Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Ingenieria de Software

Trabajo Práctico
Verificación de Software

Sebastián Giulianelli

Rosario, Santa Fe, Argentina

3 de febrero de 2025

Índice

1. Problema elegido

Sistema de gestión de procesos

Un proceso puede estar en tres estados: **activo**, **pasivo** o **muerto**.

La señal Activate pasa el proceso de **pasivo** a **activo**; la señal Kill lo pasa de **pasivo** a **muerto**; la señal Suspend lo pasa de **activo** a **pasivo**. La señal KillNow hace que un proceso **activo** pase a estar **muerto**.

El estado inicial de un proceso es **pasivo**.

En un cierto sistema cada proceso se identifica por un identificador de proceso. El sistema presenta una interfaz que permite suscribir un proceso a una de las señales mencionadas en el párrafo anterior. Es decir, si una de las señales aparecen, el sistema la comunica a todos los procesos suscritos a esa señal (lo que implica que todos ellos cambian de estado de acuerdo a las reglas mencionadas más arriba). Puede suscribirse un proceso a más de una señal simplemente utilizando la interfaz varias veces. El usuario de la interfaz es quien determina el identificador para el proceso que se esté suscribiendo; el sistema rechazará la suscripción si el identificador ya está usado para la misma señal.

El sistema establece que uno de los procesos es el primario; cuando aún no se ha suscrito ningún proceso el primario es un proceso especial llamado idle. Cada vez que llega una señal Activate, el sistema elige aleatoriamente entre los procesos afectados el que será el nuevo proceso primario, si no encuentra, el proceso primario pasa a ser idle.

2. Especificación en Z

2.1. Designaciones

- i es un identificador de proceso $\approx i \in PID$
- El proceso se encuentra en estado pasivo $\approx pasivo$
- El proceso se encuentra en estado activo $\approx activo$
- El proceso se encuentra en estado muerto $\approx muerto$
- El proceso $i?$ está suscrito a la señal Activate $\approx i? \in suscritos(Activate)$
- El proceso $i?$ está suscrito a la señal Suspend $\approx i? \in suscritos(Suspend)$
- El proceso $i?$ está suscrito a la señal Kill $\approx i? \in suscritos(Kill)$
- El proceso $i?$ está suscrito a la señal KillNow $\approx i? \in suscritos(KillNow)$
- Existe un proceso especial $\approx idle$
- El proceso primario del sistema es $prim$, los estados de los procesos son ps y los procesos por señal estan en $suscritos$ $\approx SISTEMA(prim, ps, suscritos)$
- Se crea proceso con id $i?$ en el sistema $\approx SistNuevoProceso(i?)$
- Se suscribe proceso con id $i?$ a la señal $s?$ en el sistema $\approx SistSuscProceso(i?, s?)$
- Aparece la señal Activate en el sistema $\approx SistActivate$
- Aparece la señal Suspend en el sistema $\approx SistSuspend$
- Aparece la señal Kill en el sistema $\approx SistKill$
- Aparece la señal KillNow en el sistema $\approx SistKillNow$

2.2. Tipos y definiciones axiomáticas

Identificadores de proceso.

$[PID]$

Estados de los procesos y señales del sistema.

$Estado ::= activo \mid pasivo \mid muerto$

$Senal ::= Activate \mid Suspend \mid Kill \mid KillNow$

Estados finales de cada operación

$Msg ::= ok \mid errProcExist \mid errProcSuscrito \mid sinCambios \mid errProcNoExistente$

Tipo del sistema.

$SISTEMA$

$prim : PID$

$ps : PID \pfun Estado$

$suscritos : PID \leftrightarrow Senal$

Definición axiomática para el proceso especial.

$idle : PID$

2.3. Esquemas

Estado inicial del sistema.

