Derivación de secuencias de test para protocolos de comunicaciones

Lic. Leonardo Marty †

Ing. Fernando Brum††

Resumen

En los últimos años se ha incrementado abruptamente el uso de redes de datos públicas que permiten interconectar sistemas heterogéneos. Los sistemas conectados a una red se comunican entre sí mediante un conjunto de reglas comunes que se denominan protocolos.

La implementación de un protocolo es una tarea verdaderamente compleja que requiere gran esfuerzo. Dicha implementación se obtiene de una especificación estándar permitiendo derivar varias implementaciones diferentes.

Estas razones motivan la necesidad de servicios de test cuyo objetivo es establecer si una implementación concreta satisface su definición. En este trabajo se intenta automatizar el proceso de derivación de test a partir de la especificación, mediante la construcción de un conjunto de algoritmos que abstraigan el procedimiento ejecutado por un experto humano.

† Escuela Superior Latinoamericana de Informática (ESLAI). Parque Pereyra Iraola, Prov. de Buenos Aires, Argentina.
Ayudante del Instituto de Computación de la Facultad de Ingeniería, Av. Julio Herrera y Reissig 565, Montevideo, Uruguay. (598) (2) 71-42-44/47
Dirección electrónica: marty@incouy.edu.ar

†† Profesor del Instituto de Computación de la Facultad de Ingeniería, Montevideo, Uruguay.
Dirección electrónica: brum@incouy.edu.ar
Director del proyectos de INTERFASE S.A., Zabala 1378, Montevideo, Uruguay. (598) (2) 96-11-43. FAX (598)(2) 96-29-65.

1. Introducción

En los últimos años, se ha incrementado abruptamente el uso de grandes redes de datos públicas. Estas, permiten interconectar sistemas heterogéneos para soportar aplicaciones distribuídas.

Los sistemas adheridos a una red, se comunican entre sí utilizando un conjunto de reglas comunes y convenciones que se denominan protocolos. El modelo de referencia ISO/OSI [ISO 83], define una arquitectura de siete niveles con el fin de estandarizar los sistemas de comunicaciones.

La implementación de un protocolo, es una tarea verdaderamente compleja que requiere, por consiguiente, un gran esfuerzo. Dicha implementación, se obtiene generalmente de una especificación estándar (informal o formal), por lo tanto, estos documentos pueden derivar a varias implementaciones distintas.

Estas razones, sumadas al creciente número de equipos heterogéneos que se pueden interconectar hoy en día, motivan la necesidad de servicios de tests. Una secuencia de tests para un protocolo, tiene por objetivo establecer cuándo una implementación concreta : IUT (Implementation Under Test), satisface su definición. La realización de estos tests implica muchas dificultades, no sólo técnicas, sino tambien políticas y legales. ISO ha intentado establecer un orden en este caos, mediante la publicación de un estándar compuesto de 5 documentos distintos [OSI 88] [OSI 89] [OSI 89a].

La derivación de una secuencia de tests adecuada, a partir de la especificación del protocolo, es un proceso largo y tedioso, realizado generalmente en forma manual por un experto humano.

En este trabajo, se intenta automatizar este proceso, mediante la construcción de un conjunto de algoritmos que abstraigan el procedimiento ejecutado por el experto humano.

El enfoque seguido, intenta mantener una cierta independencia del lenguaje de especificación, aunque se basa firmemente en el modelo de

máquinas de estados extendidas. Para obtener esta independencia, se realiza la derivación a partir de un grafo asociado al protocolo estudiado, y no directamente de la especificación del mismo. La obtención del grafo a partir de la especificación en cualquier lenguaje basado en un enfoque de máquinas de estados extendidas como puede ser SDL o ESTELLE, se puede realizar automáticamente en forma muy sencilla.

2. Sistemas de Tests de Protocolos

Existe una gran variedad de sistemas de test que se pueden aplicar sobre una implementación de un protocolo. En este trabajo se intenta testear protocolos que siguiendo el modelo OSI, se encuentran estructurados en diferentes capas, testeando una sola capa o nivel por vez. Este tipo de tests, se denominan SLCT (Single Layer Conformance Tests).

Existen distintos métodos para la realización de este tipo de tests:

- Método local: En este caso el sistema de test, tiene acceso directo a todas las interfases de la implementación.
- Método distribuído: Cuando se tiene acceso a la interfase superior de la IUT, pero no a las interfases inferiores de la IUT (exceptuando el Nivel Enlace).
- Método remoto: No se posee ningún acceso a las interfases superiores de la IUT, solamente se tiene acceso a la interfase inferior de la IUT.

El primer método es el utilizado por el propio implementador del sistema, quien tiene acceso a todas sus componentes. El segundo método surge, por ejemplo, cuando se contrata a una entidad externa para que realice el test. En ese caso no se le permite al sistema de test, el acceso a toda la implementación, sino a las interfase superior e inferior del mismo.

