UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

SERIE “A”

TRABAJOS DE INFORMÁTICA

Nº 3/09

Estructura de Interfaces para Sistemas con Seguridad Multi-nivel

Matías Lee - Pedro R. D’Argenio

Editores: Pedro R. D’argenio – Gabriel Infante López

CIUDAD UNIVERSITARIA – 5000 CÓRDOBA

REPÚBLICA ARGENTINA

Estructura de Interfaces para Sistemas con Seguridad Multi-nivel

Matías Lee, Pedro R. D'Argenio*
*FaMAF, Universidad Nacional de Córdoba (*y CONICET)*
Ciudad Universitaria - 5000, Córdoba - Argentina.
email: {dargenio,lee}@famaf.unc.edu.ar

Abstract

En este trabajo introducimos una nueva forma de modelar interfaces de sistemas: las Estructuras de Interfaces para Seguridad (EIS). Las mismas están basadas en los autómatas de interfaces *[5] y permiten definir cuales acciones son confidenciales y cuales no. En este nuevo modelo, una EIS será segura si satisface la propiedad de no-interferencia. Además se presenta un algoritmo para obtener una interfaz segura, restringiendo los servicios (acciones de entrada) que ésta ofrece.*

1. Introducción

La *interfaz* de un sistema (de hardware o de software) es todo lo relacionado a los métodos y formas que tiene un sistema para interactuar con su entorno. Hoy en día es natural ver como distintos sistemas interactúan con otros para realizar tareas más complejas. Un ejemplo de esto puede verse en la web, donde distintos servicios se "unen" para realizar nuevos servicios más complejos.

Una buena forma de describir interfaces, debería permitir analizar como será la interacción de un sistema con otros sistemas. De esta forma podríamos predecir si un sistema compuesto por sistemas más simples cumple con nuestras exigencias. A partir de aquí, usaremos los términos *"sistema"* e *"interfaz"* para referirnos a la descripción/abstracción de la interfaz del sistema real.

Teniendo en cuenta distintas exigencias, se han definido distintas formas de especificar interfaces. En las distintas definiciones, dos interfaces son *compatibles* si cuando estas se *componen*, es decir interactúan entre si, existe una forma de lograr la exigencia deseada. En *Interface Automata* [5] la exigencia es lograr una buena comunicación entre los sistemas a componer, entonces dos interfaces no son compatibles si existe una interacción en donde una interfaz manda un mensaje a la otra y ésta no está lista para recibirlo. En *Resource Interfaces* [3] existen recursos limitados para realizar una tarea, luego las interfaces no son compatibles si juntas analicemos necesitan más recursos que los disponibles. En *Timed Interfaces* [6] es posible expresar las expectaciones de un sistema con respecto al tiempo que se debe esperar para recibir un estímulo del ambiente. En este caso, si una interfaz no cumple las expectaciones de la otra, entonces éstas no son compatibles. En este trabajo definimos un nuevo tipo de interfaz: las *Estructuras de Interfaz para Seguridad* (EIS). En estas existen *acciones con baja confidencialidad*, las cuales pueden ser observadas/manipuladas por todos los usuarios, y acciones privadas o *acciones con alta confidencialidad*, las cuales sólo son observadas/manipuladas por los usuarios con *mayor jerarquía* del sistema. Los usuarios de mayor jerarquía serán llamados *usuarios altos* y el resto *usuarios bajos*. En este contexto, la *exigencia deseada* será la propiedad de *no-interferencia* [11] [9] [16], la cual informalmente establece que los usuarios bajos, sólo observando y manipulando las acciones de su nivel, no pueden obtener ningún tipo de información o interferir sobre las actividades del nivel alto.

Ilustremos esto con un ejemplo. Supongamos que se desea desarrollar una aplicación, en la cual los usuarios de la aplicación son monitoreados por supervisores. Sobre esta aplicación existen dos condiciones: tiene que ser reutilizable para distintas tareas y los usuarios que son monitoreados no pueden saber cuando son monitoreados, ni tampoco tener ninguna información sobre quien es la persona que los monitorea.

La aplicación podría estar representada por $\hat{S}$ en la Figura 1: se recibe una tarea (*newTask*), se empieza la tarea (*startTask*), se termina la tarea (*endTask*) y se continúa con una nueva tarea (*newTask*). Antes de empezar con una tarea, el supervisor puede em-

Trabajo soportado por ANPCyT PICT 26135, CONICET PIP 6391, y SeCyT-UNC.

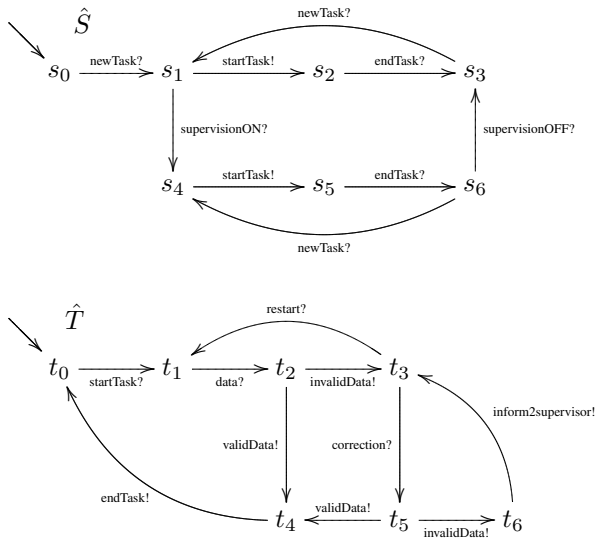

Figure 1. $\hat{S}$ representa una aplicación, de uso general, donde los usuarios pueden ser supervisados. $\hat{T}$ representa una tarea particular que puede ser realizada dentro de la aplicación $\hat{S}$

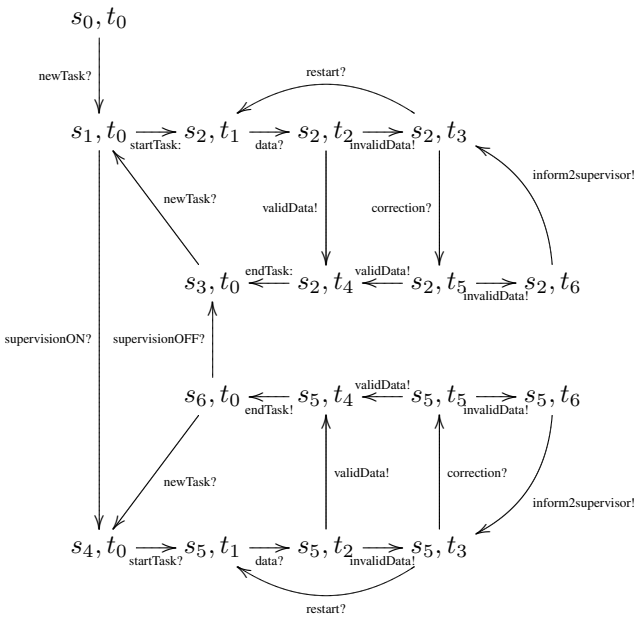

Figure 2. Representación informal de $\hat{S}$ y $\hat{T}$ interactuando.

pezar a monitorear el trabajo (*supervisionON*), y si el supervisor está monitoreando la tarea, cuando se termina la tarea, este puede dejar de monitorear (*supervisionOFF*). La nueva tarea, *newTask*, es un estímulo que espera recibir $\hat{S}$, gráficamente esto se representa por la etiqueta acompañada del símbolo ?; $\hat{S}$ no se encarga de realizar la tarea, pues ésta debe ser realizada por otro módulo para garantizar le reusabilidad de las tareas de monitoreo, entonces $\hat{S}$ debe informar a otro módulo el inicio de un nuevo trabajo, por lo cual *startTask* es una acción de salida, este hecho se remarca en el gráfico, mediante el símbolo !. $\hat{S}$ espera ser informado la finalización de la tarea mediante la acción *endTask* antes de empezar con una nueva tarea, luego *endTask* es un acción de entrada en $\hat{S}$. Las acciones *supervisionON* y *supervisionOFF* abstraen todo el proceso relacionado con el registro de actividad de usuarios. El monitoreo empieza y termina cuando el supervisor lo decide, entonces estas son acciones de entrada. Ambas acciones no son vistas por los usuarios de la aplicación. Notemos que el usuario de $\hat{S}$, no ve diferencias entre el sistema cuando esta siendo monitoreado y cuando no. Esto se debe a que el usuario ve el mismo funcionamiento de la aplicacion, sin importar si el sistema está, o no, siendo monitoreado: (*newTask, starTask, endTask*)*. Entonces, $\hat{S}$ satisface la condición que exige que el usuario no tenga conocimiento sobre si esta o no siendo monitoreado.

En la Figura 1, $\hat{T}$ representa un módulo que podría ser usado con $\hat{S}$. El módulo posee el siguiente funcionamiento: espera que se le informe que hay que empezar con una tarea (*startTask?*), luego espera que se ingresen datos (*data?*), si los datos son correctos (*validData!*), se termina la tarea y se informa de este hecho (*endTask!*); si los datos son incorrectos se informa de este hecho (*invalidData!*), se puede recibir una corrección (*correction?*) o optar por reiniciar la carga de datos (*restart?*). Si la corrección de datos se realiza bien (*validData!*), se termina la tarea (*endTask!*); si la corrección de datos es errónea, el módulo genera un informe extra para el supervisor (*inform2supervisor!*) informando de la falla, y se reinicia el proceso de carga de datos.

La cuestión ahora es si el módulo de tarea $\hat{T}$ es compatible con el sistema $\hat{S}$. Para dar una respuesta a esta pregunta, deberíamos analizar si los módulos pueden interactuar bien juntos. Para esto, analicemos como se comportan los módulos en paralelo y sincronicemoslos en las transiciones comunes. El resultado de la interacción está representado en la Figura 2. En este nuevo sistema, las acciones que se utilizan para la comunicación entre $\hat{S}$ y $\hat{T}$ pasan a ser internas y por lo tanto estas acciones ya no son vistas por el entorno. Las etiquetas de acciones internas son acompañadas por el símbolo ":".

Intuitivamente, podemos decir que los módulos tra-

bajan bien juntos, pero, nuestro sistema compuesto, ¿sigue satisfaciendo los requerimientos de seguridad?. Notemos que si el usuario ingresa mal los datos, decide corregirlos y vuelve a ingresar mal los datos, el sistema automáticamente genera un informe extra para el supervisor, el cual es independiente de los informes realizados por $\hat{S}$. Luego, el usuario bajo puede manipular al sistema, con las acciones que están a su alcance, para que éste realice acciones con información confidencial: generar informes extras para un supervisor particular. Entonces existe una vulnerabilidad que puede ser explotada para averiguar la identidad del supervisor. Por lo tanto, el sistema compuesto ya no satisface los requerimientos de seguridad.

