Lógica proposicional en Isabelle/HOL

Sofía Santiago Fernández

actualizado el 15 de diciembre de 2019

Comentario 1: Falta la introducción.

Índice

Capítulo 1

Sintaxis

1.1 Fórmulas

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

Comentario 3: He quitado la palabra "continuación" del fichero castellano.tex ya que no dejaba cargar el documento

notation *insert* (- $\triangleright$ - [*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 le 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 sinctáticas 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.

Comentario 4: Incluir el árbol de formación.

A continuación, veamos su representación en Isabelle

```
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 constructures sobre un tipo *'a* cualquiera:

Fórmulas básicas:

- *Atom* :: *'a* ⇒ *'a formula*
- $\bot$ :: *'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.

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

A continuación veremos varios ejemplos de fórmulas y el conjunto de sus variables proposicionales obtenido mediante *atoms*. Se observa que, por 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 (¬ (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:

Definición 1.1.2 *Sea $\mathcal{P}$ una propiedad sobre fórmulas que verifica las si-*

guientes 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 $\mathcal{P}$.

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.

Comentario 5: Poner bien cada regla.

$$\frac{\begin{array}{ccc} & \bigwedge x.\ P\ (Atom\ x) & \\ P\ \bot & \bigwedge x.\ \dfrac{P\ x}{P\ (\neg\ x)} & \bigwedge x1a\ x2.\ \dfrac{P\ x1a \wedge P\ x2}{P\ (x1a \wedge x2)} \\ \bigwedge x1a\ x2.\ \dfrac{P\ x1a \wedge P\ x2}{P\ (x1a \vee x2)} & & \bigwedge x1a\ x2.\ \dfrac{P\ x1a \wedge P\ x2}{P\ (x1a \rightarrow x2)} \end{array}}{P\ formula}$$

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 1.1.3 *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 1.1.4 *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* $\Longrightarrow$ *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 simplificacion 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 $\bot$. 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.

$\square$

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* ⇒ *'a formula list* **where**
subformulae (*Atom k*) = [*Atom k*]
| *subformulae* ⊥ = [⊥]

| *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* $(F \longrightarrow G) = (F \longrightarrow G)$ # *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* $((Atom\ p \longrightarrow Atom\ q) \vee Atom\ r) =$
$[(Atom\ p \longrightarrow Atom\ q) \vee Atom\ r,\ Atom\ p \longrightarrow Atom\ q,\ Atom\ p,$
$Atom\ q,\ Atom\ r]$
by *simp*

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

have *subformulae* $(Atom\ p \vee Atom\ p) =$
$[Atom\ p \vee Atom\ p,\ Atom\ p,\ Atom\ p,\ Atom\ p]$
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* $\Rightarrow$ $'a$ *formula set* **where**
setSubformulae $F \equiv$ *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. 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 $setSubformulae\ (Atom\ p \lor Atom\ p) = \{Atom\ p \lor Atom\ p,\ Atom\ p\}$
    by *simp*

  have $setSubformulae\ ((Atom\ p \longrightarrow Atom\ q) \lor Atom\ r) =$
      $\{(Atom\ p \longrightarrow Atom\ q) \lor Atom\ r,\ Atom\ p \longrightarrow Atom\ q,\ Atom\ p,$
      $Atom\ q,\ Atom\ r\}$
  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*: $set\ (x\ \#\ ys) = \{x\} \cup set\ ys$
  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 (¬ F) = {¬ F} ∪ setSubformulae F
proof -
  have setSubformulae (¬ F) = set (¬ F # subformulae F)
    by (simp only: subformulae.simps(3))
  also have ... = {¬ F} ∪ set (subformulae F)
    by (simp only: set-insert)
  finally show ?thesis
    by this
qed

lemma setSubformulae-and:
  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(4))
  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-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:
```

$setSubformulae\ (F1 \longrightarrow F2)$
$\quad = \{F1 \longrightarrow F2\} \cup (setSubformulae\ F1 \cup setSubformulae\ F2)$
proof $-$
have $setSubformulae\ (F1 \longrightarrow F2)$
$\qquad = set\ ((F1 \longrightarrow F2)\ \#\ (subformulae\ F1\ @\ subformulae\ F2))$
by $(simp\ only\!: subformulae.simps(6))$
also have $\ldots = \{F1 \longrightarrow F2\} \cup (set\ (subformulae\ F1\ @\ subformulae\ F2))$
by $(simp\ only\!: set\text{-}insert)$
also have $\ldots = \{F1 \longrightarrow F2\} \cup (setSubformulae\ F1 \cup setSubformulae\ F2)$
by $(simp\ only\!: set\text{-}append)$
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.

