Trabajo Práctico para Ingeniería de Software

Antonio Locascio

1. Requerimientos

Se describen los requerimientos para una *lista de control de acceso* (ACL) para archivos.

La ACL guarda los permisos sobre un archivo que poseen los distintos usuarios y grupos. Los posibles permisos son de lectura y escritura. La interfaz de la ACL debe permitir:

- Agregar un permiso a un usuario,
- Agregar un permiso a un grupo,
- Verificar si un usuario es lector (tanto si el usuario tiene permiso de lectura o si pertence a un grupo con este permiso),
- Verificar si un usuario es escritor (tanto si el usuario tiene permiso de escritura o si pertence a un grupo con este permiso).

El sistema provee una función que asocia usuarios a grupos.

2. Especificación

Para empezar, se dan las siguientes designaciones.

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

g es un grupo $\approx g \in GROUP$

r es un permiso $\approx r \in PERM$

ans es una respuesta a una consulta $\approx ans \in ANS$

Conjunto de grupos a los que u pertenece $\approx userGroups\ u$

Permisos guardados para el usuario $u \approx usrs\ u$

Permisos guardados para el grupo $g \approx grps\ g$

Luego, se introducen los tipos que se utilizan en la especificación.

$$[USER, GROUP]$$

$$PERM ::= r \mid w$$

$$ANS ::= yes \mid no$$

Además, se presenta la siguiente definición axiomática. Esta representa la función que asocia usuarios a grupos que está disponible en el sistema. Se asume que su dominio es el conjunto de todos los usuarios del sistema.

To appear in EPTCS.

$$userGroups : USER \pfun \mathbb{P}\, GROUP$$

Con lo anterior, se define el espacio de estados de la ACL junto a su estado inicial.

$$\begin{array}{l}
\underline{ACL} \\
usrs : USER \pfun \mathbb{P}\, PERM \\
grps : GROUP \pfun \mathbb{P}\, PERM
\end{array}$$

$$\begin{array}{l}
\underline{ACLInit} \\
ACL \\
\hline
usrs = \emptyset \\
grps = \emptyset
\end{array}$$

Como todos los usuarios registrados se encuentran en el dominio de *userGroups*, debe valer el siguiente invariante.

$$\begin{array}{l}
\underline{ACLInv} \\
ACL \\
\hline
\mathrm{dom}(usrs) \subseteq \mathrm{dom}(userGroups)
\end{array}$$

A continuación se modelan las operaciones requeridas. En primer lugar se define la que permite agregar un usuario con un permiso a la ACL. Para ello, se dan esquemas para manejar si el usuaruio ya está en la lista o no. Además, se agrega un esquema para modelar el posible error.

$$\begin{array}{l}
\underline{AddNewUserRight} \\
\Delta ACL \\
u? : USER \\
r? : PERM \\
\hline
u? \notin \mathrm{dom}(usrs) \\
u? \in \mathrm{dom}(userGroups) \\
usrs' = usrs \oplus \{u? \mapsto \{r?\}\} \\
grps' = grps
\end{array}$$

$$\begin{array}{l}
\underline{AddExistingUserRight} \\
\Delta ACL \\
u? : USER \\
r? : PERM \\
\hline
u? \in \mathrm{dom}(usrs) \\
usrs' = usrs \oplus \{u? \mapsto \{r?\} \cup (usrs(u?))\} \\
grps' = grps
\end{array}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{UserDoesNotExist}} \\
\Xi ACL \\
u? : USER \\
\hline
u? \notin \operatorname{dom}(userGroups) \\
\hline
\end{array}
$$

$$AddUserRight == AddNewUserRight \lor AddExistingUserRight \lor UserDoesNotExist$$