El método remoto, como su nombre lo indica es el que se utiliza cuando se desea realizar un test a distancia de la IUT. En ese caso no se tiene más acceso

que al nivel físico, a través de un cierto medio de comunicación. Este es el método más utilizado para el testeo de los protocolos de la familia X.25 [Lin 90]. Este será el método de test adoptado en este trabajo.

Los tests que se pueden realizar sobre la IUT se pueden clasificar en tests dinámicos o estáticos. Los aspectos estáticos se concentran en la correcta formación de las PDU (Protocol Data Units) y en la consistencia entre la implementación y las opciones de servicios que dice ofrecer la IUT. Los aspectos dinámicos de conformance se centran en el comportamiento de la IUT al interactuar con otras implementaciones (en particular el sistema de test). Son los tests dinámicos los que se generarán automáticamente.

Existen varios trabajos publicados sobre generación automática de secuencias de test a partir de especificaciones de protocolos con máquinas de estados (FSMs Finite State Machines) [N&M 81][S&D 85][Gon 70][Cho 78]. Un estudio comparativo detallado se encuentra en [S&L 89].

El problema en estos trabajos, no radica en el método seleccionado, sino en el mecanismo de especificación utilizado. Las FSMs no tienen la expresividad suficiente. El problema de la explosión de estados, hace inmanejable aún la especificación de pequeños protocolos. Además, estos mecanismos no poseen forma de expresar eventos dependientes de timers, o canales de comunicación, etc.

Sin embargo los métodos de generación utilizados, ofrecen un conjunto de ideas muy buenas. El enfoque a seguir consistirá entonces en extender estas ideas para aplicarlas a especificaciones en lenguajes como SDL [Bel 88] [CCI 88] o ESTELLE [B&D 88] [ISO 86]. Estos lenguajes ofrecen EFSM (Extended Finite State Machines) como forma de escribir procesos, que en nuestro caso serán las capas del protocolo dado.

Se debe estudiar entonces, en qué forma estos lenguajes extienden las FSMs, y de qué forma estas extensiones afectan los mecanismos de generación de tests propuestos.

3. Características del enfoque presentado

Un experto humano, al generar una secuencia de test, debe ir llevando en su mente, el estado de la IUT, entendiendo por estado, el valor que tienen determinadas variables, o el rango de valores en el que puede estar el valor de una variable en un momento dado.

Esta necesidad de tener en cuenta de alguna forma el valor de las variables y la forma en que son alteradas, es la causa de la introducción de un análisis semántico de la especificación.

Se entiende por análisis semántico de la especificación, a un análisis que requiere conocimientos sobre los dominios de valores a los que pertenecen las variables, y sus propiedades. Existen varios trabajos en este tema que realizan un análisis puramente sintáctico [Ura 87] [P&G 90] [Arm 89].

El proceso a automatizar es sumamente complejo, por lo que se hace muy difícil una automatización total del mismo. La idea que se intenta seguir, consiste entonces en intentar obtener por ejemplo, un noventa por ciento de las secuencias que un experto generaría manualmente, en forma automática.

Se darán entonces un conjunto de reglas prácticas y heurísticas, que aplicadas en los algoritmos de generación, realizarán búsquedas inteligentes de las secuencias de tests deseadas, imitando la forma de pensar de los expertos, y no realizando búsquedas exhaustivas que pueden llevar demasiado tiempo o no acabar.

De esta manera, los algoritmos presentados siempre terminan retornando uno de estos tres diagnósticos:

- EXITO: Si se completó satisfactoriamente la búsqueda solicitada.
- FRACASO: Si el algoritmo falla, no encuentra la secuencia de test buscada, pero no puede afirmar que dicha secuencia no existe.
- IMPOSIBLE: Si demuestra que la secuencia solicitada, no existe.

4. El Modelo Utilizado

Para mantener una cierta independencia del lenguaje de especificación utilizado, se presenta un modelo que abstrae, de la especificación de un

protocolo la información relevante para la generación de secuencias de tests.

Esta abstracción consistirá en manipular un protocolo como un grafo dirigido. La obtención del grafo asociado a una especificación se puede hacer en forma mecánica de una manera muy sencilla.

DEFINICION 1:

Una *especificación* E de un protocolo es una 4-upla compuesta por:
$E = < S, C, V, T >$ donde :

- $S(E) = S$ es un conjunto de estados.
- $C(E) = C$ es un conjunto de canales tipados.
- $V(E) = V$ es un conjunto de variables tipadas.
- $T(E) = T$ es un conjunto de transiciones.

El conjunto de estados, representa a los estados sintácticos ya sea de una especificación SDL o ESTELLE.

Los canales de comunicación representan o bien las *signal routes* asociadas a un proceso SDL, o los *ips* asociados a un modulo ESTELLE.

Tanto los tipos de las variables, como de los canales asociados a una especificación, se hacen accesibles a través de funciones que se llamarán: *TipoCan* y *TipoVar*.

Las transiciones de una especificación, representan la abstracción de una transición sintáctica de SDL o ESTELLE.

