UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S.I. DE TELECOMUNICACIÓN

CONTRIBUCIÓN A LA FORMALIZACIÓN DE LA FASE DE EJECUCION DE PRUEBAS

Tesis Doctoral

Gabriel Huecas Fernández-Toribio

Abril de 1995

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS

E.T.S.I. DE TELECOMUNICACIÓN

CONTRIBUCIÓN A LA FORMALIZACIÓN DE LA FASE DE EJECUCION DE PRUEBAS

Tesis Doctoral

Autor: Gabriel Huecas Fernández-Toribio
Ingeniero de Telecomunicación
Director: José Antonio Mañas Argemí
Doctor Ingeniero de Telecomunicación

Abril de 1995

T E S I S D O C T O R A L

CONTRIBUCIÓN A LA FORMALIZACIÓN DE LA FASE DE EJECUCIÓN DE PRUEBAS

T R I B U N A L C A L I F I C A D O R

Presidente:

Vocales:

Secretario:

Vocales Suplentes:

Acuerdan otorgar la calificación de:

Madrid, de de 1995

A Paqui,
a mis padres y hermanas.

Resumen

En el campo de la Ingeniería de Protocolos es fundamental el papel que han tomado los organismos normalizadores de Servicios y Sistemas de Comunicaciones, como ISO e ITU. En este entorno, las Técnicas de Descripción Formal son un mecanismo clave para el diseño y especificación de dichos protocolos.

Esta actividad ha surgido, en gran parte, debida a las necesidades de interconectividad, que está alcanzando niveles difícilmente imaginables hace pocos años: se pretende que sistemas heterogéneos y completamente diferentes cooperen y trabajen de forma distribuida o, simplemente, que intercambien volúmenes de información cada vez mayores.

El surgimiento de normas supone un cambio en la mentalidad tanto en el desarrollo como en la producción para los fabricantes de sistemas de comunicaciones.

Surgen normas y recomendaciones a partir de iniciativas públicas orientadas a proporcionar normas en los servicios y protocolos de comunicaciones; normas que los fabricantes deben cumplir y organismos independientes deben certificar u homologar.

Existen dos campos de actuación bien diferentes: por un lado, las normas deben ser precisas y no contener ambigüedades. Por otro, es necesario comprobar que el producto se atiene a la norma. Este proceso se realiza en base a unas pruebas denominadas de *Conformidad*.

En el primer campo es el causante directo del desarrollo de las FDTs. El segundo, ha provocado que ISO normalice un entorno específico y una metodología para el desarrollo y ejecución de Pruebas de Conformidad: la norma ISO-9646.

LOTOS es una de las FDTs normalizadas por ISO. Con una base fuertemente matemática basada en Álgebras de Procesos, se ha utilizado para especificar sistemas de comunicaciones durante los últimos años.

En este entorno tiene lugar el desarrollo de la presente tesis. Como objetivos fundamentales se ha trabajado en:

- conceptualización y subsiguiente formalización del proceso de ejecución de Pruebas de Conformidad y elementos integrantes en las arquitecturas de pruebas.
- definición y formalización de los componentes de las pruebas.
- definición de una métrica de cobertura que aproveche la existencia de especificaciones formales como elemento de referencia para la generación de las pruebas de conformidad.

Los estudios desarrollados han dado como fruto las siguientes contribuciones:

- Formalización del concepto de *Punto de Control y Observación*, definido en ISO-9646, punto de importancia capital en el acceso al producto bajo pruebas.
- Desarrollo de una métrica de cobertura adecuada la la FDT LOTOS. Debido a la particular semántica de composición de procesos y al general concepto de sincronización de dicho lenguaje, las adaptaciones de métricas convencionales de Ingeniería Software no son suficientes.
- Formalización del concepto de *prueba correcta* como aglutinador de todos los elementos formalizados que deben integrarse en una prueba.
- Formalización del proceso de ejecución de pruebas sobre productos reales, con la consiguiente interpretación de resultados bajo la óptica de *asignación de veredictos*, definido por ISO-9646.

El capítulo 1 presenta en profundidad los objetivos de la tesis enmarcados en el entorno de trabajo. El capítulo 2 presenta una perspectiva de campo e introduce las bases de la investigación.

En el capítulo 3 se estudian en detalle los *PCOs*, formalizándolos y proporcionando una taxonomía de los modelos más interesantes.

En el capítulo 4 se hace una análisis de las métricas de cobertura convencionales adaptadas al lenguaje LOTOS como introducción a la propuesta Métrica de Combinación de Eventos. Posteriormente, en el capítulo 5 se presenta una posible implementación de un algoritmo que implemente el cálculo de dichas medidas.

El capítulo 6 presenta la formalización del proceso de ejecución de pruebas y lo que entendemos por *prueba correcta*. Aquí se definen todos los componentes que debe incluir una Prueba de Conformidad, especificados formalmente en LOTOS.

El capítulo 7 presenta las conclusiones de este trabajo, así como las vías de investigación futura. Además, se ofrece un breve panorama de trabajos que validan las presentes contribuciones.

El apéndice A es un glosario de términos usados en la presente tesis. El apéndice B contiene la especificación formal de los tipos de datos usados en la formalización de los PCOs. El apéndice C contiene dos ejemplos de aplicación de la métrica de cobertura.

Abstract

In the Protocol Engineering field the rol of standardization organizations of Communicating Services and Systems, such as ISO or ITU, is fundamental. In this context, Formal Description Techniques are a key tool in the design and specification of such protocols.

This activity has arisen, mainly, due to the interconnectivity needs, that are reaching levels which were unimaginable a few years ago: it is intended that heterogeneous, and completely different systems will cooperate and interconnect in a distributed way or, simply, will interchange greater and greater amounts of data.

Standards imply a change in the way of thinking both in the development and in the production for manufacturers of communicating systems.

Standards and recommendations originate from public initiatives, oriented towards providing standards of services and protocols; standards that have to be accomplished by manufacturers and that should be certified or homologated by independent laboratories.

There exist two very different aspects: firstly, norms should be precise and not contain ambiguities. Secondly, it is necessary to check that the product fulfills the norm. This process is performed by executing tests called *Conformance Tests*.

The first field is the direct originator of the development of FDTs. The second one has induced ISO to standardize a specific framework and a methodology for the development and execution of Conformance Testing: ISO-9646.

LOTOS is one of the FDTs standardized by ISO. Having a strong mathematical support based on Process Algebras, it has been used to specify communicating systems for several years.

In this framework the current PhD Thesis has been developed. As main objectives, we have worked on:

- Conceptualization and subsequent formalization of the Conformance Test Execution process and the elements belonging to the test architectures.
- Definition and Formalization of the test elements.
- Definition of a coverage metric that takes advantage of the existence of formal specifications as a reference element to generate conformance tests.

The studies carried out have produced the following results:

- Formalization of the *Points of Control and Observation* concept, defined in ISO-964. They are the key to access the product under test.
- Development of a coverage metric suited to the LOTOS FDT. Due to the particular semantic of process composing and the rendez-vous synchronization concept of the language, adapting conventional Software Engineering metrics are not sufficient for measuring how much specification has been covered.
- Formalization of the concept of *correct test* as collector of all formalized elements to be integrated in the test.
- Formalization of the test execution process on real products, with the subsequent interpretation of result under the optic of *verdict assignment*, defined by ISO-9646.

Chapter 1 presents in depth the objectives of the Thesis. Chapter 2 shows a field study and introduces the research basis.

In chapter 3 *pcos* are studied in detail, formalizing them and providing a taxonomy of the most interesting models.

In chapter 4 an analysis of conventional coverage metrics adapted to the LOTOS language is carried out as an introduction to the proposed Event Composition Metric. Subsequently, in chapter 5 an implementation of the algorithm to calculate such a measure is described.

Chapter 6 formalize the test execution process and what is known as a *correct test*. Here, components belonging to a conformance test are defined and formally specified in LOTOS.

Chapter 7 presents the conclusions of this work, as well as future researching fields. in addition, a brief overview of papers that validate the current contributions are described.

Appendix A is a glossary of terms used in the thesis. Appendix B contains the data types used in the PCOs formalization. Appendix C contains two examples of application of the proposed metric.

Agradecimientos

Quisiera agradecer a José Antonio Mañas Argemí como tutor, y a Tomás Robles Valladares como principal colaborador, el intercambio de ideas, soporte, animo y apoyo mostrados durante todo el tiempo de trabajo con ellos, en particular para la elaboración de esta Tesis. No puedo olvidar en este apartado al mejor cibernauta del hiperespacio, Joaquín Salvachúa Rodríguez, cuyas ideas y contribuciones también has sido esenciales para este trabajo.

A David Fernández Cambronero, amigo y compañero de despacho, que me animó a escribir y me ayudó a formatear la presente memoria.

A Juan Carlos Dueñas López, amigo, compañero de trabajo y cuñado, con el que pasé tan buenos ratos en la carrera y me proporcionó el presente formato de memoria.

A los proyectistas Pedro Jara e Ignacio Salas, que colaboraron en la validación de la metodología y sufrieron la realización de prototipos.

Al resto de miembros del Departamento de Ingeniería de Sistemas Telemáticos, sin los cuales no hubiera existido un entorno donde crear esta tesis.

A *Pink Floyd*, *Mike Oldfield* y *el coro de monjes de la abadía de Santo Domingo de Silos*, que me proporcionaron la concentración necesaria para escribir la presente memoria.

A mi familia y mi mujer, Paqui, cuyo constante apoyo y cariño hicieron que mereciera la pena todos los esfuerzos por elaborar la presente contribución.

Demasiado viejo para el rock’n’roll
Demasiado joven para morir.
Jethro Tull

Contenido

Indice de figuras. **xix**

Indice de tablas. **xix**

1 Introducción y Objetivos **1**
1.1 - Introducción 1
1.2 - Pruebas de Conformidad 2
1.3 - Entorno de la Tesis 3
1.4 - Objetivos de la Tesis 3
1.5 - Estructura de la Memoria 4

2 Estado del Arte **7**
2.1 - Introducción 7
2.2 - Entorno y Metodología de Pruebas en OSI 7
2.2.1 - Ejecución de las Pruebas de Conformidad 9
2.3 - Técnicas de Descripción Formal 10
2.3.1 - El lenguaje SDL 11
2.3.2 - El lenguaje ESTELLE 12
2.3.3 - El lenguaje LOTOS 12

3 Modelado y Caracterización de los Puntos de Control y Observación **19**
3.1 - Introducción 19
3.2 - Características de un PCO 20
3.2.1 - Propiedades de Alto Nivel 20
3.2.2 - Propiedades Elementales: Definición y Formalización 22
3.3 - Clasificación de Modelos de PCOs Reales 24
3.4 - Especificación de PCOs en LOTOS 26
3.4.1 - Anotaciones 26
3.4.2 - Anotaciones para Modelización de PCOs 26
3.4.3 - Lenguaje de Anotaciones para Modelización de PCOs 28

4 Análisis de Cobertura **33**
4.1 - Introducción 33
4.2 - Cobertura de Comportamiento 34
4.2.1 - Cobertura de Puertas 35

4.2.2 - Cobertura de Patrones de Acciones . . . 35
4.2.3 - Cobertura de Predicados y Guardas . . . 36
4.2.4 - Cobertura de Composición de Eventos . . . 37
4.3 - Definición de la Métrica de Composición de Eventos . . . 39
4.4 - Conclusiones . . . 47

5 Algoritmo para Cálculo de Cobertura 49
5.1 - Introducción . . . 49
5.2 - Identificación sintáctica de eventos simples de LOTOS . . . 49
5.3 - Reconocimiento de Patrones de Sincronización de Comportamientos (PSC) 50
5.3.1 - Definición de los PSC . . . 53
5.4 - Esbozo del Resto del Capítulo . . . 53
5.5 - Formalización del Algoritmo de Cálculo de Cobertura . . . 54
5.5.1 - Consideraciones previas . . . 54
5.5.2 - Nomenclatura . . . 56
5.5.3 - Función de Identificación de Expresiones LOTOS IB . . . 60
5.5.4 - Función de extracción de ofertas Of . . . 61
5.5.5 - Función de activación de ofertas gl . . . 61
5.5.6 - Función de Extracción de PSC RC . . . 65
5.5.7 - Proyección del Árbol de Sincronización . . . 65
5.5.8 - Definición de Igualdad de PSCs . . . 68
5.5.9 - Función de Cálculo de Combinaciones . . . 68
5.5.10- Terminación y Completitud del Algoritmo . . . 68
5.5.11- Definición en PASCAL del algoritmo . . . 70
5.5.12- Ejemplos de Aplicación . . . 71

6 Arquitectura de un Sistema de Ejecución de Pruebas 77
6.1 - Introducción . . . 77
6.2 - Generación de Casos de Prueba . . . 80
6.2.1 - Propósitos de Prueba . . . 80
6.2.2 - Temporizadores . . . 81
6.2.3 - Selección de Datos de Prueba . . . 81
6.2.4 - Clases de Pruebas . . . 81
6.3 - Otras Aproximaciones . . . 83
6.4 - Ejecución de las Pruebas . . . 83
6.4.1 - Relación entre la formalización y su concreción . . . 84
6.4.2 - Imposición versus Observación . . . 85
6.4.3 - Seguimiento Sobre la Especificación de Referencia . . . 86
6.4.4 - Bloqueo . . . 86
6.4.5 - Pruebas correctas . . . 87
6.4.6 - Etiquetado de Pruebas en LOTOS . . . 88
6.4.7 - Cobertura . . . 88
6.4.8 - Asignación de veredictos . . . 90
6.5 - Formalización de la Ejecución de Pruebas . . . 90

6.6 - Conclusiones 95

7 Conclusiones **97**
7.1 - Conclusiones 97
7.2 - Vías de Investigación Futuras 98

A GLOSARIO **101**

B Tipos Básicos para la Formalización de los PCOs **105**

C Ejemplo de Aplicación de la Métrica de Composición de Eventos **109**
C.1 - Ejemplo de Aplicación de Medidas de Cobertura: La Criba de Eratóstenes 109
C.1.1 - La Criba de Eratóstenes 109
C.1.2 - Descripción de la Especificación 110
C.1.3 - Especificación LOTOS de la Criba de Eratóstenes 110
C.1.4 - Aplicación de Medida de Cobertura de Puertas 111
C.1.5 - Aplicación de Medida de Cobertura de Patrones de Acción 112
C.1.6 - Aplicación de la Métrica de Composición de Eventos 113
C.2 - Ejemplo de Aplicación de Medidas de Cobertura: El Protocolo Abracadabra 118
C.2.1 - El Protocolo Abracadabra 118
C.2.2 - Interfaz de la Especificación 120
C.2.3 - Cálculo de Posibles Combinaciones 120

Bibliografía **124**

Curriculum **130**

Indice de Tablas

4.1 Función de Extracción de Nombres de Puertas de un Comportamiento . . 45

5.1 Identificación de Subexpresiones LOTOS por Numeración 60
5.2 Obtención de Ofertas Simples 61
5.3 Obtención de Ofertas en Comportamientos Compuestos 62

6.1 Asignación de veredictos 91

Indice de Figuras

2.1 Proceso de certificación de Conformidad 8
2.2 Descomposición Jerárquica de los Conjuntos de Pruebas 9
2.3 Métodos normalizados para ejecución de Pruebas 11
2.4 Fases del aplanado semántico de una especificación LOTOS 14

3.1 Los PCOs como interfaz entre Probador e IUT 23
3.2 PCO síncrono y asíncrono 23
3.3 PCO simétrico y asimétrico 24
3.4 PCO con Capacidad de Almacenamiento 24
3.5 PCO con Pérdidas 25
3.6 Taxonomía de algunos modelos de PCOs 25

5.1 Transformación de Árboles 67

6.1 Ingeniería de Protocolos de Comunicaciones 78
6.2 Escenario de Pruebas de Conformidad 79
6.3 Organización funcional del probador 84
6.4 Ejemplo de elaboración de casos de prueba 85
6.5 Prueba correcta completamente anotada 89
6.6 Relación entre la IUT, su modelo y la especificación de referencia 92

C.1 Árbol de Sincronización de la especificación Eratos, expandido hasta localización de PSCs repetidos 117
C.2 Árbol de Sincronización del protocolo Abracadabra, expandido hasta localización de PSCs repetidos 123

Capítulo 1

Introducción y Objetivos

1.1 - Introducción

Hoy en día, la interconectividad de sistemas está alcanzando cotas difícilmente imaginables hace pocos años. Cada vez más, la necesidad de comunicarse más rápido, más fiablemente, más cómodamente y con mayor volumen de datos acapara grandes inversiones en nuevas tecnologías y adaptación de infraestructuras que van quedando obsoletas. Por otro lado, el usuario final, más cerca del ciudadano normal, heterogeiniza las necesidades de comunicación. Esto deriva en que los fabricantes se ven obligados a diversificar la oferta en sus productos y servicios, así como a interconectarse con máquinas heterogéneas y diferentes, muchas veces fabricadas por otros.

Este último hecho hizo surgir los organismos de normalización, en un intento de asegurar una calidad e interconectividad al usuario final. Pero tampoco se podía olvidar la libertad de competencia del fabricante en cuanto a proporcionar una oferta variada y diversificada.

El surgimiento de normas ha supuesto un punto de inflexión en los métodos de trabajo y producción en los fabricantes de sistemas de comunicaciones. El fabricante pierde algo de creatividad en cuanto a las funcionalidades de sus productos a cambio de una mejora en la capacidades de interoperatibilidad de sus productos con otros. Para ello, es necesario que el fabricante siga la norma al pie de la letra.

En cierto modo, se pretenden conseguir dos objetivos fundamentales: que los productos sean compatibles para beneficio del usuario y que, a su vez, los fabricantes dispongan de cierta libertad para proporcionar servicios de valor añadido sobre lo que propone la norma.

Ya en 1970, en “un entorno experimental, académico y con patrocinio militar”, la experiencia ARPA [AR86] demostró la potencia de una normalización de interfaces. Los primeros pasos de normalización fueron iniciativas privadas de fabricantes, en concreto IBM con su SNA y DEC con DECNET. No eran normas fijas, sino que las arquitecturas e interfaces fueron evolucionando según los avances de la técnica.

En este periodo surgen normas y recomendaciones a través de CCITT (hoy ITU), que

ya no constituían iniciativas privadas sino que era un esfuerzo conjunto de los principales PTTs del mundo. Finalmente, ISO desarrolló su modelo arquitectónico de interconexión de sistemas abiertos [ISO84], en la que trata de especificar una arquitectura para interconexión de sistemas heterogéneos.

Actualmente existen muchas normas de protocolos y servicios que los fabricantes deben cumplir. Sin embargo, el significado de la palabra "cumplir" puede nos ser el mismo para todo el mundo. De ahí surge la necesidad de definir cuándo un producto se atiene a una norma. El usuario va a exigir una certificación u homologación por la capacidad que tenga para funcionar con lo que ya tiene, es decir, va a estar interesado, como vimos más arriba, en la *interoperabilidad*.

Hemos visto que es fundamental disponer de normas internacionalmente aceptadas de los protocolos. Se desea que estas especificaciones sean precisas y no contengan ambigüedades. Además, deben contemplar una serie de facilidades opcionales para respetar la libertad de los fabricantes, de forma que puedan ofertar productos diferentes de los de la competencia. *A posteriori* es necesario comprobar que el producto se atiene a una norma. Este proceso, denominado homologación o certificación, se realiza en base a una serie de pruebas denominadas de *conformidad* [Lin88].

El objetivo final es asegurar la interoperabilidad de productos diferentes que *conforman* con una norma. Sin embargo, las pruebas de conformidad no garantizan la interoperabilidad. Está demostrado [Dil91] que los productos homologados tienen una alta probabilidad de interoperar con otros, y que esta probabilidad es menor para productos no conformes a la misma norma[1].

1.2 - Pruebas de Conformidad

Probar es el proceso de ejercitar un sistema con la intención de encontrar errores, por medio de la experimentación del mismo. Esta experimentación se lleva a cabo en un entorno especial con el fin de simular el funcionamiento normal y excepcional del sistema. Aunque es evidente que para sistemas de complejidad realista no se puede ser exhaustivo en el proceso de pruebas, si es cierto que se incrementa la confianza en el buen funcionamiento del producto. Puesto que el proceso de pruebas sólo es capaz de mostrar la presencia de errores, pero nunca su ausencia, no es posible demostrar la corrección de una implementación.

Las Pruebas de Conformidad son una rama de la Ingeniería de Pruebas donde una implementación es probada respecto de su especificación. Específicamente, en ISO-9646 se define que el objetivo de las Pruebas de Conformidad es: "establecer cuando una implementación conforma a la especificación de la norma pertinente". En dicha norma se establece una estructura de los conjuntos de pruebas y unos métodos abstractos de pruebas, todo ello englobado en una metodología abstracta de pruebas.

[1] Evidentemente, ni siquiera se intenta que interoperen productos que no han sido fabricados para ello.

1.3 - Entorno de la Tesis

Esta Tesis esta enmarcada en el campo de la Ingeniería Software dedicado a asegurar la *Conformidad* de un producto respecto de una norma. En particular, restringiremos el campo a la Ingeniería de Protocolos.

Por otro lado, la creciente aplicación de Técnicas de Descripción Formal (FDTs) en la especificación de protocolos de comunicaciones hacen de estas Técnicas un campo abonado y muy atractivo para el desarrollo de metodologías y herramientas orientadas a garantizar la conformidad de un producto respecto de su norma.

En efecto, gracias a las FDTs obtenemos normas no ambiguas, concisas y precisas que eliminan problemas de incoherencia, imprecisión, etc. de las normas escritas en lenguaje natural, como inglés o español. Al igual que los matemáticos desarrollan sus lenguajes simbólicos para unificar la nomenclatura y la interpretación de sus fórmulas y teoremas [Fra87] se han desarrollado estos Lenguajes de Especificación y Descripción Formal que evitan dichos problemas.

Existen diversos lenguajes, como veremos en el siguiente capítulo, pero esta Tesis utilizará uno como herramienta y como punto de mira de sus objetivos; este lenguaje será LOTOS [ISO89b].

Por otro lado, y dado el creciente interés y volumen de negocio que está generando este campo, ISO ha normalizado un entorno específico y una metodología para el desarrollo y ejecución de pruebas de conformidad [ISO91a]. Estas Pruebas de Conformidad, como su nombre indica, son pruebas desarrolladas para ser pasadas por un sistema visto como caja negra orientadas a asegurar la Conformidad de dicho producto respecto de una norma. La idea básica es disponer para cada norma de una batería de pruebas completa y exhaustiva, disponibles públicamente para fabricantes y usuarios, de forma que cualquiera pudiera disponer de ellas. Evidentemente, habrán de existir unos Centro de Homologación y/o Certificación específicos, que aseguren a terceros la conformidad de un producto.

La Norma ISO-9646 será para esta Tesis punto de apoyo y referencia en todo su desarrollo.

Posteriormente, ISO lanzó un nuevo proyecto para formalizar los conceptos definidos en ISO-9646, norma que está en desarrollo y en la que participa activamente el autor de esta Tesis, dando como fruto parte del capítulo 3. Esta norma, denominada “Formal Methods in Conformance Testing”, está actualmente en la fase DIS[2] en ISO [N6294].

1.4 - Objetivos de la Tesis

Los objetivos de esta Tesis están orientados a mejorar el nivel de formalización y conceptualización de la ejecución de Pruebas de Conformidad. Tres son los principales campos de actuación:

[2]Draft International Standard

1. Formalización y Taxonomía del concepto de *Punto de Control y Observación*. En ISO-9646, se define, pero no se formaliza, este concepto de importancia capital en la ejecución de las Pruebas de Conformidad. Desafortunadamente, este es uno de los puntos más débiles de dicha norma, pues es un concepto ciertamente abierto y del cual dependen críticamente los resultados de la certificación y homologación de un producto.

2. Desarrollo de una medida de cobertura adecuada a la FDT LOTOS. Hoy en día no existe una definición de cobertura que capture la semántica de una especificación LOTOS que además sea capaz de tratar con comportamientos dinámicamente infinitos. En esta Tesis se ha desarrollado dicha medida, que por un lado proporciona un método eficaz de comparación de baterías de pruebas (cobertura *a priori*) y que por otro lado nos permite conocer el grado de ejercitamiento de la especificación a la hora de ejecutar pruebas sobre un producto (cobertura *a posteriori*).

3. Formalización de los elementos, arquitectura y conceptos participantes en la ejecución de pruebas sobre productos reales. Si bien ISO-9646 propone una metodología y entorno para desarrollo de pruebas, la fase de ejecución sigue siendo un trabajo claramente manual pero, sobre todo, específico de cada sistema de comunicaciones. En esta Tesis pretendemos formalizar la ejecución de las pruebas, proporcionando un marco teórico y conceptual para la ejecución de pruebas, entorno de ejecución e interpretación de resultados.

1.5 - Estructura de la Memoria

En el capítulo 2 se presenta un breve resumen del Estado del Arte actual referente a Pruebas de Conformidad, Desarrollo y Ejecución de Baterías de Pruebas. También se establecen las bases formales que serán utilizadas en el resto de la memoria.

En el capítulo 3 se presenta el trabajo relativo a la formalización de los PCOs. Se hará un estudio exhaustivo de las propiedades relevantes de los mismos, especificando su comportamiento en LOTOS. Además, bajo este prisma, se estudiarán los modelos de PCOs más interesantes en la práctica.

En el capítulo 4 se hace un estudio de las medidas de cobertura más clásicas como introducción y justificación de la medida propuesta. En este capítulo se desarrolla la *métrica de Composición de Eventos* y se definen las características exigibles a una prueba para poder asignarla una medida de cobertura. Asimismo, se introducen los conceptos de medida *a priori* como potencia de una batería de pruebas y *a posteriori* o medida de ejecución.

En el capítulo 5 se presenta una posible implementación para calcular la medida de Composición de Eventos.

En el capítulo 6 se estudia y define formalmente el proceso de ejecución de pruebas sobre un producto. Además, se presenta una arquitectura virtual que comprende los elementos necesarios para dicho proceso, sin implicar una implantación concreta. En este

capítulo se define lo que se entiende por una prueba *correcta*, es decir, qué propiedades ha de exigirse a una prueba orientada a asignar un veredicto según define ISO-9646.

En el capítulo 7, se presentan las conclusiones del presente trabajo, así como las líneas de investigación futura que pudieren surgir en el futuro.

En el apéndice A se presenta una lista de términos usados en la presente memoria, junto a una breve descripción de su significado.

En el apéndice B se especifican en LOTOS los tipos básicos que serán utilizados en la formalización de los PCOs.

En el apéndice C se desarrollan dos ejemplos prácticos de la aplicación de la medida de cobertura propuesta. El primero es la Criba de Eratóstenes para la determinación de los números primos. El objetivo principal es mostrar la potencia de la medida propuesta frente a otras aproximaciones más clásicas. El segundo es un ejemplo más realista, el protocolo de comunicaciones Abracadabra. Este ejemplo se ha introducido por dos razones: en primer lugar, usar un caso de complejidad real, cuyos resultados sean de aplicabilidad inmediata. En segundo lugar, un problema que suele invalidar modelos aparentemente solidos es la escalabilidad de la aplicación de las soluciones y metodologías propuestas. En efecto, es desesperanzador observar como soluciones aprobadas como válidas no son aplicables nada más que a problemas de juguete y los pequeños ejemplos usados en su demostración. Se quería demostrar que el prototipo de entorno generado en esta tesis sería capaz de manejar especificaciones de un tamaño normal.

Capítulo 2

Estado del Arte

2.1 - Introducción

En este capítulo, se presentan las bases sobre las que se fundamenta y justifica la presente Tesis. En particular, se describe la Metodología propuesta por OSI para Pruebas de Conformidad, haciendo hincapié en aquellos aspectos más relevantes en cuanto a la Ejecución de las Pruebas, puesto que es el aspecto que más interesa.

Por otro lado, se describen las FTDs en general, haciendo un breve resumen de ellas y extendiéndose profusamente en el lenguaje LOTOS. Esto es así porque es fundamental entender la semántica de dicho lenguaje y porque se presenta de forma natural la notación y elementos matemáticos que se usarán a lo largo de la memoria.

2.2 - Entorno y Metodología de Pruebas en OSI

Ya se ha visto (ver sección 1.2) cuál es el objetivo de las Pruebas de Conformidad. En esta sección se describe el tratamiento que se da en la ISO-9646. Esta Norma se divide 5 partes, cada una de ellas tratando de un aspecto del proceso de conformidad mediante pruebas:

1. La parte 1 [ISO91b] es una introducción donde se definen los conceptos más generales.

2. La parte 2 [ISO91c] describe el proceso de especificación de conjuntos de pruebas y su estructuración.

3. La parte 3 [ISO91d] define la notación TTCN para definición de conjuntos de pruebas.

4. La parte 4 [ISO91e] trata de la ejecución de pruebas sobre el producto y la interpretación de los resultados.

5. La parte 5 [ISO91f] describe los requisitos de los centros de pruebas y sus clientes durante el proceso de garantía de la conformidad.

El proceso de garantizar la conformidad de una implementación respecto de su especificación propuesto por ISO se resumen en la figura 2.1.