En este trabajo definimos formalmente los conceptos expuestos en este ejemplo. Además se diseño un algoritmo que decide si una interfaz satisface o no la propiedad de no-interferencia. En el caso de que no la satisfaga, el algoritmo informa si existe un conjunto de acciones bajas de entrada, de manera si estas acciones se prohíben, la interfaz obtenida sí cumple con la propiedad de no-interferencia. Además, el algoritmo brinda un posible conjunto de acciones a eliminar.

El trabajo esta organizado de la siguiente manera: en la Sección 2 se presentan rápidamente los Autómatas de Interface (AI); estos sirven para modelar formalmente los sistemas presentados en el ejemplo y definir formalmente la composición de interfaces. En la Sección 3 se presentan las Estructuras de Interfaz para Seguridad que están basadas en los AI. Con estas nueva versión de interface, podremos definir cuales son las acciones confidenciales y cuales las acciones públicas de un sistema. Se define formalmente qué es un sistema seguro, utilizando la propiedad de no-interferencia para esta tarea. En la Sección 4 se presenta el algoritmo que verifica si un sistema posee la propiedad de no-interferencia. En el caso de que la propiedad no sea satisfecha, el algoritmo informa si existe o no, un conjunto de acciones bajas de entrada, que al eliminarlas, se satisface la propiedad de seguridad. En la Sección 5 se hacen algunas consideraciones para extender el conjunto de acciones eliminables a todas las acciones de entrada: altas y bajas. Finalmente, la Sección 6, con las conclusiones y trabajo futuro.

2. Interface Automata

Un *Autómata de Interface* [4], [5] (Intarface Automata) es un autómata que modela la interacción de una componente de un sistema con su entorno. Esta interacción es realizada por medio de acciones de *entrada* y de *salida*. Las acciones de entrada describen el comportamiento que la componente espera (o *asume*) de su entorno, y las acciones de salida representan el comportamiento que comunica (o *garantiza*) al entorno. El autómata también ofrece una descripción causal de como las interacciones están ordenadas. Consideraremos además un tercer conjunto de acciones, las acciones *ocultas*, las cuales se utilizarán para representar el no determinismo interno, producto de la actividad interna de la componente.

Definición 1: Un *Autómata de Interface* (AI) es una tupla $S = \langle Q, q^0, A^I, A^O, A^H, \rightarrow \rangle$ donde: *(i)* Q es un conjunto de *estados* tal que $q^0 \in Q$. El estado q_0 es *estado inicial*. *(ii)* A^I, A^O y A^H son los conjuntos de *acciones de entrada*, *acciones de salida* y *acciones ocultas* respectivamente. A^I, A^O y A^H son conjuntos disjuntos a pares y denotamos con A la unión de estos conjuntos ($A = A^I \cup A^O \cup A^H$). *(iii)* $\rightarrow \subseteq Q \times A \times Q$ es la *relación de transición* la cual es *determinística para las entradas* (i.e. (q, a, q_1) y (q, a, q_2) implica $q_1 = q_2$ para toda acción $a \in A^I$ y estados $q, q_1, q_2 \in Q$.) Utilizaremos Q_S, A^I_S, $\rightarrow_S$, etc. para indicar que nos estamos refiriendo al conjunto de estados, de acciones, de transiciones, etc. del AI S.

Usaremos las siguientes notaciones: $q \xrightarrow{a} q'$ denota $(q, a, q') \in \rightarrow$, $q \xrightarrow{a}$ si existe q' tal que $q \xrightarrow{a} q'$ y $q \not\xrightarrow{a}$ en caso contrario. Una ejecución de S es una secuencia finita de estados y acciones $q_0\, a_0\, q_1\, a_1 \ldots q_n$ tal que $q_i \in Q$, $a_i \in A$ y $q_i \xrightarrow{a_i} q_{i+1}$ para $0 \leq i < n$. Diremos que la ejecución es autónoma si toda transición es de salida u oculta (la ejecución no necesita un estímulo del ambiente para realizarse).

Ejemplo 1: El sistema $\hat{S}$ está representado por la interfaz $\hat{S} = \langle Q, q^0, A^I, A^O, A^H, \rightarrow \rangle$, donde los estados son $Q = \{s_0, \cdots, s_6\}$, el estado inicial es $q_0 = s_0$, las acciones de entrada son $A^I = \{\textit{newTask}, \textit{endTasksupervisionON}, \textit{supervisionOFF}\}$, las acciones de salida son $A^O = \{\textit{startTask}\}$, no hay acciones internas, $A^H = \emptyset$, y la relación de transición $\rightarrow$ es directa de la representación de S en la Figura 1.

Composición de Autómatas de Interfaces. La composición en paralelo es la operación principal para la construcción jerárquica de sistemas. Dadas dos interfaces de autómatas, su composición define una nueva interfaz que representa el modo en el que el sistema compuesto actúa con su entorno, ocultando la comunicación entre las componentes.

Para poder componer dos interfaces vamos a establecer una restricción. La composición de dos AI solamente será posible si dada una acción común, ésta pertenece al conjunto de acciones de entrada de una interfaz y al conjunto de acciones de salida de la otra. Estas acciones serán utilizadas para sincronizar la interacción.

Definición 2: Sean S y T dos AI y sea $\textit{shared}(S,T) = (A_S \cap A_T)$ el conjunto de *acciones compartidas*. Diremos que S y T son *componibles* siempre que $\textit{shared}(S,T) = (A_S^I \cap A_T^O) \cup (A_S^O \cap A_T^I)$.

Ejemplo 2: Las interfaces $\hat{S}$ y $\hat{T}$ de la Figura 1 son componibles pues satisfacen la condición $\textit{shared}(\hat{S},\hat{T}) = \{\textit{startTask, endTask}\}$ y $(A_{\hat{S}}^I \cap A_{\hat{T}}^O) \cup (A_{\hat{S}}^O \cap A_{\hat{T}}^I) = \{\textit{endTask}\} \cup \{\textit{startTask}\}$.

El producto de dos AI componibles S y T se define de manera similar a la composición paralela en CSP: *(i)* el espacio de estados del producto es el producto de los espacios de estados de las componentes, *(ii)* las acciones comunes siempre deben sincronizarse, i.e. ambas componentes deben realizar la transición con acción común (en una interfaz será una acción de salida, en la otra de entrada), y *(iii)* las transiciones no comunes realizan todos los posibles *interleaving*. Por último, las acciones comunes se ocultan en el producto.

Definición 3: Sean S y T dos AI componibles. El *producto* $S \otimes T$ es un nuevo autómata de interfaz definido por:

- $Q_{S\otimes T} = Q_S \times Q_T$.
- $q^0_{S\otimes T} = (q^0_S, q^0_T)$.
- $A^I_{S\otimes T} = A^I_S \cup A^I_T - \textit{shared}(S,T)$.
- $A^O_{S\otimes T} = A^O_S \cup A^O_T - \textit{shared}(S,T)$.
- $A^H_{S\otimes T} = A^H_S \cup A^H_T \cup \textit{shared}(S,T)$.
- $(q_S, q_T) \xrightarrow{a}_{S\otimes T} (q'_S, q'_T)$ si alguno de los siguientes items vale:
  - $a \in A_S - \textit{shared}(S,T)$, $q_S \xrightarrow{a}_S q'_S$, y $q_T = q'_T$;
  - $a \in A_T - \textit{shared}(S,T)$, $q_T \xrightarrow{a}_S q'_T$, y $q_S = q'_S$;
  - $a \in \textit{shared}(S,T)$, $q_S \xrightarrow{a}_S q'_S$, y $q_T \xrightarrow{a}_T q'_T$.

Pueden existir estados de $S \otimes T$ para los cuales una componente, supongamos S, pueda producir una acción de salida que la otra componente no esté lista para recibir, es decir que en el estado en el que se encuentra T no se puede realizar la correspondiente acción de entrada. Entonces S viola las suposiciones de T y esto no es aceptable. Estados como el descripto serán llamados *estados de error*.

Definición 4: Sean S y T dos AI componibles. Un estado del producto $(q_S, q_T) \in Q_{S\otimes T}$ es un *estado de error* si existe una acción $a \in \textit{shared}(S,T)$ tal que $a \in A^O_S$, $q_S \xrightarrow{a}_S$ y $q_T \not\xrightarrow{a}_T$; o $a \in A^O_T$, $q_T \xrightarrow{a}_T$ y $q_S \not\xrightarrow{a}_S$.

Si el producto $S \otimes T$ no contiene estados de error alcanzables, cada componente satisface los supuestos de la otra componente y por lo tanto las interfaces funcionan correctamente juntas. La presencia de estados de error es una evidencia de que una componente está violando los supuestos de la otra interface. Pero esto no significa que no se puedan usar las interfaces en conjunto. Notemos que el comportamiento de las interfaces depende de los estímulos que ésta puede recibir, los cuales están representados por las acciones de entradas. Entonces, tal vez es posible restringir algunas transiciones con acciones de entrada en el producto $S \otimes T$ y de esta forma evitar alcanzar los estados de error. Por supuesto, puede haber casos en los que los estados de errores de $S \otimes T$ se alcanzan autónomamente (i.e. utilizando acciones de salidas u ocultas). En tales casos diremos que las interfaces S y T son incompatibles.

Definición 5: Sean S y T dos AI compatibles y sea $S \otimes T$ su producto. Un estado $(q_S, q_T) \in Q_{S\otimes T}$ es un *estado incompatible* si desde (q_S, q_T) existe una ejecución autónoma que alcanza estado de error. Si un estado no es incompatible entonces es *compatible*. Si el estado inicial de $S \otimes T$ es compatible, entonces S y T son *compatibles* ($S \sim T$). En caso contrario, S y T son incompatibles ($S \not\sim T$).

Finalmente, si dos AI son compatibles, es posible definir la interfaz para la composición de éstas. La interfaz resultante es el resultado de eliminar del producto de las interfaces todas las transiciones de entradas que tengan como destino un estado incompatible.

Definición 6: Sea S y T dos AI compatibles. La *composición* $S \parallel T$ es el AI que se obtiene al eliminar del producto $S \otimes T$ todas las transiciones $q \xrightarrow{a}_{S\otimes T} q'$ tal que *(i)* q es un estado compatible de $S \otimes T$, *(ii)* $a \in A^I_{S\otimes T}$, y *(iii)* q' es un estado incompatible de $S \otimes T$.

Ejemplo 3: La composición $\hat{S} \parallel \hat{T}$ esta representada por la Figura 2. Las acciones de entrada son: *newTask, data, correction, restart* . Las acciones de salida son: *validData, invalidData, inform2supervisor*. Las acciones ocultas son: *startTask, endTask*. Notemos que $\hat{S} \otimes \hat{T} = \hat{S} \parallel \hat{T}$, pues no existen estados de error, es decir, ninguna de las dos interfaces envía un mensaje común que la otra no está lista para recibir.