DEFINICION 2:

Una *transición* t es una 5-tupla formada por:

- FROM $(t) = s_0 \in S(E)$.
- TO $(t) = s_1 \in S(E)$.
- WHEN $(t) = < c, s_1(x_1,...,x_n) > \cup \{ \varepsilon \}$
  con $s_1(x_1,...,x_n)$: *TipoCan* (c) / $c \in C(E)$
- PROVIDED $(t) = pred\ (x_1,...,x_n)$ con *pred* un predicado
- BEGIN-END $(t) =$ *accion*

Las dos primeras componentes de una transición corresponden a estados sintácticos de la especificación ya sea SDL o ESTELLE. La primera indica el estado origen de la transición y la segunda el estado final de la misma.

La componente WHEN indica la señal que activa la transición. El caso en

que WHEN (t) = ε corresponde a una transición espontánea [Ora 88].

La componente PROVIDED indica la condición que se debe verificar para que una transición esté habilitada para ser disparada.

La última componente corresponde a la acción a realizar en caso de seleccionar dicha transición. Una acción consta de dos partes:

- Un conjunto de asignaciones.
- Un conjunto de mensajes a enviar (OUTPUTs).

Se referirá a estos por las funciones: Asignaciones (t) y OUTPUTs (t) respectivamente.

La nomenclatura utilizada deriva de ESTELLE. Del mismo modo se podría haber tomado de SDL (STATE, NEXTSTATE, INPUT, PROVIDED, TASK).

Observar que se está pidiendo, por las restricciones impuestas, que esta especificación se encuentre en forma normal. Propuestas de cómo llevar una especificación a su forma normal se presentan en [Sar 85].

A partir de una especificación E, se puede obtener un grafo asociado del mismo modo que para las FSMs.

DEFINICION 3:

Se define el *grafo* asociado a una especificación E como
$G = (\ (E\ ),\ T(E\ ),\ \varphi)$ donde: φ: $T(E\ ) \rightarrow S(E\ )^2$ /
$\varphi\ (t\ )$ = FROM $(t\ )$ x TO $(t\ )$.

Este grafo, no se debe confundir con la FSM subyacente, sino que, en cierto sentido es más potente, por las extensiones introducidas.

En un primer lugar, este grafo se diferencia de una máquina de Mealy, en que tiene asociado la noción de canal de comunicación. Una máquina de estados extendida, se puede comunicar con más de una entidad al mismo tiempo, por lo que cada mensaje debe ser cuantificado con un canal de comunicación como forma de indicar a quien se dirige y de quien proviene.

Se introduce también la noción de Timer, o de disparo de transiciones por tiempo. Se puede ver a las operaciones sobre Timers como una forma más

de envío y recepción de mensajes, definiendo al Timer como un canal de comunicación. Las operaciones *set* y *reset* equivalen a enviar mensajes a un cierto proceso por el canal TIMER, el cual responde por el mismo canal con mensajes llamados *time-outs* .

Esto implica que en la componente WHEN de una transición puede tener como canal asociado a: TIMER, en este caso, el mensaje corresponde a un *time-out* .

Esta noción puede verse también como una forma de introducir no-determinismo [Ora 88], en el sentido que estas transiciones pueden ser disparadas, fuera del control del sistema de test.

La mayor diferencia con las máquinas de Mealy radica en la introducción de variables. La idea básica consiste en que las variables de condición se introducen como forma de evitar el problema de explosión de estados. La técnica adecuada será entonces, introducir las variables en la menor dosis posible, sólo en los lugares que no utilizarlas provocaría una explosión en la cantidad de estados del sistema.

De este modo un estado de la especificación puede representar muchos estados de la FSM subyacente. La componente PROVIDED de una transición, simboliza la selección entre esa cantidad de estados de la FSM condensados en un único estado de la especificación. Del mismo modo, la acción a ejecutar dentro de la transición, al alterar las variables, representa parte de la noción de cambio de estado en la FSM subyacente.

De ninguna manera el enfoque a seguir será expandir este grafo a la FSM subyacente ya que ésto implicaría caer en el problema que se intentó solucionar al extender las FSM, la explosión de estados. La idea será más bien, realizar búsquedas inteligentes de caminos, y no búsquedas exhaustivas.

Otra forma en la cual se extendieron las FSMs fue cambiando la noción de mensaje. Estos pasaron, de ser manipulados como elementos de un conjunto finito, a ser manipulados mediante la noción de tipo. Esto permite por ejemplo, poder utilizar *pattern-matching* sobre los mensajes para distinguir un

subconjunto específico de mensajes, etc.

A continuación se definen algunos conceptos necesarios para el diseño del conjunto de algoritmos.

Se notará como *Val* al conjunto de todos los valores posibles que pueden tomar las variables de la especificación. Más adelante se definirá este conjunto con más detalle.

DEFINICION 4:

Se define una *asignación* como una función :
$\sigma: V(E) \rightarrow Val \cup \{\uparrow\} / \sigma(x) \in TipoVar(x)$.

Las asignaciones de variables a valores se extienden de la forma usual sobre los términos utilizados en las diferentes componentes de las transiciones.