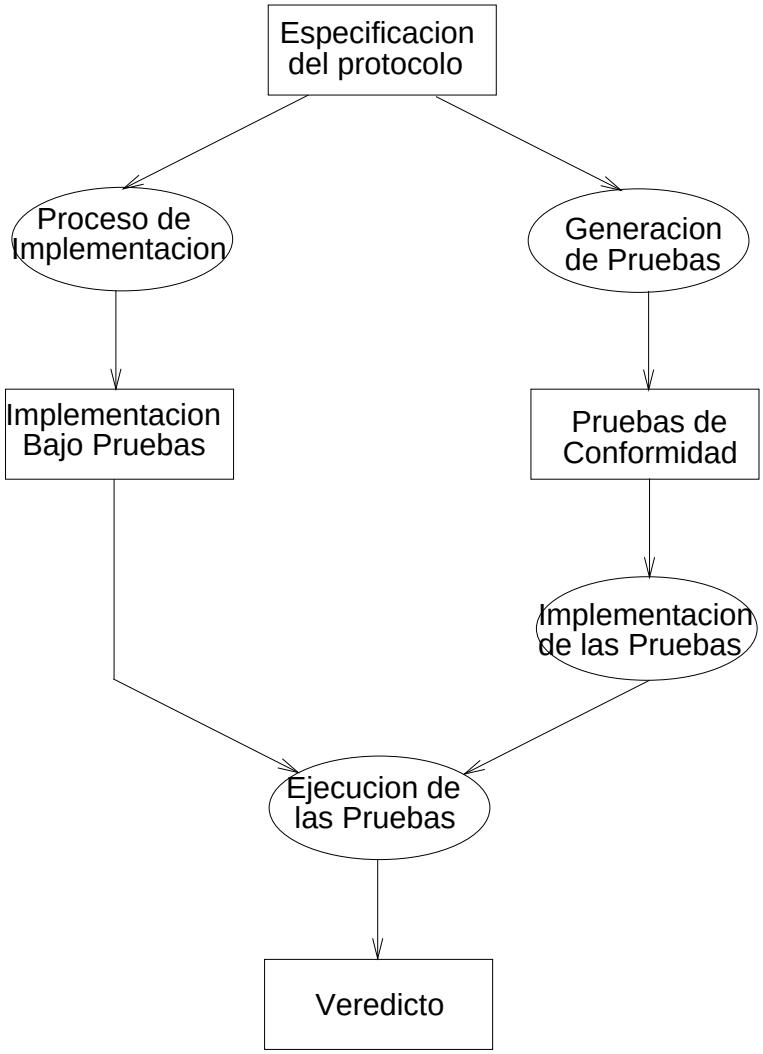


Figura 2.1: Proceso de certificación de Conformidad

La primera fase es generar una batería de pruebas (*Test Suite*) para un protocolo (OSI) en particular. Esta fase se la conoce como *Generación* o *Derivación de Pruebas*. Esta batería es abstracta en el sentido de que es completamente independiente de la implementación[1]. La segunda fase consiste en la realización de los medios de ejecución de pruebas. Esta fase se llama *Implementación de las Pruebas*. Los *Casos de Prueba Abstractos* son transformados en *Pruebas Ejecutables*, que serán particulares para un sistema de pruebas

[1]En principio, solo existirá un *Test Suite* por protocolo

en particular. La última fase es la *Ejecución de Pruebas*. Los *Casos de Prueba Implementados* son ejecutados sobre la IUT para la que fueron desarrollados. El resultado es un *veredicto* sobre la Conformidad de la IUT respecto de la Especificación del protocolo del cual se desarrollaron las pruebas. Este resultado es documentado en un *Informe de Pruebas de Conformidad del Protocolo*.

Estas Baterías de Pruebas se descomponen jerárquicamente en subconjuntos inferiores. La figura 2.2 muestra esquemáticamente esta descomposición.

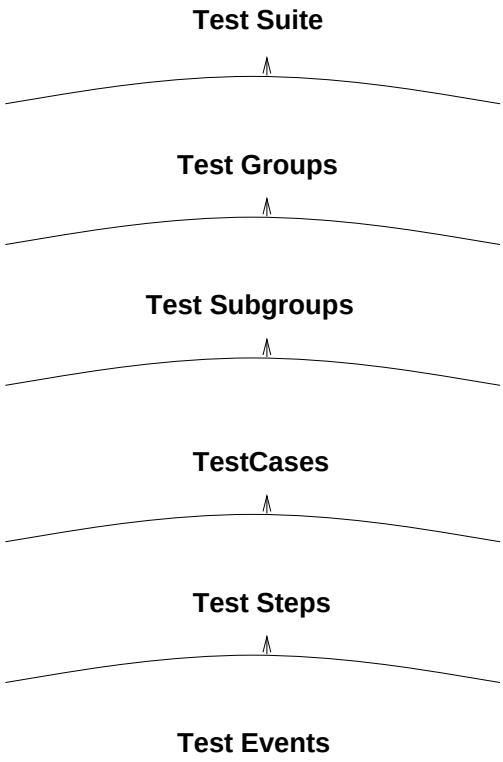


Figura 2.2: Descomposición Jerárquica de los Conjuntos de Pruebas

En esta estructuración, un elemento importante son los Casos de Prueba. Cada uno de ellos tendrá un objetivo claro y bien definido. Estos objetivos, llamados *Propósitos de Prueba*, son los que, en definitiva, nos determinan si la IUT conforma respecto de la norma.

Los Grupos de Prueba proporcionan un agrupamiento lógico de los Casos de Prueba, y se dividen en Subgrupos de Prueba. A su vez, los Casos de Prueba se componen de una secuencia de Pasos de Prueba, que cubrirán un propósito específico del Caso de Prueba. Además, es posible que un Caso de Prueba incluya otros pasos de prueba orientados a situar a la IUT en el estado requerido para comenzar el cuerpo de la prueba (*preámbulo*) y para situar a la IUT en un estado estable tras la ejecución de la prueba (*postámbulo*).

2.2.1 - Ejecución de las Pruebas de Conformidad

Las normas de OSI para protocolos de comunicaciones definen el comportamiento permitido de una entidad de protocolo en los puntos de acceso al mismo, en términos de las Unidades de Datos del Procotolo (PDUs) y de las Primitivas de Acceso al Servicio (ASPs). En el entorno de pruebas, estos puntos serán los Puntos de Acceso a la Implementación (IAPs).

En la ejecución de las pruebas sobre la IUT, dos aspectos se destacan por su influencia en los resultados a obtener y en la operatividad de la ejecución: el Método de Prueba y los Puntos de Control y Observación (PCOs).

Si la ejecución de las pruebas se realizará directamente sobre estos IAPs, entonces coincidirían plenamente con los PCOs. Este se considera el caso ideal[2] porque normalmente la Especificación del Protocolo hace referencia a los IAPs (nunca a los PCOs), por lo que las Pruebas derivadas para el sistema se referirá a los IAPs. Por otro lado, veremos que la no coincidencia entre PCOs e IAPs modificará el comportamiento de las interacciones, alejando la semántica de las sincronizaciones esperada por las pruebas de la que realmente nos presenta nuestro SUT.

El Método de Prueba define un modelo para la accesibilidad de la IUT en términos de PCOs y su lugar dentro del modelo de referencia OSI [ISO84]. Se distinguen varios aspectos que modifican y caracterizan los Métodos de Prueba [Tre92]:

- Existencia de PCOs: si IAP no son accesibles, no existirán PCOs para ellos.
- Si existen otros niveles entre los PCOs y los IAPs.
- El posicionamiento de los PCOs en el mismo sistema que el IUT.
- El funcionamiento interno del probador en terminos de la distribución de funcionalidad de pruebas sobre los Probadores Inferior y Superior, así como las reglas que definen su coordinación: los Procedimientos de Coordinación de Pruebas.

Modificando estos aspectos se obtienen diferentes Métodos de Prueba. Algunos han sido identificados y normalizados en ISO9646, para usarlos en los Entornos de Pruebas Abstractas. La característica común de estos métodos es que los IAPs inferiores de la IUT están siempre accesibles mediante el servicio de nivel inferior; los IAPs superiores pueden no ser accesibles para propósitos de pruebas.

Los Métodos normalizados son: Método Local (LS-method), Método Distribuido (DS-method), Método Coordinado (CS-method) y el Método Remoto (RS-method), que se presentan esquemáticamente en la figura 2.3.

[2]En la Norma 9646 se considera el Método de Referencia.

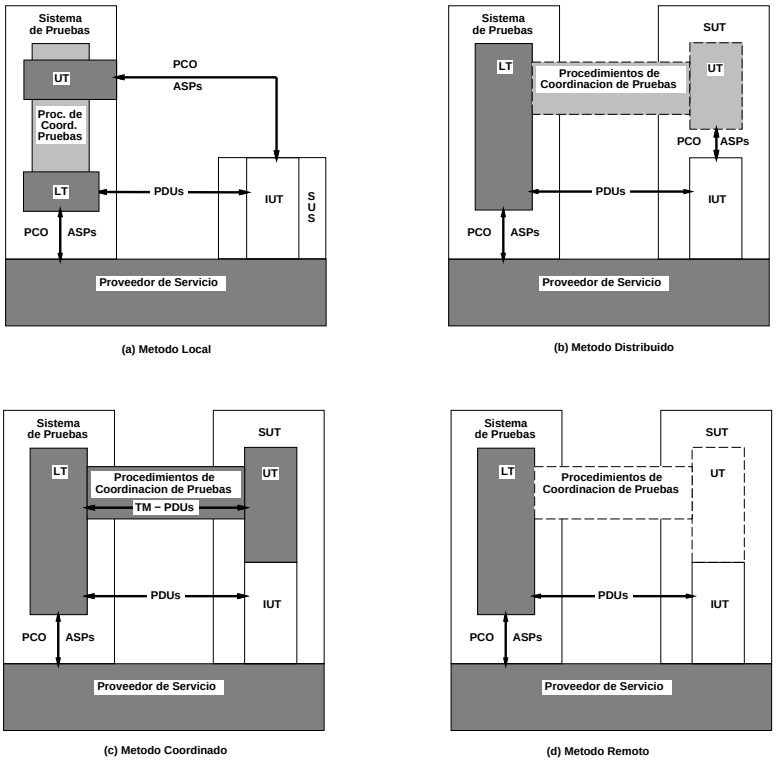


Figura 2.3: Métodos normalizados para ejecución de Pruebas

2.3 - Técnicas de Descripción Formal

La creciente complejidad de los protocolos y sistemas han provocado la necesidad de especificaciones bien definidas, completas, no ambiguas y precisas, de donde surgieron las Técnicas de Descripción Formal (FDTs). Mediante el uso de estas técnicas, el comportamiento de los protocolos es formalmente descrito con lenguajes de sintaxis y semántica formales, en vez de utilizar lenguaje natural como el Español o el Inglés. Esto evita los típicos problemas de malentendidos, diferentes interpretaciones del mismo texto, etc.

Las FDTs se basan en modelos matemáticos bien definidos, lo cual proporciona especificaciones precisas y no ambiguas, haciendo posible el análisis matemático del comportamiento de los protocolos. Actualmente, existen tres FDTs normalizadas, SDL [CCI88], estelle [ISO89a] y LOTOS [ISO89b]. Las dos primeras se basan en máquinas de estados finitos extendidos, la última se basa en teoría de álgebras de procesos.

Existen algunas otras técnicas formales en desarrollo, como el lenguaje Z [Spi89, NC88], de reciente desarrollo e incipientes aplicaciones. Debido a un fuerte formalismo matemático, se espera un fuerte impacto en su aplicación para validación y verificación, aunque todavía se encuentra en fase de estudio.

2.3.1 - El lenguaje SDL

SDL (Specification and Design Language) fu diseñado y normalizado por el CCITT para especificar y describir sistemas de telecomunicaciones [BHS91, BH89]. Aunque en sus comienzos fue utilizado para especificar programas de control, su uso se ha expandido en los últimos años, utilizándose para muchas aplicaciones. Es, con diferencia, el lenguaje más usado en la industria actualmente.

En SDL el comportamiento de un sistema se describe como el resultado de combinar los comportamientos de varios procesos. Esto procesos son máquinas de estados finitos extendidas, con modelo de comunicación mediante canales de señales. El proceso evoluciona a través de una serie de estados como consecuencia de la respuesta a las señales provenientes del exterior o como reacción a ciertas temporizaciones internas. Además, estos cambios de estado pueden acompañarse de unas señales de salidas, que comunicarán con otros procesos.

Se distingue entre descripciones (o definiciones) o instanciaciones. Una descripción caracteriza sintáctica y semánticamente una clase de objetos SDL (sistema, bloque, proceso, tipo de datos). Una instancia es una activación dinámica de una clase. Las especificaciones SDL no son "ejecutables" en una máquina real, sino que *tiempo de ejecución* se refiere a un intérprete abstracto con tiempo *simulado*. No obstante, existe herramientas que proporcionan modelos de ejecución más o menos particulares con características propias.

2.3.2 - El lenguaje ESTELLE

ESTELLE [BD87, LJ87] es una FDT basada en Modelos de Transición de Estados Extendidos, es decir, el modelo matemático subyacente es un autómata no determinístico extendido con tipos de datos tomados del lenguaje de programación Pascal [Wir71].

Más precisamente, ESTELLE puede ser visto como una extensión al Pascal de ISO [ISO83], que modela un sistema como una jerarquía estructurada de autómatas, los cuales pueden ejecutar en paralelo y comunicarse mediante el intercambio de mensajes o la compartición (restringida) de algunas variables.

Esta FDT permite separar la descripción de los interfaces de comunicación de los componentes del sistema de la descripción del comportamiento interno de cada componente.

Como en Pascal, los objetos manipulados están fuertemente tipados, permitiendo detectar estáticamente inconsistencias en la especificación.

2.3.3 - El lenguaje LOTOS

LOTOS (Language Of Temporal Ordering Specification - Lenguaje de Especificación de Ordenación Temporal) es una de las dos técnicas de descripción formal desarrollada por ISO para la especificación de sistemas distribuidos abiertos, aunque es aplicable a sistemas distribuidos y/o concurrentes en general. Fue desarrollado por el grupo ISO/TC97/SC21/WG11 entre los años 1981 y 1986. La idea básica a partir de la cual

surgió LOTOS es la descripción de un sistema mediante la definición de las relaciones temporales que constituyen el comportamiento observable de dicho sistema. La base matemática de LOTOS es el álgebra de procesos CCS (Calculus Communicating Systems) y elementos del CSP (Communication Sequential Processes) en cuanto al comportamiento. Este es independiente de la representación de tipos de datos escogida y, en particular, se eligió ACT ONE como la base para los Tipos de Datos Abstractos (ADT), cuyo fundamento matemático son las clases de equivalencias. No obstante, se hicieron ciertas modificaciones, tanto sintácticas como semánticas a dicho ADT [MH85][Mn88a, Mn88b]

Dado que esta tesis utiliza LOTOS en una doble vertiente de objetivo y vehículo de expresión, se va a profundizar más en su estudio.

Sistema de Transición Etiquetado

El modelo formal usado por LOTOS para describir un comportamiento es el de un sistema de transición etiquetado. Formalmente, un sistema de transición etiquetado o LTS (Labelled Transition System) es definido como una tupla $< S, L, T, s_0 >$, donde

- S en un conjunto (contable) no vacío de estados;
- L es un conjunto (contable) de acciones observables, también llamado conjunto de etiquetas o alfabeto de acciones;
- $T = \{\stackrel{a}{\rightarrow} \mid \stackrel{a}{\rightarrow} \subseteq S \times S \ y \ a \in A \cup \{\iota\}\}$ donde ι representa la acción interna; no siendo ésta una transición observable.
- $s_0 \in S$, considerado el estado inicial, es un elemento de S.

Consideraremos una transición como $s_i \stackrel{a}{\rightarrow} s_{i+1}$ con $s_i, s_{i+1} \in S$.

En esta memoria, se usará la siguiente notación:

L es el alfabeto de acciones observables; a, b,. . . , se usarán para sus elementos.

L^* es el conjunto de cadenas sobre L. σ', σ_1, σ_2,. . . serán sus elementos, y ε será la cadena vacía.

L' es $L \cup \{\iota\}$, es decir, el conjunto de las acciones observables extendido con la acción invisible (interna) ι .

Semántica de las Especificaciones LOTOS La interpretación formal (semántica) de una especificación LOTOS no se puede inferir directamente de su representación textual o sintáctica. La definición de esta interpretación formal se estructura en dos fases: la fase de la semántica estática y la fase de la semántica dinámica [ISO89b], según se representa en la figura 2.4.

El resultado será una especificación aplanada, donde todos los tipos de datos se hayan reunido en uno solo y todos los procesos se ven al mismo nivel de anidamiento. Además,

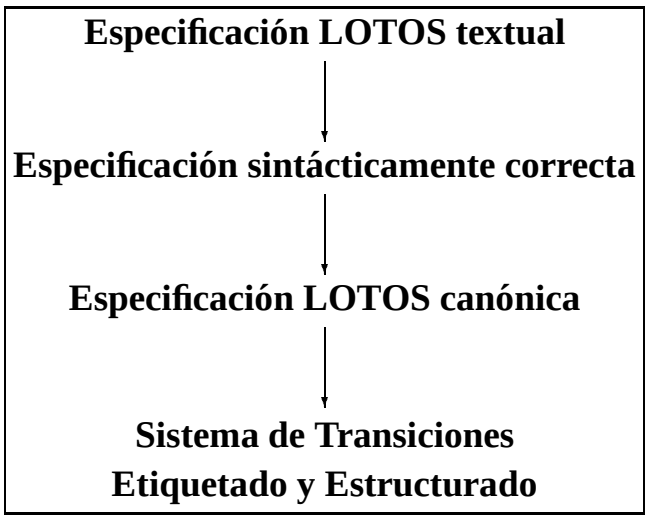


Figura 2.4: Fases del aplanado semántico de una especificación LOTOS

todos los identificadores serán sustituidos por un identificador único, por lo que queda resuelto el problema de la sobrecarga.

Formalmente, la especificación canónica se representará por una tupla:

$$CLS =< AS, BS >$$

donde:

- AS : Es una representación de la especificación algebraica de todos los tipos de datos presentes en la especificación.

  Esta contendrá tanto los nombres de los tipos de datos, como las operaciones definidas entre dichos tipos; además incorporará los axiomas que definen la semántica de las operaciones.

- BS: Será la especificación de comportamiento. Análogamente contendrá la representación de todas las definiciones de expresiones de comportamiento presentes en la especificación. Consistirá en un conjunto de pares formados por identificadores de procesos (p) y definiciones de procesos aplanados, mediante expresiones de comportamiento (B_p).

  $$BS = \{< p, B_p >\}$$

  Existirá una definición del proceso inicial p_0, correspondiente al comportamiento de la especificación que viene tras la palabra reservada **behaviour**. Este será el comportamiento por el que comenzará la determinación del comportamiento de la especificación.

De esta forma se habrá obtenido una representación clara y sin ambigüedades de todos los elementos del lenguaje. Disponiendo de una forma unívoca de referirnos a cada cosa, ya que todo poseerá un nombre único.

Interpretación de la especificación algebraica AS Aunque nuestro estudio se centra en la parte de comportamiento de LOTOS no obstante daremos un somero repaso a los conceptos fundamentales de la parte de datos. El uso de dicha especificación algebraica (AS) será necesaria para la correcta interpretación de la especificación del comportamiento (BS).

La parte correspondiente a los datos en la especificación canónica, tras el aplanado, se agrupará en una especificación algebraica AS. Esta representará un álgebra multitipada formada por un conjunto de tipos de datos (*sorts*) y un conjunto de operaciones asociadas a estos. Por construcción estos conjuntos serán finitos. Dichos tipos y operaciones son todos los que posteriormente podemos encontrarnos en la especificación de comportamiento BS.

En el proceso de aplanado se habrán realizado las operaciones de parametrización, actualización y renombramiento que puedan existir en la especificación. Asimismo las ecuaciones presentes en la especificación se habrán agrupado, de forma que habremos obtenido un conjunto de axiomas que nos definirán una relación de congruencia entre los distintos términos básicos que podamos formar con las operaciones definidas.

De esta forma diremos que dos términos básicos t_1, t_2 son congruentes si, mediante la utilización de sistema de derivación formado por los axiomas obtenidos de la especificación, nos es posible derivar su igualdad:

$$t_1 \equiv t_2 \Longleftrightarrow D_d \vdash t_1 = t_2$$

Definimos la clase de congruencia asociada a un término básico t como el conjunto de todos los términos básicos que son congruentes con él.

$$[t] = \{t' | t \equiv t'\}$$

Será este relación de congruencia la que nos permitirá el establecimiento de la igualdad de distintos términos básicos. Diremos que dos términos básicos son iguales si y solo si pertenecen a una misma clase de equivalencia.

$$t_1 = t_2 \Longleftrightarrow t_1 \in [t], t_2 \in [t]$$

La igualdad es absolutamente básica para la semántica de la especificación LOTOS, ya que muchas de las decisiones a tomar en la parte de comportamiento se basan en la comprobación de igualdades entre términos básicos. De hecho es la única relación que necesitaremos para nuestra semántica.

En la práctica lo que haremos será seleccionar un término básico como representante canónico de cada clase de congruencia. Determinando la igualdad de dos términos si podemos derivar de ellos, mediante el sistema D_d, un mismo representante canónico.

Para finalizar daremos una serie de definiciones de conjuntos que utilizaremos posteriormente en la descripción de la semántica dinámica de LOTOS. Definiremos el conjunto de términos DD como el conjunto de todos los términos básicos que existirán en nuestra álgebra. Definiremos también el conjunto DD^* como el cierre de Kleene del conjunto DD, esto es el conjunto de todas las tuplas, de cualquier orden, que nos es posible formar con los elementos de DD. Denotaremos como $Q(s)$ al conjunto cociente de un tipo de datos s.

Sistema de transición etiquetado asociado a la especificación canónica Pasaremos a establecer la relación existente entre el sistema de transición etiquetado y la especificación canónica. Veremos cómo se obtienen los distintos componentes que forman un sistema de transición etiquetado, y cómo los relacionamos entre ellos. Para ello pasaremos a detallar las relaciones que existen entre los diferentes elementos, realizando una identificación entre miembros de ambos conjuntos.

El conjunto de acciones Una transición realiza la ocurrencia de una acción $a \in A$. En nuestro caso dicha acción se identificará bien con la acción interna, o bien con una de las puertas de la especificación seguida de una lista de términos básicos perteneciente al álgebra de datos AS, a la que nos referiremos como lista de experimentos.

Por tanto definiremos el nuevo conjunto de acciones como el conjunto A' :

$$A' = \{\iota\} \cup \{gv | g \in G \cup \{\delta\}, v \in DD^*\}$$

Donde G es el conjunto de puertas que aparecen en la especificación, δ la puerta asociada con la acción terminación (**exit**) y ι la puerta asociada a la acción interna.

El conjunto de estados y sus relaciones de transición Dentro del estándar ISO se establecen una serie de axiomas y reglas de derivación que establece un sistema de derivación (D_b) sobre expresiones LOTOS canónicas. De esta manera podemos decir que una expresión de comportamiento B, perteneciente al conjunto de todas las expresiones de comportamiento BE, es capaz de realizar la transición a si es posible, mediante el sistema de derivación D_b, derivar que :

$$D_b \vdash B_1 \xrightarrow{a} B_2$$

Este mecanismo nos define, de forma implícita, una aplicación entre el conjunto de las expresiones de comportamiento (BE) y el conjunto de todos los estados existentes en el sistema de transición asociado (S):

$$\mathcal{S} : BE \rightarrow S$$

Realmente no asociaremos un estado con cualquier expresión válida de comportamiento que podamos formar, sino que únicamente realizaremos dicha asociación con las

expresiones de comportamiento obtenibles mediante nuestro sistema de derivación D_b a partir de nuestra especificación (B_{spec}).

Dicho conjunto será el de todas la expresiones derivables de nuestra especificación ($DER(B_{spec})$). Formalmente este será el conjunto mínimo que verifica:

- $B \in DER(B)$.
- Si $B_1 \in DER(B)$ y $D_b \vdash B_1 \xrightarrow{a} B_2$ para algún a, entonces $B_2 \in DER(B)$.

A cada elemento de este conjunto podremos asociarle un estado de nuestro sistema de transición.

Esto quiere decir que el estado representado por la expresión de comportamiento B_1 y el estado representado por B_2 estarán dentro de la relación de transición a ($B_1 \xrightarrow{a} B_2$) si podemos derivar dicha transición entre ambos estados:

$$T(D_b) = \{\xrightarrow{a} \subseteq S \times S | \mathcal{S}(B_1) \xrightarrow{a} \mathcal{S}(B_2), D_b \vdash B_1 \xrightarrow{a} B_2\}$$

De esta forma el sistema de derivación definido en el estándar ISO tiende el puente que nos permite relacionar el mundo de las álgebras asociadas a las especificaciones de datos y comportamiento (AS, BS) con el de los sistema de transición etiquetados.

El estado inicial del sistema El estado inicial (s_0) se definirá por el comportamiento asociado al proceso inicial (p_0), en el cual se habrán ejemplarizado las variables formales por términos básicos, así como sustituido las puertas formales por las reales.

$$s_0 = \mathcal{S}([t_1/x_1, \ldots, t_n/xs_n]B_{p0}[h_1/g_1, \ldots, h_m/g_m]), < p_0, B_{p0} > \in BS$$

Donde $t_1, \ldots, t_n$ son los términos básicos; $x_1, \ldots, x_n$ son las variables correspondientes a los parámetros formales del proceso; $g_1, \ldots, g_m$ las puertas formales del proceso y $h_1, \ldots, h_m$ las puertas reales del proceso.

Por tanto el sistema de transición etiquetado asociado a una especificación LOTOS canónica será :

$$< DER(B), A', T(D_b), s_0 >$$

Capítulo 3

Modelado y Caracterización de los Puntos de Control y Observación

Este capítulo trata de establecer una caracterización formal de los Puntos de Control y Observación (PCOs), de forma que establezcamos una base para su estudio y comparación.

Posteriormente se identificarán unos modelos reales de sincronización que nos permita establecer una taxonomía de los PCOs que consideramos más relevantes.

Con este trabajo pretendemos proporcionar un nuevo enfoque en el estudio del grado de capacidad de prueba de una IUT respecto de su especificación de referencia.

3.1 - Introducción

Como ya se ha visto en el capítulo 2, según la norma ISO-9646, un PCO es "Un punto dentro del entorno de pruebas donde la ocurrencia de eventos de prueba será controlada y observada, según se define en un método abstracto de prueba". En la misma norma, se dice que un PCO es caracterizado por el conjunto de Primitivas de Servicio Abstractas (ASPs) y/o las Unidades de Datos del Protocolo (PDUs) que pueden intercambiarse en dicho PCO.

De la primera definición es claro que un PCO modela conceptualmente los límites entre un sistema y su entorno. De la segunda, se deduce que un PCO es un punto de interacción externo del sistema [Rei93]. También puede ser entendido como los mecanismos o medios por los cuales el entorno afecta a un sistema y viceversa. En definitiva, un PCO es un concepto arquitectural en el campo de los Sistemas Abiertos.

El enfoque que se adoptará para formalizar los PCOs es estudiar todas las propiedades relevantes que modifican el comportamiento del PCO. Esta propiedades dependerán tanto de la Arquitectura de Pruebas escogida para la Ejecución de las mismas como de la naturaleza del IAP, como ya se vio en la sección 2.2.

Con la identificación y definición de estas propiedades que modelan un PCO podremos establecer una clasificación de los mismos. Por lo tanto, su comparación se podrá establecer si podemos formalizar y expresar en el mismo formalismo dichas propiedades.

3.2 - Características de un PCO

La caracterización de los PCOs se hará en base a una serie de propiedades que denominaremos de Alto Nivel que los puntos de interacción pueden mostrar o no. Estas propiedades están relacionadas con el comportamiento de los puntos de interacción. Desde este punto de vista, hablaremos de Fiabilidad, Modo de Interacción, etc. Llamaremos a estas propiedades Características de Alto Nivel.

Por otro lado, la caracterización puede basarse en la naturaleza más física de los PCOs, en cuyo caso las propiedades relevantes serán Capacidad de Encolamiento, Simetría, Rechazo, etc. Llamaremos a estas propiedades Características Elementales.

Ambos enfoques son interesantes, por lo que estableceremos una función de relación entre ellos.

3.2.1 - Propiedades de Alto Nivel

En esta sección enumeraremos y definiremos aquellas propiedades consideradas relevantes para la formalización de los PCOs, en base a las características y naturaleza de las interacciones que tiene lugar en ellos.

Ya en [Rei93] se ha hecho un amplio estudio de la naturaleza de las interacciones desde el punto de vista de comunicaciones entre entidades participantes. En nuestro trabajo, haremos hincapié en cómo la naturaleza de los PCOs influyen en dichas interacciones, centrando nuestra atención en el concepto arquitectural de Punto de Interacción.

- **Fiabilidad**: Se dice que un PCO es Fiable si no crea, duplica, corrompe o pierde interacciones.

  Aunque parece que este concepto es básico desde el punto de vista de ejecución de pruebas, veremos que existen Implementaciones cuyos IAPs no serán fiables. Esto conducirá a que, independientemente del Método de Prueba, los PCOs resultantes no serán fiables, con el consiguiente efecto negativo sobre los veredictos resultantes. En el caso peor, puede ocurrir que la IUT no sea susceptible de ser conformada respecto de la Norma.

- **Modo de Interacción**:

  Dependiendo de la temporización en que el Probador y el SUT atienden a las interacciones que se intercambian, distinguiremos entre:

  - **Síncrono**: la oferta y aceptación de la interacción son simultáneas, es decir, no transcurre el tiempo entre ellas.
  - **Asíncrono**: la oferta y aceptación no son simultáneas, es decir, tendrán lugar en momentos diferentes y, como es lógico, la aceptación será posterior a la oferta o entrega.

  - **Mixto**: algunas interacciones son síncronas y otras son asíncronas. Se incluye este modo más por completitud en la definición que por encontrarlo en modelos reales de PCOs.