Comentario 6: Reescribir el siguiente enunciado y demostración.

Lema 1.2.2 $F \in Subf(F)$.

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 $\bot$. Como $Subf(\bot) = \{\bot\}$, se verifica el resultado.

Por definición del conjunto de subfórmulas de $Subf(\neg\ F)$ se tiene la propiedad para este caso, pues $Subf(\neg\ F) = \{\neg\ F\} \cup Subf(F) \Longrightarrow \neg\ F \in Subf(\neg\ F)$ 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 $subformulae\text{-}self\!: F \in setSubformulae\ F$
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 ∈ 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 apareceran en el glosario final.

Además, definiremos dos reglas adicionales que utilizaremos con frecuencia.

```
lemma subContUnionRev1:
  assumes A ∪ B ⊆ C
  shows   A ⊆ C
proof -
  have A ⊆ C ∧ B ⊆ 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 $(simp\ only$: $sup.bounded\text{-}iff)$
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.

Comentario 7: Reescribir el siguiente enunciado y su demostración.

Lema 1.2.3 *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, $A_F \subseteq Subf(F)$.*

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 $A_p = \{p\} \subseteq \{p\} = Subf(Atom\ p)$ como queríamos demostrar.

Sea la fórmula $\bot$. Como su connjunto de átomos es vacío, es claro que $A_\bot = \emptyset \subseteq Subf(\bot) = \emptyset$.

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

Sean las fórmulas F y G tales que $A_F \subseteq Subf(F)$ y $A_G \subseteq Subf(G)$. Probemos ahora $A_{F*G} \subseteq Subf(F*G)$ 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: $A_{F*G} = A_F \cup A_G \subseteq Subf(F) \cup Subf(G) \subseteq \{F*G\} \cup Subf(F) \cup Subf(G) = Subf(F*G)$ Luego, $A_{F*G} \subseteq Subf(F*G)$ 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 (‘) que calcula la imagen de un conjunto en la teoría Set.thy.

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

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.

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

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

$$f \text{ ` } \emptyset = \emptyset \tag{image-empty}$$

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) ⊆ 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 ⊆ 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(4))
  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-and)
  finally show ?thesis
    by this
qed

lemma atoms-are-subformulae-or:
```

```
  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(5))
  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-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 *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 $G \in Subf(p)$, como su conjunto de variables es $\{p\}$, se tiene $G = p$. Por tanto, se tiene el resultado.

Sea la fórmula $\perp$. Si $G \in Subf(\perp)$, 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 $G \in Subf(F)$ se verifica que el conjunto de átomos de G está contenido en el de F. Supongamos $G' \in Subf(\neg\ F)$ cualquiera, probemos que $conjAtoms(G')$ $\subseteq$ $conjAtoms(\neg\ F)$. Por definición, tenemos que $Subf(\neg\ F) = \{\neg\ F\} \cup$

Subf(F). De este modo, tenemos dos opciones: $G' \in \{\neg\, F\}$ o $G' \in \mathit{Subf}(F)$. 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 $G \in \mathit{Subf}(F1)$ se tiene que el conjunto de átomos de G está contenido en el de $F1$. Sea también $F2$ tal que dada $G \in \mathit{Subf}(F2)$ cualquiera se verifica también la hipótesis de inducción en su caso. Supongamos $G' \in \mathit{Subf}(F1{*}F2)$ 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 $\mathit{Subf}(F1{*}F2) = \{F1{*}F2\} \cup (\mathit{Subf}(F1) \cup \mathit{Subf}(F2))$, se desglosan tres casos posibles para G': Si $G' \in \{F1{*}F2\}$, entonces $G' = F1{*}F2$ y se tiene la propiedad. Si $G' \in \mathit{Subf}(F1) \cup \mathit{Subf}(F2)$, entonces por propiedades de conjuntos: $G' \in \mathit{Subf}(F1)$ o $G' \in \mathit{Subf}(F2)$. Si $G' \in \mathit{Subf}(F1)$, 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 $G' \in \mathit{Subf}(F2)$ se demuestra de la misma forma.

$\square$

Formalizado en Isabelle:

lemma $G \in$ *setSubformulae F* $\Longrightarrow$ *atoms G* $\subseteq$ *atoms F*
  oops

Veamos su demostración estructurada.

lemma *subformulas-atoms-atom*:
  assumes $G \in$ *setSubformulae* (*Atom x*)
  shows *atoms G* $\subseteq$ *atoms* (*Atom x*)
proof −
  have $G \in \{$*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
    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
    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 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
    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 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
    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 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 aplicativa automática.

lemma $G \in setSubformulae\ F \Longrightarrow atoms\ G \subseteq atoms\ F$
  by $(induction\ F)\ auto$

