Segunda parte de la primera práctica

Fecha de entrega: 22 de Diciembre 2006

1 Introducción

Una *especificación* es una descripción precisa de las requerimientos del cliente escrita en una determinada notación (lenguaje). Se dice que la notación es formal cuando posee una sintáxis *formal* así como una semántica formal (e.g. Coq). Si tiene una sintáxis formal sólamente, se dice que es un lenguaje *semi–formal* (e.g. UML).

Los requerimientos pueden ser funcionales, de eficiencia o de implementación. En este ejemplo sencillo solo trataremos con requerimientos funcionales.

Una especificación viene a ser una suerte de *contrato* entre el cliente y el especificador. Por lo tanto el cliente debe ser capaz de entender la especificación en orden a *validarla* pero normalmente los clientes no estarán versados en lenguajes especializados para entender la especificación. Existen varios métodos para salvar esta dificultad: trasladarla, con las mínimas imprecisiones posibles, al lenguaje natural del cliente o, si es ejecutable probarla en varios escenarios con el cliente. El uso de ejemplos y contraejemplos es una buena técnica para asegurar que el cliente y el especificador se entienden el uno al otro.

En la medida que la especificación es también un contrato entre el especificador y el implementador, se supone que éste comprende bien el lenguaje especializado de la especificación. También resulta esencial, que el implementador disponga de la descripción en lenguaje natural porque ello le ayudará a trasladar las distintas ideas a conceptos propios del dominio de la aplicación.

Asegurar al máximo que la especificación coincide con las necesidades del cliente es un aspecto fundamental. Claro que esta coincidencia no se puede demostrar completamente y por eso este proceso, llamado validación, consiste precisamente en probar *propiedades* sobre la especificación que responden, casi siempre, a precisiones o demandas del cliente. Cuantas más propiedades solicitadas podamos probar a partir de la especificación más confianza tendremos en la coincidencia de la misma con los requerimientos expuestos.

Finalmente, debe ser posible probar que la implementación satisface la especificación. En este sentido conviene utilizar métodos que permitan, como hace Coq, bien extraer automáticamente el código directamente de la especificación o bien demostrar un teorema que pruebe dicha satisfacción.

Desafortunadamente no siempre es posible demostrar que un sistema completo satisface su especificación y para ello en algunos casos se recurre a otros métodos semiformales como el *testing*, no para probar pero sí para ganar confianza en que la implementación se adecúa bastante a la especificación.

2 Guía de teléfonos electrónica

Supongamos que deseamos construir una especificación formal en Coq de una guía de teléfonos electrónica descrita por el cliente, en un primer momento, por los siguientes requerimientos informales[1]:

- la guía debe almacenar los números de teléfono de una ciudad
- Dado un nombre, debe ser posible encontrar un número de teléfono asociado
- Debe ser posible añadir y borrar entradas en la guía

Vemos que hay tres tipos de entidades que se mencionan: guías, números de teléfonos y nombres; una guía define una asociación entre nombres y números de teléfono. Se necesitan tres operaciones, que podemos llamar `EncTel` (encontrar teléfono), `AddTel` (añadir teléfono) y `BorrTel` (borrar teléfono). `EncTel` toma una guía y un nombre y devuelve el número de teléfono asociado a ese nombre en esa guía. La funcionalidad exacta de las otras dos es menos clara, por lo tanto debemos tomar algunas decisiones de diseño. El cliente decide que `AddTel` tomará una guía, un nombre y un número de teléfono y añadirá la asociación de ese nombre y número a dicha guía. Y `BorrTel` debería tomar una guía y un nombre y borrar la asociación de ese nombre y su teléfono de esa guía (si tal asociación existe)