- **Estrategia**: cuando existe una serie de interacciones listas para ser entregadas, existen diferentes estrategias de entrega:

  - **Exclusividad**: cuando una interacción ocurre, las demás son inhibidas, en el sentido de que ya nunca ocurrirán (a menos, claro está, que vuelvan a ser ofertadas).
  - **Equidad**: todas las interacciones tendrán lugar alguna vez.
  - **Temporización**: las interacciones se acompañan de unos plazos de entrega o ejecución, transcurrido el cual, desaparecerán.
  - **Prioridad**: las interacciones pueden tener diferente prioridad de ejecución.

- **Cardinalidad**: dependiendo del número de participantes en una interacción[1], distinguiremos entre:

  - **Binaria**: 2 participantes.
  - **n-aria**: más de dos participantes.

- **Intercambio de Valores**: los valores intercambiados en la interacción pueden ser impuestos por un participante o negociados entre todos. Por lo tanto, diferenciaremos entre:

  - **Paso de Valores**: un participante impone un(os) valor(es) al resto.
  - **Negociación de Valores**: los valores son decididos entre todos los participantes.

- **Tipado**: el conjunto de posibles tipos de interacción puede estar restringido.

- **Rango de valores**: los datos intercambiados en una interacción pueden tener restringido su valor en un cierto rango.

- **Selección**: la elección de la siguiente interacción puede depender de restricciones a su tipo o a sus parámetros.

Existen una serie de relaciones entre estas propiedades, de forma que algunas imponen una modificación o incluso una restricción total a otras. Por ejemplo, es claro que una interacción asíncrona no puede negociar valores. Asimismo, la negociación de valores impone una sincronía en la interacción.

[1] En Pruebas de Conformidad, solamente participarán dos entidades, la IUT y el Probador

Como vemos, estas propiedades no hacen referencia directa a la naturaleza física del PCO. Sin embargo, existe una estrecha relación, puesto que va a depender de las capacidades elementales del PCO el que estas propiedades existan o no. Veamos primero cuáles son las características elementales del PCO para así establecer la relación entre unas y otras.

3.2.2 - Propiedades Elementales: Definición y Formalización

Estas propiedades hacen referencia a la naturaleza física del PCO. Enumeraremos y definiremos formalmente las propiedades elementales como base para expresar las Propiedades Abstractas. Primero, veremos una definición intuitiva de lo que entendemos de ellas, y posteriormente daremos una expresión formal. Y, puesto que gran parte de la contribución de esta tesis esta relacionada con LOTOS, usaremos ésta Técnica de Descripción Formal para nuestro propósito.

Se utilizarán las siguientes definiciones en LOTOS de tipos básicos, que se encuentran en el apéndice B:

- *Booleanos*. Con la semántica del Álgebra de Boole.
- *Natural*. Con la semántica de los números enteros y positivos, incluyendo el 0 (conjunto N).
- *Interacción*. Son las interacciones que se pueden intercambiar en los PCOs. En principio, la semántica dependerá de la aplicación, por lo que sólo dispondremos de la signatura.
- *Set_of_Inter*. Conjunto de interacciones. Si existen relaciones de orden dentro del conjunto estaremos hablando de Colas FIFO, LIFO, y otras estrategias de entrada y salida de elementos. Típicamente, la estrategia usada en el almacenamiento de interacciones es de Cola FIFO, aunque pueden existir otras.

Además de estos tipos básicos, generalizaremos el uso de un **sort** especial **any**, que en LOTOS sólo aparece en la construcción **exit**. Sin embargo, aquí también aparecerá en las acciones como aceptación de valores sin especificación del **sort**.

La especificación LOTOS completa de estos tipos se encuentra en el apéndice B.

El entorno en el que nos movemos en la formalización se representa en la figura 3.1:

- **Sincronicidad**: se dice que un PCO es Síncrono si todos los participantes se ven envueltos en el mismo momento en las interacciones que tienen lugar en él, entendiendo que ninguno podrá llevar a cabo ninguna actividad hasta que la interacción presente este terminada.

  El comportamiento formal de un PCO síncrono y asíncrono se presenta en la figura 3.2.

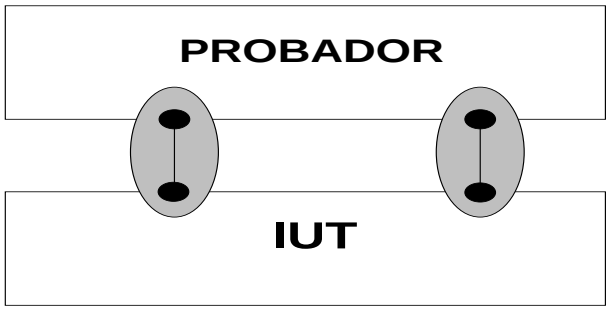


Figura 3.1: Los PCOs como interfaz entre Probador e IUT

process PCOsynch [interchange]:**noexit** := interchange; PCOsynch [interchange] **endproc**	**process** PCOasynch [send, recv]:**noexit** := send; recv; PCOasynch [send, recv] **endproc**

Figura 3.2: PCO síncrono y asíncrono

- **Simetría**: se dice que un PCO es Simétrico si todos los participantes conectados a él lo ven igual, es decir, no existe la posibilidad de hacer una distinción entre ellos. Entre otras implicaciones, esto significa que todos los participantes pueden, en principio, comenzar o aceptar una interacción.

  El concepto de simetría se expresa en diversas facetas de un comportamiento LOTOS. Por un lado, existe cierta asimetría cuando uno de los participantes ofrece un valor y otro lo acepta. Por otro lado, también existe asimetría cuando los participantes ven diferente al PCO. Por lo tanto, tendremos en la figura 3.3 dos formas de ver la simetría (diferentes aspectos):

- **Capacidad de Almacenamiento**: se dice que un PCO tiene Capacidad de Almacenamiento si puede mantener varias interacciones sin perderlas. De esta forma, en general un participante puede continuar con otras acciones. No hay espera ni bloque de los participantes atendiendo a una interacción. En la figura 3.4 se especifica este comportamiento.

- **Rechazo**: se dice que un PCO tiene Capacidad de Rechazo si puede impedir a un participante ofertar una interacción.

- **Pérdidas**: Se dice que un PCO puede causar la Pérdida de una interacción si ésta es aceptada en su oferta pero no hay reacción subsiguiente, es decir, no se entrega a ninguno de los destinatarios. En general, se entiende que el ofertante no recibe información referente a esta pérdida.

  El comportamiento de este PCO se especifica en la figura 3.5.

La principal diferencia entre Ignorar y Perder una interacción es que la primera está

```
process OneWay [recv. send] : noexit :=
    recv ? Inter : ANY; send ! Inter; OneWay [recv, send]
endproc

process PcoAsimetrico [UpperIn, LowerOut] : noexit :=
    OneWay [UpperIn, LowerOut]
endproc

process PcoSimetrico [UpperIn, LowerOut, Lowerin, UpperOut] : noexit :=
        OneWay [UpperIn, LowerOut]
    |||
        OneWay [LowerIn, UpperOut]
endproc
```

Figura 3.3: PCO simétrico y asimétrico

```
process PCO [recv, send] (InterPendientes : SET_of_Inter) : noexit :=
    recv ? Inter: ANY;
        PCO [recv, send] (Add (Inter, InterPendientes))
        []
        [IsNotEmpty (InterPendientes)] ->
            send ! Saca (InterPendientes);
            PCO [send, recv] (Saca (InterPendientes))
endproc
```

Figura 3.4: PCO con Capacidad de Almacenamiento

relacionada con los participantes que reciben la interacción, mientras que la segunda es responsabilidad del PCO.

3.3 - Clasificación de Modelos de PCOs Reales

En esta sección vamos a estudiar, mediante la formalización propuesta, diferentes modelos de PCOs que se dan en la realidad. La idea básica es poder comparar dichos modelos y establecer cuáles son las Propiedades Elementales de los mismos, de forma que se puedan comparar y por lo tanto, prever cuáles serán los principales puntos que hagan disminuir la capacidad del SUT de ser probado.

A la hora de escoger los PCOs a estudiar, nos centraremos en aquellos que entendemos de más uso en el campo de la Ingeniería Software, específicamente en Pruebas de Conformidad y Validación de Protocolos, puesto que este es el objetivo principal de la

```
process PCO [recv, send] : noexit :=
        recv ? Inter: ANY;
        send ! Inter;
        PCO [recv, send]
    []
        recv ? Inter: ANY;
        PCO [recv, send]
endproc
```

Figura 3.5: PCO con Pérdidas

Norma ISO9646.

Estos modelos serán los representados por las sincronizaciones en lenguajes de Especificación y lenguajes de programación, ambos usados para describir y realizar Protocolos. Dentro de los lenguajes de Especificación tendremos las FDTs LOTOS, Estelle y SDL. En los lenguajes de programación hemos elegido el Ada como más representativo, de amplio uso en Ingeniería Software Aeroespacial y en Ingeniería de Protocolos y porque incluye la comunicación entre tareas como elemento propio del lenguaje.

Como punto de referencia, también se han incluido dos modelos de componentes físicos que nos pueden aparecer en Sistemas Bajo Pruebas: una linea de comunicaciones con Protocol Ethernet y una linea de conexión Hardware entre circuitos integrados.

La información se ha organizado en forma de tabla (figura 3.6 donde se explicitan las Propiedades Elementales que nos presentan estos PCOs[2].

	Sincronía	Simetría	Cola	Rechazo	Ignora	Pérdidas	Temporización	Prioridad
LOTOS	Si	Si	No	Si	No	No	No	No
SDL	No	No	Si	No	Si	No	Si	Si
Estelle	No	No	Si	No	Si	No	Si	No
TTCN	No	No	Si	No	No	No	Si	No
Hardware	Si	No	No	No	Si	No	Si	Si
Ada	Si	No	Si	No	No	No	Si	Si
Ethernet	No	No	Si	No	Si	Si	No	No
RPC UNIX	Si	No	x	Si	No	Si	Si	No

x: puede implementarse con o sin cola

Figura 3.6: Taxonomía de algunos modelos de PCOs

[2]Las RPCs corresponden al modelo UNIX, ver [Blo92]

3.4 - Especificación de PCOs en LOTOS

3.4.1 - Anotaciones

Hemos visto que LOTOS puede ser usado para caracterizar formalmente las distintas propiedades que presentan los PCOs. Sin embargo, es una aproximación nada práctica el usar dichas especificaciones en sistemas que requieran PCOs con memoria, por ejemplo. En definitiva, la semántica de LOTOS impone un modelo de sincronización que no es, con mucho, el más general presentado por los sistemas de comunicaciones reales. En este sentido es deseable enriquecer el lenguaje con construcciones que determinen el tipo de sincronización adecuada a nuestro sistema.

En definitiva se busca un lenguaje que pueda modelar cualquier tipo de sincronización de forma directa y sin construcciones indirectas. No obstante, no se busca una redefinición del lenguaje, por varios motivos: Primero, la definición del lenguaje debe ser normalizada, sin derivar dialectos distintos. Segundo, el lenguaje se está redefiniendo actualmente por ISO. Tercero, existe ya un mecanismo utilizado para ampliaciones no normalizadas del lenguaje llamado anotaciones [MdMSA93]. Este mecanismo introduce primitivas en el lenguaje en forma de comentarios especiales, de forma que son transparentes a las herramientas LOTOS, excepto aquellas que se ven afectadas por la modificación introducida (usualmente, generadores de código).

Actualmente las anotaciones se están utilizando para:

- entrada/salida de datos del exterior, introducción de temporización y prioridades [Sal93].
- modificación y adaptación del código generado para los tipos de datos cuando la eficiencia (tiempo y/o memoria) de los compiladores no es adecuada [Vei93].
- introducción de cálculos estadísticos y tratamiento probabilístico [MN91].

La sintaxis de las anotaciones es muy sencilla: se introducen como comentarios a la especificación, de forma que las herramientas normalmente no las consideran. Sólo aquellas que se ven afectadas las reconocen y las interpretan. Para diferenciarlas de los comentarios normales, se añade el carácter "|" al empezar y acabar el comentario. Además, se adjunta una palabra clave que las identifica, mas el texto necesario si la anotación lo requiere. Como ejemplos: `(*| C printf ("%s\n", kd_draw ($1) |*)`, `(*| delay 10 |*)`, ...

3.4.2 - Anotaciones para Modelización de PCOs

Para adecuar la semántica de las sincronizaciones de LOTOS a los distintos modelos de PCOs se proponen las anotaciones descritas más abajo, teniendo siempre como modelo por defecto la sincronización LOTOS. Por lo tanto, las anotaciones están orientadas a aquellas características que el lenguaje LOTOS no contempla. Estas anotaciones solo

deberán afectar a las puertas externas de la especificación, puesto que son éstas los puntos de sincronización con el entorno.

Se proponen las siguientes anotaciones:

- `priority`

  En LOTOS, cuando se ofrecen varias ofertas, se elige una de forma equitativa (es decir, no hay distinción por prioridades). Esta anotación caracteriza una puerta externa con un valor entero de prioridad. Las acciones que no lleven esta anotación se les supone una prioridad 0. A la hora de elegir que acción se ofertará, se elige aquella de mayor prioridad. Entre ofertas de la misma prioridad, se elige con la semántica habitual de LOTOS.

- `asynchronous`

  Este anotación indica que la oferta es asíncrona, es decir, no se espera que el entorno reaccione, por lo que el sistema puede seguir evolucionando. Otra implicación es que no puede existir negociación de valores.

- `in, out`

  Esta anotación afecta al modo en que se tratan los valores ofertados en una oferta. Así, `in` indica que la interacción es de entrada y `out` de salida. LOTOS, por defecto, no impone ninguna restricción al sentido de los datos, tomándose entonces como de entrada o salida indistintamente. Esta anotación es obligatoria cuando la oferta en que aparecen se ve afectada por una anotación `asyn\-ch\-ro\-nous`.

- `timeout`

  Siendo value un valor de tipo real, que dependerá de la plataforma de ejecución. Esta anotación indica que pasados value unidades de tiempo, las acciones sobre la puerta dejarán de ser ofertadas.

- `queue`

  Esta anotación indica que las sincronizaciones sobre la puerta afectada serán encoladas y quedarán a la espera de ser consumidas. Esta anotación podrá estar acompañada de una serie de atributos que caracterizarán el tratamiento ante casos de colas llenas o vacías, según la siguiente lista:

  - `queue`: cola ilimitada en tamaño.
  - `queue` #: donde # es una valor entero. En este caso la cola tiene una capacidad máxima de # posiciones.
  - `queue full => lost`: Este atributo indica que si la cola está llena, las siguientes interacciones se perderán.
  - `queue full => wait`: Este atributo indica que si la cola está llena, el enviante queda a la espera, sin posibilidad de evolucionar.

- `empty => default value`: Si la cola está vacía, la interacción por defecto es `value`.
- `empty => wait`: Si la cola está vacía, el receptor queda en espera de la interacción.
- `empty => nosynch`: Si la cola está vacía, la interacción no tiene lugar.

Los atributos caracterizados como `full` aplican sobre interacciones de salida (`out`), mientras que los caracterizados como `empty` aplican a interacciones de entrada (`in`).

3.4.3 - Lenguaje de Anotaciones para Modelización de PCOs

En esta sección se propone un lenguaje para anotar una especificación. De esta forma, no es necesario modificar el propio texto LOTOS, sino que las herramientas relacionadas con el interfaz de la especificación (típicamente, generadores de código, generadores de pruebas, etc) pueden utilizar esta información sin modificaciones visibles para el resto de herramientas.

Sintaxis Las sintaxis se describe mediante las reglas de producción siguientes:

```
rule
  module ::= _annotated_act_denot_list

rule
  _annotated_act_denot_list ::= [ _annotated_act_denot + ]
```

Cada anotación corresponde a una de las explicadas en la sección anterior. Se asocian a denotaciones de acción (action-denotation), tomados de la definición del lenguaje LOTOS:

```
rule
  _annotated_act_denot ::= _action_denotation
                           _annotation_list

rule
  _action_denotation ::= _gate_identifier
                         [ _experiment_offer_list ]
                         [ _selection_predicate ]

rule
  _experiment_offer_list ::= [ _experiment_offer + ]

rule
```

```
  _experiment_offer ::= ?‘’ _identifier_declaration_list
                        ’:’ _sort_identifier

rule
  _experiment_offer ::= !‘’ _value_expression

rule
  _selection_predicate ::= ’[’ _value_expression ’=’
                               _value_expression ’]’

rule
  _selection_predicate ::= ’[’ _value_expression ’]’

rule
  _value_expression ::= [ _value_expression * ]

rule
  _value_expression_list ::= [ _value_expression + ’,’ ]

rule
  _identifier_declaration_list ::= [ _identifier + ’,’ ]

rule
  _sort_identifier ::= IDENTIFIER

rule
  _gate_identifier ::= IDENTIFIER

rule
  _annotation_list ::= [ annotation + ]
```

Las anotaciones propiamente dichas se relacionan a continuación:

```
rule
  annotation ::= _priority_annotation

rule
  _priority_annotation ::= "PRIORITY" _integer_value

rule
  annotation ::= _asynchronous_annotation

rule
  _asynchronous_annotation ::= "ASYNCHRONOUS"
```

```
rule
  annotation ::= _inout_annotation

rule
  _inout_annotation ::= "IN"

rule
  _inout_annotation ::= "OUT"

rule
  annotation ::= _timeout_annotation

rule
  _timeout_annotation ::= "TIMEOUT" _integer_value

rule
  annotation ::= _queue_annotation

rule
  _queue_annotation ::= "QUEUE" [ _attribution_list ]

rule
  _attribution_list ::= [ _attribution + ]

rule
  _attribution ::= _integer_value

rule
  _attribution ::= "FULL => lost"

rule
  _attribution ::= "FULL => wait"

rule
  _attribution ::= "EMPTY => default" _value_expression

rule
  _attribution ::= "EMPTY => wait"

rule
  _attribution ::= "EMPTY => nosynch"
```

Semántica Se describe a continuación la semántica aplicable a las reglas de producción:

- Para cada `action_denotation`, cada anotación puede aparecer una sola vez.
- `in, out`. Estas dos anotaciones son claramente excluyentes. Una y solo una de ellas debe aparecer.
- La anotación `queue` puede ir acompañada de un entero que indica su capacidad máxima. En ese caso, el atributo `FULL` debe ir acompañando a `queue`, indicando la política de tratamiento de colas llenas. Evidentemente, esto sólo aplica a puertas caracterizadas como `out`.
- Siempre se debe indicar la política de tratamiento de colas vacias. Por lo tanto, si existe `queue` debe ir acompañada del atributo `empty`. Esto solo aplica a puertas que, además de cola, estén caracterizadas como `in`.

Capítulo 4

Análisis de Cobertura

4.1 - Introducción

LOTOS es un lenguaje concebido para describir el comportamiento de sistemas con un alto nivel de abstracción, bastante lejano de detalles de realización. Por esta razón, una especificación LOTOS de un sistema será usado como referencia por implementadores, centros de homologación, etc. Es fundamental que dicha especificación no contenga errores y capture todos los requisitos de conformidad del sistema; en otras palabras, la especificación deberá ser sistemáticamente validada para eliminar cualquier defecto: fallos en el comportamiento, sobreespecificación o infraespecificación del sistema, etc.

La validación es un proceso que se suele llevar a cabo mediante la ejecución de baterías de pruebas, derivadas de los requisitos del sistema y de la especificación informal.

Cuando se desarrolla una batería de pruebas para un sistema específico, se necesita evaluar que partes de la especificación son ejercitadas o probadas. Estas medidas son, en Ingeniería Software convencional, medidas de cobertura. Por otro lado, la cobertura medirá la potencia de una batería de pruebas. Esta potencia tiene una doble vertiente. Por un lado, como "capacidad" de discernir diferentes comportamiento dentro de una especificación, consideraremos un valor de medida *a priori*. Por otro lado, la especificación formal será usada posteriormente para servir de referencia en la ejecución de pruebas sobre productos reales. Tendremos una asignación de cobertura a la ejecución de cada prueba o cobertura *a posteriori*.

En un mundo ideal, todos y cada uno de los posibles comportamientos y cada uno de los posibles valores de los datos podrían ser probados. Sin embargo, tanto uno como otro suelen ser infinitos, haciendo impracticable el examen de todos los casos posibles. No obstante, se debe fijar un límite deseable y alcanzable que nos determine una forma de medir el grado de confianza otorgado a un conjunto de pruebas. Arbitrariamente, se define dicho límite como el 100% de cobertura sobre la especificación. De esta forma, sin haber pasado ninguna prueba, diremos que tenemos una cobertura del 0%. A medida que se ejecuten pruebas sobre el sistema, la cobertura irá aumentando de forma monótona no decreciente, hasta alcanzar dicho 100%. En todo caso, por arbitrario que sea el criterio del

límite, se debe cumplir que $0 \leq C \leq 100$, para cualquier valor de cobertura C expresado porcentualmente.

En definitiva, lo único que se le impone a una medida de cobertura es que su crecimiento sea monótono no decreciente y que no sobrepase el valor del 100% (lo que vendría a suponer que el criterio esta mal definido). Este límite del 100% nos indica cuando debemos parar de ejecutar pruebas.

Los criterios para definir la cobertura son, como hemos dicho, artificiales. No obstante, deben reflejar alguna característica o modelar algún aspecto que ayuden a interpretar los valores obtenidos en la asignación de cobertura a una Batería de Pruebas o a una ejecución de una prueba simple. Por ello, se le pide que:

- sea fácil de definir.
- sea fácil de medir.
- sea fácil de entender.

Una consideración a tener en cuenta al hablar de ejecución de pruebas es si el sistema se ve como una caja blanca o negra. En la caso de validación de una especificación formal, estaremos comprobando si ésta cumple ciertos requisitos, por lo que estaremos interesados en pruebas transparentes o estructurales, es decir, de caja blanca.

Sin embargo, en el caso de pruebas de conformidad, existe una tercera entidad involucrada, el producto o Implementación Bajo Pruebas (IUT). En cuanto a lo que se refiere el producto, sólo nos interesan pruebas de caja negra: no importa cómo esta hecho, ni de qué se compone, sino si su comportamiento observable es el mismo que la Especificación de Referencia, que por lo demás, si será perfectamente transparente.

En definitiva, independientemente del objetivo de nuestras pruebas, la especificación formal siempre será vista como un objeto transparente. Deseamos definir una métrica de cobertura que sea finita y que capture los elementos esenciales de un comportamiento descrito en lenguaje LOTOS. Basaremos esta definición en el texto de la especificación: puesto que éste es finito, debería ser fácil definir métricas finitas.

El resto del capítulo está organizado como sigue: la sección 4.2 presenta un estudio de posibles métricas de cobertura que se pueden definir en base a los elementos del lenguaje. Se estudian sus puntos fuertes y sus debilidades, para terminar presentando y justificando nuestra propuesta.

La sección 4.3 define formalmente la medida de cobertura tanto *a priori* como *a posteriori* en función de la parte de la especificación cubierta por la prueba. En esta se incluyen unos ejemplos que justifican el uso de nuestra métrica.

4.2 - Cobertura de Comportamiento

A la hora de definir métricas posibles sobre una especificación textual LOTOS, deberemos basarlas sobre los componentes de dicho texto [Hor94]. En LOTOS, un comportamiento se describe mediante:

- acciones elementales
- predicados
- guardas

En realidad, la diferencia entre predicados y guardas radica esencialmente en el ámbito de los identificadores que pueden usarse. En ambos casos, contienen una expresión *booleana*, que se evaluará a un valor *true* o *false*. En el primer caso, la acción asociada podrá ser observada por (sincronizará con) el entorno del proceso. En caso contrario, esa acción no tendrá lugar. Por lo tanto, en las métricas analizadas, serán considerados como iguales.

4.2.1 - Cobertura de Puertas

Como concepto elemental del lenguaje LOTOS, y parte esencial en la semántica de la descripción de un comportamiento, definimos la métrica más elemental como marcación de los nombres de las puertas de la especificación.

Se define esta cobertura *a priori* como el número de puertas que una batería de pruebas es capaz de ejercitar (nombres de acciones que participan en las acciones contenidas en la batería). En cuanto a la cobertura *a posteriori*, se define como el número de puertas ejercitadas en la ejecución de una prueba.

En una especificación hay un número finito de puertas, lo que nos delimita el limite del 100%. Por otro lado, este límite es fácil de calcular: simplemente se cuenta el número de nombres de puertas de la especificación. La noción es también intuitiva: cada acción observada de la especificación marca las puertas que las componen. Como vemos, cumple todos los requisitos antes enumerados.

En cuanto a la alcanzabilidad del dicho 100%, es posible que en la especificación exista **texto muerto** y/o nombres definidos pero no usados. Esto hará que la cobertura *a posteriori* sea menor o igual que la *a priori*, defecto no deseable.

Por otro lado, el concepto de nombre de puerta es insuficiente para describir una acción en LOTOS: éstas pueden ir acompañadas de una negociación de valores que no tiene porque ser iguales, pero que esta cobertura interpreta sin distinción.

Por último, puede ser imposible satisfacer todos y cada uno de los predicados para que las medidas *a posteriori* de cobertura alcancen el 100%.

4.2.2 - Cobertura de Patrones de Acciones

La homogeneidad en el tratamiento de acciones con el mismo nombre pero con diferentes valores en la sincronización introducida por la métrica de Puertas no siempre responde a las posibles sincronizaciones de las acciones individuales. En efecto, en LOTOS cada acción puede observarse acompañada de una serie ordenada de valores o variables, cuyo

tipo no viene prefijado. Hemos visto que la Cobertura de Puertas confunde acciones consideradas diferentes por su asociación de tipos, y que no sincronizarán según la semántica del lenguaje. Podemos hacer esa distinción, considerando un Patrón de Acción como un nombre de puerta y una lista ordenada de tipos.

En este caso, aplican las mismas consideraciones en cuanto a la facilidad de definición y medida, así como a la intuición de la definición que en la Cobertura de Puertas.

Respecto a la alcanzabilidad del 100%, por un lado, como en el caso anterior, puede existir **texto muerto** en la especificación. Además, existe la dificultad de satisfacer todos y cada uno de los predicados[1] para llegar a todos los posibles comportamientos.

4.2.3 - Cobertura de Predicados y Guardas

Las acciones afectadas por un predicado (o guarda) solo ocurren si el predicado se satisface. En las métricas anteriores esto no supone ningún problema para cobertura *a priori*. Pero en las medidas *a posteriori* solo se tienen en cuenta los predicados que son verdaderos. Aunque esto es deseable, no es suficiente, puesto que el predicado ha sido escrito con la intención de que pueda fallar. Por lo tanto, también es relevante cubrir su fallos.

El número de predicados en un texto LOTOS es finito, evaluándose a dos posibles valores $true, false$. Por lo tanto, para un numero $N_{\mathcal{P}}$ de predicados tendremos $2^{N_{\mathcal{P}}}$ posibles evaluaciones distintas, que establecen el 100% de cobertura.

Es difícil medir la cobertura actual o *a posteriori*. Si el predicado se cumple, estamos en el caso de algunas de la coberturas ya vistas. Si falla, las acciones no son observadas, y el sistema procederá con otro comportamiento distinto (o incluso se bloqueará). Esto implica que se necesita una herramienta de ejecución simbólica capaz de evaluar cada predicado en cada posible situación de la ejecución.

Aparte de las razones apuntadas en los casos anteriores que imposibilitan alcanzar el 100% de cobertura, en este caso puede haber predicados que nunca fallen. Esto ocurre cuando se usan para filtrar datos más que para imponer condiciones. Por ejemplo, y sin importarnos el valor concreto que se obtengan:

choice r : route [] [r = GetRoute (origin, dest)]-> . . .
square ? x, r : float [r = sqrt (x)]; . . .

Por otro lado, es imposible satisfacer todos los predicados que pueden aparecer en una especificación. Por ejemplo:

$$a\ ?\ n,\ x,\ y,\ z : nat\ [(n > 2)\ and\ (z^n = x^n + y^n)];$$

Detectar y eliminar estos casos del análisis de cobertura de predicados puede ser muy difícil, cuya consecuencia inmediata es que el 100% de cobertura es ficticio e inalcanzable.

[1] Se recuerda que se utiliza Predicados para referirnos tanto a Predicados como a Guardas

4.2.4 - Cobertura de Composición de Eventos

Las métricas anteriores tratan con acciones y predicados de forma separada. Sin embargo, esto es simplificar en extremo un lenguaje que admite el llamado estilo orientado a requisitos, donde una o más acciones individuales pueden estar involucradas en un evento compuesto, teniéndose que evaluar diferentes predicados. En definitiva, la cita n-aria de LOTOS.

Por un lado deseamos que la medida de la cobertura refleje la composición de eventos LOTOS, es decir, considerando las aportaciones individuales de cada comportamiento en sincronización. Por otro lado, consideramos que la repetición en la aportación de un proceso realmente no modifica el valor de cobertura, en el sentido que ya se prueba la acción individual. Por lo tanto, desarrollaremos una medida de cobertura que teniendo en cuenta los criterios expuestos al principio de este capítulo nos satisfagan estas consideraciones.

Por desgracia, el número de componentes de una cita n-aria no puede ser determinado estáticamente, puesto que los requisitos son generados dinámicamente (mediante creación dinámica de puertas y procesos). Sin embargo, lo que si se puede determinar estáticamente son los potenciales componentes distintos de un evento. Considerando que la repetición de un componente no aporta en ejecución nada nuevo a la cobertura, es posible obtener un límite en la composición de un evento.

Determinar el límite del 100% de cobertura implica hallar cuáles son las posibles composiciones de acciones en una especificación. La primera aproximación es calcular todas las posibles combinaciones de todos los patrones de sincronización existentes en la especificación, sin repetición, y satisfaciendo las reglas de LOTOS de casamiento de tipos. En general, debido a que no todos los patrones se encuentran en el mismo ámbito y dependiendo de la estructura de sincronización de la especificación no todos estas sincronizaciones serán posibles, por lo que no se alcanzaría la cobertura del 100% ni en el diseño de una Batería de Pruebas ni en la ejecución de un caso de prueba. Este defecto debe ser eliminado de una cobertura que trate de reflejar solo las acciones posibles.

