UNR - FCEIA

LCC - Ingeniería de Software II

Trabajo Práctico Testing
Corrección : Fastest

Vallejos Vigier, Franco

V-2952/1

Febrero, 2023

Profesor:
Cristiá, Maximiliano

Introducción al Problema

De forma breve, el problema a representar es un gestor de mails sencillo. Cada usuario tiene lugar en las bandejas o buzones de este gestor el cual brinda operaciones principales como el envío, recepción y lectura de mensajes; como anexas el agregado de usuarios al gestor y el vaciado de los dichos buzones para un usuario específico. Todo integrante del gestor de mails tiene tres casillas de mensajes, enumerándolas: la casilla de nuevos, la casilla de leídos y la casilla de recibidos.

Para aumentar la dificultad, queremos que las bandejas de mails de cada usuario sean acotadas en cuanto a la capacidad de los mensajes(axiomáticamente).

Índice general

Capítulo 1

Explicación del Programa

1.1. Requerimientos del Problema

A continuación se describen los requerimientos funcionales de nuesto *Gestor de Mails* :

- Agregar un usuario al gestor,
- Mandar/Recibir mensajes entre usuarios,
- Leer y obtener un mensaje si hay alguno en la casilla de nuevo,
- Permitir a los usuarios borrar a elección sus casillas de mensajes,
- Las casillas de cada usuario pueden contener 2 mensajes como máximo
- Cuando se lee un mensaje nuevo, este pasa a la bandeja de leídos para un usuario cualquiera

La cota de la cantidad de mensajes fue elegida de forma arbitraria, pero puede ser modificada fácilmente en el axioma dónde la defino.

1.2. Designaciones del Problema

u es un usuario $\approx u \in USER$

m es un mensaje $\approx m \in MENSAJE$

Conjunto de todas las casillas de mails nuevos del gestor $\approx$ *UsersNewMails*

Casillas de nuevos para el usuario $u \approx$ *UsersNewMails* u

Conjunto de todas las casillas de mails leídos del gestor $\approx$ *UsersReadMails*

Casillas de leídos para el usuario $u \approx$ *UsersReadMails* u

Conjunto de todas las casillas de mails enviados del gestor $\approx$ *UsersSendMails*

Casillas de enviados para el usuario $u \approx$ *UsersSendMails* u

1.3. Especificación Z

Entrando ya a la especificación, primeramente recurrimos a los tipo libres en Z, hemos declarado los siguientes:

$$[USER, MENSAJE]$$

$$MSJLIST == \mathbb{P}(USER \times MENSAJE)$$

$$CASILLA == USER \pfun MSJLIST$$

Estos tipos declarados son expresivos en si mismos. Sabemos que los mensajes pueden tener múltiples representaciones, las casillas de un usuario u esta formada por un conjunto de tuplas ($user\ x\ mensaje$) por lo tanto sabemos quién le mandó un *mensaje* al usuario u, de quién el usuario u está leyendo un *mensaje* y quién le mandó un *mensaje* al usuario u

Acompañamos con una definición axiomática sobre la cota del tamaño de una casilla de mails para un usuario u

$$capMaxima : \mathbb{N}$$
$$\hline$$
$$capMaxima = 2$$

Definimos nuestro *Gestor de Mails* y su estado inicial

GestorMails

$$UsersNewMails : CASILLA$$
$$UsersReadMails : CASILLA$$
$$UsersSendMails : CASILLA$$

GestorMailsInit

$$GestorMails$$
$$\hline$$
$$UsersNewMails = \emptyset$$
$$UsersReadMails = \emptyset$$
$$UsersSendMails = \emptyset$$

Ahora hablemos de las invariantes planteadas. La primera, obviamente, es la cota del tamaño de las casillas por la cual tenemos que hacer uso del cuantificador universal porque tenemos que constatar que todo usuario que esté en las *casillas* tenga asociado una *box de mensajes* acotada. La segunda invariante es que todo usuario registrado en el gestor, tiene las tres casillas para operar.

GestorMailsInv

$GestorMails$

$\text{dom}(UsersNewMails) = \text{dom}(UsersReadMails)$
$\text{dom}(UsersReadMails) = \text{dom}(UsersSendMails)$
$\forall x : USER \mid x \in \text{dom}\ UsersNewMails \bullet \#(UsersNewMails(x)) \leq capMaxima$
$\forall x : USER \mid x \in \text{dom}\ UsersReadMails \bullet \#(UsersReadMails(x)) \leq capMaxima$
$\forall x : USER \mid x \in \text{dom}\ UsersSendMails \bullet \#(UsersSendMails(x)) \leq capMaxima$

Ahora entramos a definir las operaciones de nuestro problema, primeramente *SendMsg*

SendMsgOk

$\Delta GestorMails$
$to? : USER$
$from? : USER$
$body? : MENSAJE$

$to? \in \text{dom}(UsersNewMails)$
$from? \in \text{dom}(UsersSendMails)$
$(\#(UsersNewMails(to?))) + 1 \leq capMaxima$
$(\#(UsersSendMails(from?))) + 1 \leq capMaxima$
$UsersSendMails' = UsersSendMails \oplus \{from? \mapsto (UsersSendMails(from?)) \cup \{(to?, body?)\}\}$
$UsersNewMails' = UsersNewMails \oplus \{to? \mapsto (UsersNewMails(to?)) \cup \{(from?, body?)\}\}$
$UsersReadMails' = UsersReadMails$