En segundo lugar se hace lo mismo con la operación análoga para grupos.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddNewGroupRight}} \\
\Delta ACL \\
g? : GROUP \\
r? : PERM \\
\hline
g? \notin \operatorname{dom}(grps) \\
grps' = grps \oplus \{g? \mapsto \{r\}?\} \\
usrs' = usrs \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddExistingGroupRight}} \\
\Delta ACL \\
g? : GROUP \\
r? : PERM \\
\hline
g? \in \operatorname{dom}(grps) \\
grps' = grps \oplus \{g? \mapsto \{r?\} \cup (grps(g?))\} \\
usrs' = usrs \\
\hline
\end{array}
$$

$$AddGroupRight == AddNewGroupRight \lor AddExistingGroupRight$$

Por último, se modelan las operaciones que permiten determinar si un usuario es lector o escritor. Es necesario contemplar los casos en que el usuario posee el permiso individualmente o pertenece a un grupo que lo tiene.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsReaderUser}} \\
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \operatorname{dom}(usrs) \\
r \in usrs(u?) \\
ans! = yes \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsReaderGroup}} \\
\hline
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \mathrm{dom}(userGroups) \\
r \in \bigcup \mathrm{ran}((userGroups(u?)) \lhd grps) \\
ans! = yes \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsNotReaderNotInList}} \\
\hline
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \mathrm{dom}(userGroups) \\
u? \notin \mathrm{dom}(usrs) \\
r \notin \bigcup \mathrm{ran}((userGroups(u?)) \lhd grps) \\
ans! = no \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsNotReaderInList}} \\
\hline
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \mathrm{dom}(userGroups) \\
u? \in \mathrm{dom}(usrs) \\
r \notin usrs(u?) \\
r \notin \bigcup \mathrm{ran}((userGroups(u?)) \lhd grps) \\
ans! = no \\
\hline
\end{array}
$$

$IsNotReader == IsNotReaderNotInList \lor IsNotReaderInList$
$IsReader == IsReaderUser \lor IsReaderGroup \lor IsNotReader \lor UserDoesNotExist$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsWriterUser}} \\
\hline
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \mathrm{dom}(usrs) \\
w \in usrs(u?) \\
ans! = yes \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \mathit{IsWriterGroup} \_\_\_\_} \\
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \operatorname{dom}(userGroups) \\
w \in \bigcup \operatorname{ran}((userGroups(u?)) \lhd grps) \\
ans! = yes \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \mathit{IsNotWriterNotInList} \_\_\_\_} \\
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \operatorname{dom}(userGroups) \\
u? \notin \operatorname{dom}(usrs) \\
w \notin \bigcup \operatorname{ran}((userGroups(u?)) \lhd grps) \\
ans! = no \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \mathit{IsNotWriterInList} \_\_\_\_} \\
\Xi ACL \\
u? : USER \\
ans! : ANS \\
\hline
u? \in \operatorname{dom}(userGroups) \\
u? \in \operatorname{dom}(usrs) \\
w \notin usrs(u?) \\
w \notin \bigcup \operatorname{ran}((userGroups(u?)) \lhd grps) \\
ans! = no \\
\hline
\end{array}
$$

$$IsNotWriter == IsNotWriterNotInList \lor IsNotWriterInList$$
$$IsWriter == IsWriterUser \lor IsWriterGroup \lor IsNotWriter \lor UserDoesNotExist$$

3. Simulaciones

A continuación se presentan dos simulaciones realizadas sobre el modelo $\{log\}$. Como se utiliza el operador $\bigcup$, es necesario contar con la librería `setloglib`[1]. La primera es:

```
ug1(F) :- F = {[antonio, {g1,g2}], [locascio, {g1}]}.

aclInit(S0)                          &
ug1(UG)                              &
addUserRight(S0, antonio, w, UG, S1)  &
```

[1]Disponible en http://people.dmi.unipr.it/gianfranco.rossi/SETLOG/setloglib.slog

```
addUserRight(S1, locascio, r, UG, S2) &
addGroupRight(S2, g2, r, S3)          &
isReader(S3, antonio, R1, UG, S4)     &
isWriter(S4, locascio, R2, UG, S5).
```