El siguiente paso es decidir cómo representar estas entidades y operaciones en Coq. Si estuvieramos pogramando tendríamos que escoger alguna representación específica para los números de teléfono y para los nombres (e.g. cadenas ascci o otras más estructuradas como registros conteniendo el código de área etc.) y tendríamos que tomar algunas decisiones de diseño en este momento. Pero para crear la especificación, todo lo que necesitamos es que números de teléfono y nombres sean tipos distinguibles y saber que la igualdad entre sus elementos es decidible:

```
Section guia1.
Require Export ite.
Variable N:Set.
Variable T:Set.
Axiom eq_dec_N : ∀(x1 x2:N),{x1=x2}+{~x1=x2}.
Axiom eq_dec_T:∀(t1 t2:T),{t1=t2}+{~t1=t2}.
```

A continuación hay que decidir cómo vamos a representar las guías. Hay varias posibilidades. La primera podría ser que **una guía es simplemente una función entre nombres y teléfonos**:

```
Definition G:=N → T.
```

Esto tiene ya una consecuencia: como una guía solo registra los nombres que tienen teléfono, debemos representar, por ejemplo, mediante la asociación con un teléfono ficticio los nombres que no lo tienen:

[1] Ejemplo sacado de *A Tutorial Introduction to PVS* de Hudy Crow y otros

Variable $tf0{:}T$.

y así podemos definir la guía vacía:

Definition $guiavacia{:=}fun(nm{:}N) \Rightarrow tf0$.

Podemos definir ya la primera funcionalidad:

Definition $EncTel{:=}fun(g{:}G)\ (nm{:}N) \Rightarrow (g\ nm)$.

Vamos a definir `AddTel0` como una primera aproximación a `AddTel` veamos como podemos añadir a una guía un nombre y un teléfono. Después trataremos de comprobar que representa realmente esta idea mediante una serie de lemas:

Definition $AddTel0{:=}fun(g{:}G)(nm{:}N)(tf{:}T)(a{:}N) \Rightarrow if\ (eq\_dec\_N\ a\ nm)\ then\ tf$
$else\ (g\ a)$.

Lemma $proba{:}\forall(g{:}G)(nm{:}N)(tf{:}T),(AddTel0\ g\ nm\ tf\ nm){=}tf$.
Proof.
$intros\ g\ nm\ tf$.
$unfold\ AddTel0$.
$case\ (eq\_dec\_N\ nm\ nm)$; $auto$.
$intros$.
$elim\ n$;$auto$.
Qed.

Lemma $proba1{:}\forall(g{:}G)(nm\ a{:}N)(tf{:}T),(a{=}nm) \rightarrow (AddTel0\ g\ nm\ tf\ a){=}tf$.
Proof.
$intros\ g\ nm\ a\ tf\ H$.
$rewrite\ H$; $apply\ proba$.
Qed.

Lemma $proba2{:}\forall(g{:}G)(nm\ a{:}N)(tf{:}T),(\tilde{}a{=}nm) \rightarrow (AddTel0\ g\ nm\ tf\ a){=}(g\ a)$.
Proof.
$Admitted$.
Hint $Resolve\ proba\ proba1\ proba2$.

El cliente demanda ahora: Si añadimos un nombre `nm` con un teléfono `tfp` a una guía y después buscamos ese nombre en dicha guía deberíamos de obtener `tp`:

Lemma $EncAdd0{:}\forall(g{:}G)(nm{:}N)(tfp{:}T)$,
$(EncTel\ (AddTel0\ g\ nm\ tfp)\ nm){=}\ tfp$.
Proof.
$intros$.
$unfold\ EncTel$.
$unfold\ AddTel0$.
$case\ (eq\_dec\_N\ nm\ nm)$.

```
auto.
intros.
elim n;auto.
Qed.
```

y, efectivamente se cumple.

Definamos ahora la funcionalidad de borrar una entrada en la guía:

```
Definition BorrTel:G → N → G:=
fun (g:G) (nm:N)(x:N)=>if (eq_dec_N nm x)
then tf0 else (g x).

Lemma BorrTel1:∀ (g:G) (nm:N),(BorrTel g nm nm)=tf0.
intros.
unfold BorrTel.
rewrite ite_l;auto.
Qed.

Lemma BorrTel2:∀ (g:G)(nm x:N),~x=nm → (BorrTel g nm x)=g x.
intros.
unfold BorrTel.
rewrite ite_r;auto.
Qed.
```

Podemos ahora comprobar nuestro entendimiento de esta especificación chequeando la intuición de que añadiendo un nombre y teléfono a una guía y después borrarla debe dejar la guía como estaba.

