Trabajo Práctico
Verificación de Software

Alumna:

Sullivan, Katherine

Universidad Nacional de Rosario

2024

El problema: Sistema de Gestión de Cursos de Universidad

Se requiere la construcción de un sistema de gestión de cursos de universidad. El mismo debe ser capaz de asignar profesores a un curso, anotar alumnos a cursos para los cuales ya se encuentren asignados profesores y mostrar información de cursos existentes en la universidad.

Designaciones

s es un estudiante $\approx s \in STUDENT$

p es un profesor $\approx p \in PROFESSOR$

c es un identificador de curso $\approx c \in COURSE$

Conjunto de estudiantes anotados en el curso $c \approx enrolledStudents(c)$

Conjunto de profesores asignados al curso $c \approx assignedProfessors(c)$

Especificación Z

$[STUDENT, PROFESSOR, COURSE]$

CourseManagementSystem

$$enrolledStudents : COURSE \pfun \mathbb{P}\, STUDENT$$
$$assignedProfessors : COURSE \pfun \mathbb{P}\, PROFESSOR$$

CourseManagmentInit

$$CourseManagementSystem$$

$$enrolledStudents = \emptyset$$
$$assignedProfessors = \emptyset$$

Invariante: un estudiante no se puede anotar a un curso que no tenga profesores asignados todavía.

CourseManagmentInv

$$CourseManagementSystem$$

$$\mathrm{dom}(enrolledStudents) \subseteq \mathrm{dom}(assignedProfessors)$$

Operación que permite anotar un alumno a un curso: *EnrollStudent*.

EnrollStudentOldCourseOk

$\Delta CourseManagementSystem$

$student? : STUDENT$

$course? : COURSE$

$course? \in \text{dom}(assignedProfessors)$

$course? \in \text{dom}(enrolledStudents)$

$student? \notin enrolledStudents(course?)$

$enrolledStudents' = enrolledStudents \oplus \{course? \mapsto (enrolledStudents(course?) \cup \{student?\})\}$

$assignedProfessors' = assignedProfessors$

StudentAlreadyInCourse

$\Xi CourseManagementSystem$

$student? : STUDENT$

$course? : COURSE$

$course? \in \text{dom}(assignedProfessors)$

$course? \in \text{dom}(enrolledStudents)$

$student? \in enrolledStudents(course?)$

EnrollStudentNewCourseOk

$\Delta CourseManagementSystem$

$student? : STUDENT$

$course? : COURSE$

$course? \in \text{dom}(assignedProfessors)$

$course? \notin \text{dom}(enrolledStudents)$

$enrolledStudents' = enrolledStudents \cup \{course? \mapsto \{student?\}\}$

$assignedProfessors' = assignedProfessors$

CourseNotReady

$\Xi CourseManagementSystem$

$course? : COURSE$

$course? \notin \text{dom}(assignedProfessors)$

$$EnrollStudent == EnrollStudentOldCourseOk \lor EnrollStudentNewCourseOk \lor CourseNotReady \lor StudentAlreadyInCourse$$

Operación que permite asignar profesores a un curso: *AssignProfessor*.

AssignProfessorsOk

$$\Delta CourseManagementSystem$$
$$professors? : \mathbb{P}\, PROFESSOR$$
$$course? : COURSE$$

$$course? \notin \text{dom}(assignedProfessor)$$
$$assignedProfessors' = assignedProfessors \cup \{course? \mapsto professors?\}$$
$$enrolledStudents' = enrolledStudents$$

CourseAlreadyAssigned

$$\Xi CourseManagementSystem$$
$$professors? : \mathbb{P}\, PROFESSOR$$
$$course? : COURSE$$

$$course? \in \text{dom}(assignedProfessors)$$

$$AssignProfessors == AssignProfessorsOk \lor CourseAlreadyAssigned$$

Operación que permite mostrar información de un curso: *ShowCourseInfo*.

