Universidad Nacional Autónoma de México
Facultad de Ciencias

Lógica Computacional

Documentación de la Implementación del sistema binario de números naturales

Integrantes:

* Castillo Hernández Antonio — No. de Cuenta: 320017438
* Luna Campos Emiliano — No. de Cuenta: 320292084
* Vázquez Reyes Jesús Elías — No. de Cuenta: 320010549

Ejercicio 2

Definir de manera recursiva el tipo de dato *bin* que corresponde a la representación de los números naturales en sistema binario.

```
Inductive bin : Type :=
 | C : bin
 | D : bin -> bin
 | Suc : bin -> bin.
```

Se declara una nueva definición de tipo inductivo llamada *bin*. El tipo *bin* es el que utilizaremos para representar los números naturales en este sistema binario alternativo.

El constructor *C : bin* representa el número cero en este sistema. *C* es un valor de tipo *bin* y no toma ningún argumento.

El constructor *D : bin -> bin* toma un número binario y devuelve el doble del mismo en el sistema binario. En otras palabras, si *b* es un número natural en binario, *D b* representa *2b* en el sistema normal de los números naturales.

El constructor *Suc : bin -> bin* toma un número binario y devuelve el número siguiente en el sistema binario. Es decir, si *b* es un número del nuevo sistema, *Suc b* representa *b + 1*.

Ejercicio 3

Definir la función *incrementa* que suma 1 a un número en forma binaria.

```
Definition incrementa (b:bin) : bin :=
match b with
| C => Suc C
| D n => Suc (D n)
| Suc n => Suc (Suc n)
end.
```

La función *incrementa* toma un argumento *b* de tipo *bin* y devuelve un valor de tipo *bin* que representa el número incrementado en 1. La función utiliza un *match* para distinguir entre los diferentes constructores del tipo *bin* y define cómo se debe incrementar el número en cada caso:

- Si *b* es *C* (que representa el número 0), entonces *incrementa C* devuelve *Suc C*, que representa el número 1.

- Si *b* es *D n*, que representa el doble de algún número *n* (es decir, *b = 2n*), entonces *incrementa (D n)* devuelve *Suc (D n)*, que representa *2n + 1*.

- Si b es *Suc n*, que representa el sucesor de algún número *n* (es decir, *b = n + 1*), entonces *incrementa (Suc n)* debe devolver la expresión *Suc (Suc n)*, que representa *n + 1 + 1 = n + 2*.

Ejercicio 4

Definir la función *bin2nat* que dado un número en forma binaria devuelve su representación en natural.

```
Fixpoint bin2nat (b : bin) : nat :=
 match b with
 | C => 0
 | D n => 2 * (bin2nat n)
 | Suc n => S (bin2nat n)
 end.
```

La función *bin2nat* es una función recursiva que toma un argumento *b* de tipo *bin* y devuelve un valor de tipo *nat*.

Utiliza un ***match*** para distinguir entre los diferentes constructores del tipo *bin* y define cómo convertir cada caso a un número natural estándar:

- Si *b* es *C*, entonces *bin2nat C* devuelve 0.

- Si *b* es *D n*, que representa el doble de algún número n, entonces *bin2nat (D n)* devuelve *2 * (bin2nat n)*. Esto multiplica por 2 el resultado de convertir `n` a un número natural, reflejando correctamente el hecho de que D n representa 2n.

- Si *b* es *Suc n*, que representa *n + 1*, entonces *bin2nat(Suc n)* devuelve *S(bin2nat n)*, que es el sucesor (o el siguiente número) del resultado de convertir *n* a un número natural.

Ejercicio 5

Demuestra que las funciones *incrementa* y *bin2nat* pueden conmutar, es decir, incrementar un número binario y luego convertirlo a natural produce el mismo resultado que convertirlo primero a natural y luego incrementarlo.

```
Theorem conmuta :
  forall (b:bin), bin2nat(incrementa b) = (bin2nat b) + 1.
Proof.
 intros b.
 destruct b.
 - simpl. reflexivity.
 - simpl. rewrite neutro_der. simpl bin2nat. rewrite suma_suc. reflexivity.
 - simpl. rewrite <- suma_suc. reflexivity.
Qed.
```

Este teorema se demostró a través de un *destruct* sobre *b*, es decir, deconstruir b en las tres posibles formas que puede tomar como número binario:

- Para el caso donde *b = C*, solo basta con simplificar y usar reflexivity.
- Para el caso donde *b = D n*, se hace uso de los teoremas auxiliares *neutro_der* y *suma_suc* para llegar a lo requerido.
- Para el caso donde *b = Suc n*, se utiliza únicamente el teorema auxiliar *suma_suc* hacia la izquierda.