Lemma BorAdd0:forall(g:G)(nm:N)(tfp:T)(x:N), (BorrTel (AddTel0 g nm tfp) nm x)= (g x).

pero esto no es cierto como se ve en el contraejemplo siguiente:

```
Section contraejemplo1.
Variable nombre:N.
Variable tel1:T.
Variable nig:~(tel1=tf0).
Let g:=(AddTel0 guiavacia nombre tel1).
Lemma no: ~(BorrTel (AddTel0 g nombre tel1) nombre) nombre=(g nombre).
rewrite BorrTel1.
unfold g.
rewrite proba1;auto.
Qed.
End contraejemplo1.
```

Así pues tenemos un fallo en la especificación.

Podemos modificar esta exigencia y, llamando:

```
Definition Desconocido:=fun(g:G)(nm:N) ⇒ (g nm)=tf0.
```

planteamos el siguiente lema (interpretar y probar):

```
Lemma BorAdd0:∀(g:G)(nm:N)(tfp:T)(x:N),
(Desconocido g nm) →
(BorrTel (AddTel0 g nm tfp) nm x)=(g x).
intros.
unfold BorrTel.
unfold Desconocido in H.
Admitted.
```

Compruebe si es cierta la siguiente propiedad que nos pide el cliente: añadir un nombre y después borrarlo es lo mismo que simplemente borrarlo.

```
Lemma BorAddBor:∀(g:G)(nm:N)(tfp:T)(x:N),
(BorrTel (AddTel0 g nm tfp) nm x)=(BorrTel g nm x).
intros.
unfold BorrTel.
Admitted.
```

Compruebe si es cierto lo siguiente: después de añadir un nombre a una guía ese nombre ya no es desconocido.

3 Mejorando la especificación

Tratemos de mejorar nuestra especificación para superar los dos problemas que hemos detectado. En primer lugar `AddTel` tiene el efecto no deseado de cambiar el teléfono de un nombre cuando se añade éste nombre a guía donde ya figuraba. Y segundo, no evita que podamos añadir un nombre con el teléfono falso `tf0`.

Podemos evitar el segundo problema representando los teléfonos normales (distintos del `tf0` con el tipo :

```
Definition TN:={tf:T | ~(tf=tf0)}.
```

al que añadimos las dos conocidas proyecciones:

```
Definition p1:=fun(tfp:TN)=>let (tf,_):=tfp in tf.
Definition p2:=fun(tfp:TN)=> let (_,p)
                  return (~((p1 tfp)=tf0)) :=tfp in p.
```

y definimos la funcionalidad de añadir teléfono sólamente para los normales y evitamos cambiar el teléfono permitiendo sólamente añadir nombres que no figuraran ya con uno real:

```
Definition AddTel:G→N→TN→N→T :=
fun(g:G)(nm:N)(tfp:TN) ⇒ if (eq_dec_T (g nm) tf0)
                  then (AddTel0 g nm (p1 tfp))
                  else g.
```

Interpretar y completar las pruebas de las siguientes propiedades:

```
Lemma proban:∀(g:G)(nm:N)(tf:TN),(Desconocido g nm) →
(AddTel g nm tf nm)=(p1 tf).
Admitted.

Lemma proban_con:∀(g:G)(nm:N)(tf:TN),~(Desconocido g nm) →
(AddTel g nm tf nm)=g nm.
Proof.
intros.
unfold AddTel.
case (eq_dec_T (g nm) tf0); intros; auto.
Qed.

Lemma proba1n:∀(g:G)(nm a:N)(tf:TN),(Desconocido g nm) →
(a=nm) → (AddTel g nm tf a)=(p1 tf).
Proof.
intros g nm a tf d H.
rewrite H; apply proban;auto.
Qed.

Lemma proba2n:∀(g:G)(nm a:N)(tf:TN),
(~a=nm)->(AddTel g nm tf a)=(g a).
Proof.
intros.
unfold AddTel.
case (eq_dec_T (g nm) tf0); intros; auto.
Qed.
```

