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 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 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 *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 ($\bot$)
| $Not\ 'a\ formula$ ($\neg$)
| $And\ 'a\ formula\ 'a\ formula$ (**infix** $\wedge$ *68*)
| $Or\ 'a\ formula\ 'a\ formula$ (**infix** $\vee$ *68*)
| $Imp\ 'a\ formula\ 'a\ formula$ (**infixr** $\rightarrow$ *68*)

Como podemos observar en la definición, *formula* es un tipo de datos recursivo que se entiende como un árbol que relaciona elementos de un tipo $'a$ cualquiera del alfabeto proposicional. En ella, los constructores del tipo son los siguientes:

Fórmulas básicas :

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

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* que preceden a los símbolos de notación de los nodos nos señalan que son infijos, 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. De manera equivalente, daremos la siguiente definición.

Definición 1.2 *Sea F una fórmula proposicional. Entonces, se define conjAtoms(F) como el conjunto de los átomos que aparecen en F.*

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

$atoms\ (Atom\ x1) = \{x1\}$

$atoms\ \bot = \emptyset$

$atoms\ (\neg\ x3) = atoms\ x3$

$atoms\ (x41 \wedge x42) = atoms\ x41 \cup atoms\ x42$

$atoms\ (x51 \vee x52) = atoms\ x51 \cup atoms\ x52$

$atoms\ (x61 \longrightarrow x62) = atoms\ x61 \cup atoms\ x62$

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 (¬ (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 1.3 *Sea φ una propiedad sobre fórmulas que verifica las siguientes condiciones:*

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

Entonces, todas las fórmulas proposicionales tienen la propiedad 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.

$$[\![\bigwedge x.\ P\ (Atom\ x);\ P\ \bot;\ \bigwedge x.\ P\ x \Longrightarrow P\ (\neg\ x);\ \bigwedge x1a\ x2.\ P\ x1a \wedge P\ x2 \Longrightarrow P\ (x1a \wedge x2);\ \bigwedge x1a\ x2.\ P\ x1a \wedge P\ x2 \Longrightarrow P\ (x1a \vee x2);\ \bigwedge x1a\ x2.\ P\ x1a \wedge P\ x2 \Longrightarrow P\ (x1a \rightarrow x2)]\!] \Longrightarrow P\ formula$$

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. Por tanto, para simplificar algunas demostraciones sintácticas más extensas, expondremos la prueba estructurada únicamente para uno de los casos de conectivas binarias.

Llegamos así al primer resultado de este apartado:

Lema 1.4 *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 que tendrá la clave de la prueba del lema. Cabe añadir que la demostración seguirá el esquema inductivo relativo a la estructura de fórmula, y no el que resulta de esta definición.

Definición 1.5 *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 defición *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 equivalente 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, $conjAtoms(Atom\ p) = \{p\} = \{p\} \cup \emptyset$ es finito.

Sea la fórmula $\bot$. Entonces, $conjAtoms(\bot) = \emptyset$ y, por lo tanto, finito.

Sea F una fórmula tal que $conjAtoms(F)$ es finito. Entonces, por definición, $conjAtoms(\neg\ F) = conjAtoms(F)$ y, por hipótesis de inducción, es finito.

Consideremos las fórmulas F y G cuyos conjuntos de átomos $conjAtoms(F)$ y $conjAtoms(G)$ son finitos. Por construcción, $conjAtoms(F{*}G) = conjAtoms(F) \cup conjAtoms(G)$ para cualquier $*$ conectiva binaria. Por lo tanto, por hipótesis de inducción, $conjAtoms(F{*}G)$ es finito.