ShowCourseInfoWithStudentsOk

$$\Xi CourseManagementSystem$$
$$course? : COURSE$$
$$enrolledStudents! : \mathbb{P}\, STUDENT$$
$$assignedProfessors! : \mathbb{P}\, PROFESSOR$$

$$course? \in \text{dom}(assignedProfessors)$$
$$course? \in \text{dom}(enrolledStudents)$$
$$enrolledStudents! = enrolledStudents(course?)$$
$$assignedProfessors! = assignedProfessors(course?)$$

ShowCourseInfoWithoutStudentsOk

$\Xi CourseManagementSystem$

$course? : COURSE$

$enrolledStudents! : \mathbb{P}\, STUDENT$

$assignedProfessors! : \mathbb{P}\, PROFESSOR$

$course? \in \text{dom}(assignedProfessors)$

$course? \notin \text{dom}(enrolledStudents)$

$enrolledStudents! = \emptyset$

$assignedProfessors! = assignedProfessors(course?)$

NoInfoToShow

$\Xi CourseManagementSystem$

$course? : COURSE$

$enrolledStudents! : \mathbb{P}\, STUDENT$

$assignedProfessor! : \mathbb{P}\, PROFESSOR$

$course? \notin \text{dom}(assignedProfessors)$

$enrolledStudents! = \emptyset$

$assignedProfessor! = \emptyset$

$$ShowCourseInfo == ShowCourseInfoWithStudentsOk \lor ShowCourseInfoWithoutStudentsOk \lor NoInfoToShow$$

Especificación Z normalizada

Teniendo en cuenta lo visto en Ingeniería del Software I, cambiarán las definiciones de las variables de estado y los invariantes, puesto que que en Z las funciones parciales no son un tipo.

CourseManagementSystem

$enrolledStudents : COURSE \leftrightarrow \mathbb{P}\, STUDENT$

$assignedProfessors : COURSE \leftrightarrow \mathbb{P}\, PROFESSOR$

CourseManagmentInv

CourseManagementSystem

$\text{dom}(\mathit{enrolledStudents}) \subseteq \text{dom}(\mathit{assignedProfessors})$

$\mathit{enrolledStudents} \in \mathit{COURSE} \pfun \mathbb{P}\, \mathit{STUDENT}$

$\mathit{assignedProfessor} \in \mathit{COURSE} \pfun \mathit{PROFESSOR}$

Traducción a {log}

La traducción a {log} (aceptada por el typechecker) se encuentra en el archivo `courseMgmt.pl`.

Ejecución de simulaciones en {log}

Planteamos dos simulaciones: una directa y una inversa. Las presentamos a continuación pero también se encuentran disponibles en el archivo `simulaciones.pl`.

Simulación 1

Definimos la siguiente simulación directa:

```
dec([P0, P1, P2, P3], ap) &
dec([S0, S1, S2, S3, S4, S5], es) &
dec([StudentsPedag, StudentsDer, StudentsPsico], ss) &
dec([ProfessorsPedag, ProfessorsDer, ProfessorsPsico], pp) &
courseMgmtInit(S0, P0) &
assignProfessors(P0, {professor:pauloFreire}, course:pedagogiaCritica, P1) &
assignProfessors(P1, {professor:savitribaiPhule, professor:simonedeBeauvoir},
                course:derechosDeLaMujer, P2) &
assignProfessors(P2, {professor:jeanPiaget, professor:levVygotsky},
                course:derechosDeLaMujer, P3) &
enrollStudent(S0, P3, student:leo, course:epistemologia, S1) &
enrollStudent(S1, P3, student:maya, course:derechosDeLaMujer, S2) &
enrollStudent(S2, P3, student:maya, course:pedagogiaCritica, S3) &
enrollStudent(S3, P3, student:leo, course:pedagogiaCritica, S4) &
```

```
enrollStudent(S4, P3, student: leo, course:psicopedagogia, S5) &
showCourseInfo(S5, P3, course:pedagogiaCritica, StudentsPedag, ProfessorsPedag) &
showCourseInfo(S5, P3, course:derechosDeLaMujer, StudentsDer, ProfessorsDer) &
showCourseInfo(S5, P3, course:psicopedagogia, StudentsPsico, ProfessorsPsico).
```