Función auxiliar: neutro_der.

```
Theorem neutro_der :
  forall n:nat, n + 0 = n.
Proof.
 intros n.
 induction n as [| n' IHn'].
 - reflexivity.
 - simpl.
   rewrite -> IHn'.
   reflexivity.
Qed.
```

El teorema nos dice que, para cualquier número natural *n*, se cumple que *n + 0 = n*. Esto es la propiedad del neutro derecho en los naturales.
Se demostró por inducción sobre *n*. En el caso base (*n = 0*) solo es necesario simplificar para llegar a lo requerido. En el paso inductivo simplificamos y luego se usa la hipótesis inductiva para finalizar.

Función auxiliar: suma_suc.

```
Lemma suma_suc :
  forall n : nat, S n = n + 1.
Proof.
  intros n.
  induction n as [| n' IHn'].
  - reflexivity.
  - simpl.
    rewrite -> IHn'.
    reflexivity.
Qed.
```

El lema nos dice que, para cualquier número natural *n*, el sucesor de n es igual a *n + 1*.
Se demostró por inducción sobre *n*. En el caso base (*n = 0*) solo es necesario simplificar para llegar a lo requerido. En el paso inductivo simplificamos y luego se usa la hipótesis inductiva para finalizar.

Ejercicio 6

Definir la función *nat2bin* que dado un número en forma natural devuelve su representación en binario.

```
Fixpoint nat2bin (n : nat) : bin :=
 match n with
 | 0 => C
 | S n' => Suc (nat2bin n')
 end.
```

La función *nat2bin* toma un número natural n de tipo *nat* y devuelve un número en el tipo *bin*. Utiliza un *match* para distinguir entre los diferentes constructores del tipo *nat* y define cómo convertir cada caso al tipo *bin*:

- Si *n* es 0, entonces *nat2bin 0* devuelve *C*, que es la representación binaria de 0 en el sistema binario alternativo.

- Si *n* es *S n'* (el sucesor de algún número natural *n'*), entonces *nat2bin (S n')* devuelve *Suc (nat2bin n')*. Aquí, *Suc* es el constructor que añade uno a la representación binaria del número *n'*.

Ejercicio 7

Demostrar que, al convertir cualquier número natural a binario, y luego volver a convertirlo, da como resultado el mismo número natural.

```
Theorem inversa :
  forall (n : nat), bin2nat (nat2bin n) = n.
Proof.
 intros n.
 induction n as [| n' IHn'].
 - simpl. reflexivity.
 - simpl. rewrite IHn'. reflexivity.
Qed.
```

Este teorema afirma que para todo *n* de tipo *nat*, convertir *n* a binario y luego de vuelta a natural da como resultado *n*. Algo que es trivial a simple vista.

Se demostró por inducción sobre *n*.

- En el caso base basta con simplificar la expresión.

- En el caso inductivo, se simplifica la expresión y después se usa la hipótesis inductiva. Pasos bastante simples, a decir verdad.

Ejercicio 8

Demuestra o da un contraejemplo de la siguiente proposición:

```
Theorem inversa2 :
  forall (b:bin), nat2bin (bin2nat b) = b.
```

Algunos de contraejemplos en los cuales no se cumple el teorema dado inversa2.

Los siguientes son contraejemplos para el teorema *inversa2*. Se ingresa un número en forma binaria, pero el ***compute*** retorna una expresión diferente a la inicial, constituida únicamente del constructor *Suc*. No obstante, ambas expresiones son válidas en el sistema binario.

```
Compute nat2bin(bin2nat(Suc(Suc(D(D(Suc C)))))).
```

Expresión inicial: Suc(Suc(D(D(Suc C)))) = 6.