La restricción de funciones sobre subconjunto $V(E) \supset X$ se notarán como $\sigma \mid X$.

Se introduce entonces, la noción de estado de la IUT como :

DEFINICION 5:

Un *estado* en el grafo asociado es un par $< s , \sigma >$ donde
$s \in S(E)$ y σ una asignación.

No se debe confundir la noción de estado de la IUT, con la noción de estado de la especificación notado como s. Se llamará al primero estado semántico y al segundo estado sintáctico de la especificación.

Esta noción de estado, refleja la idea de estado de la FSM subyacente al grafo dirigido, donde un estado se debe dar no sólo por el estado sintáctico de la EFSM, sino también por el valor que tiene cada una de las variables.

DEFINICION 6:

Se define el conjunto de las *Variables Alteradas* por una transición como : VarsAlts $(t) = \{ x \in V(E) / x <-$
$exp \in$ BEGIN-END $(t) \}$

DEFINICION 7:

Se define el conjunto de las *Variables Locales* a una transición como: VarsLocales (t) = Variables (WHEN (t))

DEFINICION 8:

Se define el conjunto de las *variables de condición* de una transición como: VarsCondición (t) = Variables (PROVIDED (t))

DEFINICION 9:

Se dice que un estado $< s , \sigma >$ *satisface* una transición t, lo que se anota como:

$< s , \sigma > \models t \leftrightarrow$
FROM (t) = s &
$\exists \sigma_1 / \sigma((\sigma_1 | \text{VarLocales } (t)) \text{ (PROVIDED } (t)) = \text{True.}$

Se dice que la asignación σ_1 satisface t en el estado $< s , \sigma >$. Esto se notará usualmente de la siguiente manera: σ_1: $\pi \models t$. Que un estado satisfaga una transición, significa que si la IUT se encuentra en dicho estado, entonces la transición está habilitada para ser disparada. La señal que se debe enviar para provocar esta transición debe ser: σ (WHEN (t)), con σ una asignación que satisfaga a t.

Los caminos en el grafo $G(E)$ se notarán como $(t_1,...,t_n)$. Esta secuencias de transiciones debe cumplir que:

$$\forall i \; 1 <= i <= n \quad \text{FROM } (t_i) = \text{TO } (t_{i-1})$$

DEFINICION 10:

Se define la *función de transformación* τ como la función que dado un estado, una transición y una asignación que satisface la transición en ese estado, retorna el nuevo estado al que se pasa luego de ejecutar las acciones de la transición escogida.

La definición de esta función, depende de las acciones que se permitan ejecutar en cada transición. Esto puede abarcar desde asignaciones a estructuras de control más complejas.

DEFINICION 11:

Se dice que una secuencia $(t_1,...,t_n)$ de transiciones es un *camino efectivo*

desde $\pi_0 = \langle s , \sigma \rangle$ a $t \leftrightarrow$

$\exists (\pi_0, \pi_1, \ldots, \pi_n)$ y $(\sigma_1, \ldots, \sigma_n)$ /

$\sigma_{i-1}, \pi_i \models t_i$ &

$\pi_n \models t$ &

$\pi_i = \tau(\pi_{i-1}, t_i, \sigma_i)$

La idea consiste en que un camino efectivo es un camino en la FSM subyacente, son los caminos que efectivamente se pueden recorrer en el grafo dirigido asociado.

5. Los Algoritmos de Generación

5.1. Introducción

En esta sección se presentan un conjunto de algoritmos para la generación automática de las secuencias de tests, a partir del grafo asociado a una especificación.

Las secuencias que se intentan generar, son los tests de comportamiento, que se encargan de verificar que cada transición esté correctamente implementada. Es decir que para cada transición de la especificación se generará un test, que consiste en provocar en la IUT la ejecución de la misma y estudiar su comportamiento.

Aplicando las ideas propuestas para FSMs, para cada transición t se generará una secuencia de la siguiente forma:

- PREAMBULO $\propto$ TEST $\propto$ CHEQUEOFINAL

donde $\propto$ denota la concatenación de secuencias.

A pesar de que se hable continuamente de secuencias de test, el término correcto debiera ser árboles o digrafos. De todos modos el término secuencia hace alusión a que sólo un camino, el que conduce al test, es el que verdaderamente interesa de la estructura.

DEFINICION 12:

Un *preámbulo* consiste en un camino efectivo en el grafo subyacente a la especificación, desde un estado inicial, a la transición a testear.

El test consiste simplemente en enviar el mensaje : σ(WHEN(t)) donde

σes una asignación que satisface *t*.

El mayor problema a resolver, en la ejecución de esta parte del test, es resolver que diagnóstico emitir una vez que se obtiene una respuesta de la IUT al estímulo enviado.

El chequeo final consiste en verificar que la ejecución de la transición *t* dejó a la IUT en el estado TO (*t*).