y obtenemos el resultado esperado (único):

```
P0 = {},
P1 = {[course:pedagogiaCritica,{professor:pauloFreire}]},
P2 = {[course:pedagogiaCritica,{professor:pauloFreire}],
    [course:derechosDeLaMujer,{professor:savitribaiPhule,professor:simonedeBeauvoir}]},
P3 = {[course:pedagogiaCritica,{professor:pauloFreire}],
    [course:derechosDeLaMujer,{professor:savitribaiPhule,professor:simonedeBeauvoir}]},
S0 = {},
S1 = {},
S2 = {[course:derechosDeLaMujer,{student:maya}]},
S3 = {[course:pedagogiaCritica,{student:maya}],[course:derechosDeLaMujer,{student:maya}]},
S4 = {[course:derechosDeLaMujer,{student:maya}],
    [course:pedagogiaCritica,{student:leo,student:maya}]},
S5 = {[course:derechosDeLaMujer,{student:maya}],
    [course:pedagogiaCritica,{student:leo,student:maya}]},
StudentsPedag = {student:leo,student:maya},
StudentsDer = {student:maya},
StudentsPsico = {},
ProfessorsPedag = {professor:pauloFreire},
ProfessorsDer = {professor:savitribaiPhule,professor:simonedeBeauvoir},
ProfessorsPsico = {}
```

Simulación 2

Generamos la siguiente simulación inversa:

```
dec([P0, P1, P2], ap) &
dec([S0, S1, S2], es) &
dec([Course1, Course2], course) &
```

```
dec([Profs1, Profs2], pp) &
dec([Student1, Student2], student) &
assignProfessors(P0, Profs1, Course1, P1) &
assignProfessors(P1, Profs2, Course2, P2) &
enrollStudent(S0, P2, Student1, Course1, S1) &
enrollStudent(S1, P2, Student2, Course2, S2) &
P2 = {[course:fonoaudiologia, {professor:mariaMontessori}],
    [course:literatura, {professor:juanaManso}]} &
S2 = {[course:fonoaudiologia, {student:ailen}],
    [course:literatura, {student:camila}]}.
```

y la primer solución obtenida es:

```
P0 = {},
P1 = {[course:fonoaudiologia,{professor:mariaMontessori}]},
P2 = {[course:fonoaudiologia,{professor:mariaMontessori}],
    [course:literatura,{professor:juanaManso}]},
S0 = {[course:fonoaudiologia,{}],[course:literatura,{}]/_N2},
S1 = {[course:fonoaudiologia,{student:ailen}],[course:literatura,{}]/_N1},
S2 = {[course:fonoaudiologia,{student:ailen}],
    [course:literatura,{student:camila}]},
Course1 = course:fonoaudiologia,
Course2 = course:literatura,
Profs1 = {professor:mariaMontessori},
Profs2 = {professor:juanaManso},
Student1 = student:ailen,
Student2 = student:camila
Constraint: dom(_N2,_N6), set(_N6),
[course:fonoaudiologia,{}]nin _N5, set(_N5),
un(_N5,_N1,_N2), set(_N2), dom(_N5,_N4), rel(_N5), set(_N4),
subset(_N4,{course:fonoaudiologia}), course:fonoaudiologia nin _N3,
[course:fonoaudiologia,{}]nin _N2,
comp({[course:fonoaudiologia,course:fonoaudiologia]},_N2,{}), rel(_N2),
[course:fonoaudiologia,{student:ailen}]nin _N1, dom(_N1,_N3), set(_N3),
subset(_N3,{course:literatura}), set(_N1), [course:literatura,{}]nin _N1,
comp({[course:literatura,course:literatura]},_N1,{}), rel(_N1)
```

Generado de las condiciones de verificación

En la versión final del archivo el comando `check_vcs_courseMgmt` realiza todas las descargas de prueba automáticamente y no fue necesario agregar ninguna hipótesis.

Sobre la interacción con el VCG

En una versión anterior, para las suboperaciones de enrollStudent y assignProfessors que no cambiaban el estado, no tuve en cuenta el poner como parámetro la variable de estado actualizada (EStudents_ y AProfessors_, respectivamente) y agregar la sentencia `Var_ = Var` porque pensé que podía obviarlo.

En esta primera versión luego de ver que fallaban los invariantes relacionados con estas operaciones, usé el comando `findh`, que, con razón, me devolvió la lista vacía para las hipótesis faltantes en cada caso. Una vez añadido lo explicado en el párrafo anterior las descargas de prueba fueron automáticas y no fue necesario volver a usar el comando.

Demostración de lema de invariancia con Z/EVES

Decidí probar el siguiente teorema:

theorem EnrollCourseMgmtInv

$$CourseManagementInv \land EnrollStudent \Rightarrow CourseManagementInv'$$

realizando la siguiente prueba:

proof[*EnrollCourseMgmtInv*]

split EnrollStudentNewCourseOk;

cases;

reduce;

equality substitute enrolledStudents′;

reduce;

next;

split EnrollStudentOldCourseOk;

cases;

reduce;

equality substitute enrolledStudents′;

reduce;

next;

split CourseNotReady;

cases;

reduce;

next;

reduce;

next;

■

que simplemente consiste en separar la prueba por casos, en cada uno de ellos aplicar reducciones, y en los dos casos donde se modifica la varible de estado enrolledStudents hacer el reemplazo de la versión modificada de la variable por su definición y volver a reducir.