Por otro lado existe un defecto estructural cuando se tratan todas las combinaciones por igual, puesto que no se diferencian construcciones bien diferentes:

g experimentos ; g experimentos
g experimentos || g experimentos
g experimentos ||| g experimentos
g experimentos [] g experimentos
g experimentos >> g experimentos

No sólo estamos interesado en diferenciar entre las diferentes puertas declaradas en la especificación. Si el especificador ha considerado relevante la repetición de una acción, el criterio de cobertura no debería ocultar dichas diferencias. Por lo tanto, refinaremos la idea de “etiqueta”, de forma que no solo nos interesarán diferentes composiciones de eventos, sino que también distinguiremos composiciones de eventos sintácticamente iguales con participaciones de diferentes partes de la especificación. Esta diferenciación es fácil de hacer si tenemos en cuenta la “posición” relativa de los eventos (por ejemplo,

diferenciando por el número de linea de cada acción y suponiendo una sola acción por linea). El ejemplo siguiente nos clarificará lo expuesto:

$$\begin{aligned} B &= P[a]||Q[a] \\ P[x] &= x; x; P[x] \\ Q[y] &= y; Q[y] \end{aligned}$$

Con lo dicho al principio de la sección, existe un evento compuesto a por las acciones individuales x (de P) e y (de Q). Sin embargo, el especificador considera relevante explicitar el comportamiento repetitivo de P, aunque semánticamente es totalmente equivalente haber escrito $P[x] = x; P[x]$. Diferenciando sintácticamente las acciones de B, resulta:

$$\begin{aligned} B_0 &= B_1[a]||B_3[a] \\ B_1[x] &= x; B_2[x] \\ B_2[x] &= x; B_1[x] \\ B_3[y] &= y; B_3[y] \end{aligned}$$

Con lo que tenemos que son dos las combinaciones posibles de acciones individuales, con las participaciones de las expresiones de comportamiento $\{(B_1, B_3), (B_2, B_3)\}$. Evidentemente, lo expuesto con el operador de secuenciación pasa con el resto de operadores LOTOS. Por lo tanto, y en lo que a combinaciones se refiere, de ahora en adelante, cuando hagamos mención a las etiquetas de una especificación nos referiremos a los patrones de sincronización (nombre de puerta más secuencia de tipos) teniendo en cuenta, además, su posición.

La evidencia de esta necesidad de diferenciar entre las dos etiquetas del proceso $P\ [x]$ se muestra en el siguiente ejemplo:

$$\begin{aligned} B &= P[a]||Q[a] \\ P[x] &= x; x; \mathbf{stop} \\ Q[y] &= y; Q[y] \end{aligned}$$

En este caso, si no hubiera diferencia entre las dos etiquetas de P, no se estaría considerando la mitad del comportamiento posible.

En resumen, se quiere una cobertura que sea capaz de:

- comprobar todas las posibles sincronizaciones de una especificación, y sólo las que realmente estén comprendidas en ella, de forma que las medidas *a posteriori* (en ejecución) sea igual a la medida *a priori* (calculada sobre el papel).
- discernir entre distintas formas de componer un mismo evento.
- eliminar la infinitud de posibles transiciones debida a los tipos de datos, esto es, la observación del mismo evento con diferentes valores. En efecto, una transición en los que se presenta un valor entero tendrá infinitas posibilidades de ocurrencia, para los infinitos valores que puede tomar el entero.

- absorber la repetición del mismo patrón de sincronización en un evento compuesto, es decir, que la participación repetida del mismo componente no nos incremente la medida de cobertura.

La solución aportada consiste en derivar todas los posibles combinaciones de acciones de forma constructiva trabajando sobre la especificación LOTOS. La idea es hacer un análisis estructural de la especificación que nos permita calcular dicho conjunto de combinaciones y, por lo tanto, determinar el límite del 100% de cobertura. Por otro lado, dicho análisis será tenido en cuenta a la hora de computar la cobertura de la ejecución de una prueba, considerando los eventos ofrecidos según alguna de las posibles combinaciones ya calculadas.

En la siguiente sección veremos cómo se define formalmente la cobertura *a priori* y *a posteriori* de una prueba sobre una especificación.

4.3 - Definición de la Métrica de Composición de Eventos

Las características de la medida de cobertura expuestas en la sección anterior se pueden resumir en tres puntos:

1. identificación biunívoca de acciones sobre el texto de la especificación.
2. caracterización de acciones por el identificador de la puerta y la lista de identificadores de tipos que la acompaña.
3. eliminación de la infinitud introducida por las clases de equivalencia de los tipos de datos.

El primer punto se resuelve con la identificación única de subexpresiones en la especificación. Los dos puntos restantes se resuelven mediante la representación adecuada de las ofertas de las subexpresiones de comportamiento. Esta representación nos lleva a modificar ligeramente la definición de las reglas de inferencia de ISO8807.

Esta transformación de LOTOS a LOTOS básico ya ha sido estudiada en [Mad91] donde se llegó a construir una herramienta que implementaba dicha transformación [Mad92].

Identificación única de una subexpresión Una expresión de comportamiento estará compuesta por dos tipos de subexpresiones: las subexpresiones de acciones y las subexpresiones de operadores.

Definiremos el conjunto $Act(B)$ como el conjunto de todas las subexpresiones incluidas dentro de una expresión de comportamiento B.

Para podernos referir unívocamente a cada una de estas subexpresiones dentro de la expresión total de todo el comportamiento, definiremos una función que nos proporcionará un identificador único, en la forma de un entero, para cada uno de los elementos b del conjunto Act de una expresión B.

$$\mathbf{Id} : b \rightarrow N \ , \ b \in Act(B)$$

Algunas expresiones pertenecientes al conjunto $Act(B)$ pueden ser idénticas, por lo cual una posible forma de construir el identificador único es denotar mediante enteros la posición de la subexpresión dentro de la expresión total. Puesto que dos subexpresiones distintas nunca podrán estar en la misma posición, este modo de identificación será unívoco.

En general, el uso del identificador de subexpresiones tiene como objeto diferenciar acciones cuya composición sintáctica es distinta. Los eventos simples que componen dicha acción aparecen en expresiones de comportamiento del tipo $B_1 =$ *action_denotation*; B_2. Por esta razón, para simplificar la notación, supondremos que nos referimos a cada expresión de comportamiento LOTOS por su identificador único, representado por B_1, B_2, ..., en vez de $\mathbf{Id}(B_1)$, $\mathbf{Id}(B_2)$, ..., respectivamente.

Representación de una oferta

Definición 1 *Definiremos una oferta como:*

$$g^{\{id_1,\ldots,id_m\}}_{<s_1,\ldots,s_n>}$$

Donde sus componentes serán:

- g: la puerta sobre la cual se realiza la sincronización ($g \in G \cup \{\iota, \delta\}$), Siendo G el conjunto de puertas definidas en la especificación.
- $< s_1, \ldots, s_n >$: la lista de los tipos (**sorts**) que formarán el patrón de acción. Nótese que es aquí donde ignoramos la información de los valores concretos que maneja la especificación, estando interesados únicamente en sus tipos.
- $\{id_1, \ldots, id_m\}$: el conjunto de subexpresiones de acción que participan en la oferta. Cada subexpresión estará representada por su identificador único.

Asimismo definiremos una oferta especial, la oferta nula: O_NIL. Identificaremos el conjunto de ofertas formado por la oferta nula con el conjunto vacío:

$$\{O\_NIL\} = \emptyset$$

Denotaremos como O al conjunto de todas las ofertas, incluida la oferta nula. Denotaremos como OS al conjunto de todos los conjuntos formables con elementos de O.

Reglas de inferencia

Definición 2 *srt es una función sobre los experimentos tal que:*

- $srt(!\ v_i) = sort(v_i)$
- $srt(?\ x_i : S) = S$
- $srt(\textbf{any}\ S) = S$

Definición 3 *Definimos un sistema de derivación D'_b mediante la siguiente lista de axiomas y reglas de inferencia:*

- **inacción** $B = \mathbf{stop}$ no se definen reglas de derivación.
- **terminación** $B = \mathbf{exit}\ (E_1, \ldots, E_n)$

  $\mathbf{exit} - \delta^{\{B\}}_{<srt(E_1),\ldots,srt(E_n)>} \rightarrow \mathbf{stop}$[2].
- **prefijo de acción** $B = g\ v_1, \ldots, v_n\ [P]; B_1$

  Sea B_1 una expresión de comportamiento y $B = g\ v_1, \ldots, v_n\ [P]; B_1$ la expresión de comportamiento resultante de anteceder la acción $g\ v_1, \ldots, v_n\ [P]$ a B_1, con P un predicado cualquiera:

  $B - g^{\{B\}}_{<srt(v_1),\ldots,srt(v_n)>} \rightarrow B_1$

  Si la acción es interna, $B = \mathbf{i}; B_1$:

  $B - \mathbf{i}^{\{B\}}_{<>} \rightarrow B_1$
- **selección** $B = B_1\ []\ B_2$

  Sean B_1, B_2 expresiones de comportamiento, y $B = B_1\ []\ B_2$ la expresión de comportamiento *selección* entre B_1 y B_2:

$$\frac{B_1 \overset{g^I_S}{\rightarrow} B'_1}{B_1\ []\ B_2 \overset{g^I_S}{\rightarrow} B'_1} \qquad \frac{B_2 \overset{g^I_S}{\rightarrow} B'_2}{B_1\ []\ B_2 \overset{g^I_S}{\rightarrow} B'_2}$$

- **composición** $B = B_1\ |[g_1, \ldots, g_n]|\ B_2$

  Sean B_1 y B_2 expresiones de comportamiento y $\{g_1, \ldots, g_n\}$ una lista de nombres de puertas. Entonces $B = B_1\ |[g_1, \ldots, g_n]|\ B_2$ es el comportamiento resultante de sincronizar B_1 y B_2 sobre $\{g_1, \ldots, g_n\}$.

$$\frac{B_1 \overset{g^I_S}{\rightarrow} B'_1, g \notin \{g_1, \ldots, g_n, \delta\}}{B_1\ |[g_1, \ldots, g_n]|\ B_2 \overset{g^I_S}{\rightarrow} B'_1 |[g_1, \ldots, g_n]| B_2}$$

[2]Recordemos de la sección 4.3 que usamos B_1 como abreviatura de $\mathbf{Id}(B_1)$

$$\frac{B_2 \stackrel{g_S^I}{\rightarrow} B_2', g \notin \{g_1, \ldots, g_n, \delta\}}{B_1 \; |[g_1, \ldots, g_n]| \; B_2 \stackrel{g_S^I}{\rightarrow} B_1 |[g_1, \ldots, g_n]| B_2'}$$

$$\frac{B_1 \stackrel{g_S^{I_1}}{\rightarrow} B_1', B_2 \stackrel{g_S^{I_2}}{\rightarrow} B_2', g \in \{g_1, \ldots, g_n, \delta\}}{B_1 \; |[g_1, \ldots, g_n]| \; B_2 \stackrel{g_S^{I_1 \cup I_2}}{\rightarrow} B_1' |[g_1, \ldots, g_n]| B_2'}$$

- **ocultamiento** $B = \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1$

  Sea B_1 una expresión de comportamiento y $B = \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1$ la expresión de comportamiento resultante de ocultar las transiciones sobre $\{g_1, \ldots, g_n\}$ en el comportamiento B_1.

$$\frac{B_1 \stackrel{g_S^I}{\rightarrow} B_1', g \notin \{g_1, \ldots, g_n\}}{\mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1 \stackrel{g_S^I}{\rightarrow} \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1'}$$

$$\frac{B_1 \stackrel{g_S^I}{\rightarrow} B_1', g \in \{g_1, \ldots, g_n\}}{\mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1 \stackrel{\mathbf{i}_S^I}{\rightarrow} \mathbf{hide}\ g_1, \ldots, g_n]|\ \mathbf{in}\ B_1'}$$

- **habilitación** $B = B_1 >> B_2$

  Sean B_1, B_2 expresiones de comportamiento y $x_1, \ldots, x_n$ declaraciones de variables. Entonces $B = B_1 >> \mathbf{accept}\ x_1, \ldots, x_n\ \mathbf{in}\ B_2$ es el comportamiento resultante de habilitar el comportamiento B_2 tras la terminación de B_1, con el paso de parámetros $x_1, \ldots, x_n$.

$$\frac{B_1 \stackrel{g_S^I}{\rightarrow} B_1', g \neq \delta}{B_1 >> B_2 \stackrel{g_S^I}{\rightarrow} B_1' >> \mathbf{accept}\ x_1, \ldots, x_n\ \mathbf{in}\ B_2}$$

$$\frac{B_1 \stackrel{\delta_S^I}{\rightarrow} B_1', g = \delta}{B_1 >> B_2 \stackrel{\mathbf{i}_S^I}{\rightarrow} B_2}$$

  Es importante resaltar que no se instancian los parámetros $x_1, \ldots, x_n$ con los correspondientes valores $E_1, \ldots, E_n$ porque no estamos interesados en los valores concretos, sino sólo en los tipos de los mismos.

- **deshabilitación** $B = B_1 [> B_2$

  Sean B_1, B_2 expresiones de comportamiento y $B = B_1 [> B_2$ el comportamiento resultante de deshabilitar B_1 ante la evolución de B_2.

$$\frac{B_1 \stackrel{g_S^I}{\rightarrow} B_1'}{B_1 [> B_2 \stackrel{g_S^I}{\rightarrow} B_1' [> B_2}$$

$$\frac{B_2 \stackrel{g_S^I}{\rightarrow} B_2'}{B_1 [> B_2 \stackrel{g_S^I}{\rightarrow} B_2'}$$

- **instanciación** $B = p[g_1, \ldots, g_m](v_1, \ldots, v_r)$

  Sea $p[g_1, \ldots, g_m](v_1, \ldots, v_r)$ una instanciación de proceso y $B = p[g_1, \ldots, g_m](v_1, \ldots, v_r)$ el resultado de instanciar p con las puertas actuales $[g_1, \ldots, g_m]$ y los valores actuales $(v_1, \ldots, v_r)$.

$$\frac{B_p \stackrel{h^I_{<s_1,\ldots,s_n>}}{\rightarrow} B_p', < p, B_p > \in BS}{p[g_1, \ldots, g_m](v_1, \ldots, v_r) \stackrel{R(h)^I_{<s_1,\ldots,s_n>}}{\rightarrow} B_p'}$$

  donde

$$R(h) = \begin{cases} g_i & \text{si } h = h_i \ (1 \leq i \leq m) \\ h & \text{en otro caso} \end{cases}$$

$$srt(v_i) = s_i \ 1 \leq i \leq r$$

$$\frac{B_1 \stackrel{g^I_{<s_1,\ldots,s_n>}}{\rightarrow} B_1'}{B[S] - R(g)^I_{<s_1,\ldots,s_n>} \rightarrow B_1'[S]}$$

- **Sincronización total** $B = B_1 \,||\, B_2$

  La sincronización total de dos comportamientos se reescribe a:

$$B = B_1 \,|[gates(B_1) \cup gates(B_2)]|\, B_2$$

  donde la función $gates$ extrae todos los nombres de puertas de una expresión de comportamiento y se define en la tabla 4.1.

- **Entrelazamiento** $B = B_1 \,|||\, B_2$

  El comportamiento resultante de entrelazar las ofertas de dos subexpresiones se reescribe según:

  $$B = B_1 \,|[]|\, B_2$$

  Recordemos que la sincronización generalizada sincroniza en la puerta δ, que es, precisamente, la única sincronización en el entrelazado de comportamientos.

- **Declaración de variables** $B = \mathbf{let}\ vd_1, \ldots, vd_n\ \mathbf{in}\ B_1$

- **Expresión guardada** $B = [expr] \rightarrow\ B_1$

  Tanto en el caso de declaración de variables como comportamientos guardados, dado que queremos evaluar todas las posibilidades de combinaciones, los valores concretos de las ofertas no tienen ningún efecto sobre las posibilidades de sincronización, por lo que ni los valores concretos en las sincronizaciones ni las guardas (y predicados) que afectan a comportamientos subsiguientes no serán tenidas en cuenta. De esta forma, estas expresiones se reescribirán a:

  $$B = B_1$$

La función $gates$

Definición 4 *Se define la función de extracción de puertas:*

$$gates : beha \rightarrow (G^+)^*$$

Definición de Trazas

Definición 5 *Se define el conjunto de trazas sobre un sistema de derivación D'_b como*

$$\sigma = (OS)^*$$

donde σ, σ', σ_1, . . . denotan sus elementos.

Usaremos letras mayúsculas B, P, Q, R, . . . para referenciar estados sobre la especificación, μ_1, μ_n, . . . para ofertas y la siguiente notación sobre las trazas, sus estados y la alcanzabilidad de estados [Bri87a]:

- $P - \mu_1 \cdots \mu_n \rightarrow Q$ denota que existen P_i tal que $P = P_0 - \mu_1 \rightarrow P_1 - \mu_2 \rightarrow \cdots - \mu_n \rightarrow P_n = Q$.

- $P - \mu_1 \cdots \mu_n \rightarrow$ denota que existe Q tal que $P - \mu_1 \cdots \mu_n \rightarrow Q$.

$B = a; B_1$	$gates(B) = \{a\} \cup gates(B_1)$
$B = a\ d_1 \ldots d_n; B_1$	$gates(B) = \{a\} \cup gates(B_1)$
$B = \mathbf{i}; B_1$	$gates(B) = gates(B_1)$
$B = B_1\ [\ ]\ B_2$	$gates(B) = gates(B_1) \cup gates(B_2)$
$B = B_1\ \|\|\ B_2$	$gates(B) = gates(B_1) \cup gates(B_2)$
$B = B_1\ \|[g_1, \ldots, g_n]\|\ B_2$	$gates(B) = gates(B_1) \cup gates(B_2)$
$B = B_1\ \|\|\|\ B_2$	$gates(B) = gates(B_1) \cup gates(B_2)$
$B = B_1\ >>\ B_2$	$gates(B) = gates(B_1) - \{\delta\} \cup gates(B_2)$
$B = B_1\ [>\ B_2$	$gates(B) = gates(B_1) \cup gates(B_2)$
$B = \mathbf{hide}\ [g_1, \ldots, g_n]\ \mathbf{in}\ B_1$	$gates(B) = gates(B_1) - \{g_1, \ldots, g_n\}$
$B = \mathbf{let}\ vd_1, \ldots, vd_n\ \mathbf{in}\ B_1$	$gates(B) = gates(B_1)$
$B = [expr] \rightarrow\ B_1$	$gates(B) = gates(B_1)$
$B = \mathbf{stop}$	$gates(B) = \{\}$
$B = \mathbf{exit}$	$gates(B) = \{\delta\}$
$B = \mathbf{exit}\ (E_1, \ldots, E_n)$	$gates(B) = \{\delta\}$
$B = p[a_1, \ldots, a_n](E_1, \ldots, E_n)$	$gates(B) = \{a_1, \ldots, a_n\}$

Tabla 4.1: Función de Extracción de Nombres de Puertas de un Comportamiento

- $P - \mu_1 \cdots \mu_n \not\rightarrow$ denota que no $P - \mu_1 \cdots \mu_n \rightarrow$.

- $P = \varepsilon \Rightarrow Q$ denota

  $P - \tau^n \rightarrow Q$ para algún $n \geq 0$. La transición $-\tau \rightarrow$ es de crucial importancia en la representación del indeterminismo. Este indeterminismo representa los cambios en las capacidades dinámicas de los procesos sin que sean afectados por el entorno. De hecho, estos cambios solo pueden observarse indirectamente, mediante el análisis del comportamiento futuro de los procesos.

- $P = a \Rightarrow Q$ denota que existen dos procesos P_1 y P_2, tales que $P = \varepsilon \Rightarrow P_1 - a \rightarrow P_2 = \varepsilon \Rightarrow Q$.

- $P = a_1 \ldots a_n \Rightarrow Q$ denota que existen P_i con $0 \leq i \leq n$ tales que $P = P_0 = a_1 \Rightarrow \ldots = a_n \Rightarrow P_n = Q$. De forma mas compacta, $P = \sigma \Rightarrow Q$ representa que el proceso P, desde su estado inicial, puede aceptar una secuencia de acciones que le llevan al estado inicial del proceso Q. Evidentemente $P = \varepsilon \Rightarrow P\ \forall P$.

- $P = \sigma \Rightarrow$ denota que existe algún Q tal que $P = \sigma \Rightarrow Q$.

- $P = \sigma \not\Rightarrow$ denota que no $P = \sigma \Rightarrow$.

- $Tr(P)$ se usa para denotar $\{\sigma | P = \sigma \Rightarrow\}$, es decir, el conjunto de trazas que pertenecen a P.

Definición de Patrón de Acción

Definición 6 *Se definen los Patrones de Acción de una expresión de comportamiento B como:*

$$PS(B) = \{o | \exists \sigma, B' \; B = \sigma \Rightarrow B' - o \rightarrow\}$$

Es decir, los Patrones de Acción serán las ofertas que pueden observarse en una especificación.

Definición de Prueba Mensurable Para definir cuándo una prueba es mensurable, impondremos unas condiciones que requieren de las siguientes definiciones previas:

Definición 7 *Una traza σ es identificable en una expresión de comportamiento B y se denota como $\sigma \; \underline{id} \; B$ si y solo si:*

$$\begin{array}{l} \textit{si } B = \sigma \Rightarrow \quad \Rightarrow \\ \quad \forall \; Q_1, Q_2 \textit{ tal que } B = \sigma \Rightarrow Q_1, B = \sigma \Rightarrow Q_2 \Leftrightarrow Q_1 \equiv Q_2 \end{array}$$

Intuitivamente, una traza identificable termina en un estado único, es decir, la observación de dicha traza siempre llega al mismo estado.

Definición 8 *Una traza σ es terminante en una expresión de comportamiento B y se denota como $\sigma \; \underline{ter} \; B$ si y solo si:*

$$\begin{array}{l} \textit{si } B = \sigma \Rightarrow \quad \Rightarrow \\ \quad \forall a \in OS \;\; B = \sigma \Rightarrow -a \not\rightarrow \end{array}$$

Intuitivamente, una traza es terminante en una comportamiento si, al aceptarla éste, se llega a un estado a partir del cual no se puede evolucionar.

Definición 9 *Una prueba mensurable sobre la especificación (S) será una expresión de comportamiento T tal que:*

$$\forall \sigma \in Tr(T) \textit{ si } \sigma \; ter \; T \Rightarrow \sigma \; id \; S$$

Intuitivamente, una prueba es mensurable cuando todas sus trazas que consiguen terminar llegan a un estado único en S. Esto nos elimina la posibilidad de no poder determinar en qué estado se halla la especificación tras observar alguna traza terminante de T, por lo que se puede asignar de una forma unívoca una medida de cobertura.

Definición 10 *La métrica de cobertura será una función:*

$$\mathcal{M} : beha \times beha \rightarrow N^+ \cup \{0\}$$

Sea una prueba T mensurable sobre una especificación S. La medida de cobertura de T será

$$\mathcal{M}(T, S) = \frac{Card(PS(T||S))}{Card(PS(S))}$$

4.4 - Conclusiones

En este capítulo se ha dado una definición de medida de cobertura que captura los elementos semánticos de una especificación LOTOS, en cuanto a la componibilidad de eventos simples. Esta definición evita el problema de la infinitud de comportamientos (y, por tanto, de eventos) en base a la consideración de que la ejecución repetitiva del mismo evento en una misma acción observable no aporta valor en la medida de cobertura final.

La definición de la medida de cobertura se ha dado en forma de una función que computa la parte examinada de una especificación por una prueba dada. Sobre esta prueba se imponen unas restricciones, orientadas a asignar cobertura sólo en los casos en que el estado alcanzado en la identificación sea identificable (único). Para ello, hemos definido los que son trazas terminantes y trazas identificables respecto de una expresión de comportamiento.

Esta función solo asigna cobertura a una prueba *respecto* de la parte de la especificación que examina. No es necesario, como puede parecer a simple vista, exigir que $Tr(T) \in Tr(S)$, puesto que interesa poder asignar medida de cobertura a pruebas de rechazo en las que claramente esto no se cumple.

Capítulo 5

Algoritmo para Cálculo de Cobertura

5.1 - Introducción

En este capítulo se describe un algoritmo para el cálculo de la medida de cobertura propuesta. Existen diferentes aproximaciones, pero justificaremos nuestra elección en base a unos ejemplos de aplicación.

El algoritmo tiene dos facetas en su resultado: por un lado deberá determinar todas las combinaciones de acciones posibles. Por otro, deberá facilitar esta información para el cálculo del Límite de Cobertura, esto es, el valor de Cobertura al que se asigna un 100%. Además, esta información se usará para determinar las coberturas *a priori* de una batería de pruebas y la cobertura *a posteriori* de la ejecución de un caso de prueba.

Para calcular el conjunto de posibles combinaciones la primera idea que surge es aplicar los Teoremas de Expansión de LOTOS e ir registrando las acciones que se van generando. Evidentemente, llegará un momento en que no se registran "nuevas" acciones[1], pero el problema es parar de expandir. Este problema surge al tratar con especificaciones recursivas, en las cuales el ofrecimiento de acciones va a depender del valor tomado por unas guardas o ciertos predicados. Por otro lado, el hecho de que de un paso a otro de expansión no se obtengan nuevas combinaciones no implica que estas no vayan a existir. Necesitamos un mecanismo por el cual se asegure que a partir de cierto punto en la expansión de la especificación no se encontrarán nuevas combinaciones de eventos.

5.2 - Identificación sintáctica de eventos simples de LOTOS

En la medida de cobertura propuesta se tiene en cuenta que una misma acción LOTOS puede estar compuesta de muchas formas diferentes, y varios comportamientos pueden participar sobre una misma puerta. Es importante resaltar que incluso un mismo comportamiento puede participar sobre una misma puerta de formas diferentes, hecho que se expresa en la diferenciación sintáctica de la aparición de eventos sintácticamente distintos.

[1] "Nuevas" en el sentido dado en el apartado 4.2.4

Examinemos el siguiente ejemplo:

$$\begin{aligned} B &= P[a]||Q[a] \\ P[x] &= x; x; \mathbf{stop} \\ Q[y] &= y; Q[y] \end{aligned}$$

El comportamiento de esta especificación es la secuencia de dos acciones a seguidas, pero cuya observación requiere la participación del proceso P con dos construcciones diferentes. Para distinguirlas, se identifican las subexpresiones de comportamiento de la siguiente forma:

$$\begin{aligned} B_0 &= B_1[a]||B_4[a] \\ B_1[x] &= x; B_2 \\ B_2[x] &= x; B_3 \\ B_3 &= \mathbf{stop} \\ B_4[y] &= y; B_4[y] \end{aligned}$$

De esta forma se hace patente la diferencia sintáctica entre la primera y segunda a del comportamiento, entendiendo la primera como $a(B_1, B_4)$ y la segunda como $a(B_2, B_4)$.

5.3 - Reconocimiento de Patrones de Sincronización de Comportamientos (PSC)

El punto clave para la determinación de las posibles sincronizaciones aún en el caso de comportamientos recursivos es el reconocimiento de Patrones de Sincronización de Comportamientos (PSC). El razonamiento se basa en el hecho de que el operador de sincronización es el que genera nuevas combinaciones. Ahora bien, un comportamiento es infinito por instanciación recursivas de comportamientos que ya han sido ejecutados alguna vez: puesto que la especificación es finita, no pueden aparecer infinitos comportamientos distintos. Cada nueva instanciación de comportamiento se limita a repetir comportamientos que ya han sido observados (excepto, por supuesto, la primera vez). Estructuralmente, el comportamiento es nuevo en el sentido que existen más participantes en las ofertas: pero estos participantes se están repitiendo continuamente. Y ya se ha visto que en la definición de nuestra cobertura que esta repetición no tiene ningún valor.

Veamos con algunos ejemplos lo expuesto:

Ejemplo 1 Sea el comportamiento $P = x; P$. Evidentemente, la única acción ejecutable es x. Pero tras la primera instanciación, se vuelve a repetir el comportamiento, no aportando nuevas combinaciones de acciones individuales. Solo es necesario instanciar P una vez: las siguientes veces no aportaran nada en cuanto a cobertura.

Otra forma de ver que no es necesario instanciar P nada más que una vez es la siguiente: llamando Of a la función que extrae las combinaciones de eventos posibles de una expresión de comportamiento LOTOS:

$$Of(P) = \{a\} \cup Of(P)$$

Ahora bien, por el teorema de recursión de Kleene [Tar73, Plo81, Hoa94] la solución de esta igualdad es el mínimo punto fijo de la cadena $A = B$, $A = B \cup A$, $A = B \cup B \cup A$, ..., por lo que tenemos que:

$$Of(P) = \{a\}$$

Ejemplo 2 El siguiente ejemplo se desarrolla en detalle, incluida la identificación de eventos simples y la construcción del conjunto de PSC reconocidos. Sea el comportamiento[2]:

$$\begin{array}{rcl} B & = & P[a]||Q[a] \\ P[x] & = & x; x; P[x] \\ Q[y] & = & y; Q[y] \end{array}$$

El primer paso será identificar cada evento simple en la especificación (mediante identificación de la subexpresión a la que pertenecen), así como los diferentes comportamientos que la componen. El resultado es:

$$\begin{array}{rcl} B_0 & = & B_1[a]||B_3[a] \\ B_1[x] & = & x; B_2[x] \\ B_2[x] & = & x; B_1[x] \\ B_3[y] & = & y; B_3[y] \end{array}$$

Si vamos "ejecutando" el comportamiento B paso a paso, iremos obteniendo las diferentes combinación de acciones. En este ejemplo vamos a construir el conjunto de PSC reconocidos de forma detallada. En la primera instanciación de $P||Q$ obtendremos el evento a compuesto por (B_1, B_3). Llamando PR al conjunto de PSC reconocidos tenemos $PR = \{(B_1||B_3)\}$. El comportamiento resultante es $B_2||B_3$. Este comportamiento es nuevo, por lo que es de esperar nuevas combinaciones. Esto se determina porque $(B_2||B_3) \notin PR$. En efecto, observamos la acción a compuesta por (B_2, B_3), que es nueva. Añadimos este nuevo PSC al conjunto $PR = \{(B_1||B_3), (B_2||B_3)\}$. El comportamiento resultante es $B_1||B_3$. Evidentemente, las combinaciones que obtengamos a partir de aquí ya han sido obtenidas, por lo que no es necesario volver a expandir. Detenemos la búsqueda de nuevas combinaciones porque $(B_1||B_3) \in PR$.

