p}  $\cup$  atoms Bot
      by (simp only: formula.set(1))
    also have ... = {p}  $\cup$   $\emptyset$ 
      by (simp only: formula.set(2))
    also have ... = {p}
  end
end

```

```

    by (simp only: Un-empty-right)
  finally show atoms (Atom p  $\vee$   $\perp$ ) = {p}
    by this
qed

have atoms (Atom p  $\vee$   $\perp$ ) = {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:

Definición 1.2 *Sea φ una propiedad sobre fórmulas que verifica las siguientes condiciones:*

- *Las fórmulas atómicas la cumplen.*
- *La constante \perp 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 φ .

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.

Comentario 4: Poner bien cada regla.

$$\llbracket \bigwedge x. P (Atom\ x); P\ \perp; \bigwedge x. P\ x \implies P\ (\neg\ x); \bigwedge x1\ a\ x2. P\ x1\ a \wedge P\ x2 \implies P\ (x1\ a \wedge x2); \bigwedge x1\ a\ x2. P\ x1\ a \wedge P\ x2 \implies P\ (x1\ a \vee x2); \bigwedge x1\ a\ x2. P\ x1\ a \wedge P\ x2 \implies P\ (x1\ a \rightarrow x2) \rrbracket \implies P\ formula$$

Como hemos señalado, el esquema inductivo se aplicará en cada uno de los casos de los constructores, desglosándose 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.

Llegamos así al primer resultado de este apartado:

Lema 1.3 *El conjunto de los átomos de una fórmula proposicional es finito.*

Para proceder a la demostración, vamos a dar una 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 1.4 *Los conjuntos finitos son:*

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

En Isabelle, podemos formalizar el lema como sigue.

lemma *finite* (*atoms F*)

oops

Análogamente, el enunciado formalizado contiene la definición *finite S*, perteneciente a la teoría [FiniteSet.thy](#).

inductive *finite'* :: '*a set* \Rightarrow *bool*' **where**

emptyI' [*simp*, *intro!*]: *finite'* {}

| *insertI'* [*simp*, *intro!*]: *finite'* $A \Rightarrow$ *finite'* (*insert a A*)

Observemos que la definición anterior corresponde a *finite'*. Sin embargo, es análoga a *finite* de la teoría original. Este cambio de notación es necesario para no definir dos veces de manera idéntica la misma noción en Isabelle. Por otra parte, esta definición permitiría la demostración del lema por simplificación pues, dentro de ella las reglas que especifica se han añadido como tácticas de *simp* e *intro!*. Sin embargo, conforme al objetivo de este análisis, detallaremos dónde es usada cada una de las reglas en la prueba detallada.

A continuación, veamos en primer lugar 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 \perp . 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, por hipótesis de inducción, dicho conjunto es finito.

□

Veamos ahora la prueba detallada en Isabelle. Mostraré con detalle todos los casos de conectivas binarias, aunque se puede observar que son completamente equivalentes. 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* \emptyset

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 \perp)

proof —

have *finite* \emptyset

by (*simp only: finite.emptyI*)

then show *finite (atoms \perp)*

by (*simp only*: *formula.set*(2))
qed

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

lemma *atoms-finite-and*:
assumes *finite* (*atoms F1*)
finite (*atoms F2*)
shows *finite* (*atoms* ($F1 \wedge F2$))
proof –
have *finite* (*atoms F1* \cup *atoms F2*)
using *assms*
by (*simp only*: *finite-UnI*)
then show *finite* (*atoms* ($F1 \wedge F2$))
by (*simp only*: *formula.set*(4))
qed

lemma *atoms-finite-or*:
assumes *finite* (*atoms F1*)
finite (*atoms F2*)
shows *finite* (*atoms* ($F1 \vee F2$))
proof –
have *finite* (*atoms F1* \cup *atoms F2*)
using *assms*
by (*simp only*: *finite-UnI*)
then show *finite* (*atoms* ($F1 \vee F2$))
by (*simp only*: *formula.set*(5))
qed

