Trabajo Practico Final

Bautista Marelli

April 12, 2024

1 Problema propuesto

Sistema de control de Accesos. El sistema consiste en un control de ingresos a edificios o oficinas, el cual cuenta con una base de datos de los usuarios que tienen permitido acceder al establecimiento y otra tabla donde se registra el ingreso de los usuarios según el horario.

Las operaciones contempladas en este sistema son:

- **addUser** - que recibe un usuario y, si el usuario no existe en la base de datos, lo agregar para que pueda acceder al establecimiento.
- **removeUser** - que recibe un usuario y, si el usuario se encuentra dentro de la base de datos, lo elimina de los usuarios permitidos y también elimina todo registro de ingreso del usuario.
- **validateAccess** - que recibe un usuario y un timestamp y, si el usuario tiene permitido el ingreso, permite el acceso al establecimiento y registra el ingreso en el sistema.

Hay que tener en cuenta que solo se puede tener un ingreso por horario (no pueden entrar 2 personas al mismo tiempo). También, solo los usuarios permitidos pueden tener un registro de acceso.

2 Especificación Z

El type checking se realizo utilizando el CLI de Z-Eves.

2.1 Designaciones

- u es un usuario $\approx$ $USER(\mathrm{u})$.
- t es un timestamp $\approx$ $TIMESTAMP(\mathrm{t})$.
- No se puede validar 2 o más ingresos con el mismo timestamp.
- Se da de alta un usuario u (tiene habilitado el acceso) $\approx$ $AddUser(\mathrm{u})$.

- Se da de baja un usuario u (deja de estar habilitado para acceder) $\approx$ *RemoveUser*(u).
- Se valida el acceso de un usuario u en un timestamp t $\approx$ *ValidateAccess*(u, t).
- La operación se realizo con éxito $\approx$ *ok*.
- Se quiso dar de alta un usuario que ya existe $\approx$ *userAlreadyExists*.
- Se quiso dar de baja un usuario que no existe $\approx$ *userDoesNotExist*.
- Un usuario que no esta habilitado quiso validar un ingreso $\approx$ *userDoesNotHaveAccess*.
- Un usuario quiso validar un ingreso al mismo tiempo que otro $\approx$ *timestampAlreadyExists*.

2.2 Especificación

$[USER, TIMESTAMP]$
$RES ::= ok$
$\mid userAlreadyExists$
$\mid userDoesNotExist$
$\mid userDoesNotHaveAccess$
$\mid timestampAlreadyExists$

State

$users : \mathbb{P}\, USER$
$access : TIMESTAMP \pfun USER$

InvState

$State$

$\operatorname{ran} access \subseteq users$

InitialState

$State$

$users = \varnothing$
$access = \varnothing$

AddUserOk

$\Delta State$
$u? : USER$
$res! : RES$

$u? \notin users$
$users' = users \cup \{u?\}$
$access' = access$
$res! = ok$

AddUserAlreadyExists

$\Xi State$
$u? : USER$
$res! : RES$

$u? \in users$
$res! = userAlreadyExists$

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

RemoveUserOk

$\Delta State$
$u? : USER$
$res! : RES$

$u? \in users$
$users' = users \setminus \{u?\}$
$access' = access \rhd \{u?\}$
$res! = ok$

Notemos que borramos todos los registros de accesos del usuario.

RemoveUserDoesNotExist

$\Xi State$
$u? : USER$
$res! : RES$

$u? \notin users$
$res! = userDoesNotExist$

$RemoveUser \mathrel{\widehat=} RemoveUserOk \lor RemoveUserDoesNotExist$

ValidateAccessOk

$\Delta State$
$u? : USER$
$t? : TIMESTAMP$
$res! : RES$

$u? \in users$
$t? \notin \mathrm{dom}\ access$
$users' = users$
$access' = access \cup \{t? \mapsto u?\}$
$res! = ok$

Notemos que primero validamos que el timestamp no se encuentre dentro de los registros de accesos. De esta manera mantenemos que la relación parcial y no sobrescribimos información.