A continuación voy a introducir un lema que no pertenece a la teoría original de Isabelle pero facilita las siguientes demostraciones detalladas mediante contenciones en cadena.

Lema 1.2.5 *Sea G 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. Entonces, bajo las condiciones del lema se tiene que $G = p$. Por lo tanto, tienen igual conjunto de subfórmulas.

Sea la fórmula $\bot$. Entonces, $G = \bot$ y tienen igual conjunto de subfórmulas.

Sea una fórmula F tal que para toda subfórmula G, se tiene que el conjunto de subfórmulas de G está contenido en el de F. Veamos la propiedad para $\neg F$. Sea $G' \in Subf(\neg F) = \{\neg F\} \cup Subf(F)$. Entonces $G' \in \{\neg F\}$ o $G' \in Subf(F)$. Del primer caso se obtiene que $G' = \neg F$ y, por tanto, tienen igual conjunto de subfórmulas. Del segundo caso se tiene $G' \in Subf(F)$ y, por hipótesis de inducción, 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 $\neg F$ por definición, se verifica la propiedad como consecuencia de la transitividad de la contención.

Sean las fórmulas $F1$ y $F2$ tales que para cualquier subfórmula $G1$ de $F1$ el conjunto de subfórmulas de $G1$ está contenido en el de $F1$, y para cualquier subfórmula $G2$ de $F2$ el conjunto de subfórmulas de $G2$ está contenido en el de $F2$. Veamos que se verifica la propiedad para $F1{*}F2$ donde $*$ es cualquier conectiva binaria. Sea $G' \in Subf(F1{*}F2) = \{F1{*}F2\} \cup Subf(F1) \cup Subf(F2)$. De este modo, tenemos tres casos: $G' \in \{F1{*}F2\}$ o $G' \in Subf(F1)$ o $G' \in Subf(F2)$. De la primera opción se deduce $G' = F1{*}F2$ y, por tanto, tienen igual conjunto de subfórmulas. Por otro lado, si $G' \in Subf(F1)$, por hipótesis de inducción se tiene que el conjunto de subfórmulas de G' está contenido en el de $F1$. Por tanto, como el conjunto de subfórmulas de $F1$ está a su vez contenido en el de $F1{*}F2$, se verifica la propiedad por la transitividad de la contención en cadena. El caso $G' \in Subf(F2)$ es análogo cambiando el índice de la fórmula.

$\square$

Veamos su formalización en Isabelle junto con su demostración estructurada.

lemma *subContsubformulae-atom*:
  assumes $G \in$ *setSubformulae* (*Atom x*)
  shows *setSubformulae G* $\subseteq$ *setSubformulae* (*Atom x*)
proof −
  have $G \in \{$*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 $G \in$ *setSubformulae* $\bot$
  shows *setSubformulae G* $\subseteq$ *setSubformulae* $\bot$
proof −
  have $G \in \{\bot\}$
    using *assms*
    by (*simp only*: *setSubformulae-bot*)
  then have $G = \bot$
    by (*simp only*: *singletonD*)
  then show *?thesis*
    by (*simp only*: *subset-refl*)
qed

lemma *subContsubformulae-not*:
  assumes $G \in$ *setSubformulae F* $\Longrightarrow$ *setSubformulae G* $\subseteq$ *setSubformulae F*
    $G \in$ *setSubformulae* ($\neg$ *F*)
  shows *setSubformulae G* $\subseteq$ *setSubformulae* ($\neg$ *F*)
proof −
  have $G \in \{\neg F\} \cup$ *setSubformulae F*
    using *assms*(*2*)
    by (*simp only*: *setSubformulae-not*)
  then have $G \in \{\neg F\} \lor G \in$ *setSubformulae F*
    by (*simp only*: *Un-iff*)
  then show *setSubformulae G* $\subseteq$ *setSubformulae* ($\neg$ *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 1:setSubformulae G ⊆ setSubformulae F
      by (simp only: assms(1))
    also have 2:setSubformulae F ⊆ setSubformulae (¬ F)
      by (simp only: setSubformulae-not Un-upper2)
    finally show ?thesis
      using 1 2 by (simp only: subset-trans)
  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
    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 $G \in$ *setSubformulae F1*
then have *setSubformulae G* $\subseteq$ *setSubformulae F1*
by (*simp only*: *assms*(*1*))
also have $\ldots \subseteq$ *setSubformulae F1* $\cup$ *setSubformulae F2*
by (*simp only*: *Un-upper1*)
also have $\ldots \subseteq$ *setSubformulae* (*F1* $\wedge$ *F2*)
by (*simp only*: *setSubformulae-and Un-upper2*)
finally show *?thesis*
by *this*
next
assume $G \in$ *setSubformulae F2*
then have *setSubformulae G* $\subseteq$ *setSubformulae F2*
by (*rule assms*(*2*))
also have $\ldots \subseteq$ *setSubformulae F1* $\cup$ *setSubformulae F2*
by (*simp only*: *Un-upper2*)
also have $\ldots \subseteq$ *setSubformulae* (*F1* $\wedge$ *F2*)
by (*simp only*: *setSubformulae-and Un-upper2*)
finally show *?thesis*
by *this*
qed
qed
qed

lemma *subContsubformulae-or*:
assumes $G \in$ *setSubformulae F1*
$\Longrightarrow$ *setSubformulae G* $\subseteq$ *setSubformulae F1*
$G \in$ *setSubformulae F2*
$\Longrightarrow$ *setSubformulae G* $\subseteq$ *setSubformulae F2*
$G \in$ *setSubformulae* (*F1* $\vee$ *F2*)
shows *setSubformulae G* $\subseteq$ *setSubformulae* (*F1* $\vee$ *F2*)
proof $-$
have $G \in$ {*F1* $\vee$ *F2*} $\cup$ (*setSubformulae F1* $\cup$ *setSubformulae F2*)
using *assms*(*3*)
by (*simp only*: *setSubformulae-or*)
then have $G \in$ {*F1* $\vee$ *F2*} $\vee$ $G \in$ *setSubformulae F1* $\cup$ *setSubformulae F2*
by (*simp only*: *Un-iff*)
then show *?thesis*
proof