EmisorUnreachable

$\Xi GestorMails$
$from? : USER$

$from? \notin \text{dom}(UsersSendMails)$

DestinoUnreachable

$\Xi GestorMails$
$to? : USER$

$to? \notin \text{dom}(UsersNewMails)$

FullSendBox

$$\Xi GestorMails$$
$$from? : USER$$
$$\neg\ (((\#(UsersSendMails(from?))) + 1) \leq capMaxima)$$

FullNewBox

$$\Xi GestorMails$$
$$to? : USER$$
$$\neg\ (((\#(UsersNewMails(to?))) + 1) \leq capMaxima)$$

$$SendMgsNoOk \mathrel{\widehat=} EmisorUnreachable \lor DestinoUnreachable \lor FullSendBox \lor FullNewBox$$

$$SendMsg \mathrel{\widehat=} SendMsgOk \lor SendMgsNoOk$$

Definimos la operación *ReadMsg*, recordando que si leemos un mensaje tenemos que sacarlo del la casilla de nuevos y pasarlo a la de leídos, para ello usamos la diferencia planteada en el apunte:

ReadMsgOk

$$\Delta GestorMails$$
$$lector? : USER$$
$$msg! : MENSAJE$$
$$lector? \in \mathrm{dom}(UsersNewMails)$$
$$lector? \in \mathrm{dom}(UsersReadMails)$$
$$\exists\, u : USER \bullet (\exists\, m : MENSAJE \bullet ($$
$$(u, m) \in (UsersNewMails(lector?)) \land$$
$$msg! = m \land$$
$$(\#(UsersReadMails(lector?)) + 1) \leq capMaxima \land$$
$$UsersNewMails' = UsersNewMails \oplus \{lector? \mapsto (UsersNewMails(lector?)) \setminus \{(u, m)\}\} \land$$
$$UsersReadMails' = UsersReadMails \oplus \{lector? \mapsto (UsersReadMails(lector?)) \cup \{(u, m)\}\} \land$$
$$UsersSendMails' = UsersSendMails))$$

LectorNotFound

$$\Xi GestorMails$$
$$lector? : USER$$
$$(lector? \notin \mathrm{dom}(UsersNewMails)$$
$$\lor\ lector? \notin \mathrm{dom}(UsersReadMails))$$

NewBoxEmpty

$$\Xi GestorMails$$
$$lector? : USER$$
$$\#(UsersNewMails(lector?)) = 0$$

ReadBoxFull

$$\Xi GestorMails$$
$$lector? : USER$$
$$\neg\ ((\#(UsersReadMails(lector?))) + 1 \leq capMaxima)$$

$$ReadMsgNoOk \mathrel{\widehat=} NewBoxEmpty \lor ReadBoxFull \lor LectorNotFound$$

$$ReadMsg \mathrel{\widehat=} ReadMsgOk \lor ReadMsgNoOk$$

Definimos la operación de vaciado de la bandeja de leídos *CleanReadBox*:

CleanReadBoxOk

$$\Delta GestorMails$$
$$user? : USER$$
$$user? \in \text{dom}\ UsersReadMails$$
$$UsersReadMails' = UsersReadMails \oplus \{user? \mapsto \emptyset\}$$
$$UsersNewMails' = UsersNewMails$$
$$UsersSendMails' = UsersSendMails$$

CleanReadBoxNoOk

$$\Xi GestorMails$$
$$user? : USER$$
$$user? \notin \text{dom}\ UsersReadMails$$

$$CleanReadBox \mathrel{\widehat=} CleanReadBoxOk \lor CleanReadBoxNoOk$$

Definimos la operación de vaciado de la bandeja de enviados *CleanSendBox*:

CleanSendBoxOk

$\Delta GestorMails$
$user? : USER$

$user? \in \text{dom } UsersSendMails$
$UsersSendMails' = UsersSendMails \oplus \{user? \mapsto \emptyset\}$
$UsersNewMails' = UsersNewMails$
$UsersReadMails' = UsersReadMails$

CleanSendBoxNoOk

$\Xi GestorMails$
$user? : USER$

$user? \notin \text{dom } UsersSendMails$

$CleanSendBox \mathrel{\widehat=} CleanSendBoxOk \lor CleanSendBoxNoOk$

Definimos la operación de vaciado de la bandeja de nuevos *CleanNewBox*:

CleanNewBoxOk

$\Delta GestorMails$
$user? : USER$

$user? \in \text{dom } UsersNewMails$
$UsersNewMails' = UsersNewMails \oplus \{user? \mapsto \emptyset\}$
$UsersSendMails' = UsersSendMails$
$UsersReadMails' = UsersReadMails$