Se puede también hacer un razonamiento algebraico parecido al ejemplo anterior:

$$\begin{array}{l} Of(B_0) = Of(B_1||B_3) \\ Of(B_1||B_3) = \{a_{(B_1,B_3)}\} \cup Of(B_2||B_3) \\ Of(B_1||B_3) = \{a_{(B_1,B_3)}\} \cup \{a_{(B_2,B_3)}\} \cup Of(B_1||B_3) \end{array}$$

Por lo tanto, el mínimo punto fijo de esta igualdad es:

$$Of(B_1||B_3) = \{a_{(B_1,B_3)}, a_{(B_2,B_3)}\}$$

[2]Este ejemplo ya se utilizó para expresar la idea de combinación de eventos

Ejemplo 3 Existen posibilidades más complejas, que es cuando se van añadiendo comportamientos a una sincronización, como en el siguiente ejemplo:

$$\begin{array}{rcl} B & = & P[a]||Q[a] \\ P[x] & = & x; (P[x]||Q[x]) \\ Q[y] & = & y; Q[y] \end{array}$$

El resultado de esta especificación es que cada vez más procesos Q sincronizan para ofrecer la acción a. Desde el punto de vista de cobertura estamos repitiendo (ejecutando) lo mismo, con lo que carece de interés. Pero, ¿seremos capaces de reconocer un PSC a medida que vamos instanciando procesos? Veámoslo.

Como primer paso, identificamos las subexpresiones de comportamiento:

$$\begin{array}{rcl} B_0 & = & B_1[a]||B_3[a] \\ B_1[x] & = & x; B_2[x] \\ B_2[x] & = & B_1[x]||B_3[x]) \\ B_3[y] & = & y; B_3[y] \end{array}$$

Nótese que, aunque la construcción $P[x]||Q[x]$ aparece dos veces, los asignamos un identificador distinto, pues la identificación nos interesa a nivel sintáctico.

En un principio, tenemos por instanciar la sincronización $B_1||B_3$. Esto nos ofrece la acción $a(B_1, B_3)$, dejando instanciado $B_1||B_3||B_3$. Desde el punto de vista de comportamiento, esta expresión y la primera son bien diferentes. Desde el punto de vista de aportación a la acción a, tenemos que su composición es (B_1, B_3, B_3), esto es, obviando la repetición, (B_1, B_3). Por otro lado, este comportamiento no ofrecerá nuevas combinaciones: sólo hay dos etiquetas, por lo que pueden sincronizar de un solo modo (el obtenido). Por lo tanto, las combinaciones de acciones individuales que sacaremos de $B_1||B_3||B_3$ son las mismas que las de $B_1||B_3$. En efecto:

$$Of(B_1||B_3||B_3) = \{(B_1, B_3, B_3)\} = \{(B_1, B_3)\} = Of(B_1||B_3)$$

Más fácil es de ver algebráicamente, teniendo en cuenta que el mínimo punto fijo de $Of(A||B||B) = Of(A||B)$.

$$\begin{array}{l} Of(B_0) = Of(B_1||B_3) \\ Of(B_1||B_3) = \{a_{(B_1,B_3)}\} \cup Of(B_1||B_3||B_3) \\ Of(B_1||B_3) = \{a_{(B_1,B_3)}\} \cup Of(B_1||B_3) \end{array}$$

Y de nuevo tenemos que:

$$Of(B_1||B_3) = \{a_{(B_1,B_3)}\}$$

En definitiva, cuando tras observar una oferta existe una recursión de instanciación de comportamiento, se llega a una situación que debe ser reconocible como previamente

tratada para evitar su exploración. En estos ejemplos hemos visto como es deseable "reconocer" que ciertos PSC han sido instanciados anteriormente. Esto es muy fácil de hacer cuando la instanciación es exactamente la misma (ejemplos 1 y 2), puesto que simplemente se necesita una función que detecte igualdad de patrones. Para el ejemplo 3, hemos aplicado una propiedad derivada de nuestra definición de cobertura.

5.3.1 - Definición de los PSC

Los PSCs (Patrones de Sincronización de Comportamiento) son casos particulares de expresiones de comportamientos LOTOS, donde se representa solamente la información de sincronización e identificadores únicos de subexpresiones de comportamiento. En definitiva, son expresiones de comportamiento compuestas por operadores LOTOS e identificadores de comportamientos de acción. Subexpresiones de acción son prefijo de acción (con o sin predicados de selección), acción interna (**i**) y terminación (**exit**).

Así, los PSC serán del estilo de $B_1[]B_3|||B_4$ etc, contando con paréntesis para priorizar composiciones.

5.4 - Esbozo del Resto del Capítulo

Vamos a presentar la estructura del resto del capítulo, orientada a la construcción del algoritmo y a los pasos básicos de que se compone el mismo, presentando los fundamentos en los cuales se basa:

1. Antes de entrar en el algoritmo propiamente dicho, veremos unas consideraciones previas donde se decide no tener en cuenta los operadores "masivos" de LOTOS, **choice** y **par**.

2. Para poder manejar el algoritmo y definirlo formalmente, se introducirá una nomenclatura para tratar composición de ofertas, reetiquetados, etc.

3. Definiremos la función IB que implementará la función **Id** (definida en 4.3) de identificación de subexpresiones LOTOS.

4. Definiremos la función Of que dada una expresión de comportamiento LOTOS extrae todas las acciones posibles. Esta función evaluará las ofertas de subexpresiones de operadores LOTOS según el sistema de derivación D'_b.

5. Definiremos la función gl que permite obtener el estado siguiente de una expresión LOTOS a partir de un estado actual y la observación de una oferta[3].

[3]La definición de estas dos funciones son una modificación de las propuestas en [Sal93]. En dicho trabajo, por consideraciones de ejecución, la función Of se desarrollaba en dos partes. Esto no será necesario para nuestros propósitos.

6. Definiremos dos operadores de composición de expresiones LOTOS, denotados como $< + >$ y $< * >_g$. El primero será definido como la unión de dos expresiones de comportamiento. El segundo será la sincronización de dos expresiones de comportamiento sobre la puerta genérica g.

   Se estudiará una serie de propiedades de estos operadores útiles en el Reconocimiento de PSC.

7. Definiremos una transformación de expresiones de comportamiento LOTOS a expresiones compuestas con los operadores $< + >$ y $< * >_g$.

8. Definiremos la igualdad de PSCs (subexpresiones LOTOS, en definitiva) en base a las expresiones transformadas usando operadores $< + >$ y $< * >$.

9. Definiremos la función RC que obtendrá el PSC de una expresión de comportamiento.

10. En base a todos estos elementos definiremos la función de cálculo de combinaciones de eventos sobre una expresión de comportamiento. Estas combinaciones serán todas las ofertas que ofrece una especificación LOTOS en los diferentes estados a los que se puede llegar. Se utilizará la función RC para determinar cuándo no es necesario seguir extrayendo ofertas debido a la repetición de PSCs, que nos establecerá el punto fijo de esta extracción.

5.5 - Formalización del Algoritmo de Cálculo de Cobertura

5.5.1 - Consideraciones previas

En el tratamiento de las especificaciones LOTOS, no consideraremos los operadores **choice** ni **par**. La razón de ello es que son operadores relativamente complejos, que dificultan el tratamiento matemático. Dado su poco uso, y como existen unas reglas de transformación en operadores más simples, se considerará que dichas transformaciones se llevan a cabo antes del procesado para cobertura. Esto ni siquiera es necesario para el operador **choice** sobre variables, puesto que no influye en las combinaciones que serán calculadas. Por lo tanto, en este caso, no se realizará la transformación.

Por último, esto no tiene ninguna implicación sobre las medidas de cobertura, lo que apoya la idea de no tratarlos. De todas formas, y en aras de una mayor completitud, se incluyen a continuación las transformaciones de dichos operadores:

- **choice**

  Este operador se expande en dos fases: la primera consiste en separar los dominios de aplicabilidad, lo que supone una simple expansión sintáctica:

$$\textbf{choice } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_m], \ldots, p_1, \ldots, p_r \textbf{ in } [h_1, \ldots, h_s] \; [] \; B \Rightarrow$$

$$\begin{array}{l} \textbf{choice } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_n] \; [] \\ \quad \textbf{choice } t_1, \ldots, t_u \textbf{ in } [..] \; [] \\ \qquad \vdots \\ \qquad\quad \textbf{choice } p_1, \ldots, p_r \textbf{ in } [h_1, \ldots, h_s] \; [] \; B \end{array}$$

Y ahora se desdobla en operadores simples de elección:

$$\textbf{choice } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_m] \; [] \; B \Rightarrow$$

$$\begin{array}{l} \quad [g_1/k_1, \ldots, g_1/k_n] \; B \\ [] \; [g_1/k_1, \ldots, g_2/k_n] \; B \\ \quad \vdots \\ [] \; [g_m/k_1, \ldots, g_1/k_n] \; B \\ \quad \vdots \\ [] \; [g_m/k_1, \ldots, g_m/k_n] \; B \end{array}$$

- **par**

  Análogamente para el operador de paralelismo: primero desdoblamos los dominios de aplicabilidad:

$$\textbf{par } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_m], \ldots, p_1, \ldots, p_r \textbf{ in } [h_1, \ldots, h_s] \; op \; B$$

$$\begin{array}{l} \textbf{par } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_n] \; op \\ \quad \textbf{par } t_1, \ldots, t_u \textbf{ in } [..] \; op \\ \qquad \vdots \\ \qquad\quad \textbf{par } p_1, \ldots, p_r \textbf{ in } [h_1, \ldots, h_s] \; op \; B \end{array}$$

  Y ahora se desdobla en operadores simples de elección:

$$\textbf{par } k_1, \ldots, k_n \textbf{ in } [g_1, \ldots, g_m] \; op \; B$$

$$\begin{array}{l} \quad [g_1/k_1, \ldots, g_1/k_n] \; B \\ op \; [g_1/k_1, \ldots, g_2/k_n] \; B \\ \quad \vdots \\ op \; [g_m/k_1, \ldots, g_1/k_n] \; B \\ \quad \vdots \\ op \; [g_m/k_1, \ldots, g_m/k_n] \; B \end{array}$$

  donde $op \in \{||, |[g_1, \ldots, g_n]|, |||\}$.

5.5.2 - Nomenclatura

La nomenclatura usada en las siguientes secciones para la denotación de los elementos que componen una especificación LOTOS y el manejo de los mismos está tomada de la Tesis de Joaquín Salvachúa, donde se estudió la representación de instanciaciones de comportamientos para el cálculo de ofertas y resolución de citas [Sal93]. En este sentido es necesario aclarar que en el ámbito de esta Tesis, las expresiones *ofertas* y *acciones observables* son completamente sinónimas.

Para el cálculo de las posibles composiciones de eventos (u ofertas) se necesitará la representación de los mismos y la resolución de las citas de LOTOS. Para ello desarrollaremos dos funciones, una para el cálculo de ofertas dada una expresión de comportamiento que nos dará las posibles combinaciones y otra que nos determine si, dado el conjunto de ofertas calculadas en un punto de la especificación, nos diga si el comportamiento resultante ofrecerá ofertas que sean combinaciones nuevas (y, por lo tanto, susceptibles de ser calculadas). Esto se basará en el Reconocimiento de PSCs.

Los siguientes apartados presentan lo que constituye el universo de discurso de la memoria, para posteriormente poder construir la función de obtención de ofertas y así llegar al algoritmo.

Ofertas de sincronización

Por oferta de sincronización se entiende un "prototipo" de una transición. Debe entenderse que una transición que aún puede no estar totalmente definida, bien por no estar aún definida la lista de términos básicos sobre los cuales se realizará la transición o por no estar completo el conjunto de acciones que la formarán.

De esta forma, una transición sería una oferta de sincronización totalmente "madurada". Es precisamente esta característica de las ofertas la que permitirá el cálculo de transiciones. No es necesario determinar la transición desde el principio, sino que se puede sintetizar poco a poco, mediante la composición de las propuestas de los distintos participantes.

Una oferta contendrá información relativa a la puerta y la lista de términos básicos sobre las que se quiere realizar la transición, así como las subexpresiones de acciones que participarán en dicha propuesta de transición.

Composición de ofertas Recordemos que se ha definido en la sección 1 una oferta como:

$$g^{\{id_1,\ldots,id_m\}}_{<s_1,\ldots,s_n>}$$

Para representar la composición de ofertas, que más adelante veremos que identificaremos con el resultado de la aplicación de los operadores LOTOS, utilizaremos una serie de operaciones que pasamos a detallar:

- Unión de ofertas: ($\oplus$). Representa el entrelazamiento de las subexpresiones de acciones contenidas en las ofertas.

  $$\oplus : OS \times OS \rightarrow OS$$

  Directamente se corresponde con la unión de conjuntos; por tanto podemos definir la función $\oplus$ como:

  $$OS_1 \oplus OS_2 = OS_1 \cup OS_2$$

- Intersección de ofertas: ($\otimes$). Representa la sincronización entre acciones. Las acciones participantes se unen en una posible cita para realizar una sincronización común.

  $$\otimes : OS \times OS \rightarrow OS$$

  Esta función realizará la sincronización de todas las parejas resultantes del producto cartesiano de los dos conjuntos de ofertas.

  $$OS_1 \otimes OS_2 = \{\bigcup_{i,j}(o_{1_i} \diamond o_{2_j}) | \forall o_{1_i} \in OS_1, \forall o_{2_j} \in OS_2\}$$

  Donde la operación $\diamond$ es la sincronización de acciones individuales.

  Veamos la definición de esta última función. Es una función que nos proporcionará una nueva oferta a partir de la combinación de otras dos. Si la combinación no es viable el resultado será la oferta nula.

  $$\diamond : O \times O \rightarrow O$$

  Sean dos ofertas O_1, O_2:

  $$O_1 = g1^{I_1}_{<s_1,\ldots,s_n>}$$
  $$O_2 = g2^{I_2}_{<r_1,\ldots,r_m>}$$

  - Si $g1 \neq g2$ entonces $O_1 \diamond O_2 = O\_NIL$.
  - Si $g1 = \mathbf{i}$ entonces $O_1 \diamond O_2 = O\_NIL$.
  - Si $g2 = \mathbf{i}$ entonces $O_1 \diamond O_2 = O\_NIL$.
  - Si $n \neq m$ entonces $O_1 \diamond O_2 = O\_NIL$.
  - Si $s_i \neq r_i \ \ 1 \leq i \leq m_1$ entonces $O_1 \diamond O_2 = O\_NIL$.

  Si no ocurre ninguno de estos casos entonces:

  $$O_1 \diamond O_2 = g1^{I_1 \cup I_2}_{<s_1,\ldots,s_m>}$$

Operaciones relacionadas con conjuntos de ofertas Para su posterior uso pasamos a definir una serie de funciones que trabajarán sobre conjuntos de ofertas.

Definiremos el conjunto G^+ como: $G^+ = G \cup \{\delta\}$.

- Obtención del identificador de puerta de una oferta ($gate$):

$$gate : O \rightarrow G^+$$

Donde

$$gate(g_S^I) = g$$

- Selección de ofertas realizadas sobre una puertas (s).

$$s : G^+ \times OS \rightarrow OS$$

De un conjunto dado nos dará el subconjunto de las ofertas que están siendo ofrecidas sobre el nombre de puerta indicado.

$$s(g, o\_set) = \{o | o \in o\_set \ \wedge \ gate(o) = g\}$$

- Eliminación de ofertas realizadas sobre una puerta (d).

$$d : G^+ \times OS \rightarrow OS$$

De un cierto conjunto de ofertas nos proporcionará el subconjunto de ofertas que no están siendo ofrecidas sobre el nombre de puerta indicado.

$$d(g, off\_set) = \{o | o \in off\_set \ \wedge \ gate(o) \neq g\}$$

- Operación de sustitución de puertas sobre ofertas:

$$sust_1 : G^+ \times O \rightarrow O$$

Consiste en la obtención de una nueva oferta mediante la sustitución de la puerta de la oferta por la proporcionada:

$$sust_1(h, g_S^I) = h_S^I$$

- Operación de sustitución de puertas sobre ofertas:

$$sust : G^{+} \times OS \rightarrow OS$$

Consiste en el conjunto resultante de la aplicación de la función $sust_1$ a todos los elementos del conjunto de ofertas sobre el que se aplica:

$$sust(g, \{O_1, \ldots, O_n\}) = \{sust_1(g, O_i), O_i \in \{O_1, \ldots, O_n\}\}$$

- Operación de reetiquetado ($\mathcal{R}$):

$$\mathcal{R} : (G^{+})^{*} \times (G^{+})^{*} \times OS \rightarrow OS$$

Esta función nos reetiquetará todas las ofertas cuya puerta esté incluida en la primera lista por la puerta que ocupa igual posición en la segunda lista.

$$\mathcal{R}(< g_1, \ldots, g_n >, < g'_1, \ldots, g'_n >, off\_set) = \\ \bigcup_{i=1\ldots n}(d(g_i, off\_set)) \cup (\bigcup_{i=1\ldots n}(sust(g'_i, s(g_i, off\_set))))$$

- Operación de ocultación ($hide$):

$$hide : (G^{+})^{*} \times OS \rightarrow OS$$

Esta función es un caso especial de reetiquetado, usada para ocultar puertas y convertir un acción en interna (**i**).

$$hide(< g_1, \ldots, g_n >, off\_set) = \mathcal{R}(< g_1, \ldots, g_n >, < \mathbf{i}, \ldots, \mathbf{i} >, off\_set)$$

- Puertas formales de un proceso **Fg**:

$$\mathbf{Fg} : proc \rightarrow (G)^{*}$$

Esta función devuelve la lista de puertas formales del proceso que se toma como argumento.

$$\mathbf{Fg}(p) =< g_1, \ldots, g_n > \text{ si } \exists\, p[g_1, \ldots, g_n]$$

5.5.3 - Función de Identificación de Expresiones LOTOS IB

Definición 11 *Definimos la función IB :*

$$IB : beha, N, N \rightarrow beha$$

Esta función, descrita en la tabla 5.1, identifica cada subexpresión LOTOS de un comportamiento dado. Se aplica para cada uno de los procesos de una especificación de la forma:

$$\forall p \mid < p, B_p > \in D'_b \;\; B'_p = IB(B_p, p, 0)$$

donde B'_p es una expresión de comportamiento equivalente a B_p con todas sus subcomportamientos biunívocamente identificados. La identificación, por lo tanto, se hace en base a un par, el primero con la identificación del proceso y el segundo como la identificación de la subexpresión.

$B = \mathbf{stop}$	$IB(B, p, n) = \mathbf{stop}[p, n]$
$B = \mathbf{exit}$	$IB(B, p, n) = \mathbf{exit}[p, n]$
$B = \mathbf{exit}\ (E_1, \ldots, E_n)$	$IB(B, p, n) = \mathbf{exit}\ (E_1, \ldots, E_n)[p, n]$
$B = a; B_1$	$IB(B, p, n) = a[p, n]; IB(B_1, p, n + 1)$
$B = a\ d_1 \ldots d_n; B_1$	$IB(B, p, n) = ad_1 \ldots d_n[p, n]; IB(B_1, p, n + 1)$
$B = \mathbf{i}; B_1$	$IB(B, p, n) = \mathbf{i}[p, n]; IB(B_1, p, n + 1)$
$B = B_1\ op\ B_2$	$IB(B, n, n) = IB(B_1, p, n + 1)\ op[p, n]\ IB(B_2, p, last(B_1) + 1)$
con $op \in \{[], \|, \|[]\|, \|\|\|, >>, [>\}$	
$B = \mathbf{hide}\ [g_1, \ldots, g_n]\ \mathbf{in}\ B_1$	$IB(B, p, n) = \mathbf{hide}\ [g_1, \ldots, g_n]\ \mathbf{in}\ [p, n]\ IB(B_1, p, n + 1)$
$B = p[a_1, \ldots, a_n](V_1, \ldots, V_n)$	$IB(B, p, n) = p[a_1, \ldots, a_n](V_1, \ldots, V_n)[p, n]$
con $< p, B_p > \in\ BS$	
$B = \mathbf{let}\ vd_1, \ldots, vd_n\ \mathbf{in}\ B_1$	$IB(B, p, n) = \mathbf{let}\ vd_1, \ldots, vd_n\ \mathbf{in}\ [p, n]\ IB(B_1, p, n + 1)$
$B = [expr] \rightarrow\ B_1$	$IB(B, p, n) = [expr] \rightarrow\ [p, n]\ IB(B_1, p, n + 1)$

Tabla 5.1: Identificación de Subexpresiones LOTOS por Numeración

En la tabla 5.1 se utiliza la función $last$, necesaria para obtener la última asignación de identificadores en una subexpresión de comportamiento, de tal forma que no se asigne el mismo a dos eventos diferentes. Esta función será:

Definición 12 *Definimos la función $last$:*

$$last : beha \rightarrow N$$

que dada una expresión de comportamiento con identificación de eventos simples devuelve el último identificador asignado.

5.5.4 - Función de extracción de ofertas Of

Definición 13 *En esta sección definimos la función Of para la extracción de ofertas a partir de una expresión de comportamiento LOTOS.*

En aras de una mayor claridad, dividiremos en dos partes la definición de esta función. Por un lado, la tabla 5.2 contiene la extracción de ofertas a partir de comportamientos atómicos. Por otro, la tabla 5.3 contiene la obtención de ofertas en expresiones de comportamiento compuestas.

Para simplificar la presentación, seguiremos con el mismo convenio de la sección 4.3, donde se utilizaba la notación B_1 en lugar de $\mathbf{Id}(B_1)$. Aquí se utilizará en lugar del $IB(B_1)$.

$B = a; B_1$	$Of(B) = \{a^{\{B\}}_{<>}\}$
$B = a\ d_1 \ldots d_n; B_1$	$Of(B) = \{a^{\{B\}}_{<srt(d_1),\ldots,srt(d_n)>}\}$
$B = \mathbf{i}; B_1$	$Of(B) = \{\mathbf{i}^{\{B\}}_{<>}\}$
$B = \mathbf{stop}$	$Of(B) = \{O\_NIL\}$
$B = \mathbf{exit}$	$Of(B) = \{\delta^{\{B\}}_{<>}\}$
$B = \mathbf{exit}\ (E_1, \ldots, E_n)$	$Of(B) = \{\delta^{\{B\}}_{<srt(E_1),\ldots,srt(E_n)>}\}$

Tabla 5.2: Obtención de Ofertas Simples

Para el tratamiento de los operadores de sincronización, haremos uso de las funciones s y d que nos seleccionan o eliminan ofertas según una lista de puertas. Evidentemente, esta lista de puertas será la lista de sincronización del operador. No hay que olvidarse que incluso el operador de entrelazado (|||) sincroniza los comportamientos de las dos ramas en la puerta δ. El caso del operador de sincronización total (||) es mas delicado: los comportamientos se sincronizan en todas las puertas accesibles en el ámbito en que aparece, además de la puerta δ.

5.5.5 - Función de activación de ofertas gl

En esta sección definimos la función que nos permite avanzar una especificación desde un estado al siguiente según alguna de las ofertas ofrecidas.

Definición 14 *Definimos la función gl :*

$B = B_1\ op\ B_2$ donde $op \in \{[], >>, [>\}$	$Of(B) = Of(B_1) \cup Of(B_2)$
$B = B_1 \ ||| \ B_2$	$d(\delta, Of(B_1)) \oplus d(\delta, Of(B_2)) \oplus$ $(s(\delta, Of(B_1)) \otimes s(\delta, Of(B_2)))$
$B = B_1 \ [[g_1, \ldots, g_n]] \ B_2$	$(\bigcup_{i=1,\ldots,n}(d(g_i, Of(B_1)))) \oplus (\bigcup_{i=1,\ldots,n}(d(g_i, Of(B_2)))) \oplus$ $((\bigcup_{i=1,\ldots,n} s(g_i, Of(B_1))) \otimes (\bigcup_{i=1,\ldots,n} s(g_i, Of(B_2))))$
$B = B_1 \ \| \ B_2$	$(s(AG, Of(B_1)) \otimes s(AG, Of(B_2)))$ donde $AG = gates(B_1) \cup gates(B_2)$
$B = \mathbf{hide}\ [g_1, \ldots, g_n]\ \mathbf{in}\ B_1$	$Of(B) = hide(< g_1, \ldots, g_n >, Of(B_1))$
$B = \mathbf{let}\ vd_1, \ldots, vd_n\ \mathbf{in}\ B_1$	$Of(B) = Of(B_1)$
$B = [expr] \rightarrow\ B_1$	$Of(B) = Of(B_1)$
$B = p[a_1, \ldots, a_n](V_1, \ldots, V_n)$ donde $< p, B_p > \in\ BS$	$Of(B) = \mathcal{R}(< a_1, \ldots, a_n >, \mathbf{Fg}(B_p), Of(B_p))$

Tabla 5.3: Obtención de Ofertas en Comportamientos Compuestos

$$gl : O \times beha \rightarrow beha$$

Esta función proporciona la expresión de comportamiento LOTOS que representa al sistema tras la observación de la transición seleccionada.

Por lo tanto, esta función aplica según:

$$B' = gl(o, B) \text{ si } o \in Of(B)$$

es decir, sólo la definimos aplicable a una expresión de comportamiento y a una oferta extraída de dicha expresión.

Sea la oferta $O = g_S^I$ sobre una expresión de comportamiento B tal que $O \in B$. Con ello, definimos a continuación la imagen de la función gl:

- Inacción:

$$gl(O, \mathbf{stop}) = \mathbf{stop}$$

- Terminación:
  Sea la expresión $B = \mathbf{exit}\ E_1, \ldots, E_n$:
  - Si $B \in As(O)$ entonces :
    $$gl(O, B) = \mathbf{stop}$$
  - Si $B \notin As(O)$ entonces :
    $$gl(O, B) = B$$
- prefijo de acción:
  Para la expresión $B = g\ d_1, \ldots, d_n; B_1$:
  - Si $B \in As(O)$ entonces :
    $$gl(O, B) = B_1$$
  - Si $B \notin As(O)$ entonces :
    $$gl(O, B) = B$$

  Para la expresión $B = \mathbf{i}; B_1$:
  - Si $B \in As(O)$ entonces :
    $$gl(O, B) = B_1$$
  - Si $B \notin As(O)$ entonces :
    $$gl(O, B) = B$$
- Operador de selección:
  Sea $B = B_1 [] B_2$:
  - Si $\exists A \in Act(B_1) \mid A \in As(O)$ entonces :
    $$gl(O, B) = gl(O, B_1)$$
  - Si $\exists A \in Act(B_2) \mid A \in As(O)$ entonces :
    $$gl(O, B) = gl(O, B_2)$$
  - Si $\nexists A \in Act(B_1) \cup Act(B_2) \mid A \in As(O)$ entonces :
    $$gl(O, B) = gl(O, gl(O, B_1), gl(O, B_2))$$

- Operadores de composición:
  Sea $B = B_1\ op\ B_2$, donde $op \in \{||, |[g_1, \ldots, g_n]|, |||\}$:

$$gl(O, B) = gl(O, B_1)\ op\ gl(O, B_2)$$

- Operador de ocultación:
  Sea $B = \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ B_1$:

$$gl(O, B) = \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ gl(O, B_1)$$

- Operador de habilitación:
  Sea $B = B_1 >> B_2$:
  - Si $O \in ext(O')$ tal que $O' \in Of(B_1)$ y $gate(O') = \delta$ entonces :

$$gl(O, B) = B_2$$

  - Si $O \notin ext(O'), O' \in Of(B_1)$, $gate(O') = \delta$ entonces :

$$gl(O, B) = gl(O, B_1) >> B_2$$

- Operador de deshabilitación:
  Sea $B = B_1[> B_2$:
  - Si $\exists A \in Act(B_1) \mid A \in As(O)$ entonces :

$$gl(O, B) = gl(O, B_1)[> B_2$$

  - Si $\exists A \in Act(B_1) \mid A \in As(O)$ entonces :

$$gl(O, B) = gl(O, B_2)$$

  - Si $\nexists A \in Act(B_1) \cup Act(B_2) \mid A \in As(O)$ entonces :

$$gl(O, B) = B_1[> B_2$$

- Operador de reetiquetado:
  Sea $B = [g_1/h_1, \ldots, g_n/h_n]B_1$ (expresión no existente en LOTOS, sino generada por la instanciación de un proceso):

$$gl(O, B) = [g_1/h_1, \ldots, g_n/h_n]gl(O, B_1)$$

5.5.6 - Función de Extracción de PSC RC

Los PSCs, como hemos visto, no son más que un caso particular de expresiones de comportamiento LOTOS en las que sólo se tiene en cuenta la estructura o composición de comportamientos, sin importar las ofertas reales de los mismos. En definitiva, son expresiones de comportamientos en las que aparece toda la riqueza composicional de LOTOS pero se han eliminado las ofertas particulares de los comportamientos.