Generación de casos de prueba con FASTEST

Comandos corridos para obtener los casos de prueba

```
loadspec fastest.tex
selop EnrollStudent
genalltt
addtactic EnrollStudent_DNF_1 SP \oplus enrolledStudents \oplus
    \{course? \mapsto (enrolledStudents(course?) \cup \{student?\})\}
addtactic EnrollStudent_DNF_2 SP \cup
```

```
    enrolledStudents \cup \{course? \mapsto \{student?\}\}
addtactic EnrollStudent_DNF_3 SP \notin course? \notin \dom assignedProfessors
addtactic EnrollStudent_DNF_4 SP \in student? \in enrolledStudents~course?
genalltt
genalltca
```

Lo que hicimos con estos comandos fue primero cargar la especificación desde el archivo, setear la operación para la cual queremos generar los casos de pruebas y, luego, aplicar la primera táctica que se aplica por defecto en FASTEST al correr `genalltt`: la táctica que lleva nuestra operación a su forma normal disyuntiva (DNF). Esto hace que el VIS quede particionado según las precondiciones de las 4 suboperaciones que definimos (EnrollStudentOldCourseOk, EnrollStudentNewCourseOk, CourseNotReady y StudentAlreadyInCourse). Una vez obtenido este árbol, agregamos a la lista de tácticas a ser aplicadas particiones estándar para distintos operadores para las distintas hojas del árbol que habíamos obtenido y volvemos a generar el árbol para el cual si ejecutamos el comando `showtt` obtenemos el siguiente resultado:

```
EnrollStudent_VIS
  !______EnrollStudent_DNF_1
  |       !______EnrollStudent_SP_1
  |       !______EnrollStudent_SP_2
  |       !______EnrollStudent_SP_3
  |       !______EnrollStudent_SP_4
  |       !______EnrollStudent_SP_5
  |       !______EnrollStudent_SP_6
  |       !______EnrollStudent_SP_7
  |       !______EnrollStudent_SP_8
  |
  !______EnrollStudent_DNF_2
  |       !______EnrollStudent_SP_9
  |       !______EnrollStudent_SP_10
  |       !______EnrollStudent_SP_11
  |       !______EnrollStudent_SP_12
  |       !______EnrollStudent_SP_13
  |       !______EnrollStudent_SP_14
  |       !______EnrollStudent_SP_15
  |       !______EnrollStudent_SP_16
  |
  !______EnrollStudent_DNF_3
  |       !______EnrollStudent_SP_17
  |       !______EnrollStudent_SP_18
  |
  !______EnrollStudent_DNF_4
          !______EnrollStudent_SP_19
          !______EnrollStudent_SP_20
```

Figura 1: Árbol de prueba luego de ejecutar las tácticas

Una vez obtenido este árbol, ya sí procedemos con la generación de los casos de prueba, obteniendo el siguiente árbol si corremos `showtt` luego de `genalltca`:

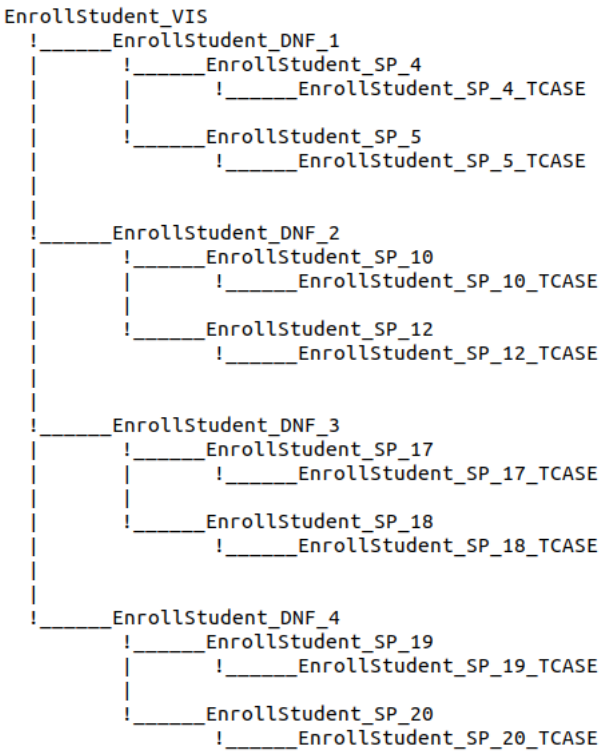

Figura 2: Árbol de prueba luego de obtener los casos de prueba abstractos

Pero, ¿y las particiones que faltan?