CleanNewBoxNoOk

$\Xi GestorMails$
$user? : USER$

$user? \notin \text{dom } UsersNewMails$

$CleanNewBox \mathrel{\widehat=} CleanNewBoxOk \lor CleanNewBoxNoOk$

Como última operación definimos el agregado de usuarios *AddUser*:

AddUserOk

$\Delta GestorMails$
$user? : USER$

$user? \notin \text{dom}(UsersNewMails)$
$user? \notin \text{dom}(UsersReadMails)$
$user? \notin \text{dom}(UsersSendMails)$
$UsersNewMails' = UsersNewMails \cup \{(user?, \emptyset)\}$
$UsersReadMails' = UsersReadMails \cup \{(user?, \emptyset)\}$
$UsersSendMails' = UsersSendMails \cup \{(user?, \emptyset)\}$

UserInMail

$\Xi GestorMails$
$user? : USER$

$(user? \in \text{dom}(UsersNewMails)$
$\lor\ user? \in \text{dom}(UsersReadMails)$
$\lor\ user? \in \text{dom}(UsersSendMails))$

$AddUserNoOk \mathrel{\widehat=} UserInMail$

$AddUser \mathrel{\widehat=} AddUserOk \lor AddUserNoOk$

Capítulo 2

{log} y Simulaciones

2.1. Cargar el {log}

Como necesitamos la libreria *setloglib.slog* dejamos un paso a paso de ejecución. Es opcional dependiendo de como se quiera correr el comando `type_check`.Pasos:

```
[setlog].
setlog(add_lib('setloglib.slog')).
setlog.
type_check.
consult('mail.pl').
```

2.2. Diferencia entre el modelo de {log} y el modelo de Z-eves/Fastest

Los modelos de Z-eves y Fastest son iguales salvando los meros comandos para definir esquemas, pero difieren con el modelo planteado en *setlog* por el tipo básico MENSAJE que pasa a ser un renombramiento del tipo base String que nos provee prolog, lo usamos para simplificar la representación de un mensaje. La especificación de cualquiera de las operaciones no se ven afectadas por el cambio. Al contrario nos es más fácil al momento de simular ya sea mandando o recibiendo mensajes y compararlos.

```
:- dec_type(mensaje, str).
```

2.3. Simulaciones

Mostramos las dos simulaciones hechas a partir del modelo en {log}.

Primera Simulación

La primera simulación, siendo la tipada es la siguiente:

```
:- dec_p_type(checkState(casilla, casilla, casilla,user,user)).
checkState(N,R,S,Usr1,Usr2) :-
  N = {[Usr1, {[Usr1,"Borrador"],[Usr2,"hola usr1"]}] , [Usr2,
  {[Usr1, "chau usr2"]}] } &
  R = {[Usr1,{}],[Usr2,{}]} &
  S = {[Usr2, {[Usr1,"hola usr1"]}] , [Usr1,{[Usr2, "chau usr2"],
  [Usr1,"Borrador"]}] }  .
```

```
:- dec_p_type(simulacionSimbolica2(user, user, casilla, casilla, casilla,
casilla, casilla, casilla,casilla, casilla,casilla,
casilla, casilla, casilla,casilla, casilla, casilla,
casilla, casilla, casilla,casilla, casilla,casilla,
casilla, casilla, casilla)).

simulacion2(Usr1,Usr2,NewBoxInit,ReadBoxInit,SendBoxInit,
            NewBox2,ReadBox2,SendBox2,NewBox3,ReadBox3,SendBox3,
            NewBox4,ReadBox4,SendBox4,NewBox5,ReadBox5,SendBox5,
            NewBox6,ReadBox6,SendBox6,NewBox7,ReadBox7,SendBox7,
            NewBox8,ReadBox8,SendBox8 ):-
checkState(NewBoxInit,ReadBoxInit,SendBoxInit,Usr1,Usr2) &
Usr1 neq Usr2 &
gestorMailsInv(NewBoxInit,ReadBoxInit,SendBoxInit) &
readMsg(NewBoxInit,ReadBoxInit,SendBoxInit,Usr1,PrimerMsg,
        NewBox2,ReadBox2,SendBox2)  &
dec(PrimerMsg,mensaje) &
PrimerMsg = "Borrador" &
readMsg(NewBox2,ReadBox2,SendBox2,Usr2,SegundoMsg,
        NewBox3,ReadBox3,SendBox3) &
dec(SegundoMsg,mensaje) &
SegundoMsg = "chau usr2" &
readMsg(NewBox3,ReadBox3,SendBox3,Usr1,TercerMsg,
        NewBox4,ReadBox4,SendBox4) &
dec(TercerMsg,mensaje) &
TercerMsg = "hola usr1" &

 cleanReadBox(NewBox4,ReadBox4,SendBox4,Usr1,NewBox5,ReadBox5,SendBox5) &
 cleanSendBox(NewBox5,ReadBox5,SendBox5,Usr1,NewBox6,ReadBox6,SendBox6) &
 cleanReadBox(NewBox6,ReadBox6,SendBox6,Usr2,NewBox7,ReadBox7,SendBox7) &
 cleanSendBox(NewBox7,ReadBox7,SendBox7,Usr2,NewBox8,ReadBox8,SendBox8) .
```