lemma *atoms-finite-imp*:
assumes *finite* (*atoms F1*)
finite (*atoms F2*)
shows *finite* (*atoms* ($F1 \rightarrow F2$))
proof –
have *finite* (*atoms F1* \cup *atoms F2*)
using *assms*
by (*simp only*: *finite-UnI*)


```

    then show finite (atoms ( $F1 \rightarrow F2$ ))
      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 1.5 *El conjunto de subfórmulas de una fórmula F , notada $\text{Subf}(F)$, se define recursivamente como:*

- $\{\perp\}$ si F es \perp .
- $\{F\}$ si F es una fórmula atómica.
- $\{F\} \cup \text{Subf}(G)$ si F es $\neg G$.

- $\{F\} \cup \text{Subf}(G) \cup \text{Subf}(H)$ 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  $\perp$  = [ $\perp$ ]
| 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 ( $\neg$  (Atom p)) = [ $\neg$  (Atom p), Atom p]
    by simp

  have subformulae ((Atom p  $\rightarrow$  Atom q)  $\vee$  Atom r) =
    [(Atom p  $\rightarrow$  Atom q)  $\vee$  Atom r, Atom p  $\rightarrow$  Atom q, Atom p, Atom
q,
  Atom r]
    by simp

  have subformulae (Atom p  $\wedge$   $\perp$ ) = [Atom p  $\wedge$   $\perp$ , Atom p,  $\perp$ ]
    by simp

  have subformulae (Atom p  $\vee$  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*(·). En Isabelle, primero hemos definido la lista de subfórmulas pues, en algunos casos, es más sencilla la prueba de resultados sobre este tipo. Sin embargo, el tipo de conjuntos facilita las pruebas de los resultados de esta sección. 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. Esta elección se debe a que el tipo *abbreviation* se trata de un sinónimo para una expresión cuyo tipo ya existe (en nuestro caso, convertir en conjunto la lista obtenida con *subformulae*). 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} ∪ 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), simp only: list.set*)

lemma *setSubformulae-bot:*

setSubformulae (\perp) = { \perp }

by (*simp only: subformulae.simps(2), simp only: 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 \vee F2)

= {F1 \vee F2} \cup (setSubformulae F1 \cup setSubformulae F2)

proof –

have $setSubformulae (F1 \vee F2)$
 $= set ((F1 \vee F2) \# (subformulae F1 @ subformulae F2))$
by (*simp only: subformulae.simps(5)*)
also have $\dots = \{F1 \vee F2\} \cup (set (subformulae F1 @ subformulae F2))$
by (*simp only: set-insert*)
also have $\dots = \{F1 \vee F2\} \cup (setSubformulae F1 \cup setSubformulae F2)$
by (*simp only: set-append*)
finally show *?thesis*
by *this*

qed

lemma *setSubformulae-imp:*

$setSubformulae (F1 \rightarrow F2)$
 $= \{F1 \rightarrow F2\} \cup (setSubformulae F1 \cup setSubformulae F2)$

proof –

have $setSubformulae (F1 \rightarrow F2)$
 $= set ((F1 \rightarrow F2) \# (subformulae F1 @ subformulae F2))$
by (*simp only: subformulae.simps(6)*)
also have $\dots = \{F1 \rightarrow F2\} \cup (set (subformulae F1 @ subformulae F2))$
by (*simp only: set-insert*)
also have $\dots = \{F1 \rightarrow F2\} \cup (setSubformulae F1 \cup setSubformulae F2)$
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 1.6 $F \in Subf(F)$.

Demostración: Por inducción en la estructura de las fórmulas. Se tienen los siguientes casos:

Sea p fórmula atómica cualquiera. Por definición de *Subf* tenemos que $Subf(p) = \{p\}$, luego se tiene la propiedad.

Sea la fórmula \perp . Como $Subf(\perp) = \{\perp\}$, se verifica el resultado.

Por definición del conjunto de subfórmulas de $Subf(\neg F)$ se tiene la propiedad para este caso, pues $Subf(\neg F) = \{\neg F\} \cup Subf(F) \implies \neg F \in$

$Subf(\neg F)$ como queríamos ver.

Análogamente, para cualquier conectiva binaria $*$ y fórmulas F y G se cumple $Subf(F * G) = \{F * G\} \cup Subf(F) \cup Subf(G)$, luego se cumple la propiedad. \square

Formalicemos ahora el lema con su correspondiente demostración detallada.