VAUserDoesNotHaveAccess

$\Xi State$
$u? : USER$
$t? : TIMESTAMP$
$res! : RES$

$u? \notin users$
$res! = userDoesNotHaveAccess$

VATimestampAlreadyExists

$\Xi State$
$u? : USER$
$t? : TIMESTAMP$
$res! : RES$

$u? \in users$
$t? \in \mathrm{dom}\ access$
$res! = timestampAlreadyExists$

$ValidateAccessError \mathrel{\widehat=} VAUserDoesNotHaveAccess \lor VATimestampAlreadyExists$
$ValidateAccess \mathrel{\widehat=} ValidateAccessOk \lor ValidateAccessError$

2.3 Normalizamos la especificación

State

$users : \mathbb{P}\ USER$
$access : TIMESTAMP \leftrightarrow USER$

InvState

$State$

$\mathrm{dom}\ access \subseteq users$
$access \in TIMESTAMP \pfun USER$

InitialState

$State$

$users = \varnothing$
$access = \varnothing$

3 SetLog {log}

Convertimos la especificación Z a codigo {log} tipado y realizamos 2 simulaciones (una tipada y otra no).

3.1 Simulación tipada

```
dec(U0, users) & dec(U1, users) & dec(U2, users) & dec(U3, users)
& dec(A0, access) & dec(A1, access) & dec(A2, access) & dec(A3, access)
& dec(X1, response) & dec(X2, response) & dec(X3, response) & dec(X4, response)
& initialState(U0, A0)
& addUser(U0, A0, user:1, X1, U1, A1)
& addUser(U1, A1, user:2, X2, U2, A2)
& validateAccess(U2, A2, user:1, timestamp:1, X3, U3, A3).
```

En esta simulación iniciamos las variables del estado, damos de alta a dos usuarios (`user:1` y `user:2`) y validamos el ingreso de uno con un timestamp dado (`timestamp:1`). El resultado es:

```
U0 = {},
U1 = {user:1},
U2 = {user:1,user:2},
U3 = {user:1,user:2},
A0 = {},
A1 = {},
A2 = {},
A3 = {[timestamp:1,user:1]},
X1 = response:ok,
X2 = response:ok,
X3 = response:ok
```

Como podemos ver, los usuario son dados de alta exitosamente y la validación del ingreso también ya que el usuario tiene permitido el ingreso y no había ningún ingreso registrado con el timestamp dado.

3.2 Simulación no tipada

```
initialState(U0, A0)
& addUser(U0, A0, user:1, X1, U1, A1)
& addUser(U1, A1, user:2, X2, U2, A2)
& validateAccess(U2, A2, user:1, timestamp:11, X3, U3, A3)
& validateAccess(U3, A3, user:1, timestamp:11, X4, U4, A4)
& validateAccess(U4, A4, user:3, timestamp:12, X5, U5, A5)
& removeUser(U5, A5, user:1, X6, U6, A6).
```

Lo que hacemos en esta simulación es:

- Inicializar las variables del estado.
- Dar de alta los usuarios `user:1` y `user:2`.
- Validar el ingreso de `user:1` en el momento `timestamp:11`.
- Validar el ingreso de `user:1` en el momento `timestamp:11`.
- Validar el ingreso de `user:3` en el momento `timestamp:12`.
- Dar de baja al usuario `user:1`.

El resultado es:

```
U0 = {},
A0 = {},
X1 = response:ok,
U1 = {user:1},
A1 = {},
X2 = response:ok,
U2 = {user:1,user:2},
A2 = {},
X3 = response:ok,
U3 = {user:1,user:2},
A3 = {[timestamp:11,user:1]},
X4 = response:timestampAlreadyExists,
U4 = {user:1,user:2},
A4 = {[timestamp:11,user:1]},
X5 = response:userDoesNotHaveAccess,
U5 = {user:1,user:2},
A5 = {[timestamp:11,user:1]},
X6 = response:ok,
U6 = {user:2/_N1},
A6 = {}
Constraint: subset(_N1,{user:1,user:2}), user:1 nin _N1, set(_N1)
```

Como podemos ver, despues de dar de alta los usuarios `user:1` y `user:2` tenemos una respuesta `ok` y en `U2` tenemos los usuarios agregados.