La generación de este tipo de tests, implica resolver los siguientes problemas:

- Encontrar el preámbulo adecuado. Como se necesita que se satisfaga la condición del PROVIDED de la transición a testear, dependiente de las variables, ya no es suficiente un análisis puramente sintáctico, sino que también es necesario un análisis semántico. Se necesita manejar la noción de estado definida en la sección anterior.
- Emitir un diagnóstico. Una vez que se lleva a la IUT al estado deseado, es necesario analizar todas las respuestas posibles debido al no-determinismo inherente a la especificación para emitir un correcto diagnóstico acorde a la respuesta obtenida.
- Distinguir cuando una transición es testeable o no. Gran cantidad de transiciones pueden depender de eventos no controlables por la IUT por lo que no es posible testearlas. Incluso estados enteros pueden ser inaccesibles, para el sistema de test.
- Seleccionar un lenguaje adecuado para la descripción de las secuencias generadas.

5.2. TTCN

Con el objetivo de tener un lenguaje estándar para la especificación del comportamiento de los sistemas de test, y las entidades a ser testeadas, la ISO definió una notación llamada TTCN (Tree and Tabular Combined Notation) estandarizada en [OSI 89].

Esta notación aspira a ser lo suficientemente general como para permitir describir tests para cualquier protocolo OSI. TTCN no tiene la intención de ser un lenguaje ejecutable, sino que esta notación permite definir una secuencia de test abstracta, independiente de la implementación. Cada secuencia abstracta se compone de una sucesión de eventos atómicos con los suficientes

detalles como para juzgar el comportamiento de una implementación del protocolo y formular un diagnóstico (PASS, FAIL, INCONCLUSIVE).

5.3. Supuestos básicos

El conjunto de algoritmos presentados, se basan en ciertos supuestos básicos. La intención es que estos supuestos sean lo suficientemente amplios como para no excluir ningún protocolo OSI.

Estos supuestos son los siguientes:

- Los tests a realizar serán tests remotos en los que el sistema de test, solamente tiene acceso a la interfase inferior de la IUT. Por esta razón, se debe indicar en la información de la arquitectura del test el nombre del canal de comunicación, controlable por el sistema de test. A este canal se le denominará : c,
- La probabilidad de ruido en la línea de test es despreciable. Este es el supuesto más fuerte que se realiza, porque no siempre es válido, pero esta suposición simplifica enormemente el trabajo.
- Existe una secuencia que se llamará *ri* (*Reset Input*) y un estado $\pi_{inic} = < s_{inic}, \sigma_{inic} >$, tal que dicha secuencia aplicada en cualquier momento, conduce a la IUT al estado inicial π_{inic}.
- El predicado $|=$ es decidible, existiendo un algoritmo para encontrar en cada caso, una asignación que satisface la transición para el estado dado.

Este último supuesto se introduce por utilizar un análisis semántico sobre la especificación del protocolo.

El pedir decidibilidad para el predicado $|=$ puede parecer restrictivo, pero la experiencia demuestra que no lo es. Incluso, en la práctica, se trabajó con restricciones mucho más fuertes que esta, como exigir que las variables de condición sean de algún tipo isomorfo a Z_n con $n \in N$. Esto permite restringir los predicados a utilizar a los operadores de comparación respectivos y las expresiones en las asignaciones, a sumas y productos. Esta condición más restrictiva que la original, sobra para todos los protocolos experimentados.

Se explica fácilmente al observar la naturaleza de las variables utilizadas en la componente PROVIDED de las transiciones, porque alcanza con

pedir dominios isomorfos a Z_n.

Estas variables son generalmente:

- booleanas: isomorfas a Z_2. (Ej : I´m Busy, etc)
- contadores de reintentos: isomorfo a $Z_{Cantintents}$.
- variables de numeración de PDU : como ser vs, vr.

Esta restricción transforma al conjunto *V*, en el conjunto de los naturales. De todos modos esta restricción no impide la utilización de tipos más complejos como *buffers* o *stacks*, puesto que generalmente la condición que se testea sobre estas variables son por ejemplo sobre su largo, (si está lleno, si está vacio), condición que se puede representar con una variable del tipo deseado.

5.4. Detección de Estados y Transiciones Inaccesibles

Existen muchas transiciones inaccesibles debido a eventos no controlables en la IUT. A grandes rasgos, se pueden distinguir:

- Variables no controlables. Las condiciones dependientes de la implementación como ser que una IUT se encuentre *busy*, se expresaron en la especificación como variables a las que se consulta su valor, pero nunca se modifican. Esas variables, se pueden detectar con un análisis sintáctico de la especificación, siendo las transiciones que dependen de estas variables en sus condiciones inaccesibles.
- Canales no controlables. Por asumir que el test es remoto controlando únicamente el canal c_i, no es posible provocar las transiciones que dependen de recepción de mensajes por canales diferentes a c_i. Estas transiciones tampoco serán controlables.
- Estados inaccesibles. Existen estados que no son accesibles porque todas las transiciones que conducen a él son inaccesibles. A su vez todas las transiciones originadas en esos estados, serán inaccesibles.

Estas condiciones, se combinan entre sí, generando más transiciones inaccesibles. Se presenta a continuación un algoritmo para detectar transiciones no testeables.