En este caso, se espera que `R1` sea *yes*, ya que `antonio` es lector por formar parte de *g2*, y que *R2* sea igual a *no*. Como se puede ver a continuación, la simulación se comporta correctamente.

```
S0 = {[usrs,{}],[grps,{}]},
UG = {[antonio,{g1,g2}],[locascio,{g1}]},
S1 = {[usrs,{[antonio,{w}]}],[grps,{}]},
S2 = {[usrs,{[antonio,{w}],[locascio,{r}]}],[grps,{}]},
S3 = {[usrs,{[antonio,{w}],[locascio,{r}]}],[grps,{[g2,{r}]}]},
R1 = yes,
S4 = {[usrs,{[antonio,{w}],[locascio,{r}]}],[grps,{[g2,{r}]}]},
R2 = no,
S5 = {[usrs,{[antonio,{w}],[locascio,{r}]}],[grps,{[g2,{r}]}]}
Constraint: set(_N5), set(_N4), dom(_N3,_N2), g2 nin _N2, set(_N1), rel(_N3), set(_N2)
```

La segunda simulación es:

```
ug2(F) :- F = {[u1, {g1,g2}], [u2, {g1}], [u3, {}], [u4, {g3}]}.

aclInit(S0)                     &
ug2(UG)                         &
addUserRight(S0, u1, w, UG, S1) &
addUserRight(S1, u3, r, UG, S2) &
addGroupRight(S2, g3, r, S3)    &
addGroupRight(S3, g3, w, S4)    &
addGroupRight(S4, g2, r, S5)    &
isWriter(S5, u1, R1, UG, S6)    &
isReader(S6, u4, R2, UG, S7).
```

y tiene como primera respuesta:

```
S0 = {[usrs,{}],[grps,{}]},
UG = {[u1,{g1,g2}],[u2,{g1}],[u3,{}],[u4,{g3}]},
S1 = {[usrs,{[u1,{w}]}],[grps,{}]},
S2 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{}]},
S3 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{[g3,{r}]}]},
S4 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{[g3,{w}]}]},
S5 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{[g3,{w}],[g2,{r}]}]},
R1 = yes,
S6 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{[g3,{w}],[g2,{r}]}]},
R2 = no,
S7 = {[usrs,{[u1,{w}],[u3,{r}]}],[grps,{[g3,{w}],[g2,{r}]}]}
Constraint: set(_N11), set(_N10), dom(_N9,_N8), g3 nin _N8, set(_N7), rel(_N9),
set(_N8), dom(_N6,_N5), g3 nin _N5, set(_N4), rel(_N6), set(_N5), dom(_N3,_N2),
g2 nin _N2, set(_N1), rel(_N3), set(_N2)
```

4. Demostraciones con $\{log\}$

Primera demostración con $\{log\}$. En primer lugar se demuestra que *AddGroupRight* pereserva el invariante $grps \in \_ \pfun \_$. Esto es equivalente al teorema:

theorem GrpsIsPfun
$grps \in \_ \pfun \_ \land AddBGroupRight \Rightarrow grps' \in \_ \pfun \_$

el cual en $\{log\}$ se escribe de la siguiente forma:

```
S = {[usrs, Us],[grps, Gr]}                &
S_ = {[usrs, Us_],[grps, Gr_]}             &
pfun(Gr)                                   &
addGroupRight(S, G, P, S_)                 &
npfun(Gr_).
```

Segunda demostración con $\{log\}$. En este caso se demuestra que *AddUserRight* preserva el invariante *ACLInv*, es decir, que vale el siguiente teorema:

theorem AddUserRightPI
$ACLInv \land AddUserRight \Rightarrow ACLInv'$

cuya traducción a $\{log\}$ es:

```
S = {[usrs, Us],[grps, Gr]}                &
S_ = {[usrs, Us_],[grps, Gr_]}             &
dom(Us, DUs) & dom(UserGroups, DUserGroups) &
subset(DUs, DUserGroups)                   &
addUserRight(S,U,P,UserGroups,S_)          &
dom(Us_, DUs_)                             &
nsubset(DUs, DUserGroups).
```