Luego, cuando intentamos validar el ingreso del `user:1` por segunda vez con el mismo timestamp, obtenemos como respuesta de error `timestampAlreadyExists` y el estado no cambia. Luego validamos el ingreso de `user:3` con timestamp `timestamp:12` y como este usuario no se encuentra dentro de `U4` (el usuario no tiene permitido el ingreso), la operación nos da como resultado `userDoesNotHaveAccess`.

Por ultimo, damos de baja a `user:1` y con esto removemos todos los registros de acceso del usuario y el resultado es `ok`.

4 VCG

La iteración con el VCG fue bastante directa. No hubo necesidad de agregar ninguna hipótesis. Los comandos utilizados para correr la iteración de VCG fueron:

```
consult("setlog.pl").
setlog.
type_check.
consult("spec.pl").
vcg("spec.pl").
consult("spec-vc.pl").
check_vcs_spec.
```

Y el resultado de esto fue lo siguiente:

```
Checking initialState_sat_invState ... OK
Checking addUser_is_sat ... OK
Checking removeUser_is_sat ... OK
Checking validateAccess_is_sat ... OK
Checking addUser_pi_invState ...
***WARNING***: using unsafe negation
```

```
OK
Checking removeUser_pi_invState ...
***WARNING***: using unsafe negation
OK
Checking validateAccess_pi_invState ...
***WARNING***: using unsafe negation
OK

Total VCs: 7 (discharged: 7, failed: 0, timeout: 0)
Execution time (discharged): 0.006925344467163086 s
yes
```

5 Teorema Z-Eves

El teorema planteado es el siguiente:

theorem ValidateAccessInv
$InvState \land ValidateAccess \Rightarrow InvState'$

Este teorema quiere decir que la operación *ValidateAccess* mantiene la invariante de estado. La demostración que llegue después de usar el CLI de Z-Eves es la siguiente:

proof[*ValidateAccessInv*
invoke ValidateAccess;
invoke ValidateAccessError;
split ValidateAccessOk;
cases;
reduce;
prove by rewrite;
next;
reduce;
disjunctive;
prove by rewrite;
next;
■

La explicación es:

- **invoke** - remplazamos la definición de *ValidateAccess*.
- **invoke** - remplazamos la definición de *ValidateAccessError*.
- **split** - generamos un nuevo objetivo del tipo if *ValidateAccessOk* then G else G.
- **cases** - separamos el problema en 2 casos.
- **reduce** - intenta reducir la formula utilizando las equivalencias.
- **prove by rewrite** - se demuestra el caso con rewrite.
- **next** - pasamos al siguiente caso.
- **reduce** - intenta reducir la formula utilizando las equivalencias.
- **disjunctive** - transformamos en la forma normal disjunctiva.
- **prove by rewrite** - se demuestra el caso con rewrite.
- **next** - pasamos al siguiente caso (terminamos).

6 Fastest

Los comandos usados para generar los casos de prueba fueron:

```
loadspec fastest.text
selop ValidateAccess
genalltt
addtactic ValidateAccess_DNF_1 SP \in u? \in users
addtactic ValidateAccess_DNF_2 SP \notin u? \notin users
addtactic ValidateAccess_DNF_3 SP \in u? \in users
genalltt
addtactic ValidateAccess_DNF_1 SP \notin t? \notin \dom access
addtactic ValidateAccess_DNF_3 SP \in t? \in \dom access
genalltt
genalltca
```

Se generaron casos de pruebas para la operación *ValidateAccess* y de esta forma descomponemos los casos por las operaciones *ValidateAccessOk*, *VAUserDoesNotHaveAccess* y *VATimestampAlreadyExists*. Luego, se aplica SP sobre $\in$ y $\notin$ (en las expresiones $u? \in users$ y $u? \notin users$). Por ultimo, volvemos aplicar las particiones estándar sobre los mismos operadores pero sobre los casos `ValidateAccess_DNF_1` y `ValidateAccess_DNF_3` (en las expresiones $t? \notin \text{dom}\ access$ y $t? \in \text{dom}\ access$).