La idea para detectar todas las transiciones inaccesibles para el sistema de test, consiste en marcar las transiciones que son directamente inalcanzables (las que dependen de canales o variables no controlables, etc.), lo que

permite visualizar qué estados son inaccesibles. De esta manera se van derivando nuevas transiciones inalcanzables (las que tienen su origen en estados ya marcados) y a su vez nuevos estados inaccesibles. Al finalizar este proceso se tienen marcadas un conjunto de transiciones que son inaccesibles al sistema de test.

Una transición marcada por este algoritmo es necesariamente no testeable, pero ésta no es una condición suficiente. Por ejemplo, en el caso de testear el nivel 2 del protocolo X.25, no se puede obligar a la IUT a emitir tramas de información, por lo tanto no se puede alterar la variable que cuenta las tramas emitidas en la IUT. Este caso no es detectado por este algoritmo.

Un caso particular ocurre con las transiciones que dependen del canal TIMER, o sea las transiciones que dependen de eventos provocados por time-outs. Estas transiciones se consideraron como accesibles, puesto que para provocarlas, simplemente se debe hacer que el sistema de test permanezca sin transmitir PDUs por un tiempo 2d + t, siendo d la demora de la transmisión y t el tiempo de time-out para dicha transición.

A pesar de esto, estas transiciones serán consideradas como incontrolables en el momento de emitir diagnósticos en un test, puesto que pueden ser provocadas al expirar el tiempo de espera, sin que el sistema de test lo desee, debido a demoras en la transmisión de un mensaje.

5.5. Algoritmos para Encontrar Preámbulos

El problema a resolver consiste en encontra un camino efectivo desde el estado inicial π_{inic} a la transición a testear t. Esto implica no sólo encontrar un camino efectivo desde π_{inic} a FROM (t), sino que debe suceder que el estado final de ese camino debe satisfacer la transición t.

Este problema se subdivide a su vez en dos problemas más simples, primero encontrar un camino efectivo desde π_{inic} al resto de los estados de la especificación. Una vez resuelto esto, se desea verificar que la IUT quede en condiciones de poder ejecutar la transición deseada, o sea que se satisfacen la

condición del WHEN.

La búsqueda del camino desde π_{inic} a cada estado de $S(E\ )$, se realiza solo una vez para cada estado, independiente de la cantidad de transiciones que se originan para cada estado. Luego, se debe verificar una vez para cada transición, que la IUT queda en condición de ejecutar la transición a testear. En caso de que no se esté en condiciones, se debe buscar un ciclo en el grafo que permita hacer válida la condición del WHEN de la transición que se desea seleccionar.

Para encontrar un camino desde el estado inicial a cualquier otro estado de $S(E\ )$, se propone aplicar el algoritmo de Dijkstra para encontrar caminos mínimos en un grafo [A&H 76] modificado de manera que los caminos que obtengan sean caminos efectivos.

ALGORITMO 1:

EstsRevis : $P(S(E))$
Expandir : STACK ($<S(E\ ), V(E) \rightarrow Val>$)

EstsRevis := { s_{inic} }
Expandir := { π_{inic} }
MIENTRAS (EXPANDIR $\neq \emptyset$) *HACER*
  $< s , \sigma >$:= Pop (Expandir)
  $\forall t \in T(E)$ / ($< s , \sigma > \models t$) & not(TO (t) $\in$ EstsRevis)
    EstsRevis := EstsRevis $\cup$ { To (t) }
    σ_1 := AsignacionSatisfact (t, σ)
    π := $\tau(< s , \sigma >, t, \sigma_1)$
    Push (Expandir, π)
RETURN (EstsRevis)

Este algoritmo encuentra un camino efectivo desde π_{inic} a cada uno de los estados en el conjunto EstsRevis en el momento de terminar su ejecución.

Este algoritmo utiliza el último supuesto realizado, la decidibilidad del predicado $\models$. Este utiliza también el algoritmo para encontrar la asignación adecuada AsignaciónSatisfact.

Trabajando con la restricción : $\forall x \in V(E\ )$ TipoVar(x) = Z_n ($n \in N$), se pueden dar reglas para decidir que asignación elegir acorde a los intereses del

test. Por ejemplo una buena regla puede ser, probar los casos de borde, que son los casos con más probabilidad de error en una implementación.

Este algoritmo, es una generalización del enfoque puramente sintáctico. La inclusión del análisis semántico en este algoritmo se realiza a través del predicado $|=$. La no utilización de este predicado, transformaría el algoritmo propuesto en el de Dijkstra. No incluir este predicado en el algoritmo 1, sería absurdo porque los caminos generados, no tienen por que ser efectivos, por lo que hay que revisarlos manualmente uno a uno, lo cual se contradice con el objetivo perseguido.

Esto no implica que no se pueda eliminar el análisis semántico, sino que se puede sustituir el predicado $|=$, por alguna condición sintáctica que implique al predicado $|=$. Una condición que se puede usar en este sentido, puede ser: $VarLocales(t) \supseteq VarProvided(t)$.