y construimos una nueva funcionalidad que hace precisamente el cambio del teléfono de un nombre sólamente si ese nombre tiene ya un número normal. Interpretar y probar:

```
Definition CambiarTel:G → N → TN → N → T :=
fun(g:G)(nm:N)(tfp:TN) ⇒ if (eq_dec_T (g nm) tf0)
                         then g
                         else (AddTel0 g nm (p1 tfp)).


Lemma EncAdd:∀(g:G)(nm:N)(tfp:TN),(Desconocido g nm)
  → (EncTel (AddTel g nm tfp) nm)= (p1 tfp).
Proof.
intros.
simpl.
unfold EncTel.
unfold AddTel.
Admitted.

Lemma BorAdd:∀(g:G)(nm:N)(tfp:TN)(x:N),(Desconocido g nm) →
```

```
(BorrTel (AddTel g nm tfp) nm x)=(g x).
intros.
case (eq_dec_N nm x).
intro e.
rewrite ← e.
rewrite BorrTel1.
auto.

Admitted.


Lemma camtelx:∀(g:G)(nm:N)(tfp:TN)(x:N),(~x=nm) →
(CambiarTel g nm tfp x)=(g x).
Proof.
intros.
unfold CambiarTel.
case (eq_dec_T (g nm) tf0); intros; auto.
Qed.

Lemma EncCam:∀(g:G)(nm:N)(tfp:TN),~(Desconocido g nm)
  → (EncTel (CambiarTel g nm tfp) nm)= (p1 tfp).
Admitted.

Lemma MiCam1:∀(g:G)(nm:N)(tfp:TN),~(g nm)=tf0 →
                ( CambiarTel g nm tfp nm)= (p1 tfp).
Proof.
intros.
unfold CambiarTel.
case (eq_dec_T (g nm) tf0); intros; auto.
elim (H e).
Qed.

Lemma MiCam2:∀(g:G)(nm:N)(tfp:TN),(g nm)=tf0 →
                ( CambiarTel g nm tfp nm)= tf0.
Proof.
intros.
unfold CambiarTel.
case (eq_dec_T (g nm) tf0); intros; auto.
elim (n H).
Qed.

Lemma ConAdd:∀(g:G)(nm:N)(tfp:TN)(x:N),
                    ~(Desconocido (AddTel g nm tfp) nm).
Proof.
intros.
```

```
unfold Desconocido.
unfold AddTel.
case (eq_dec_T (g nm) tf0).
intros.
unfold AddTel0.
case (eq_dec_N nm nm).
intros.
exact (p2 tfp).
destruct 1;auto.
auto.
Qed.

Lemma adtelcon:∀(g:G)(nm:N)(tfp1:TN),~(AddTel g nm tfp1 nm)=tf0.
Proof.
(red; intros).
(apply (ConAdd g nm tfp1 nm); auto).
Qed.

Lemma AddCam: ∀(g:G)(nm:N)(tfp1 tfp2:TN)(x:N),
    (CambiarTel (AddTel g nm tfp1) nm tfp2 x)=
    (AddTel (CambiarTel g nm tfp2) nm tfp2 x).
Admitted.

End guia1.
```

4 Nueva representación

Se trata ahora de seguir un nuevo requisito pedido por el cliente que nos dice que deberíamos de contemplar la posibilidad de que un nombre tuviera varios teléfonos. Ello nos lleva a representar una guía como una función de nombres en listas de teléfonos.

Esto tiene además la ventaja de que nos permite prescindir del teléfono virtual.

Desarrollar una especificación en la misma línea que la anterior.

5 Representación con un tipo inductivo

Finalmente, tratar de definir el tipo guía como un tipo inductivo y definir funcionalidades y expresar y probar propiedades análogas a las anteriores.