El resultado de Fastest fue el siguiente árbol de clases de pruebas:

```
ValidateAccess_VIS
  !______ValidateAccess_DNF_1
  |       !______ValidateAccess_SP_1
  |       |       !______ValidateAccess_SP_7
  |       |       |       !______ValidateAccess_SP_7_TCASE
  |       |       |
  |       |       !______ValidateAccess_SP_8
  |       |               !______ValidateAccess_SP_8_TCASE
  |       |
  |       |
  |       !______ValidateAccess_SP_2
  |               !______ValidateAccess_SP_9
  |               |       !______ValidateAccess_SP_9_TCASE
  |               |
  |               !______ValidateAccess_SP_10
  |                       !______ValidateAccess_SP_10_TCASE
  |
  |
  |
  !______ValidateAccess_DNF_2
  |       !______ValidateAccess_SP_3
  |       |       !______ValidateAccess_SP_3_TCASE
  |       |
  |       !______ValidateAccess_SP_4
  |               !______ValidateAccess_SP_4_TCASE
  |
  |
  !______ValidateAccess_DNF_3
          !______ValidateAccess_SP_5
          |       !______ValidateAccess_SP_11
          |       |       !______ValidateAccess_SP_11_TCASE
          |       |
```

```
|       !______ValidateAccess_SP_12
|               !______ValidateAccess_SP_12_TCASE
|
|
!______ValidateAccess_SP_6
        !______ValidateAccess_SP_13
        |       !______ValidateAccess_SP_13_TCASE
        |
        !______ValidateAccess_SP_14
                !______ValidateAccess_SP_14_TCASE
```

Por lo que podemos ver, no tuvimos testcases que no pudieron generarse. Los testcases generados son los siguientes:

ValidateAccess_SP_7_TCASE

ValidateAccess_SP_7

$t? = tIMESTAMP2$
$u? = uSER1$
$access = \varnothing$
$users = \{uSER1\}$

ValidateAccess_SP_8_TCASE

ValidateAccess_SP_8

$t? = tIMESTAMP1$
$u? = uSER1$
$access = \{(tIMESTAMP2 \mapsto uSER2)\}$
$users = \{uSER1\}$

ValidateAccess_SP_9_TCASE

ValidateAccess_SP_9

$t? = tIMESTAMP1$
$u? = uSER1$
$access = \varnothing$
$users = \{uSER1, uSER2\}$

ValidateAccess_SP_10_TCASE

ValidateAccess_SP_10

$t? = tIMESTAMP1$
$u? = uSER1$
$access = \{(tIMESTAMP2 \mapsto uSER1)\}$
$users = \{uSER1, uSER2\}$

ValidateAccess_SP_3_TCASE

ValidateAccess_SP_3

$t? = tIMESTAMP2$
$u? = uSER1$
$access = \varnothing$
$users = \varnothing$

ValidateAccess_SP_4_TCASE

ValidateAccess_SP_4

$t? = tIMESTAMP2$
$u? = uSER1$
$access = \varnothing$
$users = \{uSER1\}$

ValidateAccess_SP_11_TCASE

ValidateAccess_SP_11

$t? = tIMESTAMP2$
$u? = uSER1$
$access = \{(tIMESTAMP2 \mapsto uSER3)\}$
$users = \{uSER1\}$

ValidateAccess_SP_12_TCASE

ValidateAccess_SP_12

$t? = tIMESTAMP2$
$u? = uSER1$
$access = \{(tIMESTAMP2 \mapsto uSER3), (tIMESTAMP4 \mapsto uSER3)\}$
$users = \{uSER1\}$

ValidateAccess_SP_13_TCASE

ValidateAccess_SP_13

$t? = tIMESTAMP1$
$u? = uSER1$
$access = \{(tIMESTAMP1 \mapsto uSER2)\}$
$users = \{uSER1, uSER2\}$

ValidateAccess_SP_14_TCASE

ValidateAccess_SP_14

$t? = tIMESTAMP1$
$u? = uSER1$
$access = \{(tIMESTAMP1 \mapsto uSER2), (tIMESTAMP3 \mapsto uSER2)\}$
$users = \{uSER1, uSER2\}$