En esta primera simulación podemos ver que los estados iniciales de las casillas tienen que ser los que valida `checkState()` de esa forma vamos a ir leyendo uno a uno los mensajes que tienen Usr1 y Usr2 en su casillas de nuevos, una vez que lo leen pasan el mensaje de la casilla de nuevos a la casilla de leídos (lo podemos constatar con los estados resultantes), checkeamos que los mensajes sean los esperados y limpiamos las casillas. Podemos observar las ternas de casillas (New,Read,Send) en el resultado a

continuación:

```
Comando de ejecucion de la simulacion2() tipada:
    simulacion2(user:usr1,user:usr2,
    {[user:usr1, {[user:usr1,"Borrador"],[user:usr2,"hola usr1"]}] ,
    [user:usr2,{[user:usr1, "chau usr2"]}] },
    {[user:usr1,{}],[user:usr2,{}]},
    {[user:usr2, {[user:usr1,"hola usr1"]}] ,
    [user:usr1,{[user:usr2, "chau usr2"],[user:usr1,"Borrador"]}] }
    ,N2,R2,S2,N3,R3,S3,N4,R4,S4,N5,R5,S5,N6,R6,S6,N7,R7,S7,N8,R8,S8) &
    dec([N2,R2,S2,N3,R3,S3,N4,R4,S4,N5,R5,S5,N6,R6,S6,N7,R7,S7,N8,R8,S8],
    casilla).

Resultado:
N2 = {[user:usr2,{[user:usr1,chau usr2]}],
      [user:usr1,{[user:usr2,hola usr1]}]},
R2 = {[user:usr2,{}],
      [user:usr1,{[user:usr1,Borrador]}]},
S2 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{[user:usr2,chau usr2],[user:usr1,Borrador]}]},
N3 = {[user:usr1,{[user:usr2,hola usr1]}],
      [user:usr2,{}]},
R3 = {[user:usr1,{[user:usr1,Borrador]}],
      [user:usr2,{[user:usr1,chau usr2]}]},
S3 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{[user:usr2,chau usr2],[user:usr1,Borrador]}]},
N4 = {[user:usr2,{}],
      [user:usr1,{}]},
R4 = {[user:usr2,{[user:usr1,chau usr2]}],
      [user:usr1,{[user:usr1,Borrador],[user:usr2,hola usr1]}]},
S4 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{[user:usr2,chau usr2],[user:usr1,Borrador]}]},
N5 = {[user:usr2,{}],
      [user:usr1,{}]},
R5 = {[user:usr2,{[user:usr1,chau usr2]}],
      [user:usr1,{}]},
S5 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{[user:usr2,chau usr2],[user:usr1,Borrador]}]},
N6 = {[user:usr2,{}],
      [user:usr1,{}]},
R6 = {[user:usr2,{[user:usr1,chau usr2]}],
```

```
      [user:usr1,{}]},
S6 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{}]},
N7 = {[user:usr2,{}],
      [user:usr1,{}]},
R7 = {[user:usr1,{}],
      [user:usr2,{}]},
S7 = {[user:usr2,{[user:usr1,hola usr1]}],
      [user:usr1,{}]},
N8 = {[user:usr2,{}],
      [user:usr1,{}]},
R8 = {[user:usr1,{}],
      [user:usr2,{}]},
S8 = {[user:usr1,{}],
      [user:usr2,{}]}
```

Sin hacer extensa la explicación podemos fácilmente ver cómo las casillas se van vaciando extrayendo 1 a 1 sus mensajes y luego limpiamos las casillas.