```

lemma subformulae-self:  $F \in setSubformulae\ F$ 
proof (induction F)
  case (Atom x)
  then show ?case
    by (simp only: singletonI setSubformulae-atom)
next
  case Bot
  then show ?case
    by (simp only: singletonI setSubformulae-bot)
next
  case (Not F)
  then show ?case
    by (simp add: insertI1 setSubformulae-not)
next
  case (And F1 F2)
  then show ?case
    by (simp add: insertI1 setSubformulae-and)
next
  case (Or F1 F2)
  then show ?case
    by (simp add: insertI1 setSubformulae-or)
next
  case (Imp F1 F2)
  then show ?case
    by (simp add: insertI1 setSubformulae-imp)
qed

```

La demostración automática es la siguiente.

```

lemma  $F \in setSubformulae\ F$ 
by (induction 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 aparecerán 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  $A \cup B \subseteq C \implies A \subseteq C$ 
  by simp

```

```

lemma  $A \cup B \subseteq C \implies B \subseteq C$ 
  by simp

```

Veamos ahora los distintos resultados sobre subfórmulas.

Lema 1.7 *Sean F una fórmula proposicional y A_F el conjunto de las fórmulas atómicas formadas a partir de cada elemento del conjunto de variables proposicionales de F . Entonces, $A_F \subseteq \text{Subf}(F)$.*

Por tanto, las fórmulas atómicas son subfórmulas.

Demostración: La prueba seguirá el esquema inductivo para la estructura de fórmulas. Veamos cada caso:

Consideremos la fórmula atómica p cualquiera. Entonces, su conjunto de átomos es $\{p\}$. De este modo, el conjunto A_p correspondiente será $A_p = \{p\} \subseteq \{p\} = \text{Subf}(\text{Atom } p)$ como queríamos demostrar.

Sea la fórmula \perp . Como su conjunto de átomos es vacío, es claro que $A_{\perp} = \emptyset \subseteq \text{Subf}(\perp) = \emptyset$.

Sea la fórmula F tal que $A_F \subseteq \text{Subf}(F)$. Probemos el resultado para $\neg F$. Por definición tenemos que los conjuntos de variables proposicionales de F y $\neg F$ coinciden, luego $A_{\neg F} = A_F$. Además, $\text{Subf}(\neg F) = \{\neg F\} \cup \text{Subf}(F)$. Por tanto, por hipótesis de inducción tenemos: $A_{\neg F} = A_F \subseteq \text{Subf}(F) \subseteq \{\neg F\} \cup \text{Subf}(F) = \text{Subf}(\neg F)$, luego $A_{\neg F} \subseteq \text{Subf}(\neg F)$.

Sean las fórmulas F y G tales que $A_F \subseteq \text{Subf}(F)$ y $A_G \subseteq \text{Subf}(G)$. Probemos ahora $A_{F * G} \subseteq \text{Subf}(F * G)$ para cualquier conectiva binaria $*$. Por un lado, el conjunto de átomos de $F * G$ es la unión de sus correspondientes conjuntos de átomos, luego $A_{F * G} = A_F \cup A_G$. Por tanto, por hipótesis de inducción y definición del conjunto de subfórmulas, se tiene: $A_{F * G} = A_F \cup A_G \subseteq \text{Subf}(F) \cup \text{Subf}(G) \subseteq \{F * G\} \cup \text{Subf}(F) \cup \text{Subf}(G) = \text{Subf}(F * G)$. Luego, $A_{F * G} \subseteq \text{Subf}(F * G)$ como queríamos demostrar. □

En Isabelle, se especifica como sigue.

lemma *Atom ‘ atoms F* \subseteq *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. Dicho conjunto es equivalente al conjunto A_F del enunciado del lema. Para ello emplea el infijo $'$ definido como notación abreviada de (\cdot) que calcula la imagen de un conjunto en la teoría [Set.thy](#).

$$f \text{ ' } A = \{y \mid \exists x \in A. y = f x\} \quad (\text{image-def})$$

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

notepad
begin