3. Estructuras de Interfaces para Seguridad

En esta sección extendemos los AI para trabajar con seguridad. Para realizar esto vamos a separar las acciones visibles en dos clases: acciones públicas o *acciones con baja confidencialidad*, las cuales podrán ser observadas por todos los usuarios, y acciones privadas o *acciones con alta confidencialidad*, las cuales podrán ser observadas sólo por usuarios con permisos "especiales" del sistema. El administrador del sistema o usuario que superan cierta jerarquía son ejemplos de usuarios especiales. Denominaremos *usuarios bajos* a aquellos usuarios que no pueden ver las acciones de alta confidencialidad y *usuarios altos* a aquellos que

sí. Nos referiremos con acciones altas (bajas) a las acciones con alta (baja) confidencialidad.

Una vez realizada la separación, es posible caracterizar qué sistemas o componentes proveen una manipulación segura de la información. Diremos que una componente manipula de forma segura la información si ésta posee la propiedad llamada *no-interferencia* [11]. Un sistema posee esta propiedad si un usuario de bajo nivel sólo observando y manipulando las acciones de su nivel no puede obtener o inferir ninguna información sobre la ejecución de las actividades del nivel alto.

Definimos a continuación las Estructuras de Interfaces para Seguridad (EIS). Luego adaptamos la definición de no-interferencia de [9] a nuestro contexto.

Definición 7 (EIS): Una *Estructura de Interface para Seguridad (EIS)* es una tupla $\langle S, A^h, A^l\rangle$ donde S es un AI y A^h y A^l son conjuntos disjuntos de acciones tales que $A^h \cup A^l = A^O \cup A^I$. El conjunto A^h será el *conjunto de las acciones altas* y A^l el *conjunto de las acciones bajas*.

Cuando sea necesario, utilizaremos A^h_S y A^l_S en vez de A^h y A^l para expresar que nos referimos a las acciones altas y bajas de la EIS S. Denotaremos con $A^{X,m}$ al conjunto $A^X \cap A^m$ donde $X \in \{I, O\}$ y $m \in \{h, l\}$. Además utilizaremos la terminología utilizada en los AI para EIS cuando ésta tenga sentido para el AI subyacente a una EIS. Por ejemplo, dos EIS $\langle S, A^h_S, A^l_S\rangle$ y $\langle T, A^h_T, A^l_T\rangle$ son componibles sólo si S y T son componibles.

Una componente es segura si un usuario bajo no puede distinguir la ocurrencia de acciones altas. En otras palabras, un sistema es seguro si para el usuario bajo, el sistema ejecutado sin realizar las acciones altas se *comporta de igual modo* que el sistema ejecutado sin mostrar las acciones ocultas (i.e. las acciones altas no pueden ser observadas por el usuario bajo). En nuestro modelo, diremos que dos componentes se comportan de igual modo si ellas son *débilmente bisimilares* (weak bisimilar). Para poder formalizar estas ideas, definamos los operadores de *ocultamiento* y *restricción* y recordemos la definición de *bisimulación débil*.

Definición 8: Dado un AI A y un conjunto de acciones $X \subseteq A^I_S \cup A^O_S$, definimos:

- la *restricción* de X enanalicemos S como
  $S\backslash X = \langle Q_S, q^0_S, A^I_S - X, A^O_S - X, A^H_S, \rightarrow_{S\backslash X}\rangle$
  donde $q \xrightarrow{a}_{S\backslash X} q'$ sii $q \xrightarrow{a}_S q'$ y $a \notin X$.
- la *ocultamiento* de X en S como
  $S/X = \langle Q_S, q^0_S, A^I_S - X, A^O_S - X, A^H_S \cup X, \rightarrow_S\rangle$.

Dado un EIS $\mathcal{S} = \langle S, A^h_S, A^l_S\rangle$ definimos la *restricción* de X en $\mathcal{S}$ como $\mathcal{S}\backslash X = \langle S\backslash X, A^h_S - X, A^l_S - X\rangle$ y la *ocultamiento* de X en $\mathcal{S}$ como $\mathcal{S}/X = \langle S/X, A^h_S - X, A^l_S - X\rangle$.

Usamos $q_0 \stackrel{\varepsilon}{\Longrightarrow} q_n$ para denotar la existencia de acciones $a_0, \cdots, a_{n-1} \in A^H$ y estados $q_1, \cdots, q_{n-1}$ tales que $q_i \xrightarrow{a_i} q_{i+1}$ para $0 \leq i < n$; i.e. existe una transición autónoma desde q_0 a q_n con sólo transiciones ocultas. Si $a \notin A^H$, entonces $q \stackrel{a}{\Longrightarrow} q'$ denotará que existen q_1 y q_2 tal que $q \stackrel{\varepsilon}{\Longrightarrow} q_1 \xrightarrow{a} q_2 \stackrel{\varepsilon}{\Longrightarrow} q'$.

Definición 9: Sean S y T dos AI. Una relación $R \subseteq Q_S \times Q_T$ es una *bisimulación (débil)* entre S y T si $q^0_S \; R \; q^0_T$ y para todo $q_S \in Q_S$ y $q_T \in Q_T$, $q_S \; R \; q_T$ implica:

- para toda acción $a \in A_S$ y estado $q'_S \in Q_S$, $q_S \xrightarrow{a}_S q'_S$, implica que existe $q'_T \in Q_T$ tal que $q_T \stackrel{a}{\Longrightarrow}_T q'_T$ y $q'_S \; R \; q'_T$.
- para toda acción $a \in A_T$ y estado $q'_T \in Q_T$, $q_T \xrightarrow{a}_T q'_T$, implica que existe $q'_S \in Q_S$ tal que $q_S \stackrel{a}{\Longrightarrow}_S q'_S$ y $q'_S \; R \; q'_T$.

Diremos que S y T son *bisimilares*, notación $S \approx T$, si existe una bisimulación entre S y T. Más aún, diremos que dos EIS $\mathcal{S}$ y $\mathcal{T}$, notación $\mathcal{S} \approx \mathcal{T}$, siempre que los AI subyacentes lo sean.

Ahora nos encontramos en condiciones de dar nuestra definición de seguridad.

Definición 10 ([9]): Sea $\mathcal{S}$ una EIS.

(i) $\mathcal{S}$ es *bisimulation-based strong non-deterministic non-interefrence (BSNNI)* si $\mathcal{S}\backslash A^h \approx \mathcal{S}/A^h$.

(ii) $\mathcal{S}$ es *bisimulation-based non-deterministic non-interefrence (BNNI)* si $\mathcal{S}\backslash A^{I,h}/A^{O,h} \approx \mathcal{S}/A^h$.

Notemos que existe una diferencia entre las dos definiciones. BSNNI representa la propiedad ya descripta: un sistema es BSNNI si un usuario bajo no puede distinguir, sólo observando las acciones bajas,cuando el sistema ejecuta acciones altas (éstas están ocultas) y cuando no (éstas son eliminadas). Por el otro lado, tenemos la definición de BNNI, la cual aparenta ser más débil. (En realidad, ambas definiciones son incomparable ver [9].) En este caso, sólo las acciones altas de entrada son eliminadas y las acciones altas de salida siguen ocurriendo de forma oculta en ambas interfaces. Ésta variante es considerada ya que la misma es más apropiada para el contexto de AI, donde sólo las acciones bajas son controlables.

Ejemplo 4: En la interfaz $\hat{\mathcal{S}}$ las acciones altas son *supervisionON* y *supervisionOFF*, el resto de las acciones son bajas. En la Figura 3 se grafica la interfaz $\hat{\mathcal{S}}$ con las acciones altas restringidas y con las acciones altas ocultas. Las acciones de entrada, salidas e internas son acompañadas por los símbolos ?, ! y : respectivamente. Sólo se grafican los estados alcanzables. $\hat{\mathcal{S}}$ es BSNNI y BNNI. La bisimulación para ambos casos está dada por: $(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3), (s_1, s_4), (s_2, s_5)$ y (s_3, s_6).

Ahora podemos extender la composición de AI a

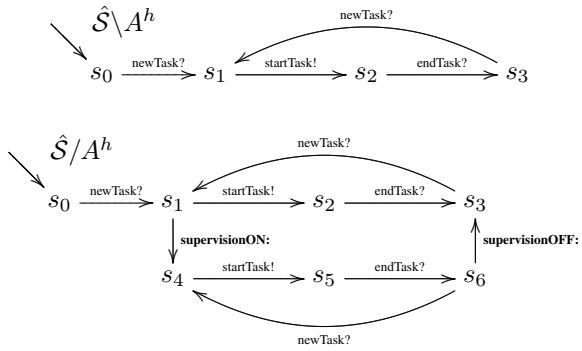

Figure 3. La interfaz $\hat{S}$ con las acciones altas restringidas: $\hat{S}\backslash A^h$. La interfaz $\hat{S}$ con las acciones altas ocultas: $\hat{S}/A^h$.

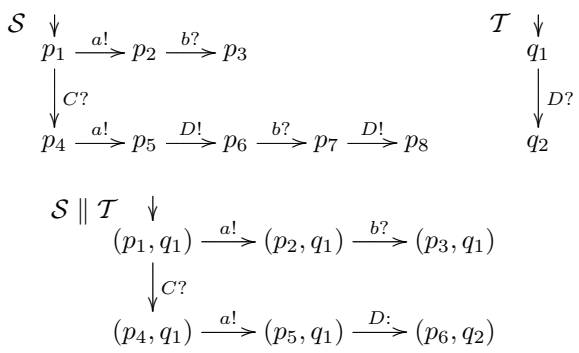

Figure 4. $\mathcal{S}$ y $\mathcal{T}$ son BSNNI y BNNI pero su composición no lo es.

EIS. La misma es bastante natural y es la siguiente:

Definición 11: Dada dos EIS $\mathcal{S} = \langle S, A_S^h, A_S^l \rangle$ y $\mathcal{T} = \langle T, A_T^h, A_T^l \rangle$, su *composición* $\mathcal{S} \parallel \mathcal{T}$ es una nueva EIS $\langle S \parallel T, (A_S^h \cup A_T^h) - \mathit{shared}(S,T), (A_S^l \cup A_T^l) - \mathit{shared}(S,T) \rangle$.

En la Figura 4 se grafican dos EIS: $\mathcal{S}$ y $\mathcal{T}$. Las acciones altas son representadas con letras mayúsculas y las acciones bajas con letras minúsculas. Notemos que ambos EIS son tanto BSNNI como BNNI. En la misma Figura se puede ver $\mathcal{S} \parallel \mathcal{T}$, el resultado de componer $\mathcal{S}$ con $\mathcal{T}$. Aunque $\mathcal{S}$ y $\mathcal{T}$ son ambos BSNNI y BNNI, notemos que $\mathcal{S} \parallel \mathcal{T}$ no es BSNNI, ni BNNI. Luego, las propiedades BSNNI y BNNI no son preservadas por la composición.