$\square$

Veamos ahora la prueba detallada en Isabelle del resultado que, aunque es sencillo, nos muestra un ejemplo claro de la estructura inductiva que nos acompañará en las siguientes demostraciones. En este primer lema mostraré con detalle de 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 ⊥)
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 1.6 *El conjunto de subfórmulas de una fórmula F, notada $Subf(F)$, se define recursivamente como:*

- $\{\bot\}$ *si* F *es* $\bot$.
- $\{F\}$ *si* F *es una fórmula atómica.*
- $\{F\} \cup Subf(G)$ *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 (¬ F)    = (¬ F) # subformulae F
| subformulae (F ∧ G)  = (F ∧ G) # subformulae F @ subformulae G
| subformulae (F ∨ G)  = (F ∨ G) # subformulae F @ subformulae G
| subformulae (F → G)  = (F → G) # subformulae F @ subformulae G
```

Observemos que, en la definición anterior, # es el operador que añade un elemento al comienzo de una lista y @ concatena varias listas. Siguiendo con los ejemplos, apliquemos *subformulae* en las distintas fórmulas. En particular, al tratarse de una lista pueden aparecer elementos repetidos como se muestra a continuación.

```
notepad
begin
  fix p :: 'a

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

  have subformulae (¬ (Atom p)) = [¬ (Atom p), Atom p]
```

by *simp*

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

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

have *subformulae* $(Atom\ p \vee Atom\ p) =$
$[Atom\ p \vee Atom\ p,\ Atom\ p,\ Atom\ p]$
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, $Subf(\cdot)$ es equivalente a esta nueva definición. La justificación para este cambio en el tipo reside en las propiedades sobre conjuntos que facilitan las demostraciones de los resultados que mostraremos a continuación, frente a las listas. Algunas de estas ventajas 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* $(Atom\ p \vee Atom\ p) = \{Atom\ p \vee Atom\ p,\ Atom\ p\}$
by *simp*

have *setSubformulae* $((Atom\ p \longrightarrow Atom\ q) \vee Atom\ r) =$
$\{(Atom\ p \longrightarrow Atom\ q) \vee Atom\ r,\ Atom\ p \longrightarrow Atom\ q,\ Atom\ p,\ Atom$
$q,$
$Atom\ r\}$
by *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, vamos a probar que *setSubformulae* es equivalente a *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 (⊥) = {⊥}
    by (*simp only*: *subformulae.simps*(*2*), *simp only*: *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 → F2)
   = {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
proof −
  have setSubformulae (F1 → F2)
        = set ((F1 → F2) # (subformulae F1 @ subformulae F2))
    by (simp only: subformulae.simps(6))
  also have ... = {F1 → F2} ∪ (set (subformulae F1 @ subformulae F2))
    by (simp only: set-insert)
  also have ... = {F1 → F2} ∪ (setSubformulae F1 ∪ setSubformulae F2)
    by (simp only: set-append)
  finally show ?thesis
    by this
qed
```

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

Lema 1.7 $F \in \mathit{Subf}(F)$.

Demostración: Procedamos por inducción sobre la estructura de fórmula probando los correspondientes tipos.

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

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

Por definición del conjunto de subfórmulas de $\mathit{Subf}(\neg\ F)$ se tiene la propiedad para este caso, pues $\mathit{Subf}(\neg\ F) = \{\neg\ F\} \cup \mathit{Subf}(F) \Longrightarrow \neg\ F \in \mathit{Subf}(\neg\ F)$ como queríamos ver.

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

$\square$

Formalicemos ahora el lema con su correspondiente demostración detallada.

lemma *subformulae-self*: $F \in \mathit{setSubformulae}\ F$
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 ⊆ C
    by (rule conjunct1)
qed