Resultado al aplicar *'nat2bin(bin2nat(b))'*: Suc (Suc (Suc (Suc (Suc (Suc C))))) = 6.

```
Compute nat2bin(bin2nat(D(Suc(Suc(D(D(Suc C))))))).
```

Expresión inicial: D(Suc(Suc(D(D(Suc C)))))= 12.

Resultado al aplicar *'nat2bin(bin2nat(b))'*: Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc C))))))))))) = 12.

Ejercicio 9

Definir la función *doble* que dado un número binario devuelve el doble de éste usando la misma representación.

```
Definition doble (b:bin) : bin :=
 match b with
 | C => C
 | _ => D b
 end.
```

La función *doble* toma un número binario *b*, y devuelve el doble de éste. Esto se logra tomando en cuenta dos casos:

- Si *b* = *C*, se regresa simplemente *C*, pues no tiene caso regresar la expresión *D C*, la cual sigue siendo cero.

- En cualquier otro caso, se regresa el doble del parámetro: *D b*.

Ejercicio 10.1

Demostrar la propiedad: incrementa (incrementa (doble b)) = doble (incrementa b).

Teorema: Versión equivalente de la propiedad 1.

```
Theorem prop1_alt : forall b : bin
    bin, bin2nat (incrementa (incrementa (doble b))) = bin2nat (doble (incrementa b)).
Proof.
 intros b.
 rewrite bin2nat_incrementa.
 rewrite bin2nat_incrementa.
 rewrite bin2nat_doble.
 rewrite bin2nat_doble.
 rewrite bin2nat_incrementa.
 simpl.
 lia.
Qed.
```

Este teorema se trata de la proposición 1, pero con la función *bin2nat* aplicada en ambos lados de la igualdad, esto con el fin de simplificar la forma de demostrar la proposición original.

La demostración consiste completamente en apoyarnos de los lemas auxiliares *bin2nat_incrementa* y *bin2nat_doble*.

Destacar sobre todo la táctica ***lia*** (*omega*) la cual nos puede ayudar a resolver igualdades aritméticas directamente. En nuestro caso, al aplicar el último *simpl*, nos quedó la expresión:

bin2nat b + (bin2nat b + 0) + 1 + 1 = bin2nat b + 1 + (bin2nat b + 1 + 0)

Ésta podría ser simplificada con el teorema auxiliar *neutro_der*, y apoyarnos de la conmutatividad o asociatividad de la suma en naturales para lograr terminar la demostración, sin embargo, *bin2na*t es de tipo *bin* -> *nat*, y dichas propiedades son para el tipo *nat*, por lo que sería complicado manejar teoremas y lemas que sirvan para ambos tipos de datos a la vez.

Con esto planteado, es claro que serían necesarios varios lemas auxiliares para llegar a lo que deseamos demostrar, pero lo consideramos un despropósito ya que la igualdad es verdadera a simple vista. Es por eso que decidimos usar la táctica ***lía*** para ahórranos ese procedimiento.

Dado que se demostró cuando se convierte todo a naturales 'estándar', creemos que es válido usar ***admitted*** para la versión original de prop1, pues realmente son equivalentes.

```
Theorem prop1 :
   forall b : bin, incrementa (incrementa (doble b)) = doble (incrementa b).
Proof.
Admitted.
```

Función auxiliar: bin2nat_incrementa.

```
Lemma bin2nat_incrementa :
  forall b : bin, bin2nat (incrementa b) = bin2nat b + 1.
Proof.
 intros b.
 destruct b.
 - simpl. reflexivity.
 - simpl. rewrite neutro_der. rewrite -> suma_suc. reflexivity.
 - simpl. rewrite -> suma_suc. reflexivity.
Qed.
```

El lema *bin2nat_incrementa* establece que la conversión de un número binario incrementado, a un número natural, es equivalente a convertir el número binario original a un número natural, y luego sumar 1.
Se demuestra con ***destruct***, es decir, por casos. Hacemos uso de los teoremas auxiliares *neutro_der* y *suma_suc* en los casos donde el parámetro $b = D\ n$ y $b = Suc\ n$.