Definición 15 *Definimos la función* RC

$$RC : beha \rightarrow beha$$

puesto que la única diferencia respecto de LOTOS es la desaparición de expresiones finales, la definición de esta función es como sigue:

- $[], |||, |[g_1, \ldots, g_n]|, ||, >>, [>$ quedan sin modificación.
- $B = \mathbf{stop}$ queda como "B".
- $B = \mathbf{exit}\ E_1, \ldots, E_n$ queda como "B".
- $B = g\ d_1, \ldots, g_n$ queda como "B".
- el resto de los operadores (**hide**, **let**, ...) son eliminados, puesto que en lo que estamos interesados es en las sincronizaciones de expresiones, no las modificaciones sobre ellas (**hide**, reetiquetado de puertas) o en los valores que se manejaran (**let**, guardas,...).

5.5.7 - Proyección del Árbol de Sincronización

Los operadores LOTOS presentan una gran riqueza composicional, necesaria para especificar las diferentes características temporales de protocolos. Esto se traduce en un alto grado de complejidad a la hora de manejar dichas expresiones de comportamiento y a la hora de determinar la equivalencia entre ellas.

Sin embargo, podemos simplificar esta riqueza composicional en base a una proyección de las expresiones de comportamientos existentes en la especificación en otras más sencillas, que reflejen la arquitectura de sincronización en un estado dado de la especificación para una puerta dada. El hecho de utilizar cada puerta como criterio de proyección es debido a que deseamos analizar las posibles combinaciones *distintas*, precisamente, para cada puerta.

Para esta proyección necesitamos representar tanto la sincronización como la unión de los diferentes expresiones participantes, para lo cual nos basaremos en la siguiente:

Definición 16 *Definimos los operadores* $< + >$ *y* $< * >_g$, *con* $g \in G^+$ *como:*

$$< + >: beha \times beha \rightarrow beha$$

$$< * >_g: beha \times beha \times G^+ \rightarrow beha$$

La funcionalidad de estos operadores queda definida en base a la siguiente proyección:

Definición 17 *Definimos la proyección:*

$$T : beha \times G^+ \rightarrow beha'$$

donde $beha'$ es el dominio de expresiones de comportamiento usando los operadores $< + >$ y $< * >_g$.

La transformación será $T(B, g)$:

- $B = B_1[]B_2, B_1[> B_2 \ \rightarrow \ B_1 < + > B_2$
- $B = B_1|||B_2 \ \rightarrow \begin{cases} B_1 < + > B_2 & \text{si } g \neq \delta \\ B_1 < * >_g B_2 & \text{si } g = \delta \end{cases}$
- $B = B_1|[g_1, \ldots, g_n]|B_2 \ \rightarrow \begin{cases} B_1 < + > B_2 & \text{si } g \notin \{g_1, \ldots, g_n\} \\ B_1 < * >_g B_2 & \text{si } g \in \{g_1, \ldots, g_n\} \end{cases}$
- $B = B_1||B_2 \ \rightarrow \ B_1 < * >_g B_2$
- $B = B_1 >> B_2 \ \rightarrow \ B_1$
- el resto de los operadores (**stop**, **exit**, **;**, **let**,. . .) son eliminados, puesto que en lo que estamos interesados es en las sincronizaciones de expresiones, no lo que ofrecen cada una de ellas (**stop**, **exit**, **;**) ni en los valores que se manejaran (**let**, guardas,. . .).

En definitiva, el operador $< + >$ representa la unión de subexpresiones de comportamiento, mientras que el operador $< * >_g$ representa la sincronización de subexpresiones de comportamiento en una puerta dada g.

Como el operador $< * >_g$ aplica sobre una sola puerta, lo que obtendremos será una instanciación de la expresión de comportamiento LOTOS que refleja la composición sobre la puerta g. En definitiva, obtenemos una serie de instantáneas de las expresiones de comportamiento sesgadas por una puerta en concreto. De esta forma, veremos en la siguiente sección el establecimiento de la igualdad de PSCs en base a la igualdad de todas las instantáneas obtenidas sesgando las expresiones para todas las puertas existentes en la especificación.

En la figura 5.1 se presenta un ejemplo simple de transformación de una expresión de comportamiento LOTOS.

Estos dos operadores tienen las siguientes propiedades:

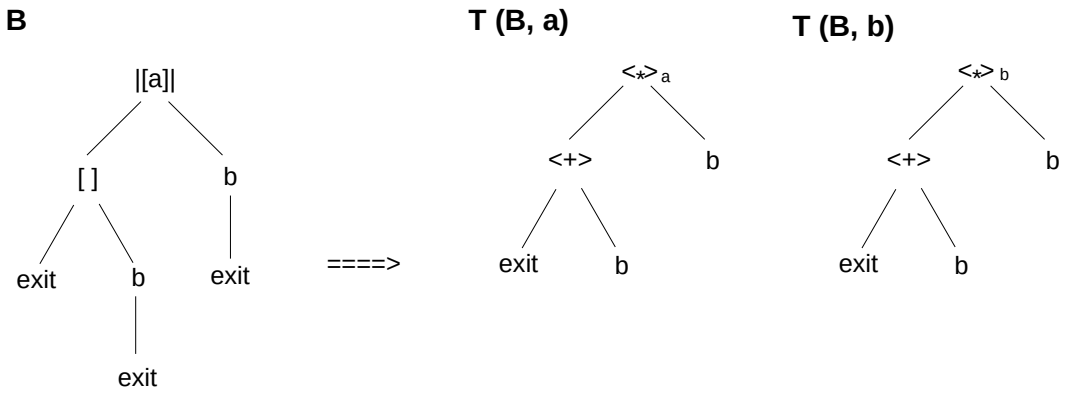


Figura 5.1: Transformación de Árboles

- Asociatividad:

$$(A <*>_g B) <*>_g C = A <*>_g (B <*>_g C)$$

$$(A <+> B) <+> C = A <+> (B <+> C)$$

- Conmutatividad:

$$A <+> B = B <+> A$$

$$A <*>_g B = B <*>_g A$$

- Distributividad:

$$A <*>_g (B <+> C)) = (A <*>_g B) <+> (A <*>_g C)$$

- Extracción de ofertas:

$$\begin{array}{ll} Of(B1 <+> B2) = & Of(B1) \cup Of(B2) \\ Of(B1 <*>_g B2) = & d(g, Of(B_1)) \cup d(g, Of(B_2)) \cup \\ & (s(g, Of(B_1)) \otimes s(g, Of(B_2))) \end{array}$$

- Absorción respecto de Of:

$$Of(A <+> A) = Of(A)$$

$$Of(A <*>_g A) = Of(A)$$

Estas dos últimas propiedades son de crucial importancia en nuestro análisis, puesto que sobre ellas se basa la identificación de PSC incluyendo la absorción de repetición de los mismos eventos en una oferta dada.

5.5.8 - Definición de Igualdad de PSCs

Definición 18 *Definimos la igualdad de PSCs como*

$$p1 = p2 \quad \Leftrightarrow \quad \forall g \in G^{+}\ T(p1, g) = T(p2, g)$$

5.5.9 - Función de Cálculo de Combinaciones

Con todos estos elementos, podemos definir el algoritmo orientado al cálculo de todas las combinaciones posibles en una especificación LOTOS. El algoritmo es:

1. Identificamos las expresiones de comportamiento de la especificación: $B'_0 = IB(B_0)$.
2. Inicializamos el conjunto de patrones de sincronización y el de ofertas a vació: $PSCS = \emptyset,\ OS = \emptyset$.
3. Hallamos las ofertas de la expresión de comportamiento: $OSa = Of(B'_0)$.
4. Si $OSa = \emptyset$ termina el proceso. Si no, añadimos las ofertas al conjunto OS ($OS = OS \cup OSa$) y pasamos al siguiente punto.
5. Hallamos el patrón de sincronización: $PSCa = RC(B'_0)$.
6. Si $PSCa \in PSCS$ termina el proceso. Si no, añadimos el patrón reconocido al conjunto $PSCS$ ($PSCS = PSCS \cup \{PSCa\}$) y pasamos al siguiente punto.
7. Para cada acción de OSa determinamos el comportamiento resultante de avanzar la especificación según cada oferta y volvemos al punto 3.

5.5.10 - Terminación y Completitud del Algoritmo

El algoritmo descrito es una de las posibles implementaciones de la función de cálculo de combinaciones de una expresión de comportamiento LOTOS. En esta sección vamos a demostrar dos características fundamentales para considerarlo como tal algoritmo: la terminación del mismo ante cualquier especificación y su completitud. Entendemos por completitud que el algoritmo calcule todas las combinaciones de la especificación.

Finitud Tanto la terminación como la completitud del algoritmo están directamente relacionada con el hecho de contar con un número finito de PSCs en una especificación.

Para una especificación dada, existe un número finito de subexpresiones de acción, componentes básicos de los PSCs. Por lo tanto, dado que el número de operadores LOTOS es finito, se pueden combinar dinámicamente, sin tener en cuenta repeticiones, de una forma finita (y contable).

En el resto de la sección, llamaremos N al cardinal del conjunto de PSC existentes en una especificación.

Terminación Cada vez que el algoritmo encuentra un nuevo comportamiento (denominado en el algoritmo $PSCa$) pueden darse tres casos:

- que PSCa sea **stop**. En este caso el algoritmo termina por el paso 4, ya que $OSa = \emptyset$, es decir, no se obtienen nuevas combinaciones.
- que PSCa pertenezca a alguno de los ya reconocidos. En este caso, el paso 6 del algoritmo nos asegura su parada, puesto que $PSCa \in PSC$ (no obtendremos, por tanto, nuevas combinaciones).
- que el PSC hallado sea nuevo y diferente a **stop**. En este caso, el algoritmo continuará la exploración, pero el PSC pasará a formar parte de los Patrones reconocidos, por el paso 6 ($PSCS = PSCS \cup \{PSCa\}$). Esto significa que hemos incrementado card(PSCS), acercándose a N.

Por lo tanto, en cada ejecución del algoritmo o bien termina o bien incrementa el cardinal de PSCS. Como este número tiene una cota máxima, N, el algoritmo termina.

Completitud Al terminar el cómputo del algoritmo, el conjunto OS debe contener todas las posibles combinaciones. O lo que es lo mismo, el conjunto $PSCS$ debe contener todas los PSCs de las especificación. En efecto, por los pasos 3 y 4 se asegura que para cada PSC obtenido incluimos todas las combinaciones que dicho PSC genera en el conjunto OS.

Con la excepción del PSC inicial, tenemos que cualquier PSC de la especificación será generado a partir de un PSC previo evolucionado según una de las ofertas de éste último. Esto sugiere la posibilidad de una demostración por inducción. Para ello necesitaremos un *paso inicial* y un *paso inductivo*.

El paso inicial será considerar el PSC inicial de la especificación. El paso inductivo consistirá en demostrar que un PSC_{n+1} es computado (es decir, sus ofertas serán incluidas en OS) si el PSC_n del que proviene ha sido computado.

- Paso inicial:

  A lo largo de la exposición se ha identificado como B_0 al PSC inicial de una especificación. El paso 2 del algoritmo inicializa $PSCS$ y OS a vacío. En una especificación dada existen dos posibilidades:

  - El conjunto inicial de ofertas (paso 3) es vacío. Esto quiere decir que la especificación contiene un comportamiento equivalente a **stop**, esto es, un *deadlock* o bloqueo. Por lo tanto, el sistema especificado es el más trivial de todos: aquel que no hace nada. En cualquier caso, no hay combinaciones que calcular, ergo el algoritmo calcula *todas*, es decir, ninguna[4].

[4]En mi opinión, el detalle no merece tantas palabras.

– El conjunto inicial de ofertas no es vacío. Por lo tanto, las combinaciones ofertadas serán incluidas en OS por el paso 5. Además, como $PSCS$ es vacío, claramente $PSC_0 \notin PSCS$, luego el paso 6 nos asegura el cómputo del comportamiento inicial.

- Paso inductivo:

  La hipótesis inductiva es que para cualquier PSC_{n+1} que proviene, mediante la evolución según la oferta o_{n+1} de un patrón PSC_n que ya ha sido computado (y, por tanto, pertenece a $PSCS$). Expresado matemáticamente:

$$\exists\ PSC_{n+1} \notin PSCS \mid \exists PSC_n \in PSCS\ \ PSC_n - o_{n+1} \rightarrow PSC_{n+1}$$

  Si $PSC_n \in PSCS$, entonces el algoritmo, por el paso 3 calcula todas las combinaciones ofertadas por PSC_n. Evidentemente, $OSa \neq \emptyset$ (puesto que existe, al menos, la oferta $o_{n+1} \in Of(PSC_n)$ por la hipótesis inductiva). El paso 4 del algoritmo añade dicha combinación en OS. Además, se calcula el PSC resultante (punto 5) que, según la hipótesis de inducción, es precisamente PSC_{n+1}. Como $PSC_{n+1} \notin PSCS$, por el paso 6 se añade dicho Patrón a $PSCS$.

q.e.d.

5.5.11 - Definición en PASCAL del algoritmo

La descripción textual del algoritmo puede ser especificada usando un lenguaje más cercano a una máquina orientada a computarlo, con la ventaja de que la interpretación queda unívocamente determinada. Hemos elegido PASCAL por ser un lenguaje claro, sencillo y ampliamente conocido:

```
PROGRAM cober;

TYPE
    TOfferts = RECORD
                    name : String;     (* Nombre externo de la oferta *)
                    sl : SortList;     (* Lista de sorts *)
                    partners : IdList; (* Lista de Participantes *)
               END;
    SetOfferts = SET OF TOfferts;
    SetPatterns = SET OF Pattern;

VAR
    B : Behaviour;
    PSCS : SetPatterns;
```

```
PROCEDURE GetCombi (B: Behaviour; VAR PSCS: SetPatterns; VAR OS: SetOfferts);
VAR
    off: TOfferts;
    OSa: SetOfferts;
    PSCa: Pattern;
BEGIN
    (* Paso 3 *)
    OSa := GetSpecOffers (B);
    (* Paso 4 *)
    IF OSa = SetOfferts {} THEN          (* OSa es vacio *)
        RETURN
    END;
    OS := UNION (OS, OSa);          (* Añadimos las nuevas combinaciones *)
    (* Paso 5 *)
    PSCa := RC (B);
    (* Paso 6 *)
    IF PSCa IN PSC THEN
        RETURN
    END;
    PSC := INCL (PSC, PSCa);
    (* Paso 7 *)
    WHILE (OSa <> SetOfferts {}) DO
        off := GetOneOffert (OSa);
        EXCL (off, OSa);

        GetCombi (NC (B, off), PSC, OS);
    END;
END GetCombi;

BEGIN
    (* Paso 1 *)
    B := IB (B);          (* Identificacion de expresiones de accion *)
    (* Paso 2 *)
    PSC := SetPatterns {};
    OS := SetOfferts {};
    GetCombi (B, PSC, OS);
END cober.
```

5.5.12 - Ejemplos de Aplicación

En esta sección veremos unos ejemplos simples de aplicación del algoritmo. En los apéndices C.1 y C.2 presentamos unos ejemplos más complejos: la Criba de Eratóstenes, escogida por su singular componibilidad y el protocolo Abracadabra elegido por represen-

tar un sistema de complejidad real. A continuación revisaremos los ejemplos del principio del capítulo.

Ejemplo 1 Repetimos a continuación el ejemplo de especificación:

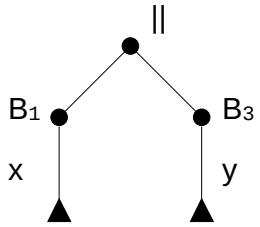
$$\begin{array}{rcl} S & = & P[a]||Q[a] \\ P[x] & = & x;x;P[x] \\ Q[y] & = & y;Q[y] \end{array}$$

Aplicamos el algoritmo paso a paso:

- **1** Identificamos subexpresiones:

$$B_0 = B_1||B_3 \quad B_1 = x;B_2 \quad B_2 = x;B_1 \quad B_3 = y;B_3$$

- **2** Inicializamos: $PSC = \emptyset,\ OS = \emptyset$. El árbol de sincronización es:



- **3a** Hallamos las ofertas:

$$OSa = \{a^{\{B_1,B_3\}}_{<>}\}$$

- **4a** Como $OSa \neq \emptyset$:

$$OS = OS \ \cup \ OSa = \{a^{\{B_1,B_3\}}_{<>}\}$$

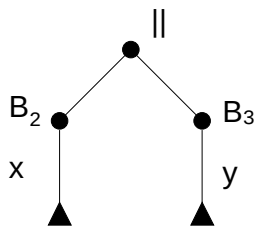
- **5a** Hallamos el patrón de sincronización:

$$PSCa = (B_1||B_2)$$

- **6a** Como $PSCa \notin PSC$:

$$PSC = PSC \ \cup \ \{PSCa\} = \{(B_1||B_3)\}$$

- **7a** Como sólo existe una acción observable, avanzamos la especificación, resultando:



- **3b** Volvemos a calcular ofertas:

$$OSa = \{a_{<>}^{\{B_2,B_3\}}\}$$

- **4b** De nuevo tenemos que $OSa \neq \emptyset$, por lo que añadimos a las ya calculadas:

$$OS = OS \cup OSa = \{a_{<>}^{\{B_1,B_3\}}, a_{<>}^{\{B_2,B_3\}}\}$$

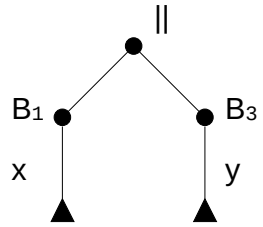
- **5b** Calculamos también el patrón de sincronización:

$$PSCa = (B_2||B_3)$$

- **6b** De nuevo tenemos que $PSCa \notin PSC$ por lo que añadimos a los ya calculados:

$$PSC = PSC \cup \{PSCa\} = \{(B_1||B_3), (B_2||B_3)\}$$

- **7b** Y para la única acción observable, el árbol de sincronización resultante de avanzar las especificación es:



- **3c** El conjunto OSa resulta:

$$OSa = \{a_{<>}^{\{B_1,B_3\}}\}$$

- **4c** Que se añade de nuevo:

$$OS = OS \cup OSa = \{a_{<>}^{\{B_1,B_3\}}, a_{<>}^{\{B_2,B_3\}}\}$$

  sin que OS resulte modificado.

- **5c** Por otro lado, el patrón de sincronización $PSCa$ es:

$$PSCa = (B_1||B_2)$$

- **6c** Como $PSCa \in PSC$ termina el proceso.

Por lo tanto, el cardinal del conjunto de Patrones de Sincronización de la especificación es $Card(OS(S)) = 2$, que nos delimita el límite del 100% de cobertura.

De esta forma obtenemos dos pruebas, que nos permiten asignar las siguientes medidas de cobertura *a priori*:

prueba	comportamiento	cobertura $\mathcal{M}(T)$
T_1	a; **stop**	50%
T_2	a; a; **stop**	100%

Ejemplo 2 Una variante del ejemplo 3 presentado al principio del capítulo es:

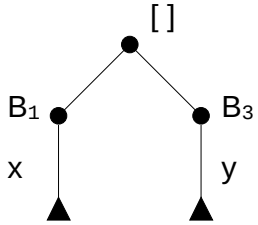
$$\begin{array}{rcl} S & = & P[a][]Q[b] \\ P[x] & = & x; (P[x]||Q[x]) \\ Q[y] & = & y; Q[y] \end{array}$$

Aplicamos el algoritmo paso a paso:

- **1** Identificamos subexpresiones:

$$B_0 = B_1[]B_3 \;\; B_1 = x; B_2 \;\; B_2 = B_1||B_3 \;\; B_3 = y; B_3$$

- **2** Inicializamos: $PSC = \emptyset,\ OS = \emptyset$. El árbol de sincronización es:



- **3a** Hallamos ofertas:

$$OSa = \{a^{\{B_1\}}_{<>}, b^{\{B_3\}}_{<>}\}$$

- **4a** Como $OSa \neq \emptyset$:

$$OS = OS \cup OSa = \{a^{\{B_1\}}_{<>}, b^{\{B_3\}}_{<>}\}$$

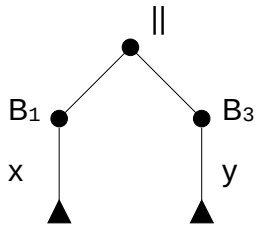
- **5a** Hallamos el patrón de sincronización:

$$PSCa = (B_1[]B_2)$$

- **6a** Como $PSCa \notin PSCS$:

$$PSCS = PSCS \cup \{PSCa\} = \{(B_1[]B_2)\}$$

- **7a1** En este caso, existen dos acciones observables. Por lo tanto, hay que procesar según el avance de cada una de las acciones. Empezando con la primera (a):



- **3b1** Volvemos a calcular ofertas:

$$OSa = \{a^{\{B_1,B_3\}}_{<>}\}$$

- **4b1** De nuevo tenemos que $OSa \neq \emptyset$, por lo que añadimos a lo ya calculado:

$$OS = OS \cup OSa = \{a^{\{B_1\}}_{<>}, b^{\{B_3\}}_{<>}, a^{\{B_1,B_3\}}_{<>}\}$$

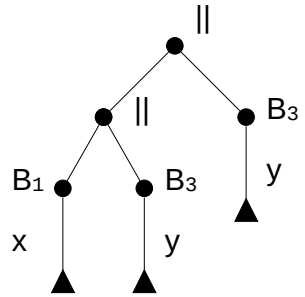
- **5b1** Volvemos a calcular el patrón resultante:

$$PSCa = (B_1||B_3)$$

- **6b1** Nuevamente tenemos que $PSCa \notin PSCS$:

$$PSCS = PSCS \cup \{PSCa\} = \{(B_1[]B_2), (B_1||B_3)\}$$

- **7b1** Y para la única acción observable de OSa, el árbol de sincronización resultante de avanzar la especificación es:



- **3c** El conjunto OSa resulta:

$$OSa = \{b^{\{B_1,B_3\}}_{<>}\}$$

- **4c** De nuevo tenemos que $OSa \neq \emptyset$, por lo que añadimos a lo ya calculado:

$$OS = OS \cup OSa = \{a^{\{B_1\}}_{<>}, b^{\{B_3\}}_{<>}, a^{\{B_1,B_3\}}_{<>}\}$$

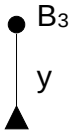
  Es decir, OS no se ve modificado.

- **5c** Y el patrón de sincronización es:

$$PSCa = (B_1||B_3||B_3) = (B_1||B_3)$$

- **6c** Como $PSCa \in PSCS$ termina el proceso por esta rama. Sin embargo, todavía quedaba una acción de la primera fase.

- **7a2** La segunda acción al finalizar el algoritmo la primera vez era b, cuyo avance produce:



- **3b2** Las ofertas son:

$$OSa = \{b^{\{B_3\}}_{<>}\}$$

- **4b2** Esta oferta no modifica el conjunto OS pues ya está incluida.

- **5b2** El patrón de sincronización resultante es:

$$PSCa = (B_3)$$

- **6b2** Tampoco PSC se ve modificado.

- **7b2** Es trivial comprobar que en la siguiente iteración se produce la detección de la inclusión de B_3 en PSC, terminando el proceso.

Por lo tanto, el cardinal del conjunto de Patrones de Sincronización de la especificación es $Card(OS(S)) = 3$, que nos delimita el límite del 100% de cobertura.

De esta forma obtenemos tres pruebas, que nos permiten asignar las siguientes medidas de cobertura *a priori*:

prueba	comportamiento	cobertura $\mathcal{M}(T)$
T_1	a; **stop**	33.33%
T_2	a; a; **stop**	66.66%
T_3	b; **stop**	33.33%

Capítulo 6

Arquitectura de un Sistema de Ejecución de Pruebas

La utilización de Técnicas de Descripción Formal constituye la base para la formalización, y subsiguiente automatización, del proceso de desarrollo de protocolos en general, y de la fase de pruebas en particular. Integrando este soporte formal con la recomendaciones de ISO para pruebas de conformidad entre especificaciones e implementaciones se presenta un sistema de ejecución de pruebas basado en LOTOS. El sistema se caracteriza por utilizar la especificación formal de referencia durante la fase de pruebas, proporcionando los resultados en base a aquella: veredictos, identificación de errores, y análisis de cobertura.

6.1 - Introducción

Las Técnicas de Descripción Formal (FDTs para abreviar) adquieren cada vez una mayor importancia como soporte para el desarrollo de protocolos de comunicación (figura 6.1). Como parte de este proceso de desarrollo, la fase de pruebas se ofrece como una de las más firmes candidatos a la formalización. Concretamente nos centraremos en el lenguaje LOTOS [Bri85], que es norma internacional [ISO89b]. De entre los múltiples aspectos de la pruebas de protocolos, vamos a concentrarnos en las pruebas de conformidad [ISO91a] para homologar implementaciones respecto de sus especificaciones formales.

Las pruebas de conformidad u homologación consisten en comprobar que la implementación de un sistema es acorde con una especificación de referencia. Usualmente se trata de sistemas desarrollados bajo el marco de una norma internacional, y su homologación es requisito previo para que sean aceptados en el mercado. La homologación persigue el objetivo de poder garantizar que un producto es capaz de operar con otros. Aunque la garantía ofrecida por un certificado de homologación no es absoluta, las posibilidades de que un producto homologado supere las pruebas de interoperabilidad son muy altas [Lin86]. Las pruebas de homologación se convierten en una condición necesaria, pero no suficiente. En cualquier caso, la legislación vigente impone la homologación

como requisito previo necesario. Las compañías proveedoras de servicios de transporte declinan cualquier responsabilidad acerca de productos no homologados. Así, las pruebas de conformidad adquieren un papel relevante en el marco de desarrollo de productos telemáticos.

Conformidad Dícese que una implementación I es conforme a una especificación S si y solo si (1) para cualquier comportamiento impuesto por la especificación S, la implementación I es capaz de llevarlo a cabo, y (2) cualquier comportamiento que la implementación se niega a realizar, esta previsto en la norma S que pueda ser rechazado. Esta definición conlleva varios aspectos:

1. las pruebas de conformidad sólo tienen en cuenta comportamientos previstos en la especificación,
2. la especificación sólo impone unos *mínimos* de obligado cumplimiento,
3. la especificación puede contener aspectos opcionales: no es obligatorio que sean realizados en todas y cada una de las implementaciones; pero de ser contemplados, deben serlo en la forma especificada.
4. los fabricantes disponen de un margen de maniobra para enriquecer su producto con facilidades que lo hagan mas atractivo al cliente, sin que las pruebas de conformidad interfieran en estos aspectos.

Dadas estas características, es perfectamente posible que dos productos correctamente homologados sean incapaces de interoperar entre si. Usualmente se debe a que los fabricantes han tomado una serie de decisiones en los aspectos opcionales que impiden la coordinación de los equipos.

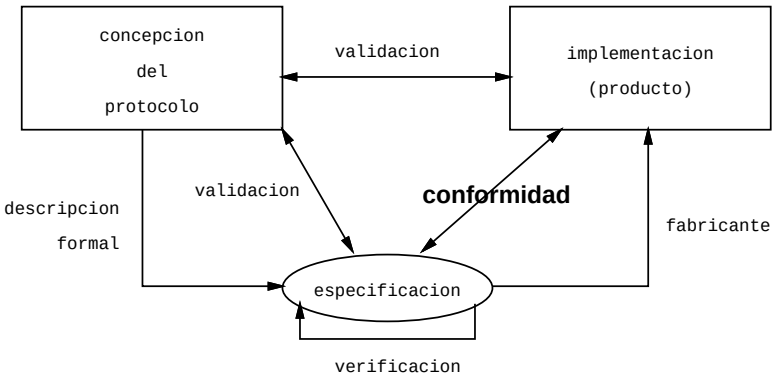


Figura 6.1: Ingeniería de Protocolos de Comunicaciones

Usualmente es imposible probar la conformidad de una implementación respecto de una especificación de referencia al 100%. El problema es planteable matemáticamente, pero indecidible en la práctica, incluso ante especificaciones descritas formalmente. Por ello se procede a una comprobación basada en pruebas. A partir de la especificación de

referencia se establecen una serie de pruebas que deben ser pasadas por el producto para su homologación. Salvo en ejemplos académicos, es imposible un análisis exhaustivo de casos, por los que las pruebas siempre terminan en la práctica antes de proporcionar una garantía absoluta de corrección. Es necesario pues disponer de alguna estimación cuantitativa de en qué medida podemos fiarnos de las pruebas realizadas, y en qué medida podemos pasar a usar (vender) el producto sin miedo a sorpresas. A estos efectos se utilizan medidas de cobertura, pasándose el mínimo número de pruebas que proporcionan una cobertura aceptable de los requisitos expresados en la especificación. Las técnicas formales proporcionan un sólido soporte de razonamiento para conseguir este objetivo.