lemma subContUnionRev2:
  assumes A ∪ B ⊆ C
  shows   B ⊆ C
proof −
  have A ⊆ C ∧ B ⊆ C
    using assms
    by (simp only: sup.bounded-iff)
  then show B ⊆ C
    by (rule conjunct2)
qed
```

Sus correspondientes demostraciones automáticas se muestran a continuación.

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

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

Veamos ahora los distintos resultados sobre subfórmulas.

Lema 1.8 *Sea F una fórmula proposicional y $conjAtoms(F)$ el conjunto de sus variables proposicionales. Sea A_F el conjunto de las fórmulas atómicas formadas a partir de cada elemento de $conjAtoms(F)$. 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 $Atom\ p$ para p una variable cualquiera. Entonces, $conjAtoms(Atom\ p) = \{p\}$. De este modo, el conjunto $A_{Atom\ p}$ correspondiente será $A_{Atom\ p} = \{Atom\ p\} \subseteq \{Atom\ p\} = Subf(Atom\ p)$ como queríamos demostrar.

Sea la fórmula $\perp$. Como $conjAtoms(\perp) = \emptyset$, es claro que $A_\perp = \emptyset \subseteq Subf(\perp) = \emptyset$.

Sea la fórmula F tal que $A_F \subseteq Subf(F)$. Probemos el resultado para $\neg F$. Por definición tenemos que $conjAtoms(\neg F) = conjAtoms(F)$, 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, $conjAtoms(F*G) = conjAtoms(F) \cup conjAtoms(G)$, luego $A_{F*G} = A_F \cup A_G$. Por tanto, por hipótesis de inducción y definición del conjunto de subfórmulas, se tiene:

$A_{F*G} = A_F \cup A_G \subseteq conjAtoms(F) \cup conjAtoms(G) \subseteq \{F*G\} \cup conjAtoms(F) \cup conjAtoms(G) = conjAtoms(F*G)$ Luego, $A_{F*G} \subseteq conjAtoms(F*G)$ como queríamos demostrar.

$\square$

En Isabelle, se especifica como sigue.

lemma *atoms-are-subformulae*: *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 $Atom\ \text{‘}atoms\ (Atom\ p \vee \bot) = \{Atom\ p\}$
  by *simp*

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

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

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

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

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

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

Una vez hechas las aclaraciones necesarias, comencemos la demostración estructurada. Esta seguirá el esquema inductivo señalado con anterioridad. Debido a la extensión de la prueba demostraremos de manera detallada únicamente el caso de conectiva binaria de la conjunción. El resto son totalmente equivalentes y los dejaré indicados de manera automática. Observemos que los casos básicos de *Atom x* y ⊥ podrían demostrarse de manera directa únicamente mediante simplificación.

```
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:
  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** *auto*
next
  case (*Imp F1 F2*)
  then show *?case* **by** *auto*
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 1.9 *Sea* $G \in \mathit{Subf}(F)$*, entonces el* $\mathit{conjAtoms}(G) \subseteq \mathit{conjAtoms}(F)$*.*

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

Sea *Atom p* una fórmula atómica cualquiera. Si $G \in \mathit{Subf}(\mathit{Atom}\ p)$, como $\mathit{conjAtoms}(\mathit{Atom}\ p) = \{\mathit{Atom}\ p\}$, se tiene $G = \mathit{Atom}\ p$. Por tanto, $\mathit{conjAtoms}(G) = \mathit{conjAtoms}(\mathit{Atom}\ p) \subseteq \mathit{conjAtoms}(\mathit{Atom}\ p)$.

Sea la fórmula $\bot$. Si $G \in \mathit{Subf}(\bot)$, como $\mathit{conjAtoms}(\bot) = \{\bot\}$, se tiene $G = \bot$. Por tanto, $\mathit{conjAtoms}(G) = \mathit{conjAtoms}(\bot) \subseteq \mathit{conjAtoms}(\bot)$.

Sea la fórmula F cualquiera tal que para cualquier subfórmula $G \in \mathit{Subf}(F)$ se verifica $\mathit{conjAtoms}(G) \subseteq \mathit{conjAtoms}(F)$. Supongamos $G' \in \mathit{Subf}(\neg\ F)$ cualquiera, probemos que $\mathit{conjAtoms}(G') \subseteq \mathit{conjAtoms}(\neg\ F)$. Por definición, tenemos que $\mathit{Subf}(\neg\ F) = \{\neg\ F\} \cup \mathit{Subf}(F)$. 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 tiene el resultado. Observando el segundo caso, por hipótesis de inducción, se tiene $\mathit{conjAtoms}(G') \subseteq \mathit{conjAtoms}(F)$. Además, como $\mathit{conjAtoms}(\neg\ F) = \mathit{conjAtoms}(F)$, se obtiene $\mathit{conjAtoms}(G') \subseteq \mathit{conjAtoms}(\neg\ F)$ como queríamos probar.