```

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

have  $Atom\ 'atoms\ (Atom\ p \vee \perp) = \{Atom\ p\}$ 
  by simp

have  $Atom\ 'atoms\ ((Atom\ p \rightarrow Atom\ q) \vee Atom\ r) =$ 
   $\{Atom\ p, Atom\ q, Atom\ r\}$ 
  by auto

have  $Atom\ 'atoms\ ((Atom\ p \rightarrow Atom\ p) \vee Atom\ r) = \{Atom\ p, Atom$ 
 $r\}$ 
  by auto
end

```

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

$$f\ ' (A \cup B) = f\ ' A \cup f\ ' B \quad (image-Un)$$

$$f\ ' (\{a\} \cup B) = \{f\ a\} \cup f\ ' B \quad (image-insert)$$

$$f\ ' \emptyset = \emptyset \quad (image-empty)$$

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 $\dots = \{Atom\ x\}$

by (*simp only: image-insert image-empty*)

also have $\dots = set\ [Atom\ x]$

by (*simp only: list.set(1) list.set(2)*)

also have $\dots = set\ (subformulae\ (Atom\ x))$

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*:
 $Atom \text{ ' } atoms \perp \subseteq setSubformulae \perp$
proof –
 have $Atom \text{ ' } atoms \perp = Atom \text{ ' } \emptyset$
 by (*simp only: formula.set(2)*)
 also have $\dots = \emptyset$
 by (*simp only: image-empty*)
 also have $\dots \subseteq setSubformulae \perp$
 by (*simp only: empty-subsetI*)
 finally show *?thesis*
 by *this*
qed

lemma *atoms-are-subformulae-not*:
 assumes $Atom \text{ ' } atoms F \subseteq setSubformulae F$
 shows $Atom \text{ ' } atoms (\neg F) \subseteq setSubformulae (\neg F)$
proof –
 have $Atom \text{ ' } atoms (\neg F) = Atom \text{ ' } atoms F$
 by (*simp only: formula.set(3)*)
 also have $\dots \subseteq setSubformulae F$
 by (*simp only: assms*)
 also have $\dots \subseteq \{\neg F\} \cup setSubformulae F$
 by (*simp only: Un-upper2*)
 also have $\dots = setSubformulae (\neg F)$
 by (*simp only: setSubformulae-not*)
 finally show *?thesis*
 by *this*
qed

lemma *atoms-are-subformulae-and*:
 assumes $Atom \text{ ' } atoms F1 \subseteq setSubformulae F1$
 $Atom \text{ ' } atoms F2 \subseteq setSubformulae F2$
 shows $Atom \text{ ' } atoms (F1 \wedge F2) \subseteq setSubformulae (F1 \wedge F2)$
proof –
 have $Atom \text{ ' } atoms (F1 \wedge F2) = Atom \text{ ' } (atoms F1 \cup atoms F2)$
 by (*simp only: formula.set(4)*)
 also have $\dots = Atom \text{ ' } atoms F1 \cup Atom \text{ ' } atoms F2$
 by (*rule image-Un*)
 also have $\dots \subseteq setSubformulae F1 \cup setSubformulae F2$
 using *assms*

by (rule Un-mono)
 also have $\dots \subseteq \{F1 \wedge F2\} \cup (\text{setSubformulae } F1 \cup \text{setSubformulae } F2)$
 by (simp only: Un-upper2)
 also have $\dots = \text{setSubformulae } (F1 \wedge F2)$
 by (simp only: setSubformulae-and)
 finally show ?thesis
 by this
 qed

lemma *atoms-are-subformulae-or:*

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

lemma *atoms-are-subformulae-imp:*

assumes $\text{Atom } ' \text{atoms } F1 \subseteq \text{setSubformulae } F1$
 $\text{Atom } ' \text{atoms } F2 \subseteq \text{setSubformulae } F2$
 shows $\text{Atom } ' \text{atoms } (F1 \rightarrow F2) \subseteq \text{setSubformulae } (F1 \rightarrow F2)$
 proof –
 have $\text{Atom } ' \text{atoms } (F1 \rightarrow F2) = \text{Atom } ' (\text{atoms } F1 \cup \text{atoms } F2)$
 by (simp only: formula.set(6))
 also have $\dots = \text{Atom } ' \text{atoms } F1 \cup \text{Atom } ' \text{atoms } F2$
 by (rule image-Un)
 also have $\dots \subseteq \text{setSubformulae } F1 \cup \text{setSubformulae } F2$
 using assms
 by (rule Un-mono)