$SISTEMAInit$

$SISTEMA$

$prim = idle$

$ps = \varnothing$

$suscritos = \varnothing$

Creación de un proceso en el sistema.

$$
\begin{array}{|l}
\hline
\textit{SistNuevoProcesoOK} \\
\Delta \textit{SISTEMA} \\
i? : \textit{PID} \\
\textit{rep}! : \textit{MSG} \\
\hline
i? \notin \mathrm{dom}\ \textit{ps} \\
\textit{prim}' = \textit{prim} \\
\textit{ps}' = \textit{ps} \cup \{ i? \mapsto \textit{pasivo} \} \\
\textit{suscritos}' = \textit{suscritos} \\
\textit{rep}! = \textit{ok} \\
\hline
\end{array}
$$

Proceso existente en el sistema.

$$
\begin{array}{|l}
\hline
\textit{SistProcesoExistenteERR} \\
\Xi \textit{SISTEMA} \\
i? : \textit{PID} \\
\textit{rep}! : \textit{MSG} \\
\hline
i? \in \mathrm{dom}\ \textit{ps} \\
\textit{rep}! = \textit{errProcExist} \\
\hline
\end{array}
$$

$$\textit{SistNuevoProceso} == \textit{SistNuevoProcesoOK} \lor \textit{SistProcesoExistenteERR}$$

Suscribir un proceso a una señal.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SistSuscProcesoOK}} \\
\Delta SISTEMA \\
i? : PID \\
s? : Senal \\
rep! : MSG \\
\hline
i? \in \mathrm{dom}\ ps \\
prim' = prim \\
ps' = ps \\
suscritos' = suscritos \cup \{i? \mapsto s?\} \\
rep! = ok \\
\hline
\end{array}
$$

Proceso no existente en sistema.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SistProcNoExistenteERR}} \\
\Xi SISTEMA \\
i? : PID \\
rep! : MSG \\
\hline
i? \notin \mathrm{dom}\ ps \\
rep! = errProcNoExistente \\
\hline
\end{array}
$$

$$SistSuscProceso == SistSuscProcesoOK \lor SistProcNoExistente$$

Sistema recibe la señal Activate con procesos a activar y por lo tanto, se actualiza el proceso primario.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ SistActivateOK\ }} \\
\Delta SISTEMA \\
rep! : MSG \\
\hline
\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \neq \varnothing \\
suscritos' = suscritos \\
ps' = ps \oplus (\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \times \{activo\}) \\
prim' \in \mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \\
rep! = ok \\
\hline
\end{array}
$$

Sistema recibe la señal Activate pero no hay procesos a activar y por lo tanto no hay cambios.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ SistActivateSC\ }} \\
\Xi SISTEMA \\
rep! : MSG \\
\hline
\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) = \varnothing \\
rep! = sinCambios \\
\hline
\end{array}
$$

$$SistActivate == SistActivateOK \lor SistActivateSC$$

Sistema recibe la señal Suspend y quedan procesos activos.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ SistSuspendOK\ }} \\
\Delta SISTEMA \\
rep! : MSG \\
\hline
\mathrm{dom}\ (ps \rhd \{activo\}) \setminus \mathrm{dom}\ (suscritos \rhd \{Suspend\}) \neq \varnothing \\
suscritos' = suscritos \\
ps' = ps \oplus (\mathrm{dom}\ (ps \rhd \{activo\}) \cap \mathrm{dom}\ (suscritos \rhd \{Suspend\}) \times \{pasivo\}) \\
prim' \in \mathrm{dom}\ (ps \rhd \{activo\}) \setminus \mathrm{dom}\ (suscritos \rhd \{Suspend\}) \\
rep! = ok \\
\hline
\end{array}
$$

Sistema recibe la señal Suspend y no quedan procesos activos.

$SistSuspendIdle$

$\Delta SISTEMA$

$rep! : MSG$

$dom\ (ps \rhd \{activo\}) \setminus dom\ (suscritos\ \rhd \{Suspend\}) = \varnothing$

$suscritos' = suscritos$

$ps' = ps \oplus (dom\ (ps \rhd \{activo\}) \times \{pasivo\})$

$prim' = idle$

$rep! = ok$

$SistSuspend == SistSuspendOK \lor SistSuspendIdle$

Sistema recibe la señal Kill.

$SistKill$

$\Delta SISTEMA$

$rep! : MSG$

$suscritos' = suscritos$

$ps' = ps \oplus (dom\ (suscritos\ \rhd \{Kill\}) \cap dom\ (ps \rhd \{pasivo\}) \times \{muerto\})$

$prim' = prim$

$rep! = ok$

Sistema recibe la señal KillNow y quedan procesos activos.

SistKillNowOK

$$\Delta SISTEMA$$
$$rep! : MSG$$

$$dom\ (ps \rhd \{activo\}) \setminus dom\ (suscritos\ \rhd \{KillNow\}) \neq \varnothing$$
$$suscritos' = suscritos$$
$$ps' = ps \oplus (dom\ (suscritos\ \rhd \{KillNow\}) \cap dom\ (ps \rhd \{activo\}) \times \{muerto\})$$
$$prim' \in dom\ (ps \rhd \{activo\}) \setminus dom\ (suscritos\ \rhd \{KillNow\})$$
$$rep! = ok$$

Sistema recibe la señal KillNow y no quedan procesos activos.

SistKillNowIdle

$$\Delta SISTEMA$$
$$rep! : MSG$$

$$dom\ (ps \rhd \{activo\}) \setminus dom\ (suscritos\ \rhd \{KillNow\}) = \varnothing$$
$$suscritos' = suscritos$$
$$ps' = ps \oplus (dom\ (ps \rhd \{activo\}) \times \{muerto\})$$
$$prim' = idle$$
$$rep! = ok$$

$$SistKillNow == SistKillNowOK \lor SistKillNowIdle$$

Se presentan a continuación las invariantes de estado:

- Inv1: Todos los procesos que están suscritos a las señales fueron registrados en el sistema.
- Inv2: El proceso primario siempre va a ser un proceso en estado activo o el proceso idle.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Inv}1} \\
\hline
\textit{SISTEMA} \\
\hline
\textit{dom suscritos} \subseteq \textit{dom ps} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Inv}2} \\
\hline
\textit{SISTEMA} \\
\hline
\textit{prim} \in \textit{dom } (\textit{ps} \rhd \{\textit{activo}\}) \cup \{\textit{idle}\} \\
\hline
\end{array}$$

3. Simulaciones sobre {log}

En estas simulaciones suscribimos diferentes procesos al sistema y enviamos diferentes señales y vemos como estos cambian de estado a medida que llegan las señales.

3.1. Primera simulación tipada:

```
sistemaInit(Prim, Ps, Susc)
& sistNuevoProc(Prim, Ps, Susc, W, Rep_o1, Prim1, Ps1, Susc1)
& sistNuevoProc(Prim1, Ps1, Susc1, Z, Rep_o2, Prim2, Ps2, Susc2)
& sistSuscProceso(Prim2, Ps2, Susc2, W, activate, Rep_o3, Prim3, Ps3, Susc3)
& sistSuscProceso(Prim3, Ps3, Susc3, Z, suspend, Rep_o4, Prim4, Ps4, Susc4)
& sistSuscProceso(Prim4, Ps4, Susc4, Z, activate, Rep_o5, Prim5, Ps5, Susc5)
& sistActivate(Prim5, Ps5, Susc5, Prim6, Ps6, Susc6, Rep_o6)
& sistSuspend(Ps6, Susc6, Prim7, Ps7, Susc7, Rep_o7)
& sistSuscProceso(Prim7, Ps7, Susc7, Z, kill, Rep_o8, Prim8, Ps8, Susc8)
& sistKill(Prim8, Ps8, Susc8, Prim9, Ps9, Susc9, Rep_o9)
& dec([Rep_o1, Rep_o2, Rep_o3, Rep_o4, Rep_o5, Rep_o6, Rep_o7, Rep_o8, Rep_o9], msg)
& dec([Prim, Prim1, Prim2, Prim3, Prim4, Prim5, Prim6, Prim7, Prim8, Prim9], pid)
& dec([Ps, Ps1, Ps2, Ps3, Ps4, Ps5, Ps6, Ps7, Ps8, Ps9], ps)
```

```
& dec([Susc, Susc1, Susc2, Susc3, Susc4, Susc5, Susc6, Susc7, Susc8, Susc9],
      suscritos)
& dec([W,Z], pid).
```

Primera solución:

```
Prim = pid:idle,
Ps = {},
Susc = {},
Rep_o1 = ok,
Prim1 = pid:idle,
Ps1 = {[W,pasivo]},
Susc1 = {},
Rep_o2 = ok,
Prim2 = pid:idle,
Ps2 = {[W,pasivo],[Z,pasivo]},
Susc2 = {},
Rep_o3 = ok,
Prim3 = pid:idle,
Ps3 = {[W,pasivo],[Z,pasivo]},
Susc3 = {[W,activate]},
Rep_o4 = ok,
Prim4 = pid:idle,
Ps4 = {[W,pasivo],[Z,pasivo]},
Susc4 = {[W,activate],[Z,suspend]},
Rep_o5 = ok,
Prim5 = pid:idle,
Ps5 = {[W,pasivo],[Z,pasivo]},
Susc5 = {[W,activate],[Z,suspend],[Z,activate]},
Prim6 = W,
Ps6 = cp({W,Z},{activo}),
Susc6 = {[W,activate],[Z,suspend],[Z,activate]},
```

```
Rep_o6 = ok,
Prim7 = W,
Ps7 = {[W,activo],[Z,pasivo]},
Susc7 = {[W,activate],[Z,suspend],[Z,activate]},
Rep_o7 = ok,
Rep_o8 = ok,
Prim8 = W,
Ps8 = {[W,activo],[Z,pasivo]},
Susc8 = {[W,activate],[Z,suspend],[Z,activate],[Z,kill]},
Prim9 = W,
Ps9 = {[W,activo],[Z,muerto]},
Susc9 = {[W,activate],[Z,suspend],[Z,activate],[Z,kill]},
Rep_o9 = ok
Constraint: subset(_N1,{W,Z}), Z nin _N1, set(_N1), W neq Z
```

En esta simulación vemos como los procesos W y Z ingresan al sistema en estado pasivo, y como a medida que se suscriben y llegan las señales estos pasan por diferentes estados hasta llegar a un estado activo como proceso primario para W y muerto para Z.

3.2. Segunda simulación no tipada:

```
sistemaInit(Prim, Ps, Susc)
& sistNuevoProc(Prim, Ps, Susc, A, Rep_o, Prim1, Ps1, Susc1)
& sistNuevoProc(Prim1, Ps1, Susc1, B, Rep_o1, Prim2, Ps2, Susc2)
& sistSuscProceso(Prim2, Ps2, Susc2, C, activate, Rep_o2, Prim3, Ps3, Susc3)
& sistSuscProceso(Prim3, Ps3, Susc3, B, kill, Rep_o3, Prim4, Ps4, Susc4)
& sistNuevoProc(Prim4, Ps4, Susc4, C, Rep_o4, Prim5, Ps5, Susc5)
& sistSuscProceso(Prim5, Ps5, Susc5, C, activate, Rep_o5, Prim6, Ps6, Susc6)
& sistSuscProceso(Prim6, Ps6, Susc6, A, kill, Rep_o6, Prim7, Ps7, Susc7)
& sistActivate(Prim7, Ps7, Susc7, Prim8, Ps8, Susc8, Rep_o7)
& sistSuspend(Ps8, Susc8, Prim9, Ps9, Susc9, Rep_o8)
& sistSuscProceso(Prim9, Ps9, Susc9, C, killNow, Rep_o9, Prim10, Ps10, Susc10)
```

```
& sistKill(Prim10, Ps10, Susc10, Prim11, Ps11, Susc11, Rep_o10)
& sistKillNow(Ps11, Susc11, Prim12, Ps12, Susc12, Rep_o11).
```

Primera solución:

```
Prim = pid:idle,
Ps = {},
Susc = {},
Rep_o = ok,
Prim1 = pid:idle,
Ps1 = {[A,pasivo]},
Susc1 = {},
Rep_o1 = ok,
Prim2 = pid:idle,
Ps2 = {[A,pasivo],[B,pasivo]},
Susc2 = {},
C = A,
Rep_o2 = ok,
Prim3 = pid:idle,
Ps3 = {[A,pasivo],[B,pasivo]},
Susc3 = {[A,activate]},
Rep_o3 = ok,
Prim4 = pid:idle,
Ps4 = {[A,pasivo],[B,pasivo]},
Susc4 = {[A,activate],[B,kill]},
Rep_o4 = errProcExist,
Prim5 = pid:idle,
Ps5 = {[A,pasivo],[B,pasivo]},
Susc5 = {[A,activate],[B,kill]},
Rep_o5 = ok,
Prim6 = pid:idle,
Ps6 = {[A,pasivo],[B,pasivo]},
```

```
Susc6 = {[B,kill],[A,activate]},
Rep_o6 = ok,
Prim7 = pid:idle,
Ps7 = {[A,pasivo],[B,pasivo]},
Susc7 = {[B,kill],[A,activate],[A,kill]},
Prim8 = A,
Ps8 = {[A,activo],[B,pasivo]},
Susc8 = {[B,kill],[A,activate],[A,kill]},
Rep_o7 = ok,
Prim9 = A,
Ps9 = {[A,activo],[B,pasivo]},
Susc9 = {[B,kill],[A,activate],[A,kill]},
Rep_o8 = ok,
Rep_o9 = ok,
Prim10 = A,
Ps10 = {[A,activo],[B,pasivo]},
Susc10 = {[B,kill],[A,activate],[A,kill],[A,killNow]},
Prim11 = A,
Ps11 = {[A,activo],[B,muerto]},
Susc11 = {[B,kill],[A,activate],[A,kill],[A,killNow]},
Rep_o10 = ok,
Prim12 = pid:idle,
Ps12 = {[B,muerto],[A,muerto]},
Susc12 = {[B,kill],[A,activate],[A,kill],[A,killNow]},
Rep_o11 = ok
Constraint: A neq B
```

Esta simulación no tipada permite apreciar como el sistema respeta la invariante del proceso primario. En un principio Prim es idle, luego pasa a ser el proceso A, y luego vuelve a ser idle al no haber mas procesos activos.

El resultado de las simulaciones simbólicas agrega restricciones sobre las variables W y Z en la primer simulación y para A y B en la otra. En estos caso se toman como variables diferentes, pero otras soluciones podrían tomarlas de manera diferente y por lo tanto el resultado cambiaría.

4. Demostraciones en {log}

Para realizar las demostraciones de las invariantes en {log} aprovechamos el generador de condiciones de prueba automático(VCG), para que estas se realicen de manera correcta primero tenemos que agregar las definiciones de las negaciones de las invariantes 1 y 2:

```
dec_p_type(inv1(ps, suscritos)).
inv1(Ps, Suscritos):-
 dec([DomS, Dom], set(pid))
 & dom(Suscritos, DomS)
 & dom(Ps, Dom)
 & subset(DomS,Dom).

% negacion de inv1
dec_p_type(n_inv1(ps, suscritos)).
n_inv1(Ps, Suscritos):-
 neg(let([DomS, Dom],
   dec([DomS, Dom], set(pid))
  & dom(Suscritos, DomS)
  & dom(Ps, Dom)
  , subset(DomS,Dom))).

invariant(inv2).
dec_p_type(inv2(pid, ps)).
inv2(Prim, Ps):-
  dec(PsA, ps)
  & dec([D,U], set(pid))
```

```
  & rres(Ps, {activo}, PsA)
  & dom(PsA, D)
  & un(D, {pid:idle}, U)
  & Prim in U.

% negacion de inv2
dec_p_type(n_inv2(pid, ps)).
n_inv2(Prim, Ps):-
 neg(let([PsA, D, U],
   dec(PsA, ps)
  & dec([D,U], set(pid))
  & rres(Ps, {activo}, PsA)
  & dom(PsA, D)
  & un(D, {pid:idle}, U)
  , Prim in U)).
```

Una vez completadas las negaciones procedemos a crear las condiciones de verificación:

```
consult('spec.pl').
vcg('spec.pl').
```

En este punto, el VCG crea el archivo spec-vc.pl. Antes de ejecutar la demostración hay que agregar una hipótesis en la prueba de la última operación para la invariante 2 así el demostrador no genera ERROR:

```
  sistKill_pi_inv2(Prim,Ps,Prim,Ps,Suscritos,Prim_,Ps_,Suscritos_,Rep_o) :-
pfun(Ps) & % Hipotesis agregada
neg(
  inv2(Prim,Ps) &
  sistKill(Prim,Ps,Suscritos,Prim_,Ps_,Suscritos_,Rep_o) implies
  inv2(Prim_,Ps_)
).
```

Luego de esto, la demostración se ejecuta con el siguiente comando:

```
consult('spec-vc.pl').
check_vcs_spec(60000, try(prover_all)).
```

Esta demostración puede demorar poco mas de tres minutos, pero una vez finalizada, quedan probadas las invariantes.

5. Teorema probado en Z/EVES

La propiedad que queremos demostrar es la siguiente invariante probada anteriormente con {log}:

$$\begin{array}{|l} \multicolumn{1}{l}{Inv1} \\ \hline SISTEMA \\ \hline prim = idle \lor prim \in \mathrm{dom}(ps \rhd \{activo\}) \\ \hline \end{array}$$

Por lo tanto, tendremos que probar que el sistema preserva dicho predicado en cada posible estado del sistema. Para ello probamos el siguiente teorema:

theorem Invariante1

$$\begin{aligned} &(Inv1 \land SistNuevoProceso \Rightarrow Inv1') \\ &\land (Inv1 \land SistSuscProceso \Rightarrow Inv1') \\ &\land (Inv1 \land SistActivate \Rightarrow Inv1') \\ &\land (Inv1 \land SistSuspend \Rightarrow Inv1') \\ &\land (Inv1 \land SistKillNow \Rightarrow Inv1') \\ &\land (Inv1 \land SistKill \Rightarrow Inv1') \end{aligned}$$

Para demostrarlo, lo descomponemos en subteoremas y los vamos probando uno a uno.

theorem InvNP

$Inv1 \land SistNuevoProceso \Rightarrow Inv1'$

proof[*InvNP*]

prove by reduce;

with normalization reduce;

prove;

■

theorem InvSP

$Inv1 \land SistSuscProceso \Rightarrow Inv1'$

proof[*InvSP*]

prove by reduce;

with normalization reduce;

■

theorem InvSA

$Inv1 \land SistActivate \Rightarrow Inv1'$

proof[$InvSA$]

with normalization reduce;

prove by reduce;

cases;

apply oplusDef to expression

$ps \oplus (\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \times \{activo\})$;

prove by reduce;

apply domDefinition to expression

$\mathrm{dom}\ ((\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \times \{activo\}) \rhd \{activo\})$;

prove by reduce;

next;

apply oplusDef to expression

$ps \oplus (\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \times \{activo\})$;

prove by reduce;

apply domDefinition to expression

$\mathrm{dom}\ ((\mathrm{dom}\ (suscritos \rhd \{Activate\}) \cap \mathrm{dom}\ (ps \rhd \{pasivo\}) \times \{activo\}) \rhd \{activo\})$;

prove by reduce;

■

theorem InvSS

$Inv1 \land SistSuspend \Rightarrow Inv1'$

proof[*InvSS*]

prove by reduce;

apply inDom to predicate

$prim' \in \mathrm{dom}\ (ps \rhd \{activo\})$;

apply inDom to predicate

$prim' \in \mathrm{dom}\ ((ps \oplus (\mathrm{dom}\ (suscritos \rhd \{Suspend\}) \cap \mathrm{dom}\ (ps \rhd \{activo\}) \times \{pasivo\})) \rhd \{activo\})$;

apply oplusDef to expression

$ps \oplus (\mathrm{dom}\ (suscritos \rhd \{Suspend\}) \cap \mathrm{dom}\ (ps \rhd \{activo\}) \times \{pasivo\})$;

prove by reduce;

■

theorem InvSKN

$Inv1 \land SistKillNow \Rightarrow Inv1'$

proof[*InvSKN*]

prove by reduce;

apply inDom to predicate

$prim' \in \mathrm{dom}\ (ps \rhd \{activo\})$;

apply inDom to predicate

$prim' \in \mathrm{dom}\ ((ps \oplus (\mathrm{dom}\ (suscritos \rhd \{KillNow\}) \cap \mathrm{dom}\ (ps \rhd \{activo\}) \times \{muerto\})) \rhd \{activo\})$;

apply oplusDef to expression

$ps \oplus (\mathrm{dom}\ (suscritos \rhd \{KillNow\}) \cap \mathrm{dom}\ (ps \rhd \{activo\}) \times \{muerto\})$;

prove by reduce;

■

theorem InvSK

$Inv1 \land SistKill \Rightarrow Inv1'$

Para probar este teorema fue necesario definir un Lemma:

theorem rule Lemma $[X, Y]$

$\forall A : X;\ F : X \rightarrowtail\!\!\!\!\!\!+\ Y;\ x, y : Y \mid x \neq y \land (A, x) \in F \bullet (A, y) \notin F$

proof[*Lemma*]

prove by reduce;

apply pfunDef to predicate

$F \ \in \ X \ ⇸ \ Y;$

split $F \ \in \ X \ \leftrightarrow \ Y;$

prove by reduce;

apply inPower to predicate

$F \ {}^{\sim}\ [X,\ Y] \ \fatsemi \ [Y,\ X,\ Y]\ F \ \in \ \mathbb{P} \ (\mathrm{id} \ \ Y);$

apply compDef to expression

$F \ {}^{\sim}\ [X,\ Y] \ \fatsemi \ [Y,\ X,\ Y]\ F;$

split $\neg \ \ (A,\ y) \ \in \ \ F;$

prove by reduce;

instantiate $e \ == \ (x,\ y);$

prove by reduce;

instantiate $y\_\_0 \ == \ A;$

prove by reduce;

prove by reduce;

■

Finalmente podemos probar el último subteorema para finalizar la demostración.

proof[$InvSK$]

prove by reduce;

apply inDom to predicate

$prim \in \operatorname{dom} (ps \rhd \{activo\})$;

apply inDom to predicate

$prim' \in \operatorname{dom} ((ps \oplus (\operatorname{dom} (suscritos \rhd \{Kill\}) \cap \operatorname{dom} (ps \rhd \{pasivo\}) \times \{muerto\})) \rhd \{activo\})$;

apply oplusDef to expression

$ps \oplus (\operatorname{dom} (suscritos \rhd \{Kill\}) \cap \operatorname{dom} (ps \rhd \{pasivo\}) \times \{muerto\})$;

prove by reduce;

split $prim' \in \operatorname{dom} (ps \rhd \{pasivo\})$;

prove by reduce;

apply inDom to predicate

$prim' \in \operatorname{dom} (ps \rhd \{pasivo\})$;

prove by reduce;

use $Lemma[PID, Estado][F := ps, A := prim', x := pasivo, y := activo]$;

prove by reduce;

■

6. Comandos FASTEST

Los comandos ejecutados en Fastest para generar el árbol de pruebas son los siguientes.

```
Fastest> loadspec specft.tex

Loading specification..

Specification loaded.

Fastest> selop SistSuscProceso

Fastest> genalltt

Fastest> addtactic SistSuscProceso FT s?

Fastest> genalltt
```

En este caso trabajamos sobre la operación SistSuscProceso, donde el sistema suscribe un proceso i? a alguna señal s?. Hasta aquí creamos dos niveles de altura en el árbol de pruebas, el primero a partir de la táctica DNF en donde se separan los casos donde el proceso no pertenece a la lista de procesos y el otro donde si lo hace. El segundo nivel se genera a partir de la táctica FT en donde se asigna una hoja para cada valor de Senal sobre s?.

Luego, generamos los los casos de test y vemos el árbol resultante.

```
Fastest> genalltca
Fastest> showtt
```

```
SistSuscProceso_VIS
  !______SistSuscProceso_DNF_1
  |   !______SistSuscProceso_FT_1
  |   |   !______SistSuscProceso_FT_1_TCASE
  |   |
  |   !______SistSuscProceso_FT_2
  |   |   !______SistSuscProceso_FT_2_TCASE
  |   |
  |   !______SistSuscProceso_FT_3
  |   |   !______SistSuscProceso_FT_3_TCASE
  |   |
  |   !______SistSuscProceso_FT_4
  |      !______SistSuscProceso_FT_4_TCASE
  |
  |
  !______SistSuscProceso_DNF_2
     !______SistSuscProceso_FT_5
     |   !______SistSuscProceso_FT_5_TCASE
     |
     !______SistSuscProceso_FT_6
```

```
|    !______SistSuscProceso_FT_6_TCASE
|
!______SistSuscProceso_FT_7
|    !______SistSuscProceso_FT_7_TCASE
|
!______SistSuscProceso_FT_8
   !______SistSuscProceso_FT_8_TCASE
```

Se puede observar que Fastest genera con éxito todos los casos de prueba.

7. Esquemas Z para casos de prueba

SistSuscProceso_FT_1_TCASE

$SistSuscProceso\_FT\_1$

$ps = \{(pID1 \mapsto muerto)\}$

$suscritos = \{(pID1 \mapsto Suspend), (pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Activate$

$prim = pID1$

SistSuscProceso_FT_2_TCASE

$SistSuscProceso\_FT\_2$

$ps = \{(pID1 \mapsto muerto)\}$

$suscritos = \{(pID1 \mapsto Suspend), (pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Suspend$

$prim = pID1$

$SistSuscProceso\_FT\_3\_TCASE$

$SistSuscProceso\_FT\_3$

$ps = \{(pID1 \mapsto muerto)\}$

$suscritos = \{(pID1 \mapsto Suspend), (pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Kill$

$prim = pID1$

$SistSuscProceso\_FT\_4\_TCASE$

$SistSuscProceso\_FT\_4$

$ps = \{(pID1 \mapsto muerto)\}$

$suscritos = \{(pID1 \mapsto Suspend), (pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = KillNow$

$prim = pID1$

$SistSuscProceso\_FT\_5\_TCASE$

$SistSuscProceso\_FT\_5$

$ps = \varnothing$

$suscritos = \{(pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Activate$

$prim = pID1$

$SistSuscProceso\_FT\_6\_TCASE$

$SistSuscProceso\_FT\_6$

$ps = \varnothing$

$suscritos = \{(pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Suspend$

$prim = pID1$

$SistSuscProceso\_FT\_7\_TCASE$

$SistSuscProceso\_FT\_7$

$ps = \varnothing$

$suscritos = \{(pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = Kill$

$prim = pID1$

$SistSuscProceso\_FT\_8\_TCASE$

$SistSuscProceso\_FT\_8$

$ps = \varnothing$

$suscritos = \{(pID1 \mapsto Kill)\}$

$i? = pID1$

$s? = KillNow$

$prim = pID1$