Práctica para entregar antes del 20 de Enero[1]

La función que se define a continuación pretende calcular la "mitad entera" de un natural.

Fixpoint *div2* (n : *nat*) : *nat* :=
  match n with
  |0⇒ 0
  | S 0 ⇒ 0
  | S (S p) ⇒ S (*div2* p)
  end.

Especificamos ahora lo que significa que x dividido por 2 (división entera) sea y. En realidad lo que hacemos con el predicado que sigue es "definir" lo que significa la división entera por 2.

Definition *div2P* (x y : *nat*) : Prop := $2 \times y = x \vee 1 + 2 \times y = x$.

Debemos comprobar ahora que nuestra función satisface efectivamente esta especificación.

Para ello necesitamos, a fín de facilitar la automatización de los cálculos, importar la librería *ArithRing*.

Para demostrar el teorema de comprobación haremos un lema previo que se basa en el hecho de que al dividir por 2 un número de la forma x+2 nos da el sucesor de la "mitad entera" de x.

Require Export *ArithRing*.
Lemma *th* : $\forall$ x y : *nat*, *div2P* x y $\rightarrow$ *div2P* (S (S x)) (S y).
en este momento, el primer subobjetivo es $2\times Sy = S(Sx)$.
En esta situación:

```
  x:nat
  y:nat
H0:2*y=x \/ 1+2*y=x
  H:2*y=x
===============
 2*Sy = S(Sx)
```

[1] La entrega consistirá en un fichero .v que debe compilar en coq.

si reescribimos H, al aplicar la táctica *ring* se aplica la libreria *ArithRing* a operaciones con *nat*

Ahora, introducimos un nuevo principio de recursión sobre *nat* que "supondremos" cierto. Por eso en lugar de dar una prueba ponemos *Admitted*. Con ello posponemos la prueba de este resultado pudiendo usarlo en otras pruebas.

Theorem *Esquema* :
$\forall\ P : nat \rightarrow \mathsf{Prop},$
$P\ 0 \rightarrow P\ 1 \rightarrow (\forall\ n : nat,\ P\ n \rightarrow P\ (S\ (S\ n))) \rightarrow \forall\ n : nat,\ P\ n.$

Finalmente comprobemos que *div2 x* cumple la definición dada por el predicado div2P. Con ello tendremos la "completa certeza" de que ello es así

Theorem *compr*:$\forall\ x{:}nat,(div2P\ x\ (div2\ x)).$
Usaremos ahora el resultado *Esquema* haciendo la eliminación de x como *nat* pero, en lugar de usar el esquema de inducción estandar (*nat_ind*), usando justamente este nuevo.

Theorem *div_ent_2* : $\forall\ n : nat,\ \{p : nat \mid div2P\ n\ p\}.$
Hemos construido así el programa *div_ent_2* que tiene un tipo que es de sort Set y por lo tanto, tiene un valor constructivo (encierra un algoritmo) que podremos extraer. Con la construcción

Recursive Extraction div_ent_2.

Si se compleetan las pruebas, se obtiene el fichero ocaml:

```
 type nat =
  | O
  | S of nat

type 'a sig0 = 'a
  (* singleton inductive, whose constructor was exist *)

(** val div2 : nat -> nat **)

let rec div2 = function
  | O -> O
  | S n0 -> (match n0 with
               | O -> O
```

```
               | S p -> S (div2 p))

(** val div_ent_2 : nat -> nat sig0 **)

let div_ent_2 n =
  div2 n
```

que coincide, como se puede ver, con la extracción de la constante div2 directamente. Este es un caso muy sencillo porque en realidad ya teníamos la función programada desde el principio, y la extracción no nos da nada nuevo. Como se verá más adelante, esto no siempre ocurre y de la prueba del resultado constructivo (en este caso *div_ent_2*) se puede encontrar el programa ejecutable sin tenerlo previamente.

Para completar este ejercicio, además de probar el teorema Esquema que ha quedado sin hacer, trate de probar todos los resultados incluyendo los siguientes. Nótese que debe sustituir los *Admitted* por pruebas:

Lemma *aux1* :$\forall$ x y:*nat*, $(2 \times S\ y = S\ (S\ x)) \rightarrow 2 \times y = x$.

Lemma *aux2*: $\forall$ x y:*nat*, $(1 + 2 \times S\ y = S\ (S\ x)) \rightarrow (1{+}2{\times}y{=}x)$.

Hint *Resolve aux1 aux2*.

Lemma *th'* : $\forall\ x\ y$: *nat*, *div2P* $(S\ (S\ x))\ (S\ y) \rightarrow$ *div2P* $x\ y$.

Lemma *evenodd* :
$\forall\ n$: *nat*, $n = 0 \lor n = 1 \lor (\exists\ p : nat,\ n = S\ (S\ p))$.

Theorem *div2_le* : $\forall\ n$: *nat*, *div2* $n \leq n$.