```

also have ...  $\subseteq \{F1 \rightarrow F2\} \cup (setSubformulae F1 \cup setSubformulae F2)$ 
  by (simp only: Un-upper2)
also have ... = setSubformulae (F1  $\rightarrow$  F2)
  by (simp only: setSubformulae-imp)
finally show ?thesis
  by this
qed

```

```

lemma atoms-are-subformulae:
  Atom ' atoms F  $\subseteq$  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 1.8 Sea $G \in Subf(F)$, entonces el conjunto de átomos de G está contenido en el de F .

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

Sea p una fórmula atómica cualquiera. Si $G \in \text{Subf}(p)$, como su conjunto de variables es $\{p\}$, se tiene $G = p$. Por tanto, se tiene el resultado.

Sea la fórmula \perp . Si $G \in \text{Subf}(\perp)$, como su conjunto de átomos es $\{\perp\}$, se tiene $G = \perp$. Por tanto, se cumple la propiedad.

Sea la fórmula F cualquiera tal que para cualquier subfórmula $G \in \text{Subf}(F)$ se verifica que el conjunto de átomos de G está contenido en el de F . Supongamos $G' \in \text{Subf}(\neg F)$ cualquiera, probemos que $\text{conjAtoms}(G') \subseteq \text{conjAtoms}(\neg F)$. Por definición, tenemos que $\text{Subf}(\neg F) = \{\neg F\} \cup \text{Subf}(F)$. De este modo, tenemos dos opciones: $G' \in \{\neg F\}$ o $G' \in \text{Subf}(F)$. Del primer caso se deduce $G' = \neg F$ y, por tanto, se verifica el resultado. 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 $F1$ fórmula proposicional tal que para cualquier $G \in \text{Subf}(F1)$ se tiene que el conjunto de átomos de G está contenido en el de $F1$. Sea también $F2$ tal que dada $G \in \text{Subf}(F2)$ cualquiera se verifica también la hipótesis de inducción en su caso. Supongamos $G' \in \text{Subf}(F1 * F2)$ donde $*$ es cualquier conectiva binaria. Vamos a probar que el conjunto de átomos de G está contenido en el de $F1 * F2$.

En primer lugar, como $\text{Subf}(F1 * F2) = \{F1 * F2\} \cup (\text{Subf}(F1) \cup \text{Subf}(F2))$, se desglosan tres casos posibles para G' : Si $G' \in \{F1 * F2\}$, entonces $G' = F1 * F2$ y se tiene la propiedad. Si $G' \in \text{Subf}(F1) \cup \text{Subf}(F2)$, entonces por propiedades de conjuntos: $G' \in \text{Subf}(F1)$ o $G' \in \text{Subf}(F2)$. Si $G' \in \text{Subf}(F1)$, por hipótesis de inducción se tiene el resultado. Como el conjunto de átomos de $F1 * F2$ es la unión de los conjuntos de átomos de $F1$ y $F2$, se obtiene el resultado como consecuencia de la transitividad de contención para conjuntos. El caso $G' \in \text{Subf}(F2)$ se demuestra de la misma forma. \square

Formalizado en Isabelle:

lemma $G \in \text{setSubformulae } F \implies \text{atoms } G \subseteq \text{atoms } F$
oops

Veamos su demostración estructurada.

lemma *subformulas-atoms-atom*:
assumes $G \in \text{setSubformulae } (\text{Atom } x)$
shows $\text{atoms } G \subseteq \text{atoms } (\text{Atom } x)$
proof –
have $G \in \{\text{Atom } x\}$
using *assms*

by (*simp only: setSubformulae-atom*)
 then have $G = \text{Atom } x$
 by (*simp only: singletonD*)
 then show ?thesis
 by (*simp only: subset-refl*)
 qed