```
    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-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
    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-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
  $G \in setSubformulae\ F \Longrightarrow setSubformulae\ G \subseteq setSubformulae\ F$
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*:
  $G \in setSubformulae\ F \Longrightarrow setSubformulae\ G \subseteq setSubformulae\ F$
  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 *Sean G subfórmula de F y H subfórmula de G, entonces H es subfórmula de F.*

Demostración: La prueba está basada en el lema anterior. Hemos demostrado que como G es subfórmula de F, entonces el conjunto de átomos de G está contenido en el de F. Del mismo modo, como H es subfórmula de G, su conjunto de átomos está contenido en el de G. Por la transitividad de la contención, tenemos que el conjunto de átomos de H está contenido en el de F. Por otro lema anterior, tenemos que H pertenece a su propio conjunto de subfórmulas. Por tanto, $H \in \mathit{Subf}(H) \subseteq \mathit{Subf}(F) \Longrightarrow H \in \mathit{Subf}(F)$.

$\square$

Veamos su formalización y prueba estructurada en Isabelle.

Veamos su prueba según las distintas tácticas.

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

lemma *subsubformulae*:
  $G \in setSubformulae\ F$
  $\Longrightarrow H \in setSubformulae\ G$
  $\Longrightarrow H \in setSubformulae\ F$
  using *subContsubformulae* **by** *blast*

comentarioExplicar el cambio de enunciado

A continuación presentamos otro resultado que relaciona los conjuntos de subfórmulas según las conectivas que operen.

lemma *subformulas-in-subformulas*:
  $G \wedge H \in setSubformulae\ F$
  $\Longrightarrow G \in setSubformulae\ F \wedge H \in setSubformulae\ F$
  $G \vee H \in setSubformulae\ F$
  $\Longrightarrow G \in setSubformulae\ F \wedge H \in setSubformulae\ F$
  $G \rightarrow H \in setSubformulae\ F$
  $\Longrightarrow G \in setSubformulae\ F \wedge H \in setSubformulae\ F$

$\neg\ G \in \mathit{setSubformulae}\ F \Longrightarrow G \in \mathit{setSubformulae}\ F$
oops

Como podemos observar, el resultado es análogo en todas las conectivas binarias aunque aparezcan definidas por separado, por tanto haré la demostración estructurada para una de ellas pues el resto son análogas.