Sea *F1* fórmula proposicional tal que para cualquier $G \in \mathit{Subf}(F1)$ se tiene $\mathit{conjAtoms}(G) \subseteq \mathit{conjAtoms}(F1)$. Sea también *F2* tal que dada $G \in \mathit{Subf}(F2)$ cualquiera se tiene también $\mathit{conjAtoms}(G) \subseteq \mathit{conjAtoms}(F2)$.

Supongamos $G' \in \mathit{Subf}(F1{*}F2)$ donde $*$ es cualquier conectiva binaria. Vamos a probar que $\mathit{conjAtoms}(G') \subseteq \mathit{conjAtoms}(F1{*}F2)$.

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: $G' \in \mathit{Subf}(F1) \vee G' \in \mathit{Subf}(F2)$. Si $G' \in \mathit{Subf}(F1)$, por hipótesis de inducción se tiene $\mathit{conjAtoms}(G') \subseteq \mathit{conjAtoms}(F1)$. Como $\mathit{conjAtoms}(F1{*}F2) = \mathit{conjAtoms}(F1) \cup \mathit{conjAtoms}(F2)$, 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 *subformula-atoms*: $G \in \mathit{setSubformulae}\ F \Longrightarrow \mathit{atoms}\ G \subseteq \mathit{atoms}\ F$
  oops

Veamos su demostración estructurada. Desarrollaré la disyunción como representante del caso de las conectivas binarias, pues los demás son equivalentes.

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

lemma subformulas-atoms-bot:
  assumes G ∈ setSubformulae ⊥
  shows   atoms G ⊆ atoms ⊥
proof –
  have G ∈ {⊥}
    using assms
    by (simp only: setSubformulae-bot)
  then have G = ⊥
    by (simp only: singletonD)
```

```
  then show ?thesis
    by (simp only: subset-refl)
qed

lemma subformulas-atoms-not:
  assumes G ∈ setSubformulae F ⟹ atoms G ⊆ atoms F
          G ∈ setSubformulae (¬ F)
  shows   atoms G ⊆ atoms (¬ F)
proof −
  have G ∈ {¬ F} ∪ setSubformulae F
    using assms(2)
    by (simp only: setSubformulae-not)
  then have G ∈ {¬ F} ∨ G ∈ setSubformulae F
    by (simp only: Un-iff)
  then show atoms G ⊆ atoms (¬ F)
  proof
    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-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:
  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** *auto*
next
  case (*Or F1 F2*)
  then show *?case* **by** (*simp only*: *subformulas-atoms-or*)
next
  case (*Imp F1 F2*)
  then show *?case* **by** *auto*
qed

Por último, su demostración aplicativa automática.

lemma *subformula-atoms*: $G \in setSubformulae\ F \Longrightarrow atoms\ G \subseteq atoms\ F$
  by (*induction F*) *auto*

CORREGIDO HASTA AQUÍ

2 Glosario de reglas

2.1 Teoría de conjuntos finitos

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.

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

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 defincíón 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.

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

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.

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

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

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

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

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

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

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

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

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

$$A \subseteq A \qquad (\textit{subset-refl})$$

$$\emptyset \subseteq A \qquad (\textit{empty-subsetI})$$

$$\frac{b \in \{a\}}{b = a} \qquad (\textit{singletonD})$$

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

2.4 Lógica de primer orden

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

$$\frac{P \wedge Q}{P} \qquad (\textit{conjunct1})$$

$$\frac{P \wedge Q}{Q} \qquad (\textit{conjunct2})$$

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 4: Añadir el artículo que se usa como base.

Lista de tareas pendientes