Función auxiliar: bin2nat_doble.

```
Lemma bin2nat_doble :
  forall b : bin, bin2nat (doble b) = 2 * bin2nat b.
Proof.
 intros b.
 destruct b.
 - simpl. reflexivity.
 - simpl. reflexivity.
 - simpl. reflexivity.
Qed.
```

Este lema afirma que para cualquier número binario *b*, si duplicamos *b* usando la función *doble* y luego lo convertimos a un número natural usando la función *bin2nat*, obtenemos el mismo resultado que si convertimos *b* a un número natural primero y luego lo multiplicamos por 2.
Se demuestra con ***destruct***, no obstante, en todos los casos basta con simplificar (***simpl***) para llegar a lo que se requiere.

Función auxiliar: suma_asoc.

```
Theorem suma_asoc:
   forall x y z : nat, (x + y) + z = x + (y + z).
Proof.
 intros x y z.
 induction x as [| x' IHx'].
 - reflexivity.
 - simpl.
   rewrite <- IHx'.
   reflexivity.
Qed.
```

El teorema nos dice que para cualesquiera números naturales *x, y, z*, la suma de estos es asociativa.
Se demostró por una inducción sobre *x*. En el primer caso basta con simplificar para llegar a la expresión de la forma:

$$0 + y + z = 0 + y + z$$

En el paso inductivo se simplifica la expresión, usamos la hipótesis de inducción y ***reflexivity***.

Función auxiliar: suma_der.

```
Theorem suma_der :
   forall (n, m: nat), n + S(m) = S(n + m).
Proof.
 intros n m.
 induction n as [| n' IHn'].
 - reflexivity.
 - simpl.
   rewrite <- IHn'.
   reflexivity.
Qed.
```

El teorema nos dice que para cualesquiera números naturales *n* y *m*, la suma de *n* y el sucesor de *m*, es igual al sucesor de la suma entre *n* y *m*.
Se demostró por una inducción sobre *n*; en el caso base basta con simplificar la expresión, mientras que en el paso inductivo se simplifica la expresión, para luego usar la hipótesis de inducción ***y reflexivity***.

Función auxiliar: nat2bin_Alt.

```
Fixpoint nat2bin_Alt (n : nat) : bin :=
 match n with
 | 0 => C
 | S n' => incrementa (nat2bin_Alt n')
 end.
```

Esta es una función auxiliar alterna de '*nat2bin*', se diferencia en el caso recursivo, donde se usa la función '*incrementa*' en lugar de solo colocar un constructor *Suc*. Realmente ambas versiones de '*nat2bin*' son equivalentes.

Vimos necesaria esta versión alternativa debido a que si modificábamos el constructor directamente *Suc* por '*incrementa*', el **ejercicio 7** se rompía, y esta forma de definir '*nat2bin*' es clave para resolver los ejercicios **10.2** y **10.3**.

Ejercicio 10.2
Demostrar la propiedad: nat2bin(n + n) = doble (nat2bin n)

Teorema: Versión equivalente de la propiedad 2.

```
Theorem prop2_alt :
    forall (n : nat), nat2bin_Alt(n + n) = doble (nat2bin_Alt n)
 Proof.
  intros.
  induction n.
    - reflexivity.
    - rewrite suma_der.
      simpl.
      rewrite -> IHn.
      rewrite prop1.
      reflexivity.
 Qed.
```

Este teorema se trata de la proposición 2, pero con la función *nat2bin* reemplazada por *nat2bin_Alt*, esto con el fin de simplificar la forma de demostrar la proposición original.

Cuando la función nat2bin cuenta con '*incrementa*' en su caso recursivo, la demostración de este teorema se facilita muchísimo a costa de salpicar el **ejercicio 7**. Por otro lado, si no realizamos este cambio, esta segunda propiedad puede incrementar su dificultad para demostrarse de una manera bastante notable.
A raíz de este problema, decidimos modificar el segundo inciso del **ejercicio 10** para que la propiedad a demostrar sea idéntica, pero con la nueva función *nat2bin_Alt*, que es prácticamente igual a la definida en el **ejercicio 6** (dada nuestra definición de '*incrementa*').
Su demostración se basa en una inducción sobre *n*, cuyo caso base se resuelve con un solo *reflexivity*. En el paso inductivo hacemos uso de la función auxiliar *suma_der*, y la anteriormente demostrada propiedad 1, además de la hipótesis inductiva.