5. Demostración con Z/EVES

Nuevamente se prueba que *AddUserRight* preserva el invariante *ACLInv*, pero ahora mediante Z/EVES.

theorem AddUserRightPI
$ACLInv \land AddUserRight \Rightarrow ACLInv'$

proof[*AddUserRightPI*]
invoke AddUserRight;
split AddNewUserRight;
cases;
prove by reduce;
next;
split AddExistingUserRight;
cases;
prove by reduce;
apply inPower;
instantiate e $==$ *u*?;
prove by reduce;
next;
prove by reduce;
next;
■

En el segundo caso de la prueba, cuando se agrega un permiso a un usuario existente, se debe aplicar la regla *inPower* y luego instanciarla en *u*? para probar que $u? \in dom(userGroups)$.

6. Casos de prueba

Se generan casos de prueba para la operación *AddGroupRight*. Para ello, se utilizan los siguientes comandos de Fastest:

```
loadspec ../tp/fastest.tex
selop AddGroupRight
genalltt
addtactic AddGroupRight_DNF_1 SP \notin g? \notin \dom grps
addtactic AddGroupRight_DNF_2 SP \in g? \in \dom grps
genalltt
addtactic AddGroupRight_DNF_1 FT r?
addtactic AddGroupRight_DNF_2 FT r?
genalltt
addtactic AddGroupRight_DNF_2 SP \cup \{ r? \} \cup grps~g?
genalltt
prunett
genalltca
```

En primer lugar, se aplica la táctica DNF, que separa la clases $AddGroup\_DNF_1$ y $AddGroup\_DNF_2$, que representan los casos en donde se aplican las operaciones *AddNewGroupRight* y *AddExistingGroupRight* respectivamente.

Luego, se aplican las particiones estándar de $\notin$ y $\in$ a estas subclases, particionando cada una nuevamente en dos. Además, se utiliza la táctica FT en todas las clases para generar para cada una el caso en que el permiso es de lectura y el caso de escritura.

Por último, se aplica la SP de $\cup$ en $AddGroup\_DNF_2$ con el fin de generar todas las formas posibles de *grps g*?.

Como resultado se obtienen casos abstractos de prueba para todas las hojas menos $AddGroupRight\_FT_9$ y $AddGroupRight\_FT_{10}$. Estas corresponden al caso en que $g? \notin dom(grps)$ ($AddNewGroupRight$) y $dom(grps) \neq \{\}$, es decir, $grps$ contiene información sobre grupos distintos a $g?$. Para estas dos clases Fastest no logra encontrar casos abstractos en el tiempo fijado.

- $AddGroupRight\_VIS$
  - $AddGroupRight\_DNF_1$
    - $AddGroupRight\_SP_3$
      - $AddGroupRight\_FT_9$
      - $AddGroupRight\_FT_{10}$
    - $AddGroupRight\_SP_4$
      - $AddGroupRight\_FT_{11}$
      - $AddGroupRight\_FT_{12}$
  - $AddGroupRight\_DNF_2$
    - $AddGroupRight\_SP_1$
      - $AddGroupRight\_FT_5$
        - $AddGroupRight\_SP_{15}$
        - $AddGroupRight\_SP_{16}$
        - $AddGroupRight\_SP_{17}$
        - $AddGroupRight\_SP_{19}$
      - $AddGroupRight\_FT_6$
        - $AddGroupRight\_SP_{23}$
        - $AddGroupRight\_SP_{24}$
        - $AddGroupRight\_SP_{25}$
        - $AddGroupRight\_SP_{27}$
    - $AddGroupRight\_SP_2$
      - $AddGroupRight\_FT_7$
        - $AddGroupRight\_SP_{31}$
        - $AddGroupRight\_SP_{32}$
        - $AddGroupRight\_SP_{33}$
        - $AddGroupRight\_SP_{35}$
      - $AddGroupRight\_FT_8$
        - $AddGroupRight\_SP_{39}$
        - $AddGroupRight\_SP_{40}$
        - $AddGroupRight\_SP_{41}$
        - $AddGroupRight\_SP_{43}$