lemma *subformulas-atoms-bot*:
 assumes $G \in \text{setSubformulae } \perp$
 shows $\text{atoms } G \subseteq \text{atoms } \perp$
proof –
 have $G \in \{\perp\}$
 using *assms*
 by (*simp only: setSubformulae-bot*)
 then have $G = \perp$
 by (*simp only: singletonD*)
 then show ?thesis
 by (*simp only: subset-refl*)
 qed

lemma *subformulas-atoms-not*:
 assumes $G \in \text{setSubformulae } F \implies \text{atoms } G \subseteq \text{atoms } F$
 $G \in \text{setSubformulae } (\neg F)$
 shows $\text{atoms } G \subseteq \text{atoms } (\neg F)$
proof –
 have $G \in \{\neg F\} \cup \text{setSubformulae } F$
 using *assms*(2)
 by (*simp only: setSubformulae-not*)
 then have $G \in \{\neg F\} \vee G \in \text{setSubformulae } F$
 by (*simp only: Un-iff*)
 then show $\text{atoms } G \subseteq \text{atoms } (\neg F)$
proof
 assume $G \in \{\neg F\}$
 then have $G = \neg F$
 by (*simp only: singletonD*)
 then show ?thesis
 by (*simp only: subset-refl*)
next
 assume $G \in \text{setSubformulae } F$
 then have $\text{atoms } G \subseteq \text{atoms } F$
 by (*simp only: assms*(1))

also have $\dots = \text{atoms } (\neg F)$
 by (simp only: formula.set(3))
 finally show ?thesis
 by this
 qed
 qed

lemma *subformulas-atoms-and:*

assumes $G \in \text{setSubformulae } F1 \implies \text{atoms } G \subseteq \text{atoms } F1$
 $G \in \text{setSubformulae } F2 \implies \text{atoms } G \subseteq \text{atoms } F2$
 $G \in \text{setSubformulae } (F1 \wedge F2)$
 shows $\text{atoms } G \subseteq \text{atoms } (F1 \wedge F2)$

proof –

have $G \in \{F1 \wedge F2\} \cup (\text{setSubformulae } F1 \cup \text{setSubformulae } F2)$
 using *assms*(3)
 by (simp only: setSubformulae-and)
 then have $G \in \{F1 \wedge F2\} \vee G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$

F2

by (simp only: Un-iff)

then show ?thesis

proof

assume $G \in \{F1 \wedge F2\}$
 then have $G = F1 \wedge F2$
 by (simp only: singletonD)
 then show ?thesis
 by (simp only: subset-refl)

next

assume $G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$
 then have $G \in \text{setSubformulae } F1 \vee G \in \text{setSubformulae } F2$
 by (simp only: Un-iff)
 then show ?thesis

proof

assume $G \in \text{setSubformulae } F1$
 then have $\text{atoms } G \subseteq \text{atoms } F1$
 by (rule *assms*(1))
 also have $\dots \subseteq \text{atoms } F1 \cup \text{atoms } F2$
 by (simp only: Un-upper1)
 also have $\dots = \text{atoms } (F1 \wedge F2)$
 by (simp only: formula.set(4))
 finally show ?thesis
 by this

```

next
  assume  $G \in \text{setSubformulae } F2$ 
  then have  $\text{atoms } G \subseteq \text{atoms } F2$ 
    by (rule assms(2))
  also have  $\dots \subseteq \text{atoms } F1 \cup \text{atoms } F2$ 
    by (simp only: Un-upper2)
  also have  $\dots = \text{atoms } (F1 \wedge F2)$ 
    by (simp only: formula.set(4))
  finally show ?thesis
    by this
qed
qed
qed

lemma subformulas-atoms-or:
  assumes  $G \in \text{setSubformulae } F1 \implies \text{atoms } G \subseteq \text{atoms } F1$ 
     $G \in \text{setSubformulae } F2 \implies \text{atoms } G \subseteq \text{atoms } F2$ 
     $G \in \text{setSubformulae } (F1 \vee F2)$ 
  shows  $\text{atoms } G \subseteq \text{atoms } (F1 \vee F2)$ 
proof -
  have  $G \in \{F1 \vee F2\} \cup (\text{setSubformulae } F1 \cup \text{setSubformulae } F2)$ 
    using assms(3)
  by (simp only: setSubformulae-or)
  then have  $G \in \{F1 \vee F2\} \vee G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$ 
  by (simp only: Un-iff)
  then show ?thesis
  proof
    assume  $G \in \{F1 \vee F2\}$ 
    then have  $G = F1 \vee F2$ 
      by (simp only: singletonD)
    then show ?thesis
      by (simp only: subset-refl)
  next
    assume  $G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$ 
    then have  $G \in \text{setSubformulae } F1 \vee G \in \text{setSubformulae } F2$ 
      by (simp only: Un-iff)
    then show ?thesis
    proof
      assume  $G \in \text{setSubformulae } F1$ 
      then have  $\text{atoms } G \subseteq \text{atoms } F1$ 