Con respecto a nuestro ejemplo inicial, la composición $\hat{\mathcal{S}} \parallel \hat{\mathcal{T}}$ no es BSNNI. En la Figura 5 podemos ver a la interfaz $\hat{\mathcal{S}} \parallel \hat{\mathcal{T}}$ luego de eliminar las acciones altas: $(\hat{\mathcal{S}} \parallel \hat{\mathcal{T}})\backslash A^h$. El estado (s_2, t_6) no posee ninguna transición, pues solamente ejecutaba una transición con una acción alta (*inform2supervisor!*), por lo tanto, éste

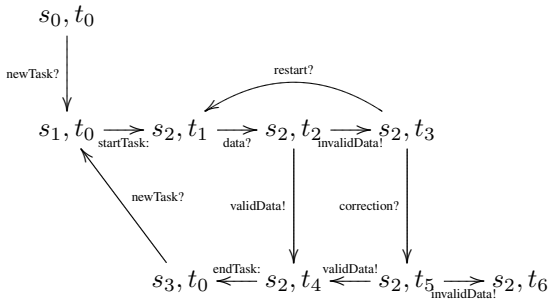

Figure 5. $(\hat{\mathcal{S}} \times \hat{\mathcal{T}})\backslash A^h$

es un estado *resumidero* (sin transiciones de salida). En la interfaz $(\hat{\mathcal{S}} \parallel \hat{\mathcal{T}})/A^h$ no existen estados resumideros, por lo tanto es claro que no existe una bisimulación entre $(\hat{\mathcal{S}} \times \hat{\mathcal{T}})\backslash A^h$ y $(\hat{\mathcal{S}} \times \hat{\mathcal{T}})/A^h$. Por otro lado, notemos que el sistema si es BNNI, ya que la transición de (s_2, t_6) con acción *inform2supervisor!* no es eliminada en $((\hat{\mathcal{S}} \times \hat{\mathcal{T}})/A^{h,O})\backslash A^{h,I}$, por ser esta una acción de salida. Luego existe una relación de bisimulación, la cual es directa.

Por último hagamos unas consideraciones sobre la composición y las propiedades de seguridad. Notemos que no tiene sentido hacer una evaluación sobre si un componente cumple las propiedades BSNNI o BNNI sin tener en cuenta el sistema donde será utilizado. Es decir, no tiene sentido hacer un análisis sobre si una interfaz es segura previo a la composición con el sistema total. Esto se debe a dos razones principales. La primera, la propiedades BSNNI y BNNI no son preservadas por las composición. La segunda, es importante el contexto donde será usado finalmente el componente, pues éste define qué acciones son confidenciales y cuales no. Si $\hat{\mathcal{T}}$ es usado en un contexto donde no es importante preservar en total secreto la identidad del supervisor, la acción *inform2supervisor* no será catalogada de confidencial y por lo tanto $\hat{\mathcal{T}}$ no presentaría ningún problema de seguridad, por estar compuesto solamente de transiciones bajas.

4. Generando EIS seguras

Como se vio en la sección anterior, es muy difícil componer sistemas y que el resultado sea un sistema seguro. Entonces, la pregunta que surge es: ¿Es posible restringir un conjunto de transiciones de una interfaz insegura con la finalidad de obtener una interfaz segura? Notemos que esta pregunta sigue el espíritu de la composición, en la cual se restringen ciertas transiciones para evitar los estados de errores. (ver Def. 6).

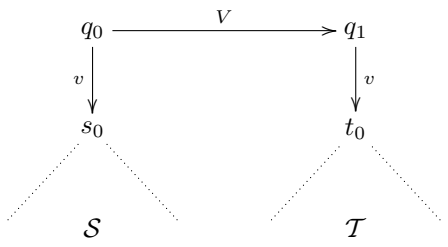

Figure 6. EIS $\mathcal{V}$ generada usando dos EIS $\mathcal{S}$ y $\mathcal{T}$.

En esta sección presentamos un algoritmo el cual verifica si una interfaz es BSNNI (o BNNI). En caso de que no se satisfaga la propiedad, verifica si es posible eliminar un conjunto de transiciones con acciones bajas de entrada y de esta forma obtener una interfaz que es BSNNI (o BNNI). Para ambos casos, BSNNI o BNNI, el algoritmo es similar, por esta razón sólo nos enfocaremos en el caso BSNNI.

Nuestro algoritmo está basado en el algoritmo de Fernandez & Mounier para verificar bisimulación *on the fly* [8]. Brevemente, nuestro algoritmo funciona de la siguiente manera: *(i)* los sistemas de transición se saturan agregando todas las transiciones *débiles* $\stackrel{a}{\Longrightarrow}$; *(ii)* se realiza un producto sincronizado entre los dos sistemas, donde la sincronización ocurre entre las acciones con igual etiqueta; *(iii)* siempre que exista una transición en una interfaz la cual no puede ser realizada por el estado en el que se encuentra la otra interfaz, se se agrega una transición a un estado especial denominado *fail*; *(iv)* si alcanzar el estado fail es inevitable (luego definiremos correctamente inevitable), el sistema de transición no es bisimilar; si siempre existe una forma de evitar el estado fail, entonces los sistemas son bisimilares. El algoritmo original se detiene en el momento en el que encuentra una evidencia de que los sistemas no son bisimilares, mientras que nuestro algoritmo no, ya que debe encontrar todos los puntos donde la bisimilaridad deja de existir, lo cual se utilizará, en la medida de lo posible, para *corregir* la interfaz de manera que sea segura.

Si se comprueba la bisimulación, entonces $\mathcal{S}$ es BSNNI (ver Teorema 5). En caso de que no se compruebe la bisimulación, el algoritmo puede decidir si existe un conjunto de transiciones bajas de entrada que al eliminarlas se logre obtener un sistema BSNNI. El algoritmo da un posible conjunto de transiciones para eliminar (ver Teorema 11).

Por último, observemos que no existen características especiales, sobre los sistemas en los cuales se va a verificar bisimulación, que puedan ser explotadas. Supongamos dos EIS $\mathcal{S}$ y $\mathcal{T}$, con estados iniciales s_0 y t_0 respectivamente. Sea v y V dos acciones tales que $v, V \notin A_S \cup A_T$. Entonces definimos una nueva EIS $\mathcal{V}$ como se ve en la Figura 6 donde las acciones altas son $A_S^h \cup A_T^h \cup \{V\}$ y el resto de las acciones visibles son bajas. Esta nueva interfaz es segura si y solo si $\mathcal{S} \backslash A^h \approx \mathcal{T}/A^h$. Este hecho simplifica la búsqueda de ejemplos, pues podemos evitar el generar una interfaz para la cual sea interesante comparar la versión con las acciones altas ocultas con la interfaz con las acciones altas restringidas, y trabajar directamente con $\mathcal{S}$ y $\mathcal{T}$ arbitrarios.

4.1. Verificando Seguridad

La *saturación* es la forma estándar de identificar que transiciones puede realizar un estado utilizando transiciones internas. La utilización de transiciones internas puede ser antes o después de ejecutar una acción visible ($q_0 \stackrel{\varepsilon}{\Longrightarrow} q_1 \stackrel{a}{\longrightarrow} q_2 \stackrel{\varepsilon}{\Longrightarrow} q_3$). Luego de saturar un sistema, el mismo tendrá nuevas transiciones, las cuales corresponden a las transiciones que pueden hacerse desde cada estado usando transiciones internas.

Notemos que los EIS poseen acciones ocultas con etiquetas, estas etiquetas no serán útiles al verificar bisimulación pues las mismas no son visibles para el entorno. Por esta razón, al saturar un EIS, se reemplazarán las acciones en A^H por una acción ε.

Antes de definir la saturación, vamos a definir un operador auxiliar. Definimos el operador $\overline{\cdot}$ sobre acciones, el cual es idempotente ($\overline{\overline{a}} = \overline{a}$) y para toda acción $a \in A$, el operador crea una nueva acción $\overline{a}$. Diremos que $\overline{a}$ *está marcada* y que a *no está marcada* caso contrario, i.e.,$a \neq \overline{a}$. Para un conjunto de acciones X, definimos $\overline{B} = \{\overline{a} \mid a \in B\}$. Este operador se utilizará para indicar que transiciones fueron agregadas cuando se realizó la saturación de una interface.

Definición 12: Sea S un AI y $B \subseteq A_S^H$. La *saturación de* S es el AI $\overline{S} = \langle Q_S, q_S^0, A_S^I \cup \overline{A_S^I}, A_S^O \cup \overline{A_S^O}, \{\varepsilon\}, \rightarrow_{\overline{S}} \rangle$ donde $\rightarrow_{\overline{S}}$ es la menor relación que satisface las reglas de la Tabla 1.

Dado una EIS $\mathcal{S}$, su *saturación*, $\overline{\mathcal{S}}$, es el EIS obtenido luego de saturar el AI subyacente.

Las primera regla (de izq. a der. de arriba a abajo) indica que un estado siempre puede realizar una transición interna a si mismo. La regla 2 agrega las transiciones de input y de output del sistema original. La regla 3 reemplaza las acciones ocultas por ε. La regla 4 realiza la saturación "hacia atrás", mientras que la regla 5 realiza la saturación "hacia adelante". El caso combinado de la saturación se logra aplicando las reglas 4 y 5, tantas veces como sean necesarias. Por ejemplo si $q_0 \stackrel{\varepsilon}{\Longrightarrow} q_1 \stackrel{a}{\longrightarrow} q_2 \stackrel{\varepsilon}{\Longrightarrow} q_3$, aplicando las regla 5, las veces que sea necesaria, se obtiene

$$q \xrightarrow{\varepsilon}_{\overline{S}} q \qquad \frac{q \xrightarrow{a}_S q' \quad a \in A_S^I \cup A_S^O}{q \xrightarrow{a}_{\overline{S}} q'} \qquad \frac{q \xrightarrow{a}_S q' \quad a \in A_S^H}{q \xrightarrow{\varepsilon}_{\overline{S}} q'}$$

$$\frac{q \xrightarrow{\varepsilon}_{\overline{S}_B} q' \quad q' \xrightarrow{\alpha}_{\overline{S}_B} q''}{q \xrightarrow{\overline{\alpha}}_{\overline{S}} q''} \qquad \frac{q \xrightarrow{\alpha}_{\overline{S}} q' \quad q' \xrightarrow{\varepsilon}_{\overline{S}_B} q''}{q \xrightarrow{\overline{\alpha}}_{\overline{S}} q''}$$

Table 1. Transiciones para la saturación S ($\alpha \in A_S^I \cup A_S^O \cup \overline{A_S^I} \cup \overline{A_S^O} \cup \{\varepsilon, \varepsilon'\}$)

$$\frac{q_S \xrightarrow{a}_S q'_S \quad q_T \xrightarrow{a}_T q'_T}{(q_S, q_T) \xrightarrow{a}_{S\times T} (q'_S, q'_T)} \quad \frac{q_S \xrightarrow{a}_S q'_S \quad q_T \xrightarrow{\overline{a}}_T q'_T}{(q_S, q_T) \xrightarrow{\overline{a}}_{S\times T} (q'_S, q'_T)} \quad \frac{q_S \xrightarrow{\overline{a}}_S q'_S \quad q_T \xrightarrow{a}_T q'_T}{(q_S, q_T) \xrightarrow{\overline{a}}_{S\times T} (q'_S, q'_T)}$$

$$\frac{q_S \xrightarrow{a}_S \quad q_T \not\xrightarrow{a}_T \quad q_T \not\xrightarrow{\overline{a}}_T}{(q_S, q_T) \xrightarrow{a}_{S\times T} \mathit{fail}} \qquad \frac{q_S \not\xrightarrow{a}_S \quad q_S \not\xrightarrow{\overline{a}}_S \quad q_T \xrightarrow{a}_T}{(q_S, q_T) \xrightarrow{a}_{S\times T} \mathit{fail}}$$

Table 2. Transiciones para el producto sincrónico ($a \in A^I \cup A^O \cup \{\varepsilon\}$)

$q_0 \stackrel{\varepsilon}{\Longrightarrow} q_1 \xrightarrow{\overline{a}}_{\overline{S}} q_3$ y aplicando después la regla 4, las veces que sea necesaria, se obtiene $q_0 \xrightarrow{\overline{a}}_{\overline{S}} q_3$.