Nos basaremos en el lema anterior *subsubformulae*.

lemma *subformulas-in-subformulas-not*:
assumes $\mathit{Not}\ G \in \mathit{setSubformulae}\ F$
shows $G \in \mathit{setSubformulae}\ F$
proof −
have $\mathit{setSubformulae}\ (\mathit{Not}\ G) = \{\mathit{Not}\ G\} \cup \mathit{setSubformulae}\ G$
by *simp* — Pendiente
then have $1{:}G \in \mathit{setSubformulae}\ (\mathit{Not}\ G)$
by (*simp add*: *subformulae-self*) — Pendiente
show $G \in \mathit{setSubformulae}\ F$ **using** *assms 1*
by (*rule subsubformulae*)
qed

lemma *subformulas-in-subformulas-and*:
assumes $G \wedge H \in \mathit{setSubformulae}\ F$
shows $G \in \mathit{setSubformulae}\ F \wedge H \in \mathit{setSubformulae}\ F$
proof (*rule conjI*)
have *1*: $\mathit{setSubformulae}\ (G \wedge H) =$
$\{G \wedge H\} \cup (\mathit{setSubformulae}\ G \cup \mathit{setSubformulae}\ H)$
by (*simp only*: *setSubformulae-and*)
then have *2*: $G \in \mathit{setSubformulae}\ (G \wedge H)$
by (*simp add*: *subformulae-self*) — Pendiente
have *3*: $H \in \mathit{setSubformulae}\ (G \wedge H)$
using *1*
by (*simp add*: *subformulae-self*) — Pendiente
show $G \in \mathit{setSubformulae}\ F$ **using** *assms 2* **by** (*rule subsubformulae*)
show $H \in \mathit{setSubformulae}\ F$ **using** *assms 3* **by** (*rule subsubformulae*)
qed

Mostremos ahora la demostración automática.

lemma *subformulas-in-subformulas*:
$G \wedge H \in \mathit{setSubformulae}\ F$
$\Longrightarrow G \in \mathit{setSubformulae}\ F \wedge H \in \mathit{setSubformulae}\ F$
$G \vee H \in \mathit{setSubformulae}\ F$

$\Longrightarrow G \in setSubformulae\ F \wedge H \in setSubformulae\ F$
$G \longrightarrow H \in setSubformulae\ F$
$\Longrightarrow G \in setSubformulae\ F \wedge H \in setSubformulae\ F$
$\neg\ G \in setSubformulae\ F \Longrightarrow G \in setSubformulae\ F$
using *subformulae-self subsubformulae* **apply** *force*
oops

Comentario 8: Completar la prueba anterior.

Comentario 9: Completar lo que falta de sección

Concluimos la sección de subfórmulas con un resultado que relaciona varias funciones sobre la longitud de la lista *subformulae F* de una fórmula F cualquiera.

lemma *length-subformulae*: *length* (*subformulae F*) = *size F*
oops

En primer lugar aparece la clase *size* de la teoría de números naturales ....

Vamos a definir *size1* de manera idéntica a como aparece *size* en la teoría.

class *size1* =
fixes *size1* :: *'a* ⇒ *nat*

instantiation *nat* :: *size1*
begin

definition *size1-nat* **where** [*simp*, *code*]: *size1* (*n*::*nat*) = *n*

instance ..

end

Como podemos observar, se trata de una clase que actúa sobre un parámetro global de tipo *'a* cualquiera. Por otro lado, *instantation* define una clase de parámetros, en este caso los números naturales *nat* que devuelve como resultado. Incluye una definición concreta del operador *size1* sobre dichos parámetros. Además, el último *instance* abre una prueba que afirma que los parámetros dados conforman la clase especificada en la definición.

Esta prueba que nos afirma que está bien definida la clase aparece omitida utilizando .. .

En particular, sobre una fórmula nos devuelve el número de elementos de la lista cuyos elementos son los nodos y las hojas de su árbol de formación.

value *size* $(n::nat) = n$
value *size* $(5::nat) = 5$

Por otro lado, la función *length* de la teoría List.thy nos indica la longitud de una lista cualquiera de elementos, definiéndose utilizando el comando *size* visto anteriormente.

abbreviation *length'* :: *'a list* $\Rightarrow$ *nat* **where**
length' $\equiv$ *size*

Las demostración del resultado se vuelve a basar en la inducción que nos despliega seis casos.

La prueba estructurada no resulta interesante, pues todos los casos son inmediatos por simplificación como en el primer lema de esta sección.