```



```

    by (rule assms(1))
  also have ...  $\subseteq$  atoms  $F1 \cup$  atoms  $F2$ 
    by (simp only: Un-upper1)
  also have ... = atoms  $(F1 \vee F2)$ 
    by (simp only: formula.set(5))
  finally show ?thesis
    by this
next
  assume  $G \in$  setSubformulae  $F2$ 
  then have atoms  $G \subseteq$  atoms  $F2$ 
    by (rule assms(2))
  also have ...  $\subseteq$  atoms  $F1 \cup$  atoms  $F2$ 
    by (simp only: Un-upper2)
  also have ... = atoms  $(F1 \vee F2)$ 
    by (simp only: formula.set(5))
  finally show ?thesis
    by this
qed
qed
qed

lemma subformulas-atoms-imp:
  assumes  $G \in$  setSubformulae  $F1 \implies$  atoms  $G \subseteq$  atoms  $F1$ 
     $G \in$  setSubformulae  $F2 \implies$  atoms  $G \subseteq$  atoms  $F2$ 
     $G \in$  setSubformulae  $(F1 \rightarrow F2)$ 
  shows atoms  $G \subseteq$  atoms  $(F1 \rightarrow F2)$ 
proof -
  have  $G \in \{F1 \rightarrow F2\} \cup (\text{setSubformulae } F1 \cup \text{setSubformulae } F2)$ 
    using assms(3)
    by (simp only: setSubformulae-imp)
  then have  $G \in \{F1 \rightarrow F2\} \vee G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$ 
  by (simp only: Un-iff)
  then show ?thesis
proof
  assume  $G \in \{F1 \rightarrow F2\}$ 
  then have  $G = F1 \rightarrow F2$ 
    by (simp only: singletonD)
  then show ?thesis
    by (simp only: subset-refl)
next

```

```

assume  $G \in \text{setSubformulae } F1 \cup \text{setSubformulae } F2$ 
then have  $G \in \text{setSubformulae } F1 \vee G \in \text{setSubformulae } F2$ 
  by (simp only: Un-iff)
then show ?thesis
proof
  assume  $G \in \text{setSubformulae } F1$ 
  then have  $\text{atoms } G \subseteq \text{atoms } F1$ 
    by (rule assms(1))
  also have  $\dots \subseteq \text{atoms } F1 \cup \text{atoms } F2$ 
    by (simp only: Un-upper1)
  also have  $\dots = \text{atoms } (F1 \rightarrow F2)$ 
    by (simp only: formula.set(6))
  finally show ?thesis
    by this
next
  assume  $G \in \text{setSubformulae } F2$ 
  then have  $\text{atoms } G \subseteq \text{atoms } F2$ 
    by (rule assms(2))
  also have  $\dots \subseteq \text{atoms } F1 \cup \text{atoms } F2$ 
    by (simp only: Un-upper2)
  also have  $\dots = \text{atoms } (F1 \rightarrow F2)$ 
    by (simp only: formula.set(6))
  finally show ?thesis
    by this
qed
qed
qed

lemma subformulae-atoms:  $G \in \text{setSubformulae } F \implies \text{atoms } G \subseteq \text{atoms } F$ 
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 aplicativa automática.