Figura 1: Árbol de clases de prueba.

Finalmente, se muestran los casos abstractos de prueba generados.

$AddGroupRight\_FT\_9\_TCASE$

$AddGroupRight\_FT\_9$

$$g? = group1$$
$$r? = r$$
$$grps = \emptyset$$
$$usrs = \emptyset$$

$AddGroupRight\_FT\_10\_TCASE$

$AddGroupRight\_FT\_10$

$$g? = group1$$
$$r? = w$$
$$grps = \emptyset$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_15\_TCASE$

$AddGroupRight\_SP\_15$

$$g? = group1$$
$$r? = r$$
$$grps = \{(group1 \mapsto \emptyset)\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_16\_TCASE$

$AddGroupRight\_SP\_16$

$$g? = group1$$
$$r? = r$$
$$grps = \{(group1 \mapsto \{w\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_17\_TCASE$

$AddGroupRight\_SP\_17$

$$g? = group1$$
$$r? = r$$
$$grps = \{(group1 \mapsto \{r, w\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_19\_TCASE$

$AddGroupRight\_SP\_19$

$$g? = group1$$
$$r? = r$$
$$grps = \{(group1 \mapsto \{r\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_23\_TCASE$

$AddGroupRight\_SP\_23$

$$g? = group1$$
$$r? = w$$
$$grps = \{(group1 \mapsto \emptyset)\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_24\_TCASE$

$AddGroupRight\_SP\_24$

$$g? = group1$$
$$r? = w$$
$$grps = \{(group1 \mapsto \{r\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_25\_TCASE$

$AddGroupRight\_SP\_25$

$$g? = group1$$
$$r? = w$$
$$grps = \{(group1 \mapsto \{r, w\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_27\_TCASE$

$AddGroupRight\_SP\_27$

$$g? = group1$$
$$r? = w$$
$$grps = \{(group1 \mapsto \{w\})\}$$
$$usrs = \emptyset$$

$AddGroupRight\_SP\_31\_TCASE$
$AddGroupRight\_SP\_31$
$g? = group1$
$r? = r$
$grps = \{(group1 \mapsto \emptyset), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_32\_TCASE$
$AddGroupRight\_SP\_32$
$g? = group1$
$r? = r$
$grps = \{(group1 \mapsto \{w\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_33\_TCASE$
$AddGroupRight\_SP\_33$
$g? = group1$
$r? = r$
$grps = \{(group1 \mapsto \{r, w\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_35\_TCASE$
$AddGroupRight\_SP\_35$
$g? = group1$
$r? = r$
$grps = \{(group1 \mapsto \{r\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_39\_TCASE$
$AddGroupRight\_SP\_39$
$g? = group1$
$r? = w$
$grps = \{(group1 \mapsto \emptyset), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_40\_TCASE$
$AddGroupRight\_SP\_40$
$g? = group1$
$r? = w$
$grps = \{(group1 \mapsto \{r\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_41\_TCASE$
$AddGroupRight\_SP\_41$
$g? = group1$
$r? = w$
$grps = \{(group1 \mapsto \{r, w\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$

$AddGroupRight\_SP\_43\_TCASE$
$AddGroupRight\_SP\_43$
$g? = group1$
$r? = w$
$grps = \{(group1 \mapsto \{w\}), (group2 \mapsto \{r\})\}$
$usrs = \emptyset$