La siguiente proposición es directa de las definición de saturación: $q \xrightarrow{\overline{a}}_{\overline{S}} q'$ sii $q \stackrel{a}{\Longrightarrow}_S q'$.

A continuación definimos el producto sincronizado, el cual se usará para verificar si dos AI son bisimilares. La idea del producto es verificar paso a paso que acciones pueden realizar cada AI. Si las interfaces son bisimilares, entonces toda transición propuesta por alguna de las interfaces podrá ser imitada por la otra. Como estamos interesados en la bisimulación débil, se compararán interfaces saturadas.

Definición 13: Sean S y T dos AI saturados, tales que $A_S^X = A_T^X = A^X$ con $X \in \{I, O\}$ y $A_S^H = A_T^H = \{\varepsilon\}$. El *producto sincronizado* de S y T es el AI $S \times T = \langle (Q_S \times Q_T) \cup \{\mathit{fail}\}, (q_S^0, q_T^0), A^I, A^O, \{\varepsilon\}, \rightarrow_{S\times T}\rangle$ donde $\rightarrow_{S\times T}$ es la menor relación que satisface las reglas de la Tabla 2.

Sean dos EIS $\mathcal{S} = \langle S, A_{\mathcal{S}}^h, A_{\mathcal{S}}^l\rangle$ y $\mathcal{T} = \langle T, A_{\mathcal{T}}^h, A_{\mathcal{T}}^l\rangle$ con S y T que satisfacen las condiciones mencionadas y además $A_{\mathcal{S}}^m = A_{\mathcal{T}}^m = A^m$ con $m \in \{l, h\}$, entonces el *producto sincronizado* de $\mathcal{S}$ y $\mathcal{T}$ está definido como el EIS $\mathcal{S} \times \mathcal{T} = \langle S \times T, A^h, A^l\rangle$.

Las primeras 3 reglas de la Tabla 2 indican que si una de las interfaces puede realizar una transición, esa transición puede ser imitada por la otra interfaz usando transiciones del sistema original o transiciones agregadas durante la saturación. Las últimas dos reglas indican que si uno de los estados puede realizar una transición y ésta no puede ser realizada por el otro estado, incluyendo las transiciones que se agregaron en la saturación, entonces se realiza una transición a un estado especial denominado *fail*.

Notemos que esta definición de producto es totalmente distinta a la definición del producto previa (ver Def. 3). En esta definición se sincronizan acciones del mismo nombre y del mismo tipo. Más aún, no existe *interleaving*; de hecho, si una acción no logra encontrar otra acción para sincronizarse, entonces ésta genera una transición al estado *fail*. Dado que todos los estados cuentan con una transición a si mismos usando ε, no existe la posibilidad de alcanzar el estado *fail* mediante una acción oculta.

Los estados en el producto que realizan una transición a un estado *fail* contiene un par de estados que no son bisimilares. Esto es claro, pues existe una transición en uno de ellos que el otro no puede realizar.

A continuación se generaliza esta idea para todos los estados del producto, luego podemos establecer si las interfaces son o no son bisimilares. Si ésta generalización sólo se limita a verificar si los pares de estados son o no son bisimilares, la misma no produciría ninguna mejora con respecto a cualquier algoritmo de verificación de bisimilaridad existente. Nuestra generalización tendrá en cuenta si la generación de un estado $(q_S, q_T) \in Q_{\overline{S}\times\overline{T}}$ con $q_S \not\approx q_T$ puede evitarse eliminando transiciones con acciones de $A^{l,I}$ de S y/o de T. Teniendo en cuenta solamente este conjunto, estamos abstrayendo el caso de limitar los servicios que ofrece la interfaz al usuario bajo para lograr obtener un sistema seguro.

Esta generalización dividirá los estados en tres

$$Fail^0_{q_S \xrightarrow{a} q'_S} = \{(q_S, q_T) : q_S \xrightarrow{a} q'_S, a \notin A^I, (q_S, q_T) \xrightarrow{a} fail\} \cup \{fail\}$$

$$Fail^0_{q_T \xrightarrow{a} q'_T} = \{(q_S, q_T) : q_T \xrightarrow{a} q'_T, a \notin A^I, (q_S, q_T) \xrightarrow{a} fail\} \cup \{fail\}$$

$$Fail^0 = \bigcup_{q_S \xrightarrow{a} q'_S \in \longrightarrow_S} Fail^0_{q_S \xrightarrow{a} q'_S} \cup \bigcup_{q_T \xrightarrow{a} q_T \in \longrightarrow_T} Fail^0_{q_T \xrightarrow{a} q'_T}$$

$$Fail^{k+1}_{q_S \xrightarrow{a} q'_S} = \{(q_S, q_T) : q_S \xrightarrow{a} q'_S, a \notin A^I, (\forall q'_T : q'_T \neq q_T \wedge (q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) : (q'_S, q'_T) \in Fail^k)\}$$

$$Fail^{k+1}_{q_T \xrightarrow{a} q'_T} = \{(q_S, q_T) : q_T \xrightarrow{a} q'_T, a \notin A^I, (\forall q'_S : q'_S \neq q_S \wedge (q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) : (q'_S, q'_T) \in Fail^k)\}$$

$$Fail^{k+1} = \bigcup_{q_S \xrightarrow{a} q'_S \in \longrightarrow_S} Fail^{k+1}_{q_S \xrightarrow{a} q'_S} \cup \bigcup_{q_T \xrightarrow{a} q_T \in \longrightarrow_T} Fail^{k+1}_{q_T \xrightarrow{a} q'_T} \cup Fail^k$$

Table 3. Conjunto de estados Fails.

$$May^0_{q_S \xrightarrow{a} q'_S} = \{(q_S, q_T) \notin Fail : q_S \xrightarrow{a} q'_S, a \in A^I, (q_S, q_T) \xrightarrow{a} fail \vee ((q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) \wedge (q'_S, q'_T) \in Fail)\}$$

$$May^0_{q_T \xrightarrow{a} q'_T} = \{(q_S, q_T) \notin Fail : q_T \xrightarrow{a} q'_T, a \in A^I, (q_S, q_T) \xrightarrow{a} fail \vee ((q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) \wedge (q'_S, q'_T) \in Fail)\}$$

$$May^0 = \bigcup_{q_S \xrightarrow{a} q'_S \in \longrightarrow_S} May^0_{q_S \xrightarrow{a} q'_S} \cup \bigcup_{q_T \xrightarrow{a} q_T \in \longrightarrow_T} May^0_{q_T \xrightarrow{a} q'_T}$$

$$May^{k+1}_{q_S \xrightarrow{a} q'_S} = \{(q_S, q_T) \notin Fail : q_S \xrightarrow{a} q'_S, (\forall q'_T : q'_T \neq q_T \wedge (q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) : (q'_S, q'_T) \in Fail \cup May^k)\}$$

$$May^{k+1}_{q_T \xrightarrow{a} q'_T} = \{(q_S, q_T) \notin Fail : q_T \xrightarrow{a} q'_T, (\forall q'_S : q'_S \neq q_S \wedge (q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T) : (q'_S, q'_T) \in Fail \cup May^k)\}$$

$$May^{k+1} = \bigcup_{q_S \xrightarrow{a} q'_S \in \longrightarrow_S} May^{k+1}_{q_S \xrightarrow{a} q'_S} \cup \bigcup_{q_T \xrightarrow{a} q_T \in \longrightarrow_T} May^{k+1}_{q_T \xrightarrow{a} q'_T} \cup Fail^k$$

Table 4. Conjuntos de estados May.

grupos: estados del producto que no contienen un par de estados bisimilares; estados que no contienen componentes bisimilares, pero pueden llegar a serlo eliminando una o más transiciones; estados que contienen un par de estados que sí son bisimilares.

La notación $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ denotará $(q_S, q_T) \xrightarrow{a} (q'_S, q'_T)$ o $(q_S, q_T) \xrightarrow{\overline{a}} (q'_S, q'_T)$.

Definición 14: Sea $S \times T$ un producto sincronizado.

- Definimos el conjunto $Fail \subseteq Q_{S\times T}$ como $Fail = \cup_{i=0}^{\infty} Fail^i$ donde $Fail^i$ está definido en la Tabla 3. Si $q \in Fail$, decimos que el par q *falló el test de bisimulación*.
- Definimos el conjunto $May \subseteq Q_{S\times T}$ como $May = \cup_{i=0}^{\infty} May^i$ donde May^i está definido en la Tabla 4. Si $q \in May$, decimos que el par q *puede pasar el test de bisimulación*. Notemos que $May \cap Fail = \emptyset$.
- Finalmente, definimos $Pass = Q_{S\times T} - (May \cup Fail)$. Si $q \in Pass$, decimos que el par q *pasa el test de bisimulación*.

En general, si el estado inicial del AI subyacente

del EIS $\mathcal{S} \times \mathcal{T}$ falla (puede pasar, pasa) el test de bisimulación, diremos que $\mathcal{S} \times \mathcal{T}$ *falla (puede pasar, pasa) el test de bisimulación*

El objetivo ahora es demostrar que pares de estados que pertenecen a *May* $\cup$ *Fail* no son bisimilares y que los estados que pertenecen a *Pass* están compuesto por pares de estados bisimilares. Demostremos la primera afirmación.