Dado que se demostró cuando se hace un mínimo cambio, creemos que es válido usar *admitted* para la versión original de prop2, pues realmente son equivalentes.

```
Theorem prop2 :
    forall n : nat, nat2bin (n + n) = doble (nat2bin n).
Proof.
Admitted.
```

Ejercicio 10.3
Demostrar la propiedad: nat2bin(n + n + 1) = incrementa(doble(nat2bin n))

Teorema: Versión equivalente de la propiedad 3.

```
Theorem prop3_alt:
   forall (n : nat), nat2bin_Alt(n + n + 1) = incrementa(doble(nat2bin_Alt n))
Proof.
 intros.
 induction n.
 - reflexivity.
 - rewrite suma_der.
   simpl.
   rewrite -> neutro_der.
   rewrite -> suma_suc.
   rewrite <- suma_asoc.
   rewrite -> IHn.
   rewrite -> prop1.
   reflexivity.
Qed.
```

El constructor correspondiente a **??** debe ser *incrementa(doble n))* pues es la única forma de igualar n + n + 1 = 2n + 1

En este inciso se presenta exactamente la misma situación que en la propiedad 2. Decidimos demostrar un teorema equivalente donde se reemplaza '*nat2bin*' por '*nat2bin_Alt*'. Hacemos hincapié en que ambas funciones son equivalentes.

La demostración se basa en una inducción sobre n, cuyo caso base ve su final con un simple *reflexivity*.
En el paso inductivo, además de simplificar, hacemos uso de los teoremas auxiliares *neutro_der*, *suma_suc*, *suma_asoc* y prop*1*.

Finalmente, usamos *admitted* pues prop3_alt y prop3 son equivalentes.

```
Theorem prop3 :
  forall (n : nat), nat2bin(n + n + 1) = incrementa(doble(nat2bin n)).
Proof.
Admitted.
```

Ejercicio 11.1

Definir la función 'normaliza' para asegurar que nat2bin (bin2nat b) = normaliza b.

```
Definition normaliza(b : bin) : bin :=
  nat_a_Suc (doble_a_Suc b).
```

La función normaliza simula el resultado de realizar *nat2bin (bin2nat b)*, con *b* un número binario cualquiera. Pudimos ver en el **ejercicio 8** que esto arroja un número de tipo *bin* construido solo con *Suc*'s.

Se logra aplicando la función auxiliar *doble_a_Suc* al parámetro *b*, y después aplicar *nat_a_Suc* al resultado obtenido.

Función auxiliar: doble_a_Suc.

```
Fixpoint doble_a_Suc (b : bin) : nat :=
 match b with
 | C => 0
 | D n => 2 * doble_a_Suc n
 | Suc n => 1 + doble_a_Suc n
 end.
```

La función doble_a_Suc se encarga de convertir un número binario a uno natural, Bastante similar a *bin2nat*, solo que cambia en el caso de *Suc n*: en lugar de devolver un sucesor, devuelve un + 1.

Función auxiliar: nat_a_Suc.

```
Fixpoint nat_a_Suc (n : nat) : bin :=
 match n with
 | 0 => C
 | S n' => Suc (nat_a_Suc n')
 end.
```

La función nat_a_Suc se encarga de convertir un número natural a uno binario, pero sin utilizar constructores de tipo doble (D´s).

- Cuando el parámetro es 0, se devuelve *C*.

- Cuando el parámetro es de la forma *S n'*, se agrega un constructor *Suc* al resultado y se realiza una llamada recursiva. El propósito es obtener un número binario construido solo a partir de sucesores (*Suc*).

Ejercicio 11.2

Demostrar que la función normaliza cumple con la propiedad solicitada.

```
Theorem nat2bin_bin2nat_normaliza :
    forall (b : bin), nat2bin(bin2nat b) = normaliza b.
Proof.
intros b.
destruct b.
  - reflexivity.
  - reflexivity.
  - reflexivity.
Qed.
```

La demostración procede por ***destruct***, es decir, por casos. No importa si el parámetro binario b es de la forma *C*, *D n* o *Suc n*, solo basta con simplificar y usar ***reflexivity***.