Rápidamente nos podemos dar cuenta de que los subárboles correspondientes a `EnrollStudent_DNF_1` y `EnrollStudent_DNF_2` cuentan con menos casos de prueba que los esperados luego de aplicar su partición estándar. Es decir, para algunas de las hojas del primer árbol mostrado no se pudieron generar casos de pruebas. Para entender por qué sucede esto procedemos a analizar los subárboles por separado.

Hojas de `EnrollStudent_DNF_1`

Podemos entender la partición de `EnrolledStudent_DNF_1` como la partición generada por las precondiciones del siguiente esquema de Z:

EnrollStudentOldCourseOk

$\Delta CourseManagementSystem$

$student? : STUDENT$

$course? : COURSE$

$course? \in \text{dom}(assignedProfessors)$

$course? \in \text{dom}(enrolledStudents)$

$student? \notin enrolledStudents(course?)$

$enrolledStudents' = enrolledStudents \oplus \{course? \mapsto (enrolledStudents(course?) \cup \{student?\})\}$

$assignedProfessors' = assignedProfessors$

Al aplicar el comando

```
addtactic EnrollStudent_DNF_1 SP \oplus enrolledStudents \oplus
    \{course? \mapsto (enrolledStudents(course?) \cup \{student?\})\}
```

se generarán 8 hijos desde el nodo `EnrolledStudent_DNF_1` cada uno representando los 8 casos de la figura a continuación (tomando $R = enrolledStudents$, $G = \{course? \mapsto (enrolledStudents(course?) \cup \{student?\})\}$).

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

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

Figura 3: Partición estándar para expresiones de la forma $R \oplus G$ detallada en el manual de usuario de Fastest

Entonces analicemos lo siguiente:

- como G en nuestro caso resulta un conjunto con único elemento (la tupla $(course?, enrolledStudents(course?) \cup \{student?\})$, sabemos que no van a existir casos donde se pueda cumplir que $G = \{\}$.
- como $R = enrolledStudents$ y $course? \in \text{dom}\, enrolledStudents$ sabemos que no van a existir casos donde $R = \{\}$.

- por los dos items anteriores sabemos que $(\text{dom}\, R \cap \text{dom}\, G) \neq \{\}$ pues ambas relaciones contienen en su dominio a *course*?. Entonces no podrá haber casos que se generen de tal manera.
- haciendo un análisis similar, como dom G es exactamente igual al conjunto con un único elemento $\{course?\}$ no puede ser verdad la contención estricta del dominio de R en el dominio de G por lo cual también descartamos el poder generar casos de prueba así.
- por último, siguiendo la misma lógica, no puede suceder que $\neg\ (\text{dom}\, G \subseteq \text{dom}\, R)$ puesto que $\text{dom}\, G = \{course?\}$ y sabemos que $course? \in \text{dom}\, R$. Así que no existirán casos de prueba que cumplan con esta sentencia.

Por lo tanto, los únicos casos que sobreviven a la generación de casos de prueba son el cuarto y quinto de los mostrados en la imagen y esos son los que podremos ver luego.

Hojas de `EnrollStudent_DNF_2`

Similar al caso anterior, podemos entender la partición de `EnrolledStudent_DNF_2` como la partición generada por las precondiciones del siguiente esquema de Z:

EnrollStudentNewCourseOk

$\Delta CourseManagementSystem$
$student? : STUDENT$
$course? : COURSE$

$course? \in \text{dom}(assignedProfessors)$
$course? \notin \text{dom}(enrolledStudents)$
$enrolledStudents' = enrolledStudents \cup \{course? \mapsto \{student?\}\}$
$assignedProfessors' = assignedProfessors$

Al aplicar el comando

```
addtactic EnrollStudent_DNF_2 SP \cup
    enrolledStudents \cup \{course? \mapsto \{student?\}\}
```

se generarán 8 hijos desde el nodo `EnrolledStudent_DNF_2` cada uno representando los 8 casos de la figura a continuación (tomando $S = enrolledStudents$, $T = \{course? \mapsto \{student?\}\}$).