Lema 1: Sea $\overline{S} \times \overline{T}$ el producto sincronizado de dos AI saturados. Si $(q_S, q_T) \in Q_{\overline{S}\times\overline{T}}$ es tal que $(q_S, q_T) \in$ *May* $\cup$ *Fail*, entonces $q_S \not\approx q_T$

Prueba: Supongamos primero que $(q_S, q_T) \in$ *Fail*. Para demostrar que (q_S, q_T) no son bisimilares, vamos a usar inducción en k con respecto a *Fail*^{k}. Si $(q_S, q_T) \in$ *Fail*^{0}, por construcción del producto sincronizado vale: $q_S \xrightarrow{a}_{\overline{S}}$, $q_T \overset{a}{\not\rightarrow}_{\overline{T}}$ y $q_T \overset{\overline{a}}{\not\rightarrow}_{\overline{T}}$o; $q_S \overset{a}{\not\rightarrow}_{\overline{S}}$, $q_S \overset{\overline{a}}{\not\rightarrow}_{\overline{S}}$ y $q_T \xrightarrow{a}_{\overline{T}}$. Luego tenemos que $q_S \xrightarrow{a}_S$ y $q_T \overset{a}{\not\Rightarrow}_T$ o; $q_T \xrightarrow{a}_T$ y $q_S \overset{a}{\not\Rightarrow}_S$. Para ambos casos $q_S \not\approx q_T$.

Supongamos $(q_S, q_T) \in$ *Fail*$^{k+1}$. Sin perdida de generalidad, supongamos $(q_S, q_T) \in$ *Fail*$^{k+1}_{q_S \xrightarrow{a} q'_S}$. Entonces existe acción $a \notin A^I$ tal que el esta q_S realiza la transición $q_S \xrightarrow{a} q'_S$ y para toda transición del producto sincronizado $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ se satisface $(q'_S, q'_T) \in$ *Fail*^{k}, por hipótesis inductiva y construcción del producto sincronizado, se obtiene que existe una acción a tal que $q_S \xrightarrow{a}_S q'_S$ y para toda transición $q_T \overset{a}{\Longrightarrow}_T q'_T$ se satisface $q'_S \not\approx q'_T$, luego $q_S \not\approx q_T$.

El caso $(q_S, q_T) \in$ *May* también se demuestra por inducción en k con respecto a *May*^{k}, aplicando un razonamiento similar. □

Antes de demostrar que si un estado del producto pasa el test de bisimulación, entonces el estado contiene un par de estados bisimilares, vamos a demostrar un lema auxiliar.

El lema demuestra que si (q_S, q_T) es un estado que pasa el test bisimulación y alguno de los estados posee una transición, supongamos $q_S \xrightarrow{a} q'_S$, entonces existe un estado (q'_S, q'_T) tal que $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ y (q'_S, q'_T) pasa el test de bisimulación.

Lema 2: Sea $\overline{S} \times \overline{T}$ el producto sincronizado de dos AI saturados. Si $(q_S, q_T) \in Q_{\overline{S}\times\overline{T}}$ es tal que $(q_S, q_T) \in$ *Pass* entonces:

1) si $q_S \xrightarrow{a} q'_S$, con $a \in A^I \cup A^O$, entonces existe un estado q'_T tal que existe una transición $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ y $(q'_S, q'_T) \in$ *Pass*.
2) si $q_T \xrightarrow{a} q'_T$, con $a \in A^I \cup A^O$, entonces existe un estado q'_S tal que existe una transición $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ y $(q'_S, q'_T) \in$ *Pass*.

Prueba: Se demostrará sólo (1) por ser (2) análogo. Si $q_S \xrightarrow{a} q'_S$, con $a \in A^I \cup A^O$ y no existe un estado q'_T tal que existe una transición $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ tal que $(q'_S, q'_T) \in$ *Pass*, entonces $(q_S, q_T) \xrightarrow{a}$ *fail* o, para todo estado (q'_S, q'_T) tal que $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$, se satisface $(q'_S, q'_T) \in$ *Fail* $\cup$ *May*. Para ambos casos vale que el estado (q_S, q_T) satisface $(q_S, q_T) \in$ *Fail* $\cup$ *May*, absurdo. □

Ahora nos encontramos en condiciones de demostrar el lema.

Lema 3: Sean $\mathcal{S}$ y $\mathcal{T}$ dos EIS tales que $\overline{S} \times \overline{T}$ pasa el test de bisimulación, entonces $S \approx T$.

Prueba: Definamos la relación $\approx \subseteq Q_S \times Q_T$ como $q_S \approx q_T$ si el estado (q_S, q_T) pasa el test de bisimulación. Los estados iniciales de S y T pertenecen a la relación ya que $\overline{S} \times \overline{T}$ pasa el test de bisimulación. Sea $(q_S, q_T) \in$ *Pass*, si $q_S \xrightarrow{a}_S q'_S$ entonces $q_S \xrightarrow{a}_{\overline{S}} q'_S$. Como $(q_S, q_T) \in$ *Pass*, por Lema 2, existe un estado q'_T tal que existe una transición $(q_S, q_T) \xrightarrow{a,\overline{a}} (q'_S, q'_T)$ con $(q'_S, q'_T) \in$ *Pass*. La existencia de esta transición implica que existe en $\overline{T}$ una transición $q_T \xrightarrow{a,\overline{a}}_{\overline{T}} q'_T$, por lo cual en la interfaz T vale $q_T \overset{a}{\Longrightarrow} q'_T$. Finalmente $(q'_S, q'_T) \in$ *Pass* implica $q'_S \approx q'_T$. El caso en que q_T realiza una transición es análogo. □

Lema 4: Sean dos EIS $\mathcal{S}$ y $\mathcal{T}$. Si dado $q_S \in Q_S$ y $q_T \in Q_T$, se satisface $q_S \approx q_T$ y (p_S, p_T) se encuentran en el producto sincronizado $\overline{S}\times\overline{T}$, entonces el estado (p_S, p_T) pasa el test de bisimulación.

Prueba: La prueba es directa utilizando el Lema 1 y el Lema 3. □

Todos los resultados son generales, utilicemoslos para verificar si una interfaz satisface alguna de nuestras propiedades de seguridad. En el caso BSNNI, se desea verificar si las interfaces $\mathcal{A}\backslash A^h_S$ y $\mathcal{A}/A^h_S$ son bisimilares, luego sólo debemos verificar si el producto sincronizado $\overline{(\mathcal{S}\backslash A^h)} \times \overline{(\mathcal{S}/A^h)}$ pasa el test de bisimulación. Para simplificar la notación definimos $P_{\mathcal{S}} = \overline{(\mathcal{S}\backslash A^h)} \times \overline{(\mathcal{S}/A^h)}$.

Teorema 5: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ un EIS. Entonces $\mathcal{S}$ es BSNNI sii $P_{\mathcal{S}}$ pasa el test de bisimulación.

Prueba: ($\Rightarrow$) Si $\mathcal{S}$ es BSNNI, entonces $\mathcal{S}\backslash A^h \approx \mathcal{S}/A^h$, por corolario 4, el producto sincronizado $P_{\mathcal{S}}$ pasa el test de bisimulación

($\Leftarrow$) Por lemma 3, vale $(\mathcal{S}\backslash A^h) \approx (\mathcal{S}/A^h)$ y por lo tanto $\mathcal{S}$ es BSNNI. □

El resultado similar vale para la propiedad BNNI:

Teorema 6: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ un EIS. Entonces $\mathcal{S}$ es BNNI sii $((S\backslash A^{h,I})/A^{h,O}) \times (S/A^h)$ pasa el test de bisimulación.

4.2. Obteniendo Sistemas Seguros

Dada una interfaz $\mathcal{S}$ tal que su producto P_S no pasa el test de bisimulación, la pregunta que surge es si se puede eliminar un conjunto de transiciones de entrada para lograr que la interfaz sí pase el test de bisimulación. Notemos que estamos repitiendo la idea utilizada para la composición: *restringir un conjunto de acciones de entrada para evitar la situación no deseada*. La división de los estados del producto sincronizado en *Fail* y *May*, nos da una clara idea sobre cuales son los estados en los cuales es posible eliminar un conjunto de transiciones y obtener un estado bisimilar, estos estados son los que pertenecen a *May*.

Primero demostremos que para un estado q, tal que $q \in$ *Fail*, *no existe* un conjunto de transiciones de entrada que se pueda eliminar de forma tal que q se transforme en un estado que no falla el test de bisimulación, i.e. $q \notin$ *Fail*.

Lema 7: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ una EIS y $P_\mathcal{S}$ su producto sincronizado. Si el estado $(q_r, q_a) \in Q_{P_\mathcal{S}}$ es tal que $(q_r, q_a) \in$ *Fail*, entonces no existe un conjunto de transiciones de entrada bajas $\rightarrow_\chi \subseteq \rightarrow_S \cap (Q_S \times A_S^{l,I} \times Q_S)$ tal que si definimos el EIS $\mathcal{S}'$ como $\mathcal{S}' = \langle\langle Q_S, q_S^0, A_S^I, A_S^O, A_S^H, (\rightarrow_S) - (\rightarrow_\chi)\rangle, A^h, A^l\rangle$, entonces se satisface $(q_r, q_a) \notin$ $Fail_{P_{\mathcal{S}'}}$.

Prueba: Si $(q_r, q_a) \in Fail^0$ es fácil ver que no existe transiciones de entradas para eliminar y evitar que el estado no pase el test de bisimulación, pues la condición para pertenecer a $Fail^0$ depende de una transición $q \xrightarrow{a} q'$ con $a \notin A^I$.

Supongamos $(q_r, q_a) \in Fail^{k+1}_{q_S \xrightarrow{a} q'_S}$. Por hipótesis inductiva, para todo estado q, no se puede evitar eliminando acciones de entrada, que $q \in$ *Fail* si $q \in Fail^k$. Además, eliminando sólo acciones de entrada, no se puede eliminar la transición $q_S \xrightarrow{a} q'_S$ pues $a \notin A^I$, por lo tanto no existe un conjunto de acciones de entradas que se puedan eliminar para evitar $(q_r, q_a) \in Fail^{k+1}_{q_S \xrightarrow{a} q'_S}$. □

A continuación definimos el conjunto de transiciones que pueden ser eliminadas para que la interfaz pase el test de bisimulación.

Definición 15: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ un EIS y $P_\mathcal{S}$ su producto sincronizado tal que $P_\mathcal{S}$ puede pasar el test de bisimulación. El conjunto de *acciones eliminables* eliminables$(P_\mathcal{S})$ esta definido en la Tabla 5.

El conjunto de transiciones eliminables son las transiciones que "conectan" un estado que pertenece a *May* con un estado que pertenece a *Fail*. Observar que si bien no se especifica en la definición, las transiciones en eliminables$(P_\mathcal{S})$ son siempre transiciones con acciones bajas de entrada, esto es consecuencia de las definiciones de los conjuntos *Fail* y *May*.

Supongamos que tenemos una interfaz que puede pasar el test de bisimulación y eliminando algún subconjunto de las transiciones eliminables generamos una nueva interfaz. Vamos a demostrar que la nueva interfaz pertenece a la categoría de *"puede pasar el test de bisimulación"* o a la categoría de *"pasa el test de bisimulación"*. En caso de obtener una interfaz que pasa el test de bisimulación, habremos obtenido el sistema deseado, en caso negativo, continuaremos eliminando transiciones hasta lograr el objetivo.