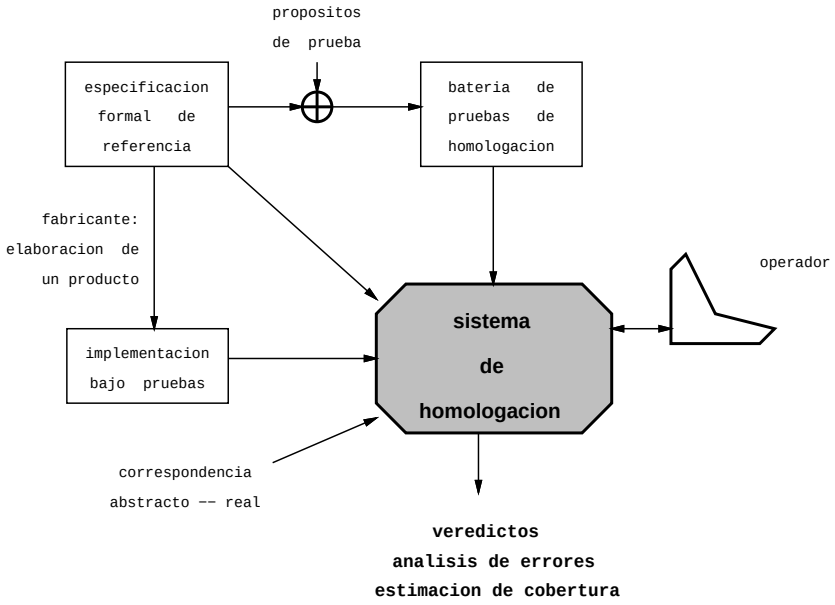


Figura 6.2: Escenario de Pruebas de Conformidad

La figura 6.2 muestra el escenario en el que nos movemos, una profundización en el marco global presentado en la figura 6.1, adaptada de [Lin89]. Se parte de una especificación de referencia descrita formalmente. Utilizando información de los propósitos de las pruebas que deseamos realizar, se obtiene una batería de casos de prueba. Estos propósitos se eligen con criterios funcionales y buscando la comprobación de aquellas facetas del producto más susceptibles de una errónea implementación, bien por su complejidad, bien porque la experiencia así lo ha constatado. Por otra parte, el fabricante ha desarrollado un producto a partir de su especificación. No nos interesa conocer el proceso de desarrollo, ni tan siquiera las interioridades del producto. Este se tratará como una caja negra que se somete a experimentación. El sistema de homologación toma en consideración todos estos elementos, así como información que le permita establecer una correspondencia entre los componentes formales y su realización práctica. Este sistema, bajo control de un operador, proporciona veredictos, análisis de cobertura e informa de los errores encontrados referidos a la especificación de referencia.

La especificación formal de referencia sirve exclusivamente al propósito de poder informar al fabricante de los errores hallados, expresados en términos de dicha especificación

que no se cumplen en su producto.

6.2 - Generación de Casos de Prueba

La obtención de la batería de pruebas es una fase previa orientada a organizar técnica y administrativamente la realización de las pruebas.

Los casos de prueba constan de un **preámbulo**, un **cuerpo** y un **postámbulo** [ISO91a]. El preámbulo es una fase preliminar en la que la implementación se lleva al estado en el que se quiere realizar la prueba. En pruebas muy sencillas, este preámbulo es nulo. El cuerpo es el objeto en sí de la prueba. Los preámbulos de pruebas complejas son el cuerpo de pruebas preliminares. Por último, el postámbulo es el conjunto de acciones que nos permiten llevar la implementación a un estado reconocible tras una prueba. A veces es tan simple como un botón de *reset*.

De acuerdo con la definición de conformidad, solamente trazas contempladas en la especificación serán probadas sobre el producto. La selección de casos de prueba se reduce a extraer trazas posibles de la especificación de referencia.

6.2.1 - Propósitos de Prueba

Las pruebas se organizan por propósitos. Estos consisten en identificar un estado inicial, al que llega gracias al preámbulo, y un objetivo que es el cuerpo de la prueba.

> Un propósito de prueba es *"una descripción en prosa de un objetivo de prueba definido con precisión, centrado en un único requisito de conformidad, de acuerdo con la especificación de la norma correspondiente"* [ISO91b]

La implementación es una caja negra sólo observable en términos de su comportamiento externo. En términos de LOTOS ésto quiere decir como eventos ocurridos en sus puertas externas. Un propósito de prueba se puede expresar como una secuencia de eventos LOTOS a observar en dichas puertas. Dos eventos son especialmente significativos: aquel que marca el final de preámbulo, y aquel que marca el éxito de la prueba. Identificados éstos, se puede extraer de la especificación formal la secuencia de eventos observables que proporcione un recorrido legal por los eventos que constituyen el propósito de la prueba. Esto se puede realizar por ejecución simbólica de la especificación de referencia [QFM87, vEE91], obteniéndose una traza o subconjunto de comportamiento que persigue los objetivos deseados [Rob91]. El caso de prueba debe guardar información de aquellos posibles caminos que el sistema puede abordar indeterminísticamente, pues impedirán conseguir el objetivo de la prueba, pero no podrán ser considerados como errores. La figura 6.4 muestra una especificación S de la que se han extraído 5 pruebas.

6.2.2 - Temporizadores

Hay que prever que la implementación falle y que durante las pruebas algún evento previsto en las trazas no se lleve a cabo. La decisión de que un evento no ocurre debe tomarse en base al vencimiento de un temporizador: "si no ocurre en T segundos, considérese que no va a ocurrir nunca". Esta información sobre tiempos de espera va estrechamente asociada a los propósitos de prueba, pudiéndose expresar bien en términos globales (idéntico criterio para todos los eventos) o particulares (usando temporizadores específicos para cada caso de prueba o, incluso, para algunos eventos).

6.2.3 - Selección de Datos de Prueba

La selección de los datos de prueba es uno de los puntos más difíciles en el diseño de las pruebas. No basta con disponer de un propósito y saber con detalle las trazas que lo comprueban, además hay que proporcionar datos concretos para llevar a cabo la ejecución. Es usualmente imposible probar todos los casos con todos los datos posibles, pues éstos suelen ser infinitos, o tan numerosos que hacen inviable en la práctica su ejercicio exhaustivo. Hay que limitarse a un subconjunto, que deberá ser mínimo, pero maximizar la probabilidad de encontrar errores en el producto. Para ello hay que partir el dominio de datos de entrada en un conjunto finito de *clases de equivalencia* de forma que pueda razonablemente suponerse (aunque nunca se sepa con certeza absoluta) que la prueba con un valor cualquiera de la clase sea equivalente a la prueba con cualquier otro valor de la misma. La identificación de estas clases de equivalencia es un proceso heurístico en el que se toma la especificación de referencia y se recorren las condiciones (guardas y predicados) de la misma. En base a su inspección, los datos quedan clasificados en clases según se cumpla o incumpla cada combinación de condiciones sobre ellos.

Una vez identificada una clase de equivalencia, se toma un elemento medio representativo de cada una de ellas, así como elementos límites, es decir valores que se hallan inmediatamente arriba o debajo de los márgenes de la clase. De esta forma, cada clase de equivalencia y cada frontera entre clases de equivalencia es probada.

6.2.4 - Clases de Pruebas

En la fase de generación de la batería de pruebas se hace un análisis exhaustivo de la especificación buscando trazas que recorran los objetivos de pruebas. De este análisis resultan los casos de prueba clasificados en tres clases:

obligatorios.- Una cierta traza de prueba es de obligado cumplimiento. Es el grupo ideal y más simple de tratar. U ocurre, o se rechaza el producto.

opcionales.- Una cierta traza de prueba es opcional, siendo decisión del fabricante implementarla o no. Resultan casos de prueba parametrizados por valores que sólo se conocerán al ir a pasar las pruebas. Estos valores los proporcionará el fabricante en

un documento conocido como PICS[1]. El operador introducirá los valores concretos del PICS para un cierto producto. Un cierto número de pruebas puede quedar anulado en base a estos valores. Típicamente, las pruebas asociadas a cierto parámetro del PICS suelen estar agrupadas.

indeterministas.- Es muy frecuente en protocolos de comunicaciones que un cierto objetivo no se pueda alcanzar, sin que ello implique un error en la implementación. Por ejemplo, una desconexión en un protocolo orientado a conexión, puede impedir que se observe la entrega de un dato; pero la desconexión iniciada por el medio es un comportamiento correcto contemplado en la norma[2].

Durante el proceso de elaboración de la batería de pruebas, combinando la especificación de referencia con los propósitos de prueba, es perfectamente detectable este tipo de posibles comportamientos. Así, es muy sencillo incluir información adicional en el caso de prueba de forma que si en vez del comportamiento deseado se observa otro comportamiento igualmente posible, se emita un veredicto de *inconcluso* [Rob91]. Sólo comportamientos inadecuados llevarán a un veredicto de error. En el ejemplo de la figura 6.4 se contemplan varios casos de pruebas obligatorias $(P1, P2, P3)$, y dos casos indeterministas $(P4, P5)$. Estas trazas expresan lo que debe o puede ocurrir. Si la implementación se sale de este marco, se dice que hay un error o fallo.

Sin embargo, debido a comportamientos no deterministas de la especificación, puede ocurrir que, al intentar ejecutar la prueba, el producto actúe de forma legal (según lo especificado) pero apartándose del propósito de la prueba. La prueba contempla estos indeterminismos dirigiendo la implementación a un estado estable pero, evidentemente, no se puede asegurar que el propósito ha sido cumplido. Por ello, se clasifican las pruebas dependiendo de si se puede asegurar el cumplimiento de su objetivo. Esta clasificación, ampliada de [dNH84] es:

- **MUST ACCEPT** La especificación es determinista en el comportamiento referente al objetivo de la prueba, por lo que la IUT debe aceptar, en todo momento, las ofertas de la misma.
- **MAY ACCEPT** La especificación admite comportamientos que se apartan del objetivo de la prueba, por lo que su terminación puede no asegurar el cumplimiento de su propósito.
- **MUST REJECT** Igual que MUST, pero la intención es que se rechace.
- **MAY REJECT** Igual que MAY, pero con la intención de que se rechace.

[1] Protocol Implementation Conformance Statement [ISO91a].

[2] Otro tipo de pruebas, por ejemplo las de prestaciones, pueden fallar si, por ejemplo, el número de desconexiones iniciadas por el medio es escandalosamente elevado, impidiendo la transferencia efectiva de información. Pero aquí nos estamos centrando en pruebas de conformidad, que no entran en aspectos de prestaciones.

6.3 - Otras Aproximaciones

Existen en la literatura una cantidad relativamente grande de artículos orientados a la generación automática de casos de prueba a partir de especificaciones formales en LOTOS. Suelen estar planteados con perfiles fuertemente teóricos, buscando más un tratamiento formal del tema [BAL+91] que una viabilidad práctica.

La linea más exitosa de estas investigaciones se basa en identificar un probador canónico [Bri87b]. Este es una "inversión" de la especificación basada en convertir los eventos decidibles por el entorno en decisiones del probador, y los eventos decidibles internamente por la especificación en decisiones de la implementación bajo prueba. Una fase posterior extraería casos concretos de prueba como trazas de este probador canónico [Tre90, WBL91]. Existen algoritmos capaces de encontrar el probador canónico [Ald90, Wez89, Wez88, Mor90] que funcionan perfectamente con especificaciones sin datos; pero que aún se ignora cómo conseguir que cubran LOTOS completo. Estos métodos tropiezan básicamente en dos puntos: (1) la infinitud de valores que pueden tomar las variables, y (2) los problemas usuales de ejecución simbólica que lleva rápidamente a problemas de resolución de teoremas. Tampoco está claro el tratamiento que reciben especificaciones que denotan autómatas infinitos. Por último, las funcionalidades del sistema hay que adivinarlas en el probador canónico, por lo que la extracción de trazas basada en propósitos de prueba y la explicación de fallos en términos de la referencia, son actividades que pueden requerir un serio esfuerzo de identificación.

No obstante, el marco teórico proporcionado por estos trabajos ha permitido clarificar los objetivos y el significado de lo que significan pruebas de conformidad, influyendo manifiestamente en la evolución de las normas internacionales [ISO91a]. Así mismo, las ideas expuestas en este artículo siguen dicho modelo teórico, si bien se ha optado por un enfoque pragmático a la hora de llevar la teoría a la práctica. Así, la extracción de trazas se realiza sobre la especificación de referencia, antes que sobre el probador canónico; y la inversión de la especificación sólo existe virtualmente durante al ejecución de las pruebas [Rob91], con datos reales, como se describe más abajo.

6.4 - Ejecución de las Pruebas

La ejecución de las pruebas consiste en la aplicación de nuestra batería de casos a un producto. Esta se realiza bajo control de un operador que introduce los parámetros necesarios de conformidad (PICS), así como los detalles de realización práctica de los eventos que se representaron en forma abstracta en la especificación de referencia (PIXIT)[3].

La figura 6.3 presenta la interrelación funcional de los componentes de prueba. El motor de la prueba es la especificación de cada caso de prueba, adecuadamente parametrizado por el PICS. El caso de prueba puede imponer un cierto comportamiento, u observar qué hace la implementación bajo pruebas. Un comportamiento se impone cuando sólo existe un posible evento en la prueba, mientras que situaciones indeterministas a decidir

[3]Protocol Implementation eXtra Information for Testing [ISO91a].

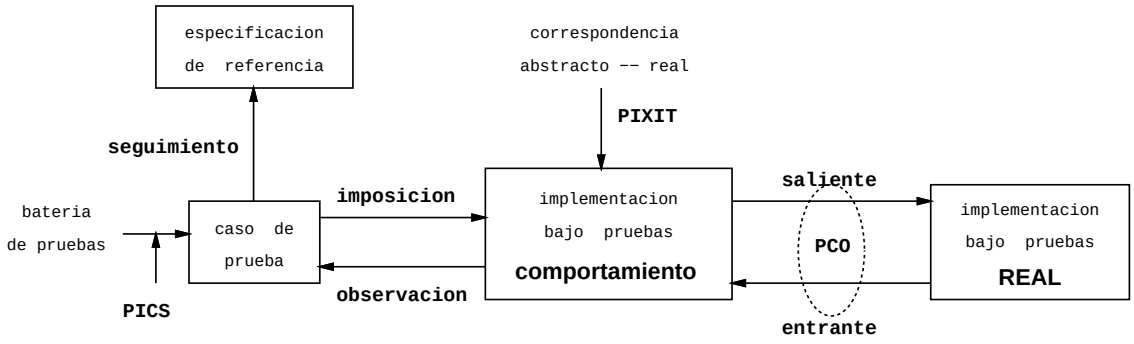


Figura 6.3: Organización funcional del probador

por el entorno se consideran observaciones pasivas del probador.

6.4.1 - Relación entre la formalización y su concreción

Todo el mecanismo de descripción formal se basa en abstraer detalles concretos de la realidad, darles un nombre simbólico, y manipular este mundo de símbolos. Todo esto es muy cómodo para realizar manipulaciones simbólicas sobre la especificación; pero tiene un límite que se alcanza cuando hay que interactuar con un producto.

La correlación entre el mundo abstracto y el concreto se realiza, en nuestro sistema, en base a un mecanismo de anotaciones [MdM89] que permite asociar aspectos externos a eventos simbólicos. Hay básicamente dos casos a contemplar: eventos que ocurren en el mundo porque la especificación así lo desea (salientes), y elementos que ocurren en el mundo y la especificación constata que han acaecido (entrantes). ISO ha propuesto un lenguaje de especificación de pruebas (TTCN [ISO91d]) que contempla esta clasificación explícitamente. Los eventos salientes se denotan por `!`, y los entrantes por `?`. Desgraciadamente, LOTOS abstrae este "detalle", lo que requiere que sea introducido posteriormente.

El mecanismo de rendezvous múltiple de LOTOS permite añadir una restricción al comportamiento en la que estos aspectos son explícitos [MdMSA93]. Los eventos se caracterizan por la puerta externa sobre la que ocurren y por el número y clase de sus argumentos. Los eventos salientes son anotados con un efecto colateral: una sentencia que se ejecuta asociada a la ocurrencia del evento. Los eventos entrantes se guardan con un predicado que indica cuándo ha entrado desde el entorno. La realización concreta de estas funciones relacionadas con el entorno viene dada por el documento PIXIT.

Esta restricción tiene la siguiente forma:

```
process entorno [puertas_externas] (PICS, PIXIT) :=
    evento_saliente (*| C efecto_externo (); |*) ;
      entorno [...] (...)
  []
    (*| wait observacion () |*) i ; evento_entrante ;
```

```
    entorno [...] (...)
[]
  ...
endproc
```

Pueden haber varios eventos salientes, caracterizados por una puerta y una serie de argumentos. Cuando el evento ocurre debido a que todas las condiciones impuestas por la especificación para su ocurrencia son satisfechas, el entorno "nota" que ha ocurrido al ejecutarse la función `efecto_externo ()`.

Cuando un evento externo ocurre, es notificado vía el predicado `observacion()`. A la especificación se le impone dicho evento (tras una transición autónoma del entorno).

6.4.2 - Imposición versus Observación

¿Quién decide qué evento ocurre? ¿El probador o el producto sujeto a pruebas? La respuesta depende de la estructura de la especificación. Esta ya fue analizada durante la fase de generación de pruebas, y la información pertinente consta en el caso de prueba. Si en un estado dado el caso sólo prevé un posible evento, la respuesta es simple: no hay nada que elegir, u ocurre ese evento o algo ha fallado (se suele decir que hay bloqueo, aspecto que trataremos más adelante). Si el caso de prueba permite una decisión indeterminista, el probador debe estar preparado para observar cualquiera de las posibilidades y emitir el veredicto correspondiente.

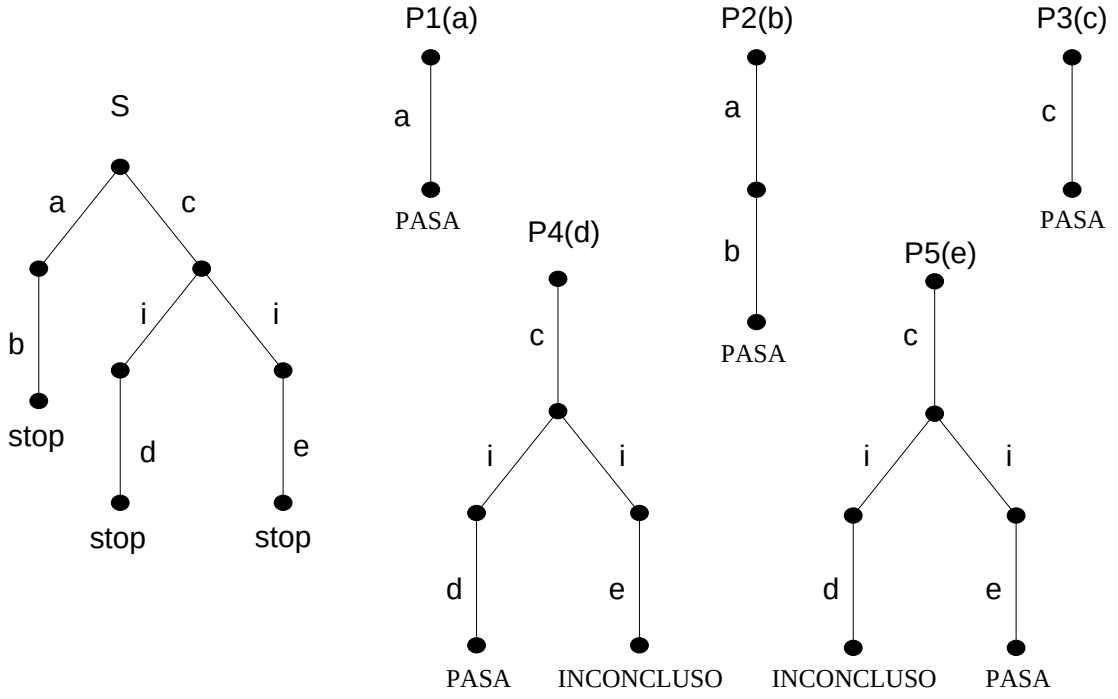


Figura 6.4: Ejemplo de elaboración de casos de prueba

La figura 6.4 presenta varias situaciones típicas. Dada una especificación S, hemos extraído varias pruebas P, identificadas por un objetivo (por ejemplo, $P1$ persigue el objetivo a). Una prueba como $P1$ dice que queremos observar a y nada más que a. No hay nada que opinar de parte de la implementación: si ocurre a, la prueba PASA. Si no, el sistema se bloquea.

Una prueba como $P4$ persigue el objetivo d. Para ello es necesario pasar por c, indiscutiblemente; pero a partir de ahí, la implementación puede optar por d (lo que sería un éxito) o por e, que no es que sea fracaso, pero ciertamente nos deja el veredicto inconcluso.

6.4.3 - Seguimiento Sobre la Especificación de Referencia

El caso de prueba es el motor de ejecución de la prueba. Según se van observando eventos, se genera un informe detallado. Si llegamos a un resultado satisfactorio (PASA), poco más hay que decir de la prueba. Pero si llegamos a un resultado inconcluso o, aún más, si llegamos a una situación de bloqueo, hay que informar al fabricante de qué ha ocurrido. Obviamente, el producto es para nosotros una caja negra y no podemos identificar errores dentro de él. Este es el problema del fabricante que tendrá que irse a su fábrica, identificar el error en su código, y corregirlo.

Pero hay que indicarle exactamente qué requisito de la especificación se ha incumplido. Para ello el sistema de pruebas realizará un seguimiento de la especificación de referencia durante la prueba. Según se observa cada evento, haremos avanzar la especificación un paso. Así, cuando la prueba se bloquee, sabremos exactamente el punto de la especificación en el que nos encontramos, y por dónde hemos llegado hasta él. Basta imprimir la traza y el estado final para proporcionar una valiosa información de qué aspecto de la especificación de referencia se incumplió, referido a ésta misma.

Este seguimiento se realiza en base a una ejecución simbólica con datos reales. Es decir, un intérprete.

6.4.4 - Bloqueo

Los casos de prueba sólo contemplan terminaciones exitosas o inconclusas. Pero en cualquier momento puede ocurrir cualquier evento no previsto o, más precisamente, que ninguno de los eventos previstos ocurran dentro del plazo marcado en el propósito de prueba. Cuando un temporizador salta decimos que se ha producido un **bloqueo**.

Un bloqueo puede ocurrir durante la ejecución del preámbulo o durante la ejecución del cuerpo de la prueba. En el primer caso se asigna un veredicto de *inconcluso,* siguiendo la costumbre de los centros de homologación y las recomendaciones de la ISO [ISO91b]. Esta asignación es arbitraria y, usualmente, conlleva la repetición de la prueba un cierto número de veces. En cuanto pasa una vez, se considera pasada y no se repite más; pero si al cabo de un número predeterminado de intentos sigue fallando el preámbulo, se le asigna un veredicto final de inconcluso. Esto también es arbitrario; pero sigue siendo práctica habitual.

Muy diferente resulta la situación en la que el probador se bloquea durante la ejecución del cuerpo de la prueba. En estos casos, el veredicto es inequívocamente FALLO. Se emite un informe del estado de la especificación de referencia en el momento del bloqueo, y se rechaza el producto[4].

6.4.5 - Pruebas correctas

Cada caso de prueba debe incluir información acerca de la consecución de su propósito. En ejecución es posible tener una serie de estados marcados como pendientes de confirmación, en el sentido de que más adelante sabremos lo que realmente está ejecutando la IUT. Por ello definimos como *prueba correcta* aquella que cumple las siguientes condiciones:

- Debe contemplar un único propósito de prueba. De aquí se deriva que:
  - La prueba debe contemplar un solo caso de terminación en FALLA o PASA. El primer caso corresponderá a pruebas de rechazo, mientras que el segundo son pruebas de aceptación.
  - Si comportamientos aceptables en la especificación apartan la ejecución del propósito de la prueba podrán existir ramas de la prueba que terminen en INCONCLUSO.
- La terminación de una prueba está limitada según su tipo. Así:
  - una prueba MUST ACCEPT debe consistir en una única traza terminante con etiqueta final PASA.
  - una prueba MAY ACCEPT debe contener más de una traza terminante, pero sólo una de ellas tendrá etiqueta PASA. Las demás tendrán forzosamente etiqueta INCONCLUSO.
  - una prueba MUST REJECT debe consistir en una única traza terminante con etiqueta final FALLA. Además se considera (por tener propósito único) que solo existirá un evento de rechazo. Todos la secuencia de eventos anteriores será considerada como preámbulo de la prueba.
  - una prueba MAY REJECT debe contener más de una traza terminante, pero sólo una de ellas con etiqueta FALLA. Igualmente que en el caso anterior, se considera que el cuerpo de la prueba comienza en el estado justamente anterior al evento a rechazar. El resto de las trazas deben terminar en INCONCLUSO.
- En cada estado de la prueba pueden existir un único tipo de eventos:

[4] A efectos prácticos, puede que se intenten algunas pruebas más para aprovechar la sesión de homologación e intentar descubrir más errores antes de devolver el producto al fabricante; pero pase lo que pase con cualquier otra prueba, el producto será rechazado.

  - **imponibles**: un evento imponible es aquel que la prueba trata de forzar su ejecución por la implementación. Si ésta lo rechaza se considera que le ejecución ha sido bloqueada. Sólo puede existir un evento imponible en un estado dado.
  - **aceptables**: un evento aceptable es aquel que la prueba se limita a observar en su ejecución por parte de la IUT. Cuando ésta ejecuta uno de tales eventos la prueba se limita a evolucionar acordemente, sincronizando adecuadamente e intercambiando datos si fuera menester. Pueden existir varios eventos aceptables en un estado dado.

- Cada traza terminal debe ser identificable sobre la prueba. En definitiva, queremos saber en que estado termina la ejecución (si ésta llega al final).

Una de las consecuencias directas es que una prueba correcta es una prueba *mensurable* sobre la especificación de referencia.

De aquí en adelante supondremos que las pruebas serán correctas.

6.4.6 - Etiquetado de Pruebas en LOTOS

A continuación se describe un ejemplo completo con todas las anotaciones propuestas de lo que se considera una prueba correcta.

En una prueba se ha de marcar el punto donde termina el preámbulo y comienza el cuerpo. Además, hay que tener en cuenta que la prueba puede no tener preámbulo. Por ello se propone la anotación `END_OF_PREAMBLE` que indicará que el preámbulo ha terminado. Esta anotación puede aparecer una sola vez en la prueba. Caso de que no aparezca, se supone que no existe preámbulo.

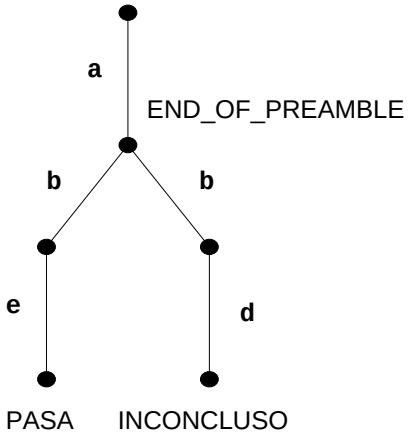
Hemos visto que la prueba debe acabar en un estado identificable pero, además, la última oferta de la prueba debe adjuntar la etiqueta de terminación. Se proponen las anotaciones `PASA`, `FALLA` e `INCONCLUSO` para los tres veredictos asignables, con las restricciones detalladas en la sección 6.4.5.

Debemos proporcionar un mecanismo para resolver la aplicación de cada acción de la prueba sobre la IUT.. Hemos visto que los eventos serán imponibles u observables. La caracterización de la controlabilidad y observabilidad ha sido estudiada en el capítulo 3, donde se proporcionaba un mecanismo de definición del interfaz que debe cumplir la especificación. Por lo tanto, se requiere haber caracterizado el interfaz con las anotaciones descritas. Así, el patrón de acción caracterizado en el interfaz como `IN` será imponible y como `OUT` será observable.

Por último, se tiene que tener en cuenta cómo se consideran los bloqueos. Evidentemente, están los debidos a reacciones erróneas de la IUT, lo que se detecta simplemente por la propia semántica de pruebas: como no es una acción incluida en la prueba el sistema determinará que es un bloque. Pero hemos comentado ya la posibilidad de que la IUT no reaccione. Esto se resuelve con un temporizador, que será introducido en el interfaz mediante la anotación `timeout #`. Si pasadas # unidades de tiempo desde la última

acción no se ha obtenido reacción por parte de la IUT, se considerará que se ha producido un bloqueo.

En la figura 6.5 se presenta un ejemplo de prueba correcta completamente anotada.




```
process test [a, b, c, d, e] : noexit :=
    a (*| END_OF_PREAMBLE |*) ;
        (b ; e (*| PASA |*) ; stop)
        []
        (b ; d (*| INCONCLUSO |*) ; stop)
endproc
```

Figura 6.5: Prueba correcta completamente anotada

6.4.7 - Cobertura

El sistema de pruebas combina aspectos de caja negra con aspectos de caja blanca. El sistema bajo pruebas se trata como una caja negra, y por tanto carecemos de conocimiento respecto de en qué medida ha sido probado. Realmente, como centro de homologación, éste no es un aspecto que nos interese lo más mínimo: es al fabricante del producto al que le preocupó en su día desarrollar pruebas estructurales de su producto. Al centro de homologación solo le preocupan las pruebas funcionales.

La especificación de referencia, por el contrario, es tratada como una caja blanca, pues las pruebas funcionales del producto se corresponden a las pruebas estructurales de la especificación[5].

Las pruebas de caja blanca son susceptibles de una calificación de cobertura, es decir del grado en que los casos de prueba ejercitan o cubren la lógica del sistema [Mye79, Bei83]. La prueba perfecta de caja blanca es aquella que cubre todos los posibles comportamientos del sistema. Desgraciadamente, sólo ejemplos académicos de simplicidad extrema permiten llegar a este ideal de perfección; mientras que cualquier sistema telemático de

[5]Quizás debiera hacerse una salvedad en lo que respecta a aquellos aspectos que pudieran adolecer de un defecto de sobreespecificación, es decir, que la especificación entre en detalles del cómo se obtiene alguna funcionalidad. Supondremos nosotros que ésto no es así, y que la especificación se limita a los mínimos necesarios de qué es lo que debe hacer el producto, y permite realmente todas las posibles implementaciones.

mediana complejidad contempla un número virtualmente infinito de posibles comportamientos, haciendo inviable su cobertura absoluta.