Incluimos a continuación la prueba automática.

lemma *length-subformulae*: *length* (*subformulae F*) = *size F*
by (*induction F*; *simp*)

Comentario 10: Hacer la prueba detallada para mostrar los teoremas utilizados.

1.3 Conectivas derivadas

En esta sección definiremos nuevas conectivas y fórmulas a partir de los ya definidos en el apartado anterior, junto con varios resultados sobre los mismos. 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* $(\top)$ **where**
$\top \equiv \bot \to \bot$

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* $\top = \emptyset$
by (*simp only*: *Top-def formula.set Un-absorb*)

Comentario 11: Añadir regla al glosario.

Comentario 12: Añadir la doble implicación como conectiva derivada.

A continuación vamos a definir dos conectivas que generalizan la conjunción y la disyunción para una lista finita de fórmulas. En particular, al ser aplicadas a listas, se definen conforme a la estructura recursiva de las mismas que se muestra a continuación.

Definición 1.3.2 *Las listas de fórmulas se definen recursivamente como sigue.*

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

Comentario 13: Esta definición es un caso particular de listas. No se si incluir la definicion de estructura e inducción general

De este modo, se definen las conectivas plurales de acuerdo a la estructura recursiva anterior. Notemos que al referirnos simplemente a disyunción o conjunción nos referiremos a la de dos elementos.

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

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

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

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

Su formalización en Isabelle es la siguiente.

Comentario 14: Da error que no localizo

Ambas nuevas conectivas se definen con el tipo funciones primitivas recursivas. Estas se basan en los dos casos descritos anteriormente: la lista vacía representada como *Nil* y la lista construida añadiendo una fórmula a una lista de fórmulas. Además, se observa en cada conectiva plural el nuevo símbolo de notación que aparece entre paréntesis.

Por otro lado, como es habitual, estas definiciones recursivas llevan asociado el correspondiente esquema inductivo de demostración. En este caso, se trata de la inducción para la estructura de lista de fórmulas.

Definición 1.3.5 *Sea $\mathcal{P}$ una propiedad sobre lista de fórmulas proposicionales que verifica las siguientes condiciones:*

- *La lista vacía la verifica.*
- *Dada una fórmula F y una lista de fórmulas Fs que la verifican, entonces $F\#Fs$ la verifica.*

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

La conjunción plural nos da el siguiente resultado.

Comentario 15: Añadir lema a mano y demostración. Falta demostración en Isabelle.

Apéndice A

Lemas de HOL usados

A.1 Teoría de conjuntos finitos

Comentario 16: Explicar la siguiente notación y recolocarla donde se use por primera vez.

Comentario 17: Falta Corregir.

A continuación se muestran resultamos 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.

$$\frac{\textit{finite } F \land \textit{finite } G}{\textit{finite } (F \cup G)} \qquad (\textit{finite-UnI})$$

A.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 defineción 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 elmento.

Además, hemos utilizado las siguientes propiedades sobre listas.

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

A.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.

$$xs \mathbin{@} ys = xs \cup ys \qquad (\textit{set-append})$$

$$a \in \{a\} \qquad (\textit{singletonI})$$

$$a \in \{a\} \cup B \qquad (\textit{insertI1})$$

$$A \cup \emptyset = A \qquad (\textit{Un-empty-right})$$

$$\frac{A \subseteq B \land B \subseteq C}{A \subseteq C} \qquad (\textit{subset-trans})$$

$$\frac{c \in A \land A \subseteq B}{c \in B} \qquad (\textit{rev-subsetD})$$

$$\frac{A \subseteq C \land B \subseteq D}{A \cup B \subseteq C \cup D} \qquad (\textit{Un-mono})$$

$$A \subseteq A \cup B \qquad (\textit{Un-upper1})$$

$$B \subseteq A \cup B \qquad (\textit{Un-upper2})$$

$$A \subseteq A \tag{subset-refl}$$

$$\emptyset \subseteq A \tag{empty-subsetI}$$

$$\frac{b \in \{a\}}{b = a} \tag{singletonD}$$

$$(c \in A \cup B) = (c \in A \vee c \in B) \tag{Un-iff}$$

A.4 Lógica de primer orden

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

$$\frac{P \wedge Q}{P} \tag{conjunct1}$$

$$\frac{P \wedge Q}{Q} \tag{conjunct2}$$

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

[3] 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 18: Añadir el artículo que se usa como base.

Lista de tareas pendientes