Utilizando en lugar del predicado $|=$, una condición de este estilo, obliga la obtención de caminos efectivos, lo que es el objetivo perseguido.

Para cada transición a testear, se debe verificar ahora, que el camino que lleva la IUT al estado origen de la transición encontrado por el algoritmo anterior, la deja en un estado π tal que $\pi |= t$.

En caso negativo debe intentar resolver el problema mediante la localización de un ciclo que cambie el estado de la IUT, hasta satisfacer la transición. La búsqueda de este ciclo se realiza mediante la utilización de una heurística adecuada.

Una heurística sumamente elemental, pero que en la práctica resulta realmente efectiva, consiste en localizar un lazo en ese estado donde se alteran las variables de condición de dicha transición.

5.6. Ejecución del Test y Emisión de Diagnóstico

En este punto, se tienen ya las secuencias que llevan a la IUT a un estado π tal que satisface la transición t, que se desea testear. Ahora se debe efectuar el test y emitir un diagnóstico adecuado.

El primer problema a resolver consiste en evitar repeticiones de pruebas sin sentido. Este caso presentado en [P&G 90] y [P&G 90a] [P&G 90b] se da cuando se tienen transiciones idénticas salvo por las acciones no observables por el sistema de test. En este caso se debe realizar una única prueba para todos estos casos.

Para resolver este problema se introduce una noción de orden entre transiciones.

DEFINICION 13:

Se dice que $t <= t_1$ si las transiciones son idénticas, salvo que los OUTPUTs de t que no utilizan el canal de test c_t están incluídos en los de t_1.

La idea es entonces testear solamente las transiciones minimales con respecto a esta relación de orden. Esto evita hacer pruebas semejantes desde el punto de vista del sistema de test, eligiendo las menores como forma de evitar la mayor cantidad de efectos no observables posibles.

De esta manera, para las transiciones seleccionadas, la realización de la prueba consiste simplemente en obtener una asignación σ que satisface t en el estado π y enviar la señal σ(WHEN (t)). De acuerdo a la señal recibida como respuesta, se debe emitir un diagnóstico (PASS, INCONCLUSIVE o FAIL).

Entonces, estando en el estado $\pi = < s , \sigma >$, al ejecutar el test σ_1 (WHEN (t)) para la transición t, el diagnóstico correspondiente se calcula de la siguiente manera:

- Si
  $< c_1, s_1(x_1, ..., x_n) > \in$ OUTPUT (t) &
  $\tau(\pi, t, \sigma_1) = < s_2 , \sigma_2 >$,
  $\sigma(\sigma_1 |$VarLocales (t)) PROVIDED (t)) = TRUE &
  entonces $\sigma_2(s_1(x_1, ..., x_n))$ genera un diagnóstico PASS.
- Si $\exists t_1 \in T(E)$ /
  FROM (t_1) = s &
  WHEN (t_1) = ε &
  $< c_1, s_1(x_1, ..., x_n) > \in$ OUTPUT (t_1) &
  $\sigma(\sigma_1 |$VarLocales (t_1)) PROVIDED (t_1)) = TRUE &
  $\tau(\pi, t_1, \sigma_1) = < s_2 , \sigma_2 >$,
  entonces $\sigma_2(s_1(x_1, ..., x_n))$ provocará un diagnóstico INCONCLUSIVE.

- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c, s_1(x_1, \ldots, x_n) \rangle$ $c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  $\sigma(\sigma_1 |$VarLocales $(t\ ))$ PROVIDED $(t_1))$ = TRUE &
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \sigma_3 \rangle$,
  entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico INCONCLUSIVE.
- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c, s_1(x_1, \ldots, x_n) \rangle$ $c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  PROVIDED $(t_1) \cap$ VarIncont $(t_1) \neq \emptyset$
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \sigma_3 \rangle$,
  entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico INCONCLUSIVE.
- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c_1, s_1(x_1, \ldots, x_n) \rangle$ $c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  $\sigma(\sigma_1 |$VarLocales $(t_1))$ PROVIDED $(t_1))$ = TRUE &
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \sigma_3 \rangle$,
  entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico PASS
- En el resto de los casos el diagnóstico debe ser FAIL.

En esta descripción, el primer item corresponde al caso esperado, el segundo a transiciones espontáneas, el siguiente a transiciones que dependen de canales o variables no controlables y el último al no-determinismo interno explícito de la especificación.

En los casos de no-determinismo explícito de la especificación, no se puede obligar a la IUT a seleccionar una de las opciones dadas. Esto implica que el diágnostico sea PASS, porque hubo un comportamiento correcto, pero eso no implica que las otras opciones no seleccionadas, se encuentren correctamente implementadas.

6. Conclusiones y Extensiones

Estas ideas fueron aplicadas a la generación de las secuencias de test para el protocolo LAPB (nivel 2 de X.25), [CCI 88a] y se compararon con un conjunto de tests generados por un experto humano.

El primer algoritmo, detectó caminos efectivos a 5 de los 9 estados de la especificación y probó que no existían tales caminos para 3 de los estados.