Clásicamente se han venido utilizando criterios de cobertura de programas basados en elementos sintácticos y constataciones simples durante la ejecución del programa:

cobertura de sentencias: Ejecución de sentencias del programa. Se persigue el objetivo de que todas se hayan ejecutado al menos una vez.

cobertura de decisión: Toma de decisiones. Se persigue el objetivo de que todas se hayan ejecutado al menos una vez, cubriéndose todas las posibles ramas de salida.

cobertura de condición: Las decisiones se desglosan en componentes, procurándose que cada componente se barra en sus dos posibles valores booleanos (cierto y falso).

cobertura de decisión/condición: Combina las dos anteriores.

Típicamente es fácil llegar a una cobertura de sentencias del 100%; pero se suele considerar como un éxito mezquino pues queda oculta la riqueza de comportamientos proporcionada por los datos, así como la utilización de segmentos de código desde diferentes puntos (procedimientos). La práctica suele aconsejar la cobertura de decisiones, pues está directamente relacionada con la cobertura del enunciado del problema. Se suele considerar un 85% como un buen nivel de cobertura.

6.4.8 - Asignación de veredictos

La asignación de veredictos descrita en la norma IS-9646 necesita de interpretación, puesto que va a depender, por un lado, de la parte de la prueba que se esté ejecutando. Por otro lado, esta interpretación es distinta para una prueba de rechazo y para una de aceptación.

Otro aspecto que modificará la asignación de veredicto es la parte de la prueba que estemos ejecutando. En efecto, si la ejecución de la prueba se interrumpe en el preámbulo de la misma no se puede aseverar el cumplimiento de su propósito. Por ejemplo, si se intenta probar que un dato se retransmite al vencer un plazo y, al ejecutar la prueba, no se establece la conexión, la prueba no termina bien, pero no se puede asegurar que los datos no se retransmiten.

Por todo ello, se proporciona la tabla 6.1 de asignación de veredictos [Rob91], relacionando la terminación de la ejecución de la prueba (filas) con la clasificación de la misma (columnas).

En la primera columna se presentan todas las posibles terminaciones de la ejecución de una prueba. Las dos primeras filas se refieren al caso en que exista un bloqueo antes de terminar la prueba. Si se está ejecutando el preámbulo, el veredicto es INCONCLUSO. Si ocurre ya en el cuerpo principal de la prueba obtenemos normalmente un FALLO, excepto en el caso de prueba MUST REJECT, puesto que precisamente la intención de la prueba es que se rechace el evento que forma el cuerpo de la prueba.

CLASIFICACIÓN

EJECUCION

	MUST ACCEPT	MAY ACCEPT	MUST REJECT	MAY REJECT
Bloqueo Preámbulo	I	I	I	I
Bloqueo Cuerpo	F	F	P	P
Termina	Etiqueta terminal			

I: Inconcluso P: Pasa F: Falla

Tabla 6.1: Asignación de veredictos

Las tres últimas filas se refieren a la terminación de la prueba. Si se llega al final, dependiendo de cómo se terminó la ejecución y del objetivo de la prueba asignamos veredictos.

6.5 - Formalización de la Ejecución de Pruebas

Todos los aspectos comentados en la sección anterior han de ser combinados en la formalización de la ejecución de pruebas. Se resalta en este caso que existe un tercer elemento en juego que no suele tomar parte en técnicas más convencionales: una especificación de referencia formalizada y manejable por herramientas.

Adoptaremos las siguientes hipótesis de partida:

- La IUT sólo ofrece/ejecuta un evento cada vez.
- La prueba es correcta (según se definió en 6.4.5).
- La prueba, en un momento dado, puede:
  - Ofrecer una acción imponible.
  - Ofertar varias acciones cuya decisión de ejecución corresponde totalmente a la IUT.
- Supondremos la existencia de la función io, que determina si una acción de una prueba es imponible o no.
- La IUT será representada por un modelo de LTS, que puede no ser conocido a priori, según la figura 6.6. La intención no es tanto construir tal modelo, sino tener un

mecanismo de representación de las ofertas de la IUT[6].

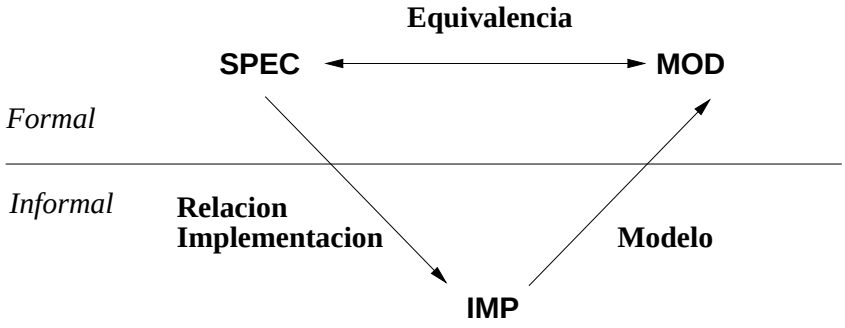
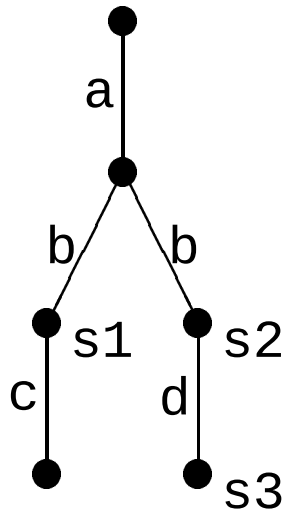


Figura 6.6: Relación entre la IUT, su modelo y la especificación de referencia

La ejecución de la prueba consiste en evolucionar en paralelo la IUT y la prueba, teniendo en cuenta que mientras que la IUT sólo ejecuta una posible secuencia de acciones, sobre la prueba se pueden seguir diferentes trazas en un momento dado. Estas trazas serán consideradas como pendientes de confirmación. Evidentemente, si la prueba está bien diseñada y termina, será posible decidir, tarde o temprano, cuál de las trazas está realmente siendo observada.

El ejemplo siguiente servirá para clarificar la cuestión:

Sea la prueba



Cuando se haya observado la secuencia `a, b` sobre la IUT, podemos estar en dos posibles estados, `s1`, `s2`, que marcaremos como pendientes. Cuando la prueba termine, por ejemplo ejecutándose `c`, no quedará duda de que el estado alcanzado será `s3`.

En definitiva, podemos tener un conjunto de estados de la prueba y y uno solo en la IUT.

[6]En particular, si este modelo estuviese disponible, se podría estudiar equivalencias tan fuertes como bisimulación y otras, con soporte matemático.

Sea T una prueba correcta, I una implementación, t el tiempo transcurrido desde la ejecución de una acción. Sean t_0 e i_0 los estados iniciales de T e I, respectivamente. Denotaremos con TS, TS', TS_i a conjuntos de estados sobre la prueba.

Dominios

Definición 19 *Definimos como resultado de la ejecución de una prueba sobre una IUT como un valor en el siguiente dominio:*

$$\mathcal{R} = (\textit{BloqueoEnPreambulo, BloqueoEnCuerpo, PASA, FALLA, INCONCLUSO})$$

Definición 20 *Definimos el estado de una ejecución de una prueba sobre una IUT como la tupla*

$$\Upsilon =< TS, i >$$

donde

- TS es un conjunto de estados pertenecientes a la prueba.
- i es un estado de la IUT.

denotaremos como $\varphi, \varphi_1, \varphi', \ldots$ a elementos de Υ.

Variables

Definición 21 *Definimos la variable*

$$preamble \in \{TRUE, FALSE\}$$

Esta variable tomará el valor $TRUE$ si la prueba se encuentra en el preámbulo y $FALSE$ en el caso contrario. Operacionalmente, comienza siempre con el valor $TRUE$ y lo cambia cuando se ejecute la acción anotada como `END_OF_PREAMBLE`.

Funciones

Definición 22 *Definimos la función de terminación T:*

$$T : offert \rightarrow \{\textit{PASA, FALLA, INCONCLUSO}\}$$

que dada una oferta terminante de una prueba devolverá la anotación asignada al resultado de la ejecución en forma de anotación, según se definió en 6.4.6.

Definición 23 *Definimos la función $\mathcal{L}$:*

$$\mathcal{L} : offert \rightarrow \{IN, OUT\}$$

que dada una acción de la prueba devuelve la anotación referente a la direccionalidad asignada en el interfaz (PCOs).

Definición 24 *Definimos la función Γ que dada una oferta devuelve el valor asignado en la anotación* `timeout`*:*

$$\Gamma : offert \rightarrow time$$

Definición 25 *Definimos la función Ofs de extracción de ofertas dado un conjunto de estados como:*

$$Ofs : state^* \rightarrow off\_set$$

con

$$OFs(TS) = \{o | o = Of(t_i)\ \forall t_i \in TS\}$$

Definición 26 *Definimos la función Ofc de elección de oferta entre una prueba y una IUT como:*

$$Ofc : \Upsilon \rightarrow offert$$

siendo:

$$Ofc(< TS_n, i_n >) = \begin{cases} O\_NIL & \text{si } t > t_i, \forall t_i = \Gamma(o_i)\ \forall o_i \in Ofs(TS_n) \\ Ofs(TS_n) \otimes Of(i_n) & \text{si } \mathcal{L}(o_i) = OUT \wedge t\ leq\ t_i, \\ & \quad \forall o_i \in Ofs(TS_n)\ \forall t_i = \Gamma(o_i) \\ Ofs(TS_n) \otimes Of(i_n) & \text{si } \mathcal{L}(o_i) = IN \wedge t\ leq\ t_i, \\ & \quad \forall o_i \in Ofs(TS_n)\ \forall t_i = \Gamma(o_i) \end{cases}$$

En el primer caso, tenemos que si el tiempo transcurrido tras la la última acción ejecutada sobrepasa todos los `timeouts` especificados se considera que existe un bloqueo. El segundo caso considera el caso de, sin existir vencimientos de plazo (al menos en alguna acción aceptable por la prueba), la IUT impone alguna acción. En definitiva se espera la reacción de la IUT, ahora bien, esa reacción debe ser acorde al propósito de la prueba.

Por último, la prueba intenta imponer una acción a la IUT. Recordemos que en este caso TS_n debe ofrecer una sola acción. Si esta acción no es ejecutada por la IUT, el resultado de $Ofs(TS_n) \otimes Of(i_n)$ será, evidentemente, O_NIL.

Definición 27 *Definimos la función A:*

$$A : state^* \times offert \rightarrow state^*$$

$$A(TS, a) = \{s_{i+1} | s_i = a \Rightarrow s_{i+1}\ \forall s_i \in TS\}$$

como la función que, dado un conjunto de estados de la prueba y una oferta ejecutada, determina el conjunto de estados resultantes en la prueba. Esta función, en definitiva, hace un seguimiento sobre la especificación de los estados de la prueba por los que está pasando la ejecución de la misma.

Definición 28 *Definimos la función $\mathcal{P}$ de paso de ejecución de una prueba sobre una IUT como:*

$$\mathcal{P} : \Upsilon \times offert \rightarrow \Upsilon$$

$$\mathcal{P}(< T, i >, o) = \{(< T', i' >) |\ T' = A(T, o),\ i' = gl(i, o)\}$$

Definición 29 *Definimos la función de ejecución de una prueba sobre una IUT como:*

$$\xi : test, IUT \rightarrow \mathcal{R}$$

donde el comienzo de la ejecución se denota como:

$$\xi(\{t_0\}, i_0)$$

La ejecución de una prueba consiste en aplicar sucesivamente las acciones de la prueba en la IUT. Esta "aplicación" supone decidir si se impone alguna acción a la IUT o se espera a observar alguna reacción de la misma. Si existe bloqueo, el resultado de la ejecución será $BloqueoEnPreambulo$ o $BloqueoEnCuerpo$ según estemos en el preámbulo o en el cuerpo de la prueba.

Una vez terminadas las acciones contempladas en la prueba, el resultado es, simplemente, la etiqueta que acompaña a la última acción ejecutada.

Formalmente:

$$\xi(< TS_n, i_n >) = \begin{cases} \mathcal{L}(o) & \text{si } Ofs(TS_n, o) = \emptyset \wedge o \neq O\_NIL \\ \textit{BloqueoEnPreambulo} & \text{si } Ofc(< TS_n, i_n >) = O\_NIL \\ & \wedge preamble = TRUE \\ \textit{BloqueoEnCuerpo} & \text{si } Ofc(< TS_n, i_n >) = O\_NIL \\ & \wedge preamble = FALSE \\ \xi(P(< TS_n, i_n >, Ofc(TS_n, i_n))) & \text{en otro caso} \end{cases}$$

6.6 - Conclusiones

La homologación de productos de comunicaciones respecto de sus especificaciones de referencia es práctica habitual que proporciona un elevado grado de confianza en que los productos que pasen las pruebas sean capaces de proporcionar un servicio efectivo durante su uso.

El uso de técnicas formales en la especificación de las normas de referencia y la existencia de una semántica operacional asociada a las mismas, permite que las pruebas de homologación sean dirigidas por la especificación, permitiendo una formalización de todos sus componentes: propósitos de pruebas, asignación de veredictos, semántica de pruebas y métricas. Todo ello con soporte de herramientas.

En este capítulo se ha proporcionado una visión global de los componentes, arquitectura y funcionalidad de un sistema de soporte de pruebas que se esta desarrollando actualmente y del cual ya existe un prototipo. Sus principales características se pueden resumir en (1) utilización sistemática de la especificación original como objeto de referencia, en particular en lo referente a análisis de cobertura e informe de errores; (2) "inversión" de las pruebas dinámicamente durante la ejecución, y (3) coordinación entre elementos abstractos y reales en tiempo de ejecución.

Por otro lado, se define el concepto de *prueba correcta*, de crucial importancia a la hora de ejecutar, interpretar resultados y asignar veredictos. En efecto, a veces se entiende por prueba cualquier traza extraída de la especificación, sin tener en cuenta factores como el indeterminismo, imponibilidad y observabilidad, y otros que afectan al proceso de ejecución.

Asimismo, se proporciona un marco formal de referencia para la ejecución de pruebas, generación de resultados y asignación de veredictos que tiene en cuenta una prueba *correcta* especificada formalmente y ejecutada sobre un producto real, al cual no se le impone nada, salvo la posibilidad, claro está, de interpretar su comportamiento (acciones y reacciones) en los mismo términos que la prueba.

Se espera así mismo que las herramientas generadas para este sistema sean útiles en el proceso de pruebas de depuración de la propia especificación de referencia (y/o de refinamientos de la misma utilizados a lo largo del desarrollo del producto). En este marco, las facetas de conexión con un producto real (correspondencia abstracto–real) pasan a un segundo término. En cambio, las herramientas de generación de casos de prueba reciben un fuerte empuje en el sentido de que sobre la especificación sujeta a análisis se estudia si permite alcanzar cierto objetivo o no.

Capítulo 7

Conclusiones

7.1 - Conclusiones

En este trabajo hemos abordado la fase de ejecución de pruebas de conformidad en un entorno en el cual se dispone de una especificación formal del sistema o producto, que consideramos como *referencia*. Se abordan desde aspectos metodológicos (medidas de cobertura) hasta operacionales (ejecución de pruebas), sin pasar por alto el hecho de que existe un entorno normalizado para generación de pruebas (ISO-9646).

El primer campo de actuación ha sido el estudio de los interfaces de pruebas y de las características que influyen en la capacidad de control y observación que tenemos sobre un sistema. El capítulo 3 describe la investigación llevada a cabo. Los resultados comprenden:

- Caracterización y modelado de PCOs.
- Formalización de las características de los PCOs.
- Desarrollo de un lenguaje de anotaciones para modelado de PCOs usando LOTOS como lenguaje de especificación.

El segundo campo de actuación ha sido el estudio de las medidas de cobertura como parámetro fundamental para evaluar qué partes de la especificación son ejercitadas. En este sentido, se incluye un nuevo elemento respecto de la Ingeniería Software convencional, que es precisamente la Especificación de Referencia. Esto da lugar a poder determinar una cobertura *a priori*, entendida como potencia de una batería de pruebas y una medida *a posteriori* o concepto convencional de cobertura de ejecución.

El capítulo 4 recoge la evaluación de diferentes coberturas de comportamientos y la propuesta Cobertura de Composición de Eventos. Asimismo, se propone un algoritmo para computar dicha métrica, descrito en el capítulo 5, donde se resuelven los problemas operativos y de implementación de dicha medida.

Además se ha realizado una herramienta que implementa dicho algoritmo, calculando sobre una especificación los eventos que contribuyen al 100% de cobertura. Como valora

añadido de la herramienta, se calcula el subárbol de sincronización de la especificación que cubre dicho 100%. De esta forma, es fácil determinar qué partes de la especificación no están cubiertas y sirve de guia para el desarrollo de pruebas orientadas por cobertura. La aplicación y resultados se comentan en el apéndice C.

Los resultados de este segundo estudio comprenden:

- Análisis de métricas de cobertura de comportamiento para el lenguaje LOTOS.
- Propuesta de una métrica de cobertura, llamada de Composición de Eventos, que capture las posibles combinaciones de eventos simples evitando la infinitud debida tanto a la repetición de comportamientos como a los tipos de datos.
- Desarrollo de un algoritmo para computar dicha métrica.

El tercer y último campo de actuación ha sido el estudio de los elementos, conceptos y métodos puestos en juego a la hora de ejecutar las pruebas de conformidad sobre protocolos de comunicaciones. El capítulo 6 recoge estos resultados, que comprenden:

- Definición de todos los elementos que componen una prueba orientada a su ejecución dentro del entorno de conformidad de la norma ISO-9646.
- Formalización de dicho elementos y propuesta de especificación de los mismos como partes significativa e inseparable de la prueba.
- Formalización del proceso de ejecución de prueba sobre un producto real, incluyendo conceptos de bloqueo, indeterminismo, observación e imposición, así como resultados de la ejecución y asignación de veredictos.

7.2 - Vías de Investigación Futuras

Entre las posibles continuaciones de la presente investigación se consideran de especial relevancia los siguientes campos:

- Investigación de la aplicación de los conceptos formalizados a otras FDTs. En particular, existe un gran interés industrial en el lenguaje SDL, para el que se disponen de potentes herramientas de especificación y análisis.
- Incorporación al propio lenguaje LOTOS de conceptos que faciliten la generación de pruebas y la caracterización de los PCOs. De esta forma, los conceptos capturados mediante anotaciones formarían parte de la semántica propia del lenguaje, facilitando enormemente el proceso de generación y ejecución de pruebas. Actualmente, ISO está trabajando en una nueva versión del lenguaje, llamada E-LOTOS, en las que algunos conceptos como el tiempo parece que serán incluidos. El presente trabajo puede formar parte de las guías de estudio que se están teniendo en consideración.

- Uno de los resultados del algoritmo de cobertura es la exploración del árbol de sincronización de una especificación. Este árbol contiene todas las participaciones de eventos simples, con lo que resulta una vía prometedora para la generación de pruebas con objetivo predeterminado de cobertura. Es particularmente interesante estudiar la generación de un árbol de sincronización mínimo en cuanto a tamaño que contenga todas las combinaciones.

- Integración de métricas de cobertura sobre los tipos de datos. Los tipos de datos con semántica no predefinida dificulta la tarea de análisis de los mismos, requiriendo pruebas específicas que comprueben la correcta implementación de los mismos.

Otras actividades En una tesis de ingeniaría no puede olvidarse en ningún momento la aplicación industrial de los resultados de las investigaciones llevadas a cabo. Por ello, proponemos:

- Desarrollo de una herramienta gráfica que integre los presentes conceptos. Los prototipos desarrollados trabajan de forma independiente, requiriendo un esfuerzo adicional fácilmente automatizable. Este entorno abarcaría desde la captura de los propósitos de prueba hasta la ejecución de las pruebas. La herramienta debe, a partir de los propósitos de prueba, generar las pruebas, facilitar su anotación, caracterización de los PCOs y ejecución de las pruebas sobre un producto real.

- Aplicación de todos los conceptos formalizados en un entorno industrial real, que valide las soluciones propuestas y las matice bajo un prisma de desarrollo en una compañía con intereses industriales.

Apéndice A

GLOSARIO

- <+>(16): Este operador compone dos expresiones de comportamiento LOTOS de forma que la expresión resultante ofrece el conjunto unión de ofertas de las dos subexpresiones.
- <*>$_g$(16): Este operador compone dos expresiones de comportamiento LOTOS de forma que la expresión resultante ofrece el conjunto sincronización en la puerta g de las ofertas de las subexpresiones.
- **Acción**: Transición entre dos estados de un LTS.
- **Conformidad**(6.1):
  - Proceso por el cual se chequea que una implementación satisface requisitos de conformidad.
  - Dícese que una implementación I es conforme a una especificación S si y solo si (1) para cualquier comportamiento impuesto por la especificación S, la implementación es capaz de llevarlo a cabo, y (2) cualquier comportamiento que la implementación rechace, está previsto en S que pueda ser rechazado.
- **Cuerpo de Prueba**(6.2): Parte de la prueba orientada a comprobar una parte específica del sistema o un requisito del mismo.
- D'_b(4.3): Ver Reglas de Inferencia.
- **Evento**: Sinónimo de acción.
- **Oferta**(1): Se define una oferta como:

$$g^{\{id_1,\ldots,id_m\}}_{<s_1,\ldots,s_n>}$$

Donde:

  - g: la puerta sobre la cual se realiza la sincronización ($g \in G \cup \{\iota, \delta\}$), Siendo G el conjunto de puertas definidas en la especificación.
  - $< s_1, \ldots, s_n >$: la lista de los tipos (**sorts**) que formarán el patrón de acción. Nótese que es aquí donde ignoramos la información de los valores concretos que maneja la especificación, estando interesados únicamente en sus tipos.
  - $\{id_1, \ldots, id_m\}$: el conjunto de subexpresiones de acción que participan en la oferta. Cada subexpresión estará representada por su identificador único.

- **srt**(2): La función srt devuelve el tipo correspondiente de uno de los experimentos de una acción dada.

- **LTS, Sistema de Transiciones Etiquetadas**(2.3.3): Un LTS es una tupla $< S, L, T, s_0 >$, donde:

  - S en un conjunto (contable) no vacío de estados;
  - L es un conjunto (contable) de acciones observables, también llamado conjunto de etiquetas o alfabeto de acciones;
  - $T = \{\xrightarrow{a} \mid \xrightarrow{a} \subseteq S \times S \ y \ a \in A \cup \{\iota\}\}$ donde ι representa la acción interna; no siendo ésta una transición observable.
  - $s_0 \in S$, considerado el estado inicial, es un elemento de S.

- **Sistema de Derivación**(3): Un sistema de derivación es un conjunto de axiomas y reglas de inferencia que permite determinar cuáles son las acciones observables en un sistema a partir de su expresión de comportamiento.

- **gates**(4): La función $gates$ devuelve las puertas utilizadas en una expresión de comportamiento LOTOS.

- **gl**(14): La función gl (*golden lion*) hace avanzar una especificación al siguiente estado dada una acción ofrecida en el estado actual. No modifica la especificación si la acción no pertenece a dicho estado.

- **IB**(11): La función IB identifica cada subexpresión LOTOS de una expresión de comportamiento dada.

- **IUT, Implementation Under Test**([ISO91b]): realización de uno o más protocolos OSI en una relación adyacente proveedor/usuario. Es parte del sistema abierto objetivo de las pruebas.

- **last**(12): La función $last$ devuelve el último identificador asignado a una expresión de comportamiento LOTOS dada. Sólo es usada como auxiliar de IB.

- **Métrica de Composición de Eventos**(10): La Métrica de Composición de Eventos es la propuesta de la presente tesis para la medida de cobertura de las pruebas, tanto en su elaboración (*a priori*) como de ejecución (*a posteriori*).

- **Of**(13): La función Of calcula las ofertas disponibles en una expresión de comportamiento LOTOS, según las reglas de inferencia definidas en 4.3.

- **Patrón de Acción**(6): Por Patrón de Acción se entiende la tupla formada por el identificador de una puerta mas una lista (vacía o no) de **sorts** de una especificación. Si un Patrón de Acción pertenece a una especificación estará asociada con una de sus ofertas. Por extensión, se suelen utilizar como sinónimos de ofertas, siempre que pertenezcan a una especificación.

- **Patrón de Sincronización de Comportamiento**(5.3.1): Los PSC son casos particulares de expresiones de comportamientos LOTOS, donde se representa solamente la información de sincronización e identificadores únicos de subexpresiones de comportamiento.

- **PCO, Point of Control and Observation**: Un punto dentro del entorno de pruebas donde la ocurrencia de eventos de prueba será controlada y observada, según se define en un método abstracto de prueba.

- **Postámbulo**(6.2): Parte de la prueba que permite llevar a la implementación a un estado reconocible.

- **PICS, Protocol Implementation Conformance Statement**: Formulario elaborado por el fabricante en el que se relacionan qué capacidades han sido implementadas en un producto dado.

- **PIXIT, Protocol Implementation eXtra Information for Testing**: Formulario elaborado por el fabricante que contiene o referencia toda la información (adicional a la proporcionada por el PICS) relacionada con la implementación y su entorno de pruebas, que permitirá la apropiada ejecución de pruebas.

- **Propósito de Prueba**(6.2.1): Un propósito de prueba es "una descripción en prosa de un objetivo de prueba definido con precisión, centrado en un único requisito de conformidad, de acuerdo con la especificación de la norma correspondiente".

- **Preámbulo**(6.2): Fase preliminar en la que se lleva a la implementación al estado en que se quiere realizar una prueba.

- **Prueba** Una prueba es el conjunto de acciones, objetivo, método y entorno de aplicación junto con los recursos que se ponen en juego para ejercitar el sistema, normalmente con la intención de encontrar fallos.

  - **Prueba Mensurable**(9): Definimos como mensurable aquella prueba que lleva a la especificación a un estado único, de forma que se le pueda asignar cobertura sobre la observación de la especificación (cobertura *a priori*).

- **Prueba correcta**(6.4.5): aquella prueba que cumple los requisitos necesarios para formalizar su ejecución y asignación de cobertura.

- **PSC**: Ver Patrón de Sincronización de Comportamiento.
- **R**(19): Dominio de posibles resultados de aplicar una prueba a una IUT. Se define $\mathcal{R} = (\text{BloqueoEnPreambulo, BloqueoEnCuerpo, PASA, FALLA, INCONCLUSO})$.
- **RC**(15): La función RC extrae el PSC de una expresión de comportamiento LOTOS dada.
- **Reglas de Inferencia**(4.3): Conjunto de reglas que componen el sistema de derivación orientado a determinar las ofertas de una expresión de comportamiento LOTOS.
- **Subexpresión de Acción**: expresión de comportamiento LOTOS que comprende acciones internas, acciones de terminación y otras acciones, con o sin predicado de selección.
- **Subexpresión de Operador**: expresión de comportamiento LOTOS compuesta por dos subexpresiones más sencillas. En general, subexpresión LOTOS que no es de acción.
- **T**(17): Se define T como la proyección de una subexpresión LOTOS a los operadores $< + >$ y $< * >_g$.
- **Trazas**(5): Se define traza como una secuencia de acciones. El Sistema de Derivación determina las trazas que pertenecen a una especificación.
  - **Identificable**(7): Se denominan trazas identificables a aquellas trazas que, tras su observación, siempre llevan a la especificación al mismo estado.
  - **Terminante**(8): Se denominan trazas terminantes a aquellas trazas que, tras su observación sobre un comportamiento, llegan a un estado tras el cual no puede ofrecerse ninguna acción.

Apéndice B

Tipos Básicos para la Formalización de los PCOs

type $Boolean$ **is**

sorts $Bool$

opns

$$\begin{array}{llll} true, & false & : & -> Bool \\ not & & : Bool & -> Bool \\ \_and\_, & \_or\_, & & \\ \_eq\_, & \_ne\_ & : Bool, \ Bool & -> Bool \end{array}$$

eqns forall $x, y : Bool$

ofsort $Bool$

$$\begin{array}{ll} not\ (true) & = false; \\ not\ (false) & = true; \\ x\ and\ true & = x; \\ x\ and\ false & = false; \\ x\ or\ true & = true; \\ x\ or\ false & = x; \\ x\ eq\ y & = (x\ or\ not\ (y))\ and\ (not\ (x)\ or\ y); \\ x\ ne\ y & = not(x\ eq\ y); \end{array}$$

endtype

type $NaturalNumber$ **is** $Boolean$

sort Nat

opns

$$
\begin{array}{llll}
0 & : & & -> Nat \\
Succ & : & Nat & -> Nat \\
\_eq\_, \quad \_ne\_ & : Nat, & Nat & -> Bool \\
\_gt\_, \quad \_ge\_ & : Nat, & Nat & -> Bool \\
\_lt\_, \quad \_le\_ & : Nat, & Nat & -> Bool \\
\_gt\_, \quad \_lt\_ & : Nat, & Nat & -> Nat
\end{array}
$$

eqns forall $m, n : Nat$

ofsort $Bool$

$$
\begin{array}{ll}
0\ eq\ 0 & = true; \\
0\ eq\ Succ(m) & = false; \\
Succ(m)\ eq\ 0 & = false; \\
Succ(m)\ eq\ Succ(n) & = m\ eq\ n; \\
m\ ne\ n & = not\ (m\ eq\ n); \\
0\ gt\ 0 & = false; \\
0\ gt\ Succ(m) & = false; \\
Succ(m)\ gt\ 0 & = true; \\
Succ(m)\ gt\ Succ(n) & = m\ gt\ n; \\
m\ ge\ n & = (m\ gt\ n)\ or\ (m\ eq\ n); \\
m\ lt\ n & = not(m\ ge\ n); \\
m\ le\ n & = not(m\ gt\ n);
\end{array}
$$

ofsort Nat

$$
\begin{array}{ll}
m\ +\ 0 & = m; \\
m\ +\ Succ(n) & = Succ(m\ +\ n