Segunda Simulación

Esta simulación la vamos a mostrar de forma NO TIPADA

```
:- dec_p_type(simulacion1(mensaje,mensaje,user,user, casilla,
                          casilla,casilla, casilla, casilla,casilla,
                          casilla, casilla,casilla, casilla, casilla,
                          casilla, casilla,casilla,casilla,casilla,
                          casilla,casilla,casilla,casilla,casilla,
                          casilla, casilla,casilla, casilla, casilla,
                          casilla)).
```

```
simulacion1(MsgPingRequest,MsgPingReply,User1,User2,
            NewBoxInit,ReadBoxInit,SendBoxInit,
            NewBoxAdd1,ReadBoxAdd1,SendBoxAdd1,
            NewBoxAdd2,ReadBoxAdd2,SendBoxAdd2,
            NewBoxPing1,ReadBoxPing1,SendBoxPing1,
            NewBoxPing2,ReadBoxPing2,SendBoxPing2,
            NewBoxPeek1,ReadBoxPeek1,SendBoxPeek1,
            NewBoxPeek2,ReadBoxPeek2,SendBoxPeek2,
            NewBoxClean1,ReadBoxClean1,SendBoxClean1,
            NewBoxClean2,ReadBoxClean2,SendBoxClean2):-
    gestorMailsInit(NewBoxInit,ReadBoxInit,SendBoxInit) &
    addUser(NewBoxInit,ReadBoxInit,SendBoxInit,User1,
            NewBoxAdd1,ReadBoxAdd1,SendBoxAdd1) &
    addUser(NewBoxAdd1,ReadBoxAdd1,SendBoxAdd1,User2,
            NewBoxAdd2,ReadBoxAdd2,SendBoxAdd2)  &
    User1 neq User2 &
    sendMsg(NewBoxAdd2,ReadBoxAdd2,SendBoxAdd2,User2,User1,MsgPingRequest,
            NewBoxPing1,ReadBoxPing1,SendBoxPing1) &%"hola soy user1".
    sendMsg(NewBoxPing1,ReadBoxPing1,SendBoxPing1,User1,User2,MsgPingReply,
            NewBoxPing2,ReadBoxPing2,SendBoxPing2) &%"un gusto user1 soy user2"
    readMsg(NewBoxPing2,ReadBoxPing2,SendBoxPing2,User2,MsgDeUser1aUser2,
            NewBoxPeek1,ReadBoxPeek1,SendBoxPeek1)&

    dec(MsgDeUser1aUser2,mensaje) &
    MsgDeUser1aUser2 = MsgPingRequest &

    readMsg(NewBoxPeek1,ReadBoxPeek1,SendBoxPeek1,User1,MsgDeUser2aUser1,
            NewBoxPeek2,ReadBoxPeek2,SendBoxPeek2)&

    dec(MsgDeUser2aUser1,mensaje) &
    MsgDeUser2aUser1 = MsgPingReply &

    cleanReadBox(NewBoxPeek2,ReadBoxPeek2,SendBoxPeek2,User1,
                 NewBoxClean1,ReadBoxClean1,SendBoxClean1) &
    cleanReadBox(NewBoxClean1,ReadBoxClean1,SendBoxClean1,User2,
                 NewBoxClean2,ReadBoxClean2,SendBoxClean2) .
```

Básicamente en esta simulación lo que queremos es partir desde el estado inicial para luego agregar dos usuarios al gestor y hacer un *ping* entre ellos con mensajes proporcionados por el usuario.

```
Comando no tipado:
simulacion1("mandame hola","hola",user1,user2,
            N0,R0,S0,N1,R1,S1,N2,R2,S2,N3,R3,S3,N4,R4,S4,N5,R5,S5,
            N6,R6,S6,N7,R7,S7,N8,R8,S8) .
Resultado:
N0 = {},
R0 = {},
S0 = {},
N1 = {[user1,{}]},
R1 = {[user1,{}]},
S1 = {[user1,{}]},
N2 = {[user1,{}],[user2,{}]},
R2 = {[user1,{}],[user2,{}]},
S2 = {[user1,{}],[user2,{}]},
N3 = {[user1,{}],[user2,{[user1,mandame hola]}]},
R3 = {[user1,{}],[user2,{}]},
S3 = {[user2,{}],[user1,{[user2,mandame hola]}]},
N4 = {[user2,{[user1,mandame hola]}],[user1,{[user2,hola]}]},
R4 = {[user1,{}],[user2,{}]},
S4 = {[user1,{[user2,mandame hola]}],[user2,{[user1,hola]}]},
N5 = {[user1,{[user2,hola]}],[user2,{}]},
R5 = {[user1,{}],[user2,{[user1,mandame hola]}]},
S5 = {[user1,{[user2,mandame hola]}],[user2,{[user1,hola]}]},
N6 = {[user2,{}],[user1,{}]},
R6 = {[user2,{[user1,mandame hola]}],[user1,{[user2,hola]}]},
S6 = {[user1,{[user2,mandame hola]}],[user2,{[user1,hola]}]},
N7 = {[user2,{}],[user1,{}]},
R7 = {[user2,{[user1,mandame hola]}],[user1,{}]},
S7 = {[user1,{[user2,mandame hola]}],[user2,{[user1,hola]}]},
N8 = {[user2,{}],[user1,{}]},
R8 = {[user1,{}],[user2,{}]},
S8 = {[user1,{[user2,mandame hola]}],[user2,{[user1,hola]}]}
```

Es mucho más simple la lectura de esta salida a comparación de la tipada. Podemos ver que los estados 0 son los iniciales, en el 1 y 2 agregamos los usuarios, en el 3 y 4 mandamos los ping (mensajes), en el 5 el user2 lee el mensaje de user1, en el 6 el user1 lee el mensaje de user2, en el 7 se limpia la casilla de leídos del user1 y en el 8 hace lo mismo el user2. Siempre que nos referimos a un número X es a la terna NX RX SX

2.4. Teoremas Tipados

La realización de la prueba de los teoremas fue realizada como lo indica la página 33 del apunte *setlog.pdf*. Dado que los teoremas necesitan la negación de esquemas nos vimos en la necesidad de negarlos explícitamente usando el *n_SchemaName*

Primer Teorema

Como primer teorema postulamos que *AddUser* como operación tiene que preservar las casillas como funciones parciales

theorem AddUser1
$(UsersNewMails \in \_ \pfun \_ \land$
$UsersSendMails \in \_ \pfun \_ \land$
$UsersReadMails \in \_ \pfun \_ \land$
$AddUser) \Rightarrow$
$(UsersNewMails' \in \_ \pfun \_ \land$
$UsersSendMails' \in \_ \pfun \_ \land$
$UsersReadMails' \in \_ \pfun \_)$