```

lemma  $G \in \text{setSubformulae } F \implies \text{atoms } G \subseteq \text{atoms } F$ 
  by (induction F) auto

```

Comentario 5: Corregido hasta aquí.

2 Glosario de reglas

2.1 Teoría de conjuntos finitos

Comentario 6: Explicar la siguiente notación y recolocarla donde se use por primera vez.

Comentario 7: Falta Corregir.

A continuación se muestran resultados relativos a la teoría [FiniteSet.thy](#). Dicha teoría se basa en la definición recursiva de *finite*, que aparece retratada en la sección de *Sintaxis*. Además, hemos empleado los siguientes resultados.

$$\frac{\text{finite } F \wedge \text{finite } G}{\text{finite } (F \cup G)} \quad (\text{finite-UnI})$$

2.2 Teoría de listas

La teoría de listas en Isabelle corresponde a [List.thy](#). Esta se fundamenta en la definición recursiva de *list*.

```
datatype (set': 'a) list' = \ Nil' ([])\ | Cons' (hd: 'a) (tl: 'a list')
(infixr # 65)\ for\ map: map\ rel: list-all2\ pred: list-all\ where\
tl [] = []\
```

COMENTARIO: NO ME PERMITE PONERLO FUERA DEL ENTORNO DE TEXTO, NI CAMBIANDO EL NOMBRE

Como es habitual, hemos cambiado la notación de la definición a *list'* para no definir dos veces de manera idéntica la misma noción. Simultáneamente se define la función de conjuntos *set* (idéntica a *set'*), una función *map*, una relación *rel* y un predicado *pred*. Para dicha definición hemos empleado los operadores sobre listas *hd* y *tl*. De este modo, *hd* aplicado a una lista de elementos de un tipo cualquiera '*a*' nos devuelve el primer elemento de la misma, y *tl* nos devuelve la lista quitando el primer elemento.

Además, hemos utilizado las siguientes propiedades sobre listas.

$$\{a\} \cup B \cup C = \{a\} \cup (B \cup C) \quad (Un-insert-left)$$

2.3 Teoría de conjuntos

Los siguientes resultados empleados en el análisis hecho sobre la lógica proposicional corresponden a la teoría de conjuntos de Isabelle: [Set.thy](#).

$$xs @ ys = xs \cup ys \quad (set-append)$$

$$a \in \{a\} \quad (singletonI)$$

$$a \in \{a\} \cup B \quad (insertI1)$$

$$A \cup \emptyset = A \quad (Un-empty-right)$$

$$\frac{A \subseteq B \wedge B \subseteq C}{A \subseteq C} \quad (subset-trans)$$

$$\frac{c \in A \wedge A \subseteq B}{c \in B} \quad (rev-subsetD)$$

$$\frac{A \subseteq C \wedge B \subseteq D}{A \cup B \subseteq C \cup D} \quad (Un-mono)$$

$$A \subseteq A \cup B \quad (Un-upper1)$$

$$\begin{array}{ll}
B \subseteq A \cup B & (Un-upper2) \\
A \subseteq A & (subset-refl) \\
\emptyset \subseteq A & (empty-subsetI) \\
\frac{b \in \{a\}}{b = a} & (singletonD) \\
(c \in A \cup B) = (c \in A \vee c \in B) & (Un-iff)
\end{array}$$

2.4 Lógica de primer orden

En Isabelle corresponde a la teoría `HOL.thy` Los resultados empleados son los siguientes.

$$\begin{array}{ll}
\frac{P \wedge Q}{P} & (conjunct1) \\
\frac{P \wedge Q}{Q} & (conjunct2)
\end{array}$$

Referencias

- [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] M. Fitting. *First-order Logic and Automated Theorem Proving*. Graduate texts in computer science. Springer, 1996.
- [3] 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>.
- [4] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A proof assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283, Springer-Verlag, 2019. En <https://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2019/doc/tutorial.pdf>.

Comentario 8: Añadir el artículo que se usa como base.

Lista de tareas pendientes

■ Comentario 1: Falta la introducción.	1
■ Comentario 2: Explicar la siguiente notación y recolocarla donde se use por primera vez.	1
■ Comentario 3: Incluir el árbol de formación.	2
■ Comentario 4: Poner bien cada regla.	5
■ Comentario 5: Corregido hasta aquí.	27
■ Comentario 6: Explicar la siguiente notación y recolocarla donde se use por primera vez.	27
■ Comentario 7: Falta Corregir.	27
■ Comentario 8: Añadir el artículo que se usa como base.	29