Empecemos demostrando un lema auxiliar, el mismo relaciona los estados que pertenecen al conjunto *Fail* de una interfaz a la cual se le eliminó *una* transición del conjunto de transiciones eliminables, con respecto a los estados que pertenecen al conjunto *Fail* de la interfaz original.

La propiedad establece lo siguiente: sea S una interfaz que puede pasar el test de bisimulación. Sea S' la interfaz S después de eliminar una transición de S que pertenece al conjunto eliminables$(P_\mathcal{S})$. El siguiente lema establece que si un estado $q \in Q_{P_{\mathcal{S}'}}$ es tal que $q \in Fail_{P_{\mathcal{S}'}}$, entonces $q \in Q_{P_\mathcal{S}}$ y $q \in Fail_{P_\mathcal{S}}$.

Lema 8: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ un EIS tal que $P_\mathcal{S}$ puede pasar el test de bisimulación y sea $q \xrightarrow{a} q'$ una transición tal que $q \xrightarrow{a} q' \in$ eliminables$(P_\mathcal{S})$. Sea $\mathcal{S}'$ el EIS que se obtiene al eliminar la transición $q \xrightarrow{a} q'$ de $\mathcal{S}$. Si el estado $(q_r, q_a) \in Q_{P_{\mathcal{S}'}}$ es tal que $(q_r, q_a) \in Fail_{P_{\mathcal{S}'}}$ entonces $(q_r, q_a) \in Q_{P_\mathcal{S}}$ y $(q_r, q_a) \in Fail_{P_\mathcal{S}}$.

Prueba: Es claro que $Q_{P_{\mathcal{S}'}} \subseteq Q_{P_\mathcal{S}}$, pues sólo eliminamos una transición.

Sea $(q_r, q_a) \in Fail^0_{P_{\mathcal{S}'}}$, entonces por definición de $Fail^0$, existe una transición con acción $a' \notin A^I$ tal que $(q_r, q_a) \xrightarrow{a'} fail$. Esto es independiente de la transición $q \xrightarrow{a} q'$, por lo tanto el estado $(q_r, q_a) \in Q_{P_\mathcal{S}}$ es tal que $(q_r, q_a) \in Fail^0_{P_\mathcal{S}}$.

Sea $(q_r, q_a) \in Fail^{k+1}_{P_{\mathcal{S}'}}$. y sin perdida de generalidad supongamos $(q_r, q_a) \in Fail^{k+1}_{q_r \xrightarrow{a'} q'_r, P_{\mathcal{S}'}}$. Para este caso se utiliza la hipótesis inductiva y se repite el razonamiento del caso base, luego $(q_r, q_a) \in Fail^{k+1}_{q_r \xrightarrow{a'} q'_r, P_\mathcal{S}}$. □

Un corolario de este lema es que luego de eliminar la transición, los estados que pasaban o podían pasar el test de simulación, siguen perteneciendo a alguna de estas dos clasificaciones.

Corolario 9: Bajo las hipótesis del lema 8, si $q \in P_\mathcal{S}$ es tal que $q \in (Pass_{P_\mathcal{S}} \cup May_{P_\mathcal{S}})$ entonces $q \in (Pass_{P_{\mathcal{S}'}} \cup May_{P_{\mathcal{S}'}})$.

Prueba: Usando el Lemma 8, $q \in Fail_{P_{\mathcal{S}'}} \Rightarrow q \in$

$$\text{eliminables}(P_{\mathcal{S}}) = \{q \xrightarrow{a} q' : (\exists \hat{q} : (q, \hat{q}) \in \mathit{May}^0_{q \xrightarrow{a} q'} \vee (\hat{q}, q) \in \mathit{May}^0_{q \xrightarrow{a} q'})\}$$

Table 5. eliminables$(P_{\mathcal{S}})$, el conjunto de acciones eliminables de $\mathcal{S}$.

$\mathit{Fail}_{P_{\mathcal{S}}}$, entonces $q \notin \mathit{Fail}_{P_{\mathcal{S}}} \Rightarrow q \notin \mathit{Fail}_{P_{\mathcal{S}'}}$. Como $q \notin \mathit{Fail}$ sii $q \in (\mathit{Pass} \cup \mathit{May})$, se puede reescribir la segunda formula como $q \in (\mathit{Pass}_{P_{\mathcal{S}}} \cup \mathit{May}_{P_{\mathcal{S}}}) \Rightarrow q \in (\mathit{Pass}_{P_{\mathcal{S}'}} \cup \mathit{May}_{P_{\mathcal{S}'}})$. □

Ahora generalizamos este resultado, en vez de eliminar una transición, eliminamos un subconjunto de eliminables$(P_{\mathcal{S}})$.

Lema 10: Sea $\mathcal{S}$ un EIS tal que $P_{\mathcal{S}}$ puede pasar el test de bisimulación y sea $\rightarrow_\chi \subseteq \rightarrow_S$ un conjunto de transiciones tal que $\rightarrow_\chi \subseteq$ eliminables$(P_{\mathcal{S}})$. Si $S' = \langle Q_S, q_S^0, A_S^I, A_S^O, A_S^H, (\rightarrow_S) - (\rightarrow_\chi)\rangle$ y $\mathcal{S}' = \langle S', A_S^h, A_S^l\rangle$, entonces $P_{\mathcal{S}'}$ puede pasar o pasa el test de bisimulación.

Prueba: La hipótesis del Lema 8 pide que la acción que se elimina pertenezca a eliminables(P_S) y la hipótesis del Lema 10 nos asegura que $\rightarrow_\chi$ satisface $\rightarrow_\chi \subseteq$ eliminables(P_S). Sea $\mathcal{S}'$ la interfaz que se obtiene de eliminar una transición de $\rightarrow_\chi$ y $\rightarrow'_\chi$ el conjunto $\rightarrow_\chi$ sin la acción que ya se eliminó. Entonces si demostramos que $\rightarrow'_\chi$ satisface $\rightarrow'_\chi \subseteq$ eliminables$(P_{S'})$ el teorema es válido, pues eliminar un conjunto de acciones es equivalente a eliminar las acciones de a una por vez.

Sea $q_r \xrightarrow{a} q'_r \in \rightarrow'_\chi$. Como $q_r \xrightarrow{a} q'_r \in \rightarrow'_\chi$ entonces $q_r \xrightarrow{a} q'_r \in \rightarrow_\chi$, por lo que existe un estado q_a tal que $(q_r, q_a) \in \mathit{May}^0_{q_r \xrightarrow{a} q'_r}$ en P_S. Si los estados alcanzados por (q_r, q_a) en P'_S utilizando la transición $q_r \xrightarrow{a} q'_r$ son los mismos estados que se alcanzan en P_S, suponer $q_r \xrightarrow{a} q'_r \notin$ eliminables(P'_S) genera un absurdo pues ésta implica que al menos uno de los estados alcanzables ya no pertenece al conjunto *Fail*, lo cual contradice al Lema 7.

Por otro lado, si el conjunto de estados alcanzables utilizando $q_r \xrightarrow{a} q'_r$ no es el mismo, entonces la transición que se eliminó es de la forma $q'_a \xrightarrow{a} q''_a$ con $q_a \overset{\varepsilon}{\Rightarrow} q'_a$. Si el conjunto $\{(q'_r, q^*_a) : (q_r, q_a) \xrightarrow{a,\overline{a}} (q'_r, q^*_a)\} \neq \emptyset$, todos los estados del conjunto pertenecen a *Fail*, sino se generaría un absurdo con el Lema 7, por lo tanto $q_r \xrightarrow{a} q'_r \in$ eliminables(P'_S). En cambio si $\{(q'_r, q^*_a) : (q_r, q_a) \xrightarrow{a,\overline{a}} (q'_r, q^*_a)\} = \emptyset$, entonces no existe $q'_a \xrightarrow{a} q''_a$ con $q_a \overset{\varepsilon}{\Rightarrow} q'_a$ por lo que $(q_r, q_a) \xrightarrow{a} \mathit{fail}$, y por lo tanto una vez más vale $q_r \xrightarrow{a} q'_r \in$ eliminables(P'_S).

□

Por último demostremos que si una ISS puede pasar el test de bisimulación, entonces existe un conjunto de transiciones que se pueden eliminar para obtener una nueva interfaz que sí lo pasa.

Teorema 11: Sea $\mathcal{S} = \langle S, A^h, A^l\rangle$ un EIS. Si $P_{\mathcal{S}}$ puede pasar el test de bisimulación, entonces existe un conjunto $\rightarrow_\chi \subseteq \rightarrow_S \cap (Q_S \times A_S^I \times Q_S)$ de transiciones de entrada tal que si $\mathcal{S}' = \langle S', A^h, A^l\rangle$, donde $S' = \langle Q_S, q_S^0, A_S^I, A_S^O, A_S^H, (\rightarrow_S) - (\rightarrow_\chi)\rangle$, entonces $\mathcal{S}'$ es BSNNI.

Prueba: Sea $\mathcal{S}^0 = \mathcal{S}$. Sea $\rightarrow^0_\chi$ un conjunto de transiciones tal que $\rightarrow^0_\chi \subseteq$ eliminables$(P_{\mathcal{S}^0})$. Sea $\mathcal{S}^{k+1}$ el resultado de eliminar las transiciones del conjunto $\rightarrow^k_\chi$ del ISS $\mathcal{S}^k$. Definamos el conjunto $\rightarrow^{k+1}_\chi \subseteq \rightarrow_S$ como un conjunto de transiciones vacío si $P_{S^{k+1}}$ pasa el test de bisimulación. En caso de no pasarlo, será un conjunto transiciones, no vacío, tal que que $\rightarrow^{k+1}_\chi \subseteq$ eliminables$(P_{S^{k+1}})$.

Por lema 10 sabemos que P_{S^k} *puede pasar el test de bisimulación* o *lo pasa*. Pero el lema no nos asegura que en algún momento efectivamente lo pase. Supongamos que nunca ocurra. Como los estados y las transiciones son finitas, entonces hay un conjunto finito de transiciones que pueden ser eliminadas.

Supongamos que en la iteración K no hay más transiciones para eliminar, es decir eliminables$(P_{\mathcal{S}^K}) = \emptyset$. Si no se obtuvo un sistema seguro entonces P^K puede pasar el test de bisimulación. Como P^K puede pasar el test de bisimulación, entonces $\mathit{May}^0 \neq \emptyset$ por lo cual eliminables$(P_{\mathcal{S}^K}) \neq \emptyset$, absurdo. Luego en alguna iteración se obtiene un sistema seguro.

Si K es el numero de iteración en el que se obtiene el P^K que pasa el test de bisimulación, entonces se define $\rightarrow_\chi = \bigcup_i^K \rightarrow^i_\chi$ y se obtiene $\mathcal{S}'$ que pasa el bisimulation test, pues $\mathcal{S}' = \mathcal{S}^K$. □

Ejemplo 5: Aplicando el algoritmo en la interfaz $\hat{\mathcal{S}} \times \hat{\mathcal{T}}$, la cual no es BSNNI, se obtiene que se deben eliminar la transiciones con acción *correction?*. De esta forma se obtiene una interfaz que es BSNNI. El resultado de eliminar estas transiciones está graficado en la Figura 7.