Lo que se traduce en $\{log\}$ como:

```
:- dec_p_type(n_gestorMailsInvPFun(casilla, casilla, casilla)).
n_gestorMailsInvPFun(UsersNewMails,UsersReadMails,UsersSendMails) :-
   neg(
       pfun(UsersNewMails) &
       pfun(UsersReadMails) &
       pfun(UsersSendMails) ).

:- dec_p_type(gestorMailsInvPFun(casilla, casilla, casilla)).
gestorMailsInvPFun(UsersNewMails,UsersReadMails,UsersSendMails) :-
   pfun(UsersNewMails) &
   pfun(UsersReadMails) &
   pfun(UsersSendMails).

:- dec_p_type(theoremAddUser1(user, casilla, casilla,casilla,
                              casilla, casilla,casilla)).
theoremAddUser1(User1,N0,R0,S0,N0_,R0_,S0_) :-
   neg((gestorMailsInvPFun(N0,R0,S0) & addUser(N0,R0,S0,User1,N0_,R0_,S0_))
        implies  gestorMailsInvPFun(N0_,R0_,S0_) ).
```

Dejamos el comando tipado de ejecución:

```
theoremAddUser1(user:user1,N0,R0,S0,N1,R1,S1) &
dec([N0,R0,S0,N1,R1,S1], casilla).
```

Dejamos en el código un teorema similar para la operación *SendMsg*

Segundo Teorema

Como segundo teoremas queremos probar que *AddUser* como operación tiene que preservar la invariante de *GestorMailsInv*. Dado que la invariante entera tiene dos partes, la primera donde se igualan los dominios y la segunda donde se checkea el largo de las casillas con cuantificadores universales. Para evitar las pruebas con los cuantificadores, solamente formulamos el teorema para la primera parte de la invariante pero no escapamos del problema tan fácil porque este teorema completo va a ser el demostrado en Z-Eves con uso de los cuantificadores.

El teorema se enuncia como: *AddUser* preserva la igualdad de los dominios de las casillas. Este teorema es importante porque en el caso de que *AddUser* efectivamente agrege un usuario queremos que lo adicione a las tres casillas.

theorem AddUser2
$$GestorMailsInvDomEq \land AddUser \Rightarrow GestorMailsInvDomEq'$$

Su traducción a $\{log\}$ es la siguiente:

```
:- dec_p_type(n_gestorMailsInvDomEq(casilla, casilla, casilla)).
n_gestorMailsInvDomEq(UsersNewMails,UsersReadMails,UsersSendMails) :-
     dec(NM, set(user)) &
     dec(RM, set(user)) &
     dec(SM, set(user)) &
     dom(UsersNewMails, NM) &
     dom(UsersReadMails,RM) &
     dom(UsersSendMails,SM) &
     neg(
     NM = RM &
     RM = SM &
     SM = NM ) .

:- dec_p_type(gestorMailsInvDomEq(casilla, casilla, casilla)).
gestorMailsInvDomEq(UsersNewMails,UsersReadMails,UsersSendMails) :-
     dec(NM, set(user)) &
     dec(RM, set(user)) &
     dec(SM, set(user)) &
     dom(UsersNewMails, NM) &
     dom(UsersReadMails,RM) &
     dom(UsersSendMails,SM) &
     NM = RM &
     RM = SM &
     SM = NM .


:- dec_p_type(theoremAddUser2(user, casilla, casilla,casilla,
                              casilla, casilla,casilla)).
theoremAddUser2(User1,N0,R0,S0,N0_,R0_,S0_) :-
neg((gestorMailsInvDomEq(N0,R0,S0) & addUser(N0,R0,S0,User1,N0_,R0_,S0_) )
     implies
     (gestorMailsInvDomEq(N0_,R0_,S0_))).
```

Dejo el comando de ejecución:

```
theoremAddUser2(user:user1,N0,R0,S0,N1,R1,S1) &
dec([N0,R0,S0,N1,R1,S1], casilla).
```

Hay un teorema extra en el código *theoremAddUser*3 que escapa a la forma $I \wedge Op \Rightarrow I'$ típica, y es más parecido a la forma $A \wedge Op \Rightarrow B$.

Capítulo 3

Z-eves y teoremas

3.1. Teoremas de ZEves

Dado que en las invariantes uso cuantificadores tuve en el camino ciertos obstáculos ya que ZEves no los maneja bien a la hora de realizar inferencias o reducirlos. Por ejemplo en esta situación:

```
Premisas: A = A'
          B = B'
          C = C'
          \forall x \in A, P(x)
          \forall y \in B, P(x)
          \forall z \in C, P(x)
\implies
          \forall x' \in A', P(x)
          \forall y' \in B', P(x)
          \forall z' \in C', P(x)
```

ZEves no lo deriva como *true*, siendo que es trivial al verlo. No podemos hacer simplificaciones o sustituciones de igualdades dentro de cuantificadores, sino que tenemos que ir más allá y utilizar normalizaciones de reducción o escritura para poder derivar *true* con los mismos.