Emitió un diagnóstico de FRACASO solo en un caso.

El segundo algoritmo, encontró el camino efectivo a 83 transiciones en un total de 119, para 35 probó que era imposible realizar un test, fallando nuevamente en un solo caso.

Se generaron entonces 83 secuencias de test para dicho protocolo. Se hace necesario un mecanismo para generar únicamente un subconjunto de las pruebas. Esto se hace evidente en los casos de excepción donde se genera una prueba para cada estado del sistema.

La preocupación debe ser -entonces- limitar la cantidad de pruebas generadas más que generar las pruebas que faltan.

Otra extensión que se debe realizar, consiste en modificar el modelo utilizado para que una asignación permita manejar el caso en el que se conoce el rango de variación de una variable pero no se sabe cual es su valor efectivo. Las asignaciones deben pasar a ser entonces funciones de variables en $P(Val)$.

Previo a implementar estos algoritmos, se realizó una implementación de un sistema de test para las secuencias generadas -en particular- para el LAPB. Dicho sistema se implementó en C++ sobre un ambiente UNIX, diseñado como un servicio para el conmutado URUPAC de ANTEL - URUGUAY.

7. Agradecimientos

A Ana María por su permanente apoyo.

A Ingrid, Jaime y Marcelo por su invalorable colaboración en la corrección y edición del presente trabajo.

8. Bibliografía

[A&H 76] A.V.Aho, J.E.Hopcroft, J.D.Ullman. The Design and Analysis of Computer Algorithms. Addison Wesley Publishing Company, 1976.

[Arm 89] F.Armengaud. Generation de specifications de tests. Validation de logiciels de systemes de telecomunications. Note Technique NT/PAA/CLC/LSC/2234 TELECOM, 1989.

[B&D 88] Budkowski. P.Dembinski. An introduccion to Estelle, Specification Laguaje for distributed systems. Computer Networks & ISDN Systems, vol 14, 1988.

[Bel 88] Ferene Belina. Introduccion to SDL, 1988.

[CCI 88] CITT Recomendation Z.100, Specification and Description Languaje SDL., ITU, X, BLUE BOOK, Annexes A-F to Z.100. Geneva, 1988.

[CCI 88a] CCITT Recomendation X.25, Intefase between Data Terminal Equipment (DTE) and Data Circuit Equipment (DCE) for Terminal Operating in the Packet Mode on Public Data Networks.ITU VIII.BLUE BOOK. Geneva, 1988.

[Cho 78] T.Chow. Testing Software Design Modeled by Finite State-Machines. IEEE Transactions on Software Eng, Vol SE-4, 1978.

[Gon 70] G.Gonene. A Method for the Design of Fault Detection Experiments. IEEE Fault Tolerant Comput, vol C-19, 1970.

[ISO 83] ISO 7498: Open System Inteconnection, Basic Reference Model, 1983.

[ISO 86] ISO. DP 9074 rev. - ESTELLE, A Formal Description Technique based on an Extended State Transition Model, 1986.

[Lin 90] R.J.Linn. Conformance Testing for OSI Protocols. Computer Networks and ISDN Systems, vol 18, pag 203-219. Elsevier Science Publishers, 1990.

[N&T 81] S.Naito. M.Tsunoyama. Fault Detection for Sequential Machines by Transmition Tours. Proc. IEEE Fault Tolerant Computers, Conf, 1981.

[Ora 88] F.Orava. Formal Semantics of SDL Specification Proc. on Protocol Specification, Testing and Verification VII. pag 143-158, 1988.

[OSI 88] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21 DIS 9641. Partes 1-2. 1988.

[OSI 89] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21N3077. Parte 3, 1989.

[OSI 89a] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21DIS 9646. Partes 4-5, 1989.

[P&G 90] Marc Phalippou. Roland Groz. From Estelle Specifications to Industrial Test Suites, Using an Empirical Approach. Forte 90 1990

[P&G 90a] Marc Phalippou. Generation de Suites de Tests TTCN pour le LapD a partir d'Estelle. Journees Firtech, 1990.

[P&G 90b] Marc Phalippou, Roland Groz Evaluation of an empirical approach for computer-aided test case generation. 3rd International Workshop on Protocol Test Systems, Washington, 1990.

[Sar 85] Sarikaya B. Bochmann G. Obtaining Normal Form Specification for Protocol, procedings COMNET'85. pag 601-612, Budapest, 1985.

[S&D 85] K.Sabnami. A.Dahbura. A New Technique for Generating Protocol Tests. AT&T Bell Laboratories, Murray Hill, N.J., USA. ACM 0-89791-164-4, 1985.

[S&L 89] D.P.SIDHU. T.K.Lung. Formal Methods for Protocol Testing: A Detailed Study. IEEE Transactions on Software Engineering, Vol 15, No 14, 1989.

[Ura 87] H.Ural. A Test Derivation Method For Protocol Conformance Testing. Proc Protocol Specification Testing and Verification VII (North Holland). pag 347-359 Elsevier Science Publishers B. V, 1987.