Una vez más, estos resultados pueden extenderse para el caso en el que la propiedad de interés sea BNNI, pues lo demostrado en esta sección es independiente de utilizar $(\mathcal{S} \backslash A^h)$ o $((\mathcal{S}/A^{h,O}) \backslash A^{h,I})$.

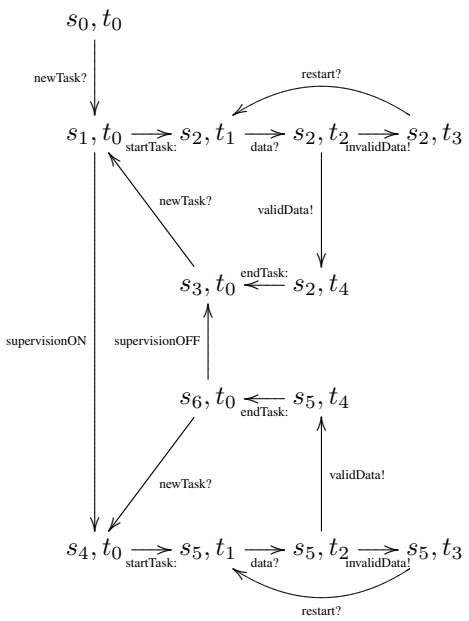

Figure 7. $\hat{\mathcal{S}} \times \hat{\mathcal{T}}$ luego de eliminar las transiciones $(s_2, t_3) \xrightarrow{\text{correction?}} (s_2, t_5)$ y $(s_5, t_3) \xrightarrow{\text{correction?}} (s_5, t_5)$.

5. Consideraciones para la generalización del conjunto de acciones a eliminar

Hasta el momento el conjunto de acciones a eliminar ha sido el conjunto de transiciones de entradas bajas. Es natural extender este conjunto al conjunto total de acciones de entrada. Al momento de implementar esta idea, surgen algunos puntos a tener en cuenta, estos se detallan a continuación.

Es claro que el procedimiento algoritmo expuesto para eliminar acciones bajas ya no sirve en un contexto en el cual las acciones altas de entrada también pueden ser eliminadas. Esto se debe a que todas las acciones altas, en el algoritmo actual, son primero ocultadas y luego reemplazadas por ε en la saturación. Esto mismo ocurre con las acciones internas de la interfaz. Por esta razón se pierde la información sobre qué acciones internas son generadas por una transición correspondiente a una acción de entrada alta. Una forma de sortear este obstáculo es reemplazar las acciones altas de entrada por otro símbolo, por ejemplo ε', y este símbolo sería utilizado sin diferenciaciones con respecto a ε en las definición del producto sincronizado.

Supongamos que ε' es el símbolo que se usa para denotar la acción interna que se puede eliminar. Si una transición interna debe ser eliminada, es porque la ejecución de esa transición lleva a una de las interfaces a un estado en el cual existe una ejecución que no puede ser realizada por alguna de ellas. Pero la misma transición que genera un *"problema"*, puede ser necesaria para que otra ejecución sea realizada por ambas interfaces. En la Figura 8 vemos un ejemplo de esto. En este ejemplo usamos el hecho de nuestras definiciones deben funcionar para el caso general (ver inicio Sección 4 y Figura 6).

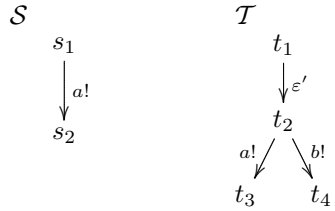

Figure 8. La transición $t_1 \xrightarrow{\varepsilon'} t_2$ permite a $\mathcal{T}$ ejecutar a y así imitar a $\mathcal{S}$, pero al mismo tiempo hace posible ejecutar una transición con $b!$, la cual no es imitable por $\mathcal{S}$.

En la Figura 8, el estado s_1 puede realizar una ejecución donde sólo se ejecuta a y alcanzar un estado que no realiza ninguna acción. Esta ejecución puede ser realizada por por t_1 utilizando previamente la transición $t_1 \xrightarrow{\varepsilon'} t_2$. Por otro lado, el estado t_1 puede ejecutar $t_1 \xrightarrow{\varepsilon'} t_2$ y luego $t_2 \xrightarrow{b} t_4$. Es claro que la última ejecución no puede ser imitada por s_1 y por lo tanto los sistemas no son bisimilares. Para evitar este problema, la única opción es eliminar la transición $t_1 \xrightarrow{\varepsilon'} t_2$, pero de esta forma t_1 ya no puede imitar la ejecución $s_1 \xrightarrow{a} s_2$. Luego este sistema no puede ser catalogado como un sistema que puede *pasar el test de bisimulación* y debería ser catalogado como un sistema que no pasa el test de bisimulación. Podemos decir que este es un *problema local*, pues si se elimina la transición $t_1 \xrightarrow{\varepsilon'} t_2$, para evitar la falla con la transición con acción b, se genera una falla que no existía relacionada con en el estado donde se elimina la transición. En este caso, en el estado (s_1, t_1).

Pero eliminar una transición con etiqueta ε' no solo puede generar un problema local, esto se puede ver en la Figura 9.

En la Figura 9, en la comparación de los estados s_3 y t_2, ambos alcanzables con una transición con acción a, existe un error debido a que t_2 luego de ejecutar ε', puede ejecutar $b!$ y este comportamiento no puede ser imitado por s_3. Eliminar la transición $t_2 \xrightarrow{\varepsilon'} t_3$ solucionaría este problema, pero generaría uno nuevo al momento de comparar s_2 y t_2 luego de eliminar la transición: s_2 aun puede realizar la transición $s_2 \xrightarrow{b!} s_4$ y esta transición ya no es imitable por t_2.

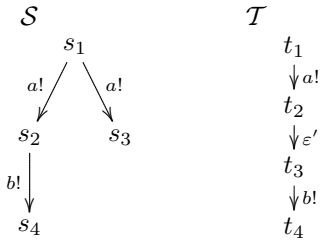

Figure 9. La transición $t_2 \xrightarrow{\varepsilon'} t_3$ permite a $\mathcal{T}$ ejecutar a y luego b y así imitar la rama izquierda de $\mathcal{S}$. Al mismo genera un conflicto con la rama derecha de $\mathcal{S}$.

6. Conclusiones y Trabajos Futuros.

Es claro que en los últimos tiempos ha habido una gran aceptación al uso de los servicios web, por esta razón es importante contar con nuevos métodos para analizar como es la composición de los mismos, más aun cuando se maneja información confidencial. Creemos que las EIS serán útiles para esta tarea; pero también sabemos que para que esto no quede solamente en un nuevo modelo teórico, deben existir formas fáciles de llevar estos análisis a la práctica. El camino más viable para lograr este objetivo, en un principio, sería la utilización de semánticas web, en las cuales es posible describir como se realiza el intercambio de mensajes entre servicios y como cooperan dos o más servicios para lograr un objetivo común [14].

El problema de generar un sistema que satisfaga la propiedad BSNNI, bajo la restricción que las acciones que se pueden eliminar son las acciones altas, fue resuelto con un enfoque distinto en [2]. En [1] se trata de generalizar el resultado sin éxito. Nuestro resultado es una nueva variante a estos resultados, pero con este nuevo enfoque, en un trabajo futuro, se presentará un algoritmo que tenga en cuenta todas las acciones de entradas al momento de restringir servicios. Esta generalización no es directa y el proceso final es más complejo.

Los trabajos futuros están orientados en tres direcciones diferentes:

- La primera dirección es reutilizar estas técnicas en el contexto de MTS (*Modal Transition System*) [15], donde se modelan el conjunto de acciones que *debe* realizar el sistema y el conjunto de acciones que *podría* realizar el sistema. Las primeras están incluidas en las segundas. Luego, las acciones que podrían ser ejecutadas por el sistema son descartadas, o incluidas en el conjunto de acciones que se deben realizar, con el objetivo de obtener un sistema seguro.
- La segunda dirección es modificar nuestra definición de EIS para modelar sistemas con mayor detalle. Este mayor detalle puede ser provisto mediante la utilizaciones de acciones con parámetros [12], o mediante la introducción de componentes probabilísticas [7].
- La última dirección esta orientada a lograr técnicas similares, a las expuestas en este trabajo, para distintas definiciones de no-interferencia. Estas definiciones son menos restrictivas y por lo tanto más fácil de lograr en la práctica [13] [10].

References

[1] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. Controller synthesis for non-interference properties. Technical Report RI-2008-01, IRCCyN, 1 rue de la Noë, BP 92101, 44321 Nantes Cedex, France, April 2008.

[2] Franck Cassez, John Mullins, and Olivier (H.) Roux. Synthesis of non-interferent systems. In *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, volume 1 of *Communications in Computer and Inform. Science*, pages 307–321. Copyright Springer, SEP 2007.

[3] Arindam Chakrabarti, Luca de Alfaro, Tom Henzinger, and Marielle Stoelinga. Resource interfaces. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*. Lecture Notes in Computer Science, Springer, January 2003.

[4] Luca de Alfaro and Thomas H. Henzinger. Interface automata. In *Proc. Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[5] Luca de Alfaro and Thomas H. Henzinger. Interface-based design. In Manfrey Broy et al., editor, *Engineering Theories of Software-Intensive Systems*, Nato Science Series. Springer, 2005.

[6] Luca de Alfaro, Tom Henzinger, and Marielle Stoelinga. Timed interfaces. In *Proceedings of the Second International Workshop on Embedded Software*. Lecture Notes in Computer Science, Springer, January 2002.

[7] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes, 1999.

[8] Jean-Claude Fernandez and Laurent Mounier. "on the fly" verification of behavioural equivalences and preorders. In *Procs. of CAV '91*, volume 575 of *LNCS*, pages 181–191. Springer, 1991.

[9] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part i: Information flow). In *Procs. of FOSAD 2000*, volume 2171 of *LNCS*, pages 331–396. Springer, 2001.

[10] Roberto Giacobazzi and Isabella Mastroeni. Generalized abstract non-interference: Abstract secure information-flow analysis for automata. In *MMM-ACNS*, pages 221–234, 2005.

[11] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[12] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 20 February 1995.

[13] Peng Li and Steve Zdancewic. Downgrading policies and relaxed noninterference. *SIGPLAN Not.*, 40(1):158–170, 2005.

[14] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the eleventh international conference on World Wide Web*, pages 77–88, 2002.

[15] Sebastian Uchitel, Greg Brunet, and Marsha Chechik. Behaviour model synthesis from properties and scenarios. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 34–43, Washington, DC, USA, 2007. IEEE Computer Society.

[16] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4(2-3):167–187, 1996.