Nuestro teorema a demostrar en ZEves, es que la operación *AddUser* preserva la invariante *GestorMailsInv*

theorem AddUserTheoremTotal
$GestorMailsInv \land AddUser \Rightarrow GestorMailsInv'$

Para ello creamos un sub teorema para la operación *AddUserOk*

theorem AddUserOkTheorem
$GestorMailsInv \land AddUserOk \Rightarrow GestorMailsInv'$

La idea de la demostración es un *divide&conquer*. Pasando a la demostración de la invariante para *AddUserOk*

proof[*AddUserOkTheorem*]
use axiom$1;
invoke;
simplify;
rewrite;
equality substitute;
rearrange;
with normalization reduce;
■

La idea es que usamos la definición del axioma de *capMaxima*, invocamos los esquemas. Debido a la gran cantidad de igualaciones entre los dominios y variables reestructuramos el contenido y el goal. Ahora quedamos en una situación similar a la explicada en el comienzo de la sección por la cual Z-Eves no la puede reducir. En dicha situación tenemos como premisas:

- Que $\theta\, GestorMails$ que cumple indudablemente la invariante por hipótesis ($UsersNewMails$, $UsersReadMails$, $UsersSendMails$)
- El elemento a agregar a los conjuntos del punto anterior es ($user?, \{\}$) que cumple con la invariante del largo de la casilla

Y del otro lado de la implicancia tenemos que $\theta\, GestorMails'$ fruto de insertar la tupla en los tres conjuntos, cumple con la invariante.

Z-eves expande la demostración si intentamos reducirla o probarla por reducción, al contrario de lo que uno espera. Por eso utilizamos la normalización de la reducción para la exprexión, gracias a esta herramienta potente podemos demostrar *AddUserOkTheorem*

Ahora pasemos a la demostración de *AddUserTheoremTotal*:

proof[*AddUserTheoremTotal*]
split AddUserOk;
cases;
use AddUserOkTheorem;
simplify;
next;
split AddUserNoOk;
cases;
invoke AddUserNoOk;
use UserInMail\$*declarationPart*;
use Xi\$*GestorMails*\$*declarationPart*;
rearrange;
invoke UserInMail;
invoke Ξ *GestorMails*;
with normalization rewrite;
next;
prove by reduce;
next;
■

En esta prueba sobre la operación total *AddUser* vamos a usar un split sobre *AddUserOk* y nos valemos del teorema pasado que sumado a una

simplificación llegamos a *true* en esa rama. Luego vamos a la rama de *AddUserNoOk* haciendo algo parecido para su homóloga positiva, llegamos a la situación comentada al inicio de la sección dónde hay una igualación de los estados pasados y futuros acompañados con una propiedad universal para todos los estados pasados. Y a pesar de haber una igualación explícita ya que no cambia el estado de *GestorMails* el prover de Z-Eves no deduce que esa propiedad universal se sigue cumpliendo. Aunque nos parezca trivial la demostración, no lo es. Hacemos uso de los teoremas que Z-Eves ya pudo deducir gracias a las declaraciones de los esquemas e invocandolos podemos reescribirlos con una normalización para llegar al *true*. Los siguientes salen triviales por *prove by reduce* porque es una simple negación de todo el antecedente.

Capítulo 4

Fastest

4.1. Casos de prueba

Realizo casos de prueba para la operación *AddUser* con la siguiente batería de operaciones

```
loadspec mailft.tex
selop AddUser
genalltt
addtactic AddUser_DNF_1 SP \notin user? \notin \dom UsersNewMails
addtactic AddUser_DNF_1 SP \notin user? \notin \dom UsersReadMails
addtactic AddUser_DNF_1 SP \notin user? \notin \dom UsersSendMails
addtactic AddUser_DNF_2 SP \in user? \in \dom UsersNewMails
addtactic AddUser_DNF_3 SP \in user? \in \dom UsersReadMails
addtactic AddUser_DNF_4 SP \in user? \in \dom UsersSendMails
genalltt
prunett
genalltca
```

Primero aplicamos la tactica DNF, obteniendo 4 de ellas:

- DNF1 : corresponde a *UserAddOk*
- DNF2 : corresponde a *UserAddNoOk* pero como es un esquema con conjunciones, este DNF hace referencia a la sentencia $user? \in \text{dom } UsersNewMails$
- DNF3 : corresponde a *UserAddNoOk* pero como es un esquema con conjunciones, este DNF hace referencia a la sentencia $user? \in \text{dom } UsersReadMails$
- DNF4 : corresponde a *UserAddNoOk* pero como es un esquema con conjunciones, este DNF hace referencia a la sentencia $user? \in \text{dom } UsersSendMails$

Y luego con la *StandardPartition* obtenemos la siguientes casos abstractos de prueba:

- *AddUser_VIS*
  - *AddUser_DNF_1*
    - *AddUser_SP_1*
      - *AddUser_SP_3*
        - *AddUser_SP_5*
          - *AddUser_SP_5_TCASE*
        - *AddUser_SP_6*
          - *AddUser_SP_6_TCASE*
      - *AddUser_SP_4*
        - *AddUser_SP_7*
          - *AddUser_SP_7_TCASE*
        - *AddUser_SP_8*
          - *AddUser_SP_8_TCASE*
    - *AddUser_SP_2*
      - *AddUser_SP_9*
        - *AddUser_SP_11*
          - *AddUser_SP_11_TCASE*
        - *AddUser_SP_12*
          - *AddUser_SP_12_TCASE*
      - *AddUser_SP_10*
        - *AddUser_SP_13*
          - *AddUser_SP_13_TCASE*
        - *AddUser_SP_14*
          - *AddUser_SP_14_TCASE*
  - *AddUser_DNF_2*
    - *AddUser_SP_15*
      - *AddUser_SP_15_TCASE*
    - *AddUser_SP_16*
      - *AddUser_SP_16_TCASE*
  - *AddUser_DNF_3*
    - *AddUser_SP_19*
      - *AddUser_SP_19_TCASE*
    - *AddUser_SP_0*
      - *AddUser_SP_20_TCASE*
  - *AddUser_DNF_4*
    - *AddUser_SP_17*
      - *AddUser_SP_17_TCASE*
    - *AddUser_SP_18*
      - *AddUser_SP_18_TCASE*

Los casos abstractos de prueba los mostramos con $showsch - tca$ y son los siguientes:

$AddUser\_SP\_5\_TCASE$

$AddUser\_SP\_5$

$UsersSendMails = \emptyset$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \emptyset$

$AddUser\_SP\_6\_TCASE$

$AddUser\_SP\_6$

$UsersSendMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \emptyset$

$AddUser\_SP\_7\_TCASE$

$AddUser\_SP\_7$

$UsersSendMails = \emptyset$
$UsersReadMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$user? = uSER1$
$UsersNewMails = \emptyset$

$AddUser\_SP\_8\_TCASE$

$AddUser\_SP\_8$

$UsersSendMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$UsersReadMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$user? = uSER1$
$UsersNewMails = \emptyset$

$AddUser\_SP\_11\_TCASE$

$AddUser\_SP\_11$

$UsersSendMails = \emptyset$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$

AddUser_SP_12_TCASE

AddUser_SP_12

$UsersSendMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$

AddUser_SP_13_TCASE

AddUser_SP_13

$UsersSendMails = \emptyset$
$UsersReadMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$user? = uSER1$
$UsersNewMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$

AddUser_SP_14_TCASE

AddUser_SP_14

$UsersSendMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$UsersReadMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$
$user? = uSER1$
$UsersNewMails = \{(uSER2 \mapsto \{(uSER1 \mapsto mENSAJE1)\})\}$

AddUser_SP_15_TCASE

AddUser_SP_15

$UsersSendMails = \emptyset$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$

AddUser_SP_16_TCASE

AddUser_SP_16

$UsersSendMails = \emptyset$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\}), (uSER3 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$

AddUser_SP_19_TCASE

AddUser_SP_19

$UsersSendMails = \emptyset$
$UsersReadMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$
$user? = uSER1$
$UsersNewMails = \emptyset$

AddUser_SP_20_TCASE

AddUser_SP_20

$UsersSendMails = \emptyset$
$UsersReadMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\}), (uSER3 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$
$user? = uSER1$
$UsersNewMails = \emptyset$

AddUser_SP_17_TCASE

AddUser_SP_17

$UsersSendMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \emptyset$

AddUser_SP_18_TCASE

AddUser_SP_18

$UsersSendMails = \{(uSER1 \mapsto \{(uSER1 \mapsto mENSAJE2)\}), (uSER3 \mapsto \{(uSER1 \mapsto mENSAJE2)\})\}$
$UsersReadMails = \emptyset$
$user? = uSER1$
$UsersNewMails = \emptyset$

Como nuestra operación *AddUser* es una conjunción de esquemas que en sus precondiciones está solamente el operador $\in$ en ellos, optamos por la técnica de partición estándar para generar el árbol de casos. En la partición del operador $\notin$ en las expresiones del estilo $x \notin A$ se va a particionar según si $A = \{\}$ o $A \neq \{\}$ tal como dice el *user guide* de fastest.

De forma obvia vamos a tener una estructura más densa de clases que nacen del *DNF_1* ya que en esta clase vamos a particionar sobre cada expresión con el operador $\notin$ que haya por lo tanto vamos a generar casos abstractos de prueba por cada combinación que se de entre las particiones de los predicados de dicho DNF. Es decir, todas las combinaciones entre las 3

casillas (*UsersSendMails*, *UsersReadMails* y *UsersNewMails*) y estas por la partición pueden adoptar 2 valores (vacío y no vacío) son 2^3 casos de prueba (5,6,7,8,11,12,13,14). De análisis similar al anterior los DNF siguientes son más simples, ahora solamente particionamos sobre el operador $\in$ en una sola expresión.

En los casos de prueba obviamente vamos a encontrar algunos que vayan en contra de los teoremas de *AddUser* que probamos en $\{log\}$, entre dichos casos son aquellos donde el dominio de las casillas son distintos. Pero a fines de testear la operación son válidos.