

Verificación de software

Trabajo Práctico Final - Ingeniería de Software II

Autor:

Juan Ignacio Farizano

Departamento de Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Rosario, Santa Fe, Argentina

24 de febrero de 2025

1. Enunciado del problema

Queremos especificar el funcionamiento del sistema *Mis Estudiantes* que utilizan equipos directivos de escuelas primarias y secundarias para inscribir y reinscribir alumnos (por ej. promoción de grado, repitente, etc) en la base de datos del Ministerio de Educación de la provincia de Buenos Aires, lo descrito es similar al sistema existente pero simplificado para mantener una dificultad razonable para este trabajo práctico.

Un directivo desea inscribir o reinscribir a un alumno en su escuela. A cada alumno se le asigna una inscripción actual que consiste en un grado escolar y un estado que representa una de las tres siguientes situaciones:

- **Inscripto:** el alumno fue inscripto y es habilitado a cursar el grado registrado
- **Promovido:** el alumno cumplió los requisitos para promocionar el grado al cual se encuentra inscripto y es habilitado a ser inscripto en el grado siguiente, o a graduarse si se encontraba en el 12avo grado
- **Repitente:** el alumno no cumplió los requisitos para promocionar de grado y debe ser inscripto en el mismo grado

Se deben especificar las siguientes operaciones:

1. **Inscribir alumno:** se registra por primera vez a un alumno y se lo inscribe a primer grado
2. **Reinscribir alumno:** se inscribe un alumno previamente registrado al grado que corresponda según su último estado de inscripción
3. **Cerrar inscripción:** la inscripción actual es cerrada y se registra una promoción o repitencia
4. **Consultar graduado:** se desea consultar si un alumno se graduó

No se diferencia entre primaria y secundaria, registramos desde 1er grado hasta 12avo, el último del secundario. Los requisitos para promocionar de grado se encuentran por fuera del sistema y no deben ser tenidos en cuenta.

2. Especificación en Z

Para comenzar, damos las designaciones:

$alumno$ es un alumno $\approx alumno \in ALUMNO$

$grado$ es un grado correspondiente a un año escolar $\approx grado \in GRADO$

$estado$ es un estado de inscripción $\approx estado \in ESTADO$

rep es un mensaje de respuesta $\approx rep \in REP$

Estado de inscripción actual correspondiente al alumno $alumno \approx inscripciones(alumno)$

Luego, definimos los tipos básicos y enumerados utilizados:

$[ALUMNO]$

$GRADO == \mathbb{N}$

$ESTADO ::= inscripto \mid promueve \mid repite$

$REP ::= alumnoEsGraduado \mid alumnoNoEsGraduado \mid alumnoNoEncontrado$

Además, damos la siguiente definición axiomática que define el último grado que debe cursar y promocionar un alumno para graduarse:

$maximoGrado : GRADO$
$maximoGrado = 12$

Definimos nuestro estado, donde *registrados* es el conjunto de alumnos que fueron registrados en el sistema, e *inscripciones* es la asignación de cada alumno a su inscripción actual, siendo esta una tupla compuesta por un grado y un estado de inscripción según lo requerido.

$MisEstudiantes$
$registrados : \mathbb{P} ALUMNO$
$inscripciones : ALUMNO \rightarrow (GRADO \times ESTADO)$

Definimos dos invariantes de estado. La primera indica que todos los alumnos registrados en el sistema deben tener un estado de inscripción actual y viceversa.

$InscripcionesInv$
$MisEstudiantes$
$dom\ inscripciones = registrados$

La segunda invariante representa que para todo alumno registrado su grado de inscripción no debe superar el máximo grado posible.

<i>MaximoGradoInv</i>
<i>MisEstudiantes</i>
$\forall \text{alumno} : \text{registrados} \bullet (\text{inscripciones alumno}).1 \leq \text{maximoGrado}$

Definimos el estado inicial del sistema:

<i>MisEstudiantesInicial</i>
<i>MisEstudiantes</i>
$\text{registrados} = \emptyset$
$\text{inscripciones} = \emptyset$

La primera operación a especificar es *InscribirAlumno*. En el caso exitoso se debe dar la condición de que el alumno especificado no se encuentre registrado en el sistema, luego se lo registra y se lo inscribe a primer grado.

<i>InscribirAlumnoOk</i>
$\Delta \text{MisEstudiantes}$
$\text{alumno?} : \text{ALUMNO}$
$\text{alumno?} \notin \text{registrados}$
$\text{inscripciones}' = \text{inscripciones} \cup \{\text{alumno?} \mapsto (1, \text{inscripto})\}$
$\text{registrados}' = \text{registrados} \cup \{\text{alumno?}\}$

El único caso de error de esta operación se da cuando el alumno ya se encontraba registrado en el sistema.

<i>InscribirAlumnoRegistradoE</i>
$\Xi \text{MisEstudiantes}$
$\text{alumno?} : \text{ALUMNO}$
$\text{alumno?} \in \text{registrados}$

$$\text{InscribirAlumno} == \text{InscribirAlumnoOk} \vee \text{InscribirAlumnoRegistradoE}$$

La siguiente operación a modelar es *ReinscribirAlumno*, esta cuenta con dos casos exitosos. El primero se da cuando el alumno se encuentra en estado de promoción y se lo inscribe en el siguiente grado a menos que haya promovido el último grado; el segundo caso exitoso se da cuando el alumno se encuentra en estado de repitencia y se lo debe inscribir en el mismo grado que se encontraba.

ReinscribirAlumnoPromovidoOk

Δ MisEstudiantes

alumno? : ALUMNO

alumno? \in registrados

$(inscripciones\ alumno?).1 < maximoGrado$

$(inscripciones\ alumno?).2 = promueve$

$inscripciones' = inscripciones \oplus \{alumno? \mapsto ((inscripciones\ alumno?).1 + 1, inscripto)\}$

$registrados' = registrados$

ReinscribirAlumnoRepitenteOk

Δ MisEstudiantes

alumno? : ALUMNO

alumno? \in registrados

$(inscripciones\ alumno?).1 \leq maximoGrado$

$(inscripciones\ alumno?).2 = repite$

$inscripciones' = inscripciones \oplus \{alumno? \mapsto ((inscripciones\ alumno?).1, inscripto)\}$

$registrados' = registrados$

Los casos de error posibles son 3: estos se dan cuando el alumno no fue registrado, si ya se ha graduado o si se intenta reinscribirlo cuando ya se encontraba inscripto.

ReinscribirAlumnoNoEncontradoE

Ξ MisEstudiantes

alumno? : ALUMNO

alumno? \notin registrados

ReinscribirAlumnoGraduadoE

Ξ *MisEstudiantes*

alumno? : *ALUMNO*

alumno? \in *registrados*

$((\text{inscripciones } \text{alumno?})).1 = \text{maximoGrado}$

$((\text{inscripciones } \text{alumno?})).2 = \text{promueve}$

ReinscribirAlumnoDobleInscripE

Ξ *MisEstudiantes*

alumno? : *ALUMNO*

alumno? \in *registrados*

$(\text{inscripciones } \text{alumno?}).2 = \text{inscripto}$

$\text{ReinscribirAlumnoE} == \text{ReinscribirAlumnoGraduadoE} \vee \text{ReinscribirAlumnoDobleInscripE}$
 $\vee \text{ReinscribirAlumnoNoEncontradoE}$

$\text{ReinscribirAlumnoOk} == \text{ReinscribirAlumnoPromovidoOk} \vee \text{ReinscribirAlumnoRepitenteOk}$

$\text{ReinscribirAlumno} == \text{ReinscribirAlumnoOk} \vee \text{ReinscribirAlumnoE}$

La tercera operación es *CerrarInscripción*, en el caso exitoso se debe dar que el alumno se encuentra registrado y su estado de inscripción sea *inscripto*, luego este estado es reemplazado por *promueve* o *repite* según indique el usuario y el grado se mantiene sin modificaciones. Se permite volver a cerrar una inscripción cerrada para permitir modificaciones o correcciones de errores del usuario.

CerrarInscripcionOk

Δ *MisEstudiantes*

alumno? : *ALUMNO*

estado? : *ESTADO*

alumno? \in *registrados*

estado? \neq *inscripto*

$\text{inscripciones}' = \text{inscripciones} \oplus \{\text{alumno?} \mapsto ((\text{inscripciones } \text{alumno?}).1, \text{estado?})\}$

$\text{registrados}' = \text{registrados}$

Los casos de error son dos: se intenta cerrar una inscripción modificando su estado por *inscripto* o el alumno indicado no se encuentra registrado en el sistema.

$CerrarInscripcionEstadoInvalidoE$
$\Xi MisEstudiantes$
$estado? : ESTADO$
$estado? = inscripto$

$CerrarInscripcionAlumnoNoEncontradoE$
$\Xi MisEstudiantes$
$alumno? : ALUMNO$
$alumno? \notin registrados$

$$CerrarInscripcionE == CerrarInscripcionEstadoInvalidoE \vee CerrarInscripcionAlumnoNoEncontradoE$$

$$CerrarInscripcion == CerrarInscripcionOk \vee CerrarInscripcionE$$

La última operación es *AlumnoEsGraduado* donde hay tres casos posible y cada uno tiene un mensaje de respuesta posible: el alumno está registrado y es graduado, el alumno está registrado y no es graduado o el alumno no está registrado en el sistema.

$AlumnoEsGraduadoSiOk$
$\Xi MisEstudiantes$
$alumno? : ALUMNO$
$rep! : REP$
$alumno? \in registrados$
$(inscripciones\ alumno?).1 = maximoGrado$
$(inscripciones\ alumno?).2 = promueve$
$rep! = alumnoEsGraduado$

AlumnoEsGraduadoNoOk

$\exists \text{MisEstudiantes}$

alumno? : *ALUMNO*

rep! : *REP*

alumno? \in *registrados*

$(\text{inscripciones } \text{alumno?}).1 \neq \text{maximoGrado} \vee (\text{inscripciones } \text{alumno?}).2 \neq \text{promueve}$

rep! = *alumnoNoEsGraduado*

AlumnoEsGraduadoNoEncontradoE

$\exists \text{MisEstudiantes}$

alumno? : *ALUMNO*

rep! : *REP*

alumno? \notin *registrados*

rep! = *alumnoNoEncontrado*

$\text{AlumnoEsGraduadoOk} == \text{AlumnoEsGraduadoSiOk} \vee \text{AlumnoEsGraduadoNoOk}$

$\text{AlumnoEsGraduado} == \text{AlumnoEsGraduadoOk} \vee \text{AlumnoEsGraduadoNoEncontradoE}$

3. {log}

La traducción de la especificación *Z* a un programa *{log}* se puede encontrar en el archivo **misestudiantes.pl**, para ejecutar este programa no es necesario consultar ninguna librería.

3.1. Simulaciones

Para realizar las siguientes simulaciones se cargó el programa con el type checker activado y una vez verificado que el programa tipa correctamente se desactivó para simplificar la ejecución de las simulaciones.

Primera simulación

`misEstudiantesInicial(R0, IO) &`


```

inscribirAlumno(R0, I0, pepe, R1, I1) &
cerrarInscripcion(R1, I1, pepe, promueve, R2, I2) &
reinscribirAlumno(R2, I2, pepe, R3, I3) &
cerrarInscripcion(R3, I3, pepe, repite, R4, I4) &
cerrarInscripcion(R4, I4, pepe, promueve, R5, I5) &
inscribirAlumno(R5, I5, pepe, R6, I6) &
alumnoEsGraduado(R6, I6, pepe, Rep1, R6, I6) &
alumnoEsGraduado(R6, I6, pepito, Rep2, R6, I6).

```

El objetivo de esta simulación es probar los usos habituales del programa, comenzando desde el estado inicial se inscribe un alumno *Pepe* y se da una serie de inscripciones cerradas y reinscripciones, incluso una ocasión donde se cerró una inscripción como repitente y luego se modificó a promoción (por ejemplo, el alumno puede haber recuperado materias luego del cierre o se corrigió una equivocación al cerrar por primera vez). Además se consulta si *Pepe* está graduado después de promover segundo grado, lo cual no sería posible, y por último se consulta por un alumno que no se encuentra registrado.

```

R0 = {},
I0 = {},
R1 = {pepe},
I1 = {[pepe,[1,inscripto]]},
R2 = {pepe},
I2 = {[pepe,[1,promueve]]},
R3 = {pepe},
I3 = {[pepe,[2,inscripto]]},
R4 = {pepe},
I4 = {[pepe,[2,repite]]},
R5 = {pepe},
I5 = {[pepe,[2,promueve]]},
R6 = {pepe},
I6 = {[pepe,[2,promueve]]},
Rep1 = alumnoNoEsGraduado,
Rep2 = alumnoNoEncontrado

```

Obtuvimos el resultado esperado, los cambios de estado fueron satisfactorios y la operación de consulta funcionó correctamente.

Segunda simulación

```
misEstudiantesInicial(R0, I0) &
inscribirAlumno(R0, I0, pepe, R1, I1) &
cerrarInscripcion(R1, I1, pepe, promueve, R2, I2) &
reinscribirAlumno(R2, I2, pepe, R3, I3) &
cerrarInscripcion(R3, I3, pepe, promueve, R4, I4) &
reinscribirAlumno(R4, I4, pepe, R5, I5) &
cerrarInscripcion(R5, I5, pepe, promueve, R6, I6) &
reinscribirAlumno(R6, I6, pepe, R7, I7) &
cerrarInscripcion(R7, I7, pepe, promueve, R8, I8) &
reinscribirAlumno(R8, I8, pepe, R9, I9) &
cerrarInscripcion(R9, I9, pepe, promueve, R10, I10) &
reinscribirAlumno(R10, I10, pepe, R11, I11) &
cerrarInscripcion(R11, I11, pepe, promueve, R12, I12) &
reinscribirAlumno(R12, I12, pepe, R13, I13) &
cerrarInscripcion(R13, I13, pepe, promueve, R14, I14) &
reinscribirAlumno(R14, I14, pepe, R15, I15) &
cerrarInscripcion(R15, I15, pepe, promueve, R16, I16) &
reinscribirAlumno(R16, I16, pepe, R17, I17) &
cerrarInscripcion(R17, I17, pepe, promueve, R18, I18) &
reinscribirAlumno(R18, I18, pepe, R19, I19) &
cerrarInscripcion(R19, I19, pepe, promueve, R20, I20) &
reinscribirAlumno(R20, I20, pepe, R21, I21) &
cerrarInscripcion(R21, I21, pepe, promueve, R22, I22) &
reinscribirAlumno(R22, I22, pepe, R23, I23) &
alumnoEsGraduado(R23, I23, pepe, Rep1, R23, I23) &
cerrarInscripcion(R23, I23, pepe, promueve, R24, I24) &
alumnoEsGraduado(R24, I24, pepe, Rep2, R24, I24) &
reinscribirAlumno(R24, I24, pepe, R25, I25) &
alumnoEsGraduado(R25, I25, pepe, Rep3, R25, I25) &
cerrarInscripcion(R25, I25, pepe, repite, R26, I26) &
alumnoEsGraduado(R26, I26, pepe, Rep4, R26, I26).
```

Esta segunda simulación consiste en inscribir al alumno *Pepe* partiendo del estado inicial y realizar el ciclo cerrar inscripción/reinscribir hasta que se encuentre cursando el último grado, una vez aquí se

consulta si está graduado a lo que se debería obtener una respuesta negativa, se cierra la inscripción con una promoción para graduarlo, se consulta de nuevo y el resultado esta vez debería ser positivo. Luego, se intenta reinscribir al alumno graduado pero esto no es permitido, por lo que consultando de nuevo se debería ver que mantiene su estado de graduado. Este estado de graduado es perdido solo cuando se decide modificar el estado del último cierre de inscripción por una repitencia, y consultado por última vez el resultado sería negativo.

```

R0 = {},
I0 = {},
R1 = {pepe},
I1 = {[pepe,[1,inscripto]]},
R2 = {pepe},
I2 = {[pepe,[1,promueve]]},
.
.
.
I22 = {[pepe,[11,promueve]]},
R23 = {pepe},
I23 = {[pepe,[12,inscripto]]},
Rep1 = alumnoNoEsGraduado,
R24 = {pepe},
I24 = {[pepe,[12,promueve]]},
Rep2 = alumnoEsGraduado,
R25 = {pepe},
I25 = {[pepe,[12,promueve]]},
Rep3 = alumnoEsGraduado,
R26 = {pepe},
I26 = {[pepe,[12,repite]]},
Rep4 = alumnoNoEsGraduado

```

Los resultados fueron acortados ya que contaban con muchos estados intermedios similares. Se obtuvieron los resultados esperados tanto en los cambios de estado satisfactorios, usos inválidos de las operaciones que no modificaron el estado y consultas respondidas correctamente.

3.2. Demostración automática de propiedades

Invocando el generador de condiciones de verificación (VCG) con el comando *vcg('misestudiantes.pl')*, es generado el archivo **misestudiantes-vc.pl**, consultando este archivo y ejecutando el comando *check_vcs_misestudiantes*, es posible descargar las condiciones de verificación.

En un principio todas las condiciones son verificadas correctamente excepto por la condición **inscribirAlumno_pi_inscripcionesPfunInv**, por lo cual fue necesario agregar la hipótesis **inscripcionesInv(Registrados, Inscripciones)** y luego esta condición fue descargada exitosamente. No fue necesario utilizar el comando *findh*.

4. Demostración de preservación de invariantes en Z/Eves

En el archivo **misestudiantes_zeves.tex** se puede encontrar la especificación vista anteriormente con la adición de la definición del teorema **InscribirAlumnoPI**, el cual verifica que la operación **InscribirAlumno** preserva el invariante de estado **InscripcionesInv**.

theorem InscribirAlumnoPI

$$InscripcionesInv \wedge InscribirAlumno \Rightarrow InscripcionesInv'$$

Utilizando la interfaz gráfica *Z/Eves* se importó el archivo con la especificación y se realizó la siguiente

demostración del teorema:

```
proof[InscribirAlumnoPI]  
  invoke InscribirAlumno;  
  split InscribirAlumnoOk;  
  simplify;  
  cases;  
  invoke InscribirAlumnoOk;  
  invoke IncripcionesInv;  
  equality substitute;  
  reduce;  
  next;  
  invoke InscribirAlumnoRegistradoE;  
  invoke  $\exists$  MisEstudiantes;  
  reduce;  
  next;
```

■

Agregando la prueba al archivo fue posible confirmar la misma utilizando la interfaz de consola.

5. Generación de casos de prueba con Fastest

En el archivo **misestudiantes_fastest.tex** se puede encontrar la especificación con su sintaxis adaptada a la aceptada por Fastest, esta es idéntica a la vista anteriormente en este informe pero debido a que la herramienta no soporta sinónimos de tipos las ocurrencias de GRADO fueron reemplazadas por N. Utilizando esta herramienta se generarán casos de prueba para la operación *ReinscribirAlumno*, los comandos ejecutados para generar los casos de prueba fueron los siguientes:

```
loadspect misestudiantes_fastest.tex  
selop CerrarInscripcion  
genalltt  
addtactic CerrarInscripcion_DNF_1 FT estado?  
addtactic CerrarInscripcion_DNF_2 FT estado?  
addtactic CerrarInscripcion_DNF_1 SP \oplus inscripciones  
  \oplus \{ alumno? \mapsto ( ( inscripciones(alumno?) ).1 , estado? ) \}  
addtactic CerrarInscripcion_DNF_3 SP \notin alumno? \notin registrados
```

```
genalltt
genalltca
```

De esta forma se carga la especificación, se selecciona la operación *CerrarInscripción* y luego son aplicadas las tácticas de testing. Lo primero que se hace es aplicar la táctica *Disjunctive Normal Form (DNF)* (utilizada automáticamente al ejecutar el comando *genalltt*) la cual separa la operación en los tres esquemas que la componen:

1. *CerrarInscripcion_DNF_1* corresponde al esquema *CerrarInscripcionOk*
2. *CerrarInscripcion_DNF_2* corresponde al esquema *CerrarInscripcionEstadoInvalidoE*
3. *CerrarInscripcion_DNF_3* corresponde al esquema *CerrarInscripcionAlumnoNoEncontradoE*

Luego para los primeros dos esquemas, los cuales reciben una variable de tipo *ESTADO* como entrada se les aplica la táctica de *Free Type (FT)* para generar un caso para cada valor posible de este tipo (siendo estos *inscripto*, *promueve* o *repite*). Por último, se aplican las tácticas de *Standard Partition (SP)* sobre los operadores \oplus y \notin . Obtenemos el siguiente árbol:

CerrarInscripcion_VIS

```
!_____CerrarInscripcion_DNF_1
|      !_____CerrarInscripcion_FT_1
|      |      !_____CerrarInscripcion_SP_4
|      |      !_____CerrarInscripcion_SP_5
|      |      !_____CerrarInscripcion_SP_6
|      |      !_____CerrarInscripcion_SP_7
|      |      !_____CerrarInscripcion_SP_8
|      |      !_____CerrarInscripcion_SP_9
|      |      !_____CerrarInscripcion_SP_10
|      |      !_____CerrarInscripcion_SP_11
|      |
|      !_____CerrarInscripcion_FT_2
|      |      !_____CerrarInscripcion_SP_12
|      |      !_____CerrarInscripcion_SP_13
|      |      !_____CerrarInscripcion_SP_14
|      |      !_____CerrarInscripcion_SP_15
|      |      |      !_____CerrarInscripcion_SP_15_TCASE
|      |      |
```

```

|      |      !_____CerrarInscripcion_SP_16
|      |      |      !_____CerrarInscripcion_SP_16_TCASE
|      |      |
|      |      !_____CerrarInscripcion_SP_17
|      |      !_____CerrarInscripcion_SP_18
|      |      !_____CerrarInscripcion_SP_19
|      |
|      !_____CerrarInscripcion_FT_3
|      !_____CerrarInscripcion_SP_20
|      !_____CerrarInscripcion_SP_21
|      !_____CerrarInscripcion_SP_22
|      !_____CerrarInscripcion_SP_23
|      |      !_____CerrarInscripcion_SP_23_TCASE
|      |
|      !_____CerrarInscripcion_SP_24
|      |      !_____CerrarInscripcion_SP_24_TCASE
|      |
|      !_____CerrarInscripcion_SP_25
|      !_____CerrarInscripcion_SP_26
|      !_____CerrarInscripcion_SP_27
|
|
!_____CerrarInscripcion_DNF_2
|      !_____CerrarInscripcion_FT_28
|      |      !_____CerrarInscripcion_FT_28_TCASE
|      |
|      !_____CerrarInscripcion_FT_29
|      !_____CerrarInscripcion_FT_30
|
!_____CerrarInscripcion_DNF_3
|      !_____CerrarInscripcion_SP_31
|      |      !_____CerrarInscripcion_SP_31_TCASE
|      |
|      !_____CerrarInscripcion_SP_32
|      |      !_____CerrarInscripcion_SP_32_TCASE

```

Los casos resaltados en rojo son aquellos que fueron podados al correr el comando *genalltca*, y en verde aquellos de los cuales resultó un caso de prueba generado.

La mayoría de los casos fueron podados, para explicar los motivos primero se explica primero el por qué de los casos más sencillos:

- La rama del caso *CerrarInscripcion_FT_1* fue podada ya que fue generada por la táctica *FT* dando el valor de inscripto a la variable *estado?*, incumpliendo la precondition *estado? ≠ inscripto* del esquema *CerrarInscripcionOk*.
- Los casos *CerrarInscripcion_FT_29* y *CerrarInscripcion_FT_30* son podados porque en estos casos se le asigna el valor *promueve* o *repite* a la variable *estado?* incumpliendo la precondition *estado? = inscripto* del esquema *CerrarInscripcionEstadoInvalidoE*.

Luego, para los siguientes casos se deben analizar los casos de la partición estándar del operador \oplus :

Standard partition for expressions of the form $R \oplus G$

$$\begin{aligned}
R &= \{\}, G = \{\} \\
R &= \{\}, G \neq \{\} \\
R &\neq \{\}, G = \{\} \\
R &\neq \{\}, G \neq \{\}, \text{dom } R = \text{dom } G \\
R &\neq \{\}, G \neq \{\}, \text{dom } G \subset \text{dom } R \\
R &\neq \{\}, G \neq \{\}, (\text{dom } R \cap \text{dom } G) = \{\} \\
R &\neq \{\}, G \neq \{\}, \text{dom } R \subset \text{dom } G \\
R &\neq \{\}, G \neq \{\}, (\text{dom } R \cap \text{dom } G) \neq \{\}, \neg (\text{dom } G \subseteq \text{dom } R), \neg (\text{dom } R \subseteq \text{dom } G)
\end{aligned}$$

donde $R = \text{inscripciones}$ y $G = \{\text{alumno?} \mapsto ((\text{inscripciones alumno?}).1, \text{estado?})\}$. A partir de esto se puede llegar a las siguiente conclusiones:

- Por la definición de G este conjunto tendrá siempre un elemento, siendo este la tupla $(\text{alumno?}, ((\text{inscripciones alumno?}).1, \text{estado?}))$, por lo que nunca se podrá dar que $G = \{\}$ y esto descarta los casos *CerrarInscripcion_FT_12*, *CerrarInscripcion_FT_14*, *CerrarInscripcion_FT_20* y *CerrarInscripcion_FT_22*.
- Otra consecuencia de que G nunca podrá ser vacío es que dentro de su definición se consulta el valor asignado a *alumno?* en la función *inscripciones* lo que significa que $R \neq \{\}$, esto descarta los casos *CerrarInscripcion_FT_13* y *CerrarInscripcion_FT_21*. Además otra consecuencia es que $\text{alumno?} \in \text{dom inscripciones}$, esto implica $\text{dom } G \subset \text{dom } R \Rightarrow \text{alumno?} \in (\text{dom } R \cap \text{dom } G)$ y descarta los casos *CerrarInscripcion_FT_17* y *CerrarInscripcion_FT_25*.
- Otro caso de la partición estándar se da con la condición $R \neq \{\}, G \neq \{\}, \text{dom } R \subset \text{dom } G$. Para que esto suceda debería existir $\text{alumno} \in \text{ALUMNO}$ tal que $\text{alumno} \in \text{dom } G \wedge \text{alumno} \notin \text{dom } R$, pero

si $alumno \in \text{dom } G$ entonces por lo visto en el ítem anterior se tiene que dar $alumno \in \text{dom } R$ y finalmente $\text{dom } R \subset \text{dom } G$ es una contradicción. Esto descarta los casos *CerrarInscripcion_SP_18* y *CerrarInscripcion_SP_26*.

- El último caso de la partición estándar viene dado por la condición $R \neq \{\}, G \neq \{\}, (\text{dom } R \cap \text{dom } G) \neq \{\}, \neg(\text{dom } G \subseteq \text{dom } R), \neg(\text{dom } R \subseteq \text{dom } G)$ y por los ítems anteriores sabemos que $\text{dom } G \subset \text{dom } R$ contradiciendo la condición. Esto descarta los casos *CerrarInscripcion_SP_19* y *CerrarInscripcion_SP_27*.

Esquemas de casos de pruebas generados

Por último, los casos de prueba generados fueron los siguientes:

<i>CerrarInscripcion_SP_15_TCASE</i>
<i>CerrarInscripcion_SP_15</i>
$registrados = \{aLUMNO1\}$
$inscripciones = \{(aLUMNO1 \mapsto (1 \mapsto repite))\}$
$estado? = promueve$
$alumno? = aLUMNO1$

<i>CerrarInscripcion_SP_16_TCASE</i>
<i>CerrarInscripcion_SP_16</i>
$registrados = \{aLUMNO1\}$
$inscripciones = \{(aLUMNO1 \mapsto (1 \mapsto repite)), (aLUMNO3 \mapsto (2 \mapsto promueve))\}$
$estado? = promueve$
$alumno? = aLUMNO1$

CerrarInscripcion_SP_23_TCASE

CerrarInscripcion_SP_23

registrados = {*aLUMNO1*}

inscripciones = {(*aLUMNO1* \mapsto (*1* \mapsto *repite*))}

estado? = *repite*

alumno? = *aLUMNO1*

CerrarInscripcion_SP_24_TCASE

CerrarInscripcion_SP_24

registrados = {*aLUMNO1*}

inscripciones = {(*aLUMNO1* \mapsto (*1* \mapsto *repite*)), (*aLUMNO3* \mapsto (*2* \mapsto *promueve*))}

estado? = *repite*

alumno? = *aLUMNO1*

CerrarInscripcion_FT_28_TCASE

CerrarInscripcion_FT_28

registrados = \emptyset

inscripciones = \emptyset

estado? = *inscripto*

alumno? = *aLUMNO1*

CerrarInscripcion_SP_31_TCASE

CerrarInscripcion_SP_31

registrados = \emptyset

inscripciones = \emptyset

estado? = *inscripto*

alumno? = *aLUMNO1*

CerrarInscripcion_SP_32_TCASE

CerrarInscripcion_SP_32

registrados = {*aLUMNO1*}

inscripciones = \emptyset

estado? = *inscripto*

alumno? = *aLUMNO1*