Standard partition for expressions of the form $S \cup T$

$$
\begin{array}{l}
S = \{\}, T = \{\} \\
S = \{\}, T \neq \{\} \\
S \neq \{\}, T = \{\} \\
S \neq \{\}, T \neq \{\}, S \cap T = \{\} \\
S \neq \{\}, T \neq \{\}, S \subset T \\
S \neq \{\}, T \neq \{\}, T \subset S \\
S \neq \{\}, T \neq \{\}, T = S \\
S \neq \{\}, T \neq \{\}, (S \cap T) \neq \{\}, \neg\,(S \subseteq T), \neg\,(T \subseteq S), T \neq S
\end{array}
$$

Figura 4: Partición estándar para expresiones de la forma $S \cup T$ detallada en el manual de usuario de FASTEST

Entonces, otra vez analicemos:

- como en nuestro caso $T = \{course? \mapsto \{student?\}\}$, entonces sabemos no se van a poder generar casos donde $T = \{\}$.
- como en nuestro caso $S = enrolledStudents$ y a su vez sabemos que $course? \notin \text{dom } enrolledStudents$, sabemos que no puede existir ninguna tupla en S con $course?$ como su primera componente. Como T solo tiene el elemento que dimos en el ítem anterior, sabemos que no se podrán generar casos de prueba tales que $T = S$, ni tales que $S \cap T \neq \{\}$ (y, como consecuencia, ni tales que suceda alguna contención hacia algún lado con S y T).

Por lo tanto, los únicos casos que sobreviven a la generación de casos de prueba son el segundo y el cuarto de los que se observan en la figura, y esos son los que veremos.

Con esto queda clara la razón por la cual FASTEST no es capaz de generar casos de prueba para varias de las hojas del árbol que primero presentamos y por la cual obtenemos el segundo árbol que presentamos luego de aplicar `genalltca`.

Esquemas Z de los casos de prueba generados

Como último paso de este trabajo, se presentan los esquemas de casos de prueba abstractos generados por FASTEST.

EnrollStudent_SP_4_TCASE

EnrollStudent_SP_4

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR2\})\}$

$course? = cOURSE1$

$enrolledStudents = \{(cOURSE1 \mapsto \emptyset)\}$

$student? = sTUDENT3$

EnrollStudent_SP_5_TCASE

EnrollStudent_SP_5

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR3\})\}$

$course? = cOURSE1$

$enrolledStudents = \{(cOURSE1 \mapsto \emptyset), (cOURSE2 \mapsto \{sTUDENT1, sTUDENT2\})\}$

$student? = sTUDENT5$

EnrollStudent_SP_10_TCASE

EnrollStudent_SP_10

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR2\})\}$

$course? = cOURSE1$

$enrolledStudents = \emptyset$

$student? = sTUDENT3$

EnrollStudent_SP_12_TCASE

EnrollStudent_SP_12

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR2\})\}$

$course? = cOURSE1$

$enrolledStudents = \{(cOURSE2 \mapsto \{sTUDENT1\})\}$

$student? = sTUDENT3$

EnrollStudent_SP_17_TCASE

EnrollStudent_SP_17

$assignedProfessors = \emptyset$

$course? = cOURSE2$

$enrolledStudents = \emptyset$

$student? = sTUDENT1$

EnrollStudent_SP_18_TCASE

EnrollStudent_SP_18

$assignedProfessors = \{(cOURSE2 \mapsto \{pROFESSOR1\})\}$

$course? = cOURSE1$

$enrolledStudents = \emptyset$

$student? = sTUDENT2$

EnrollStudent_SP_19_TCASE

EnrollStudent_SP_19

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR1, pROFESSOR2\})\}$

$course? = cOURSE1$

$enrolledStudents = \{(cOURSE1 \mapsto \{sTUDENT2\})\}$

$student? = sTUDENT2$

EnrollStudent_SP_20_TCASE

EnrollStudent_SP_20

$assignedProfessors = \{(cOURSE1 \mapsto \{pROFESSOR2\})\}$

$course? = cOURSE1$

$enrolledStudents = \{(cOURSE1 \mapsto \{sTUDENT1, sTUDENT2\})\}$

$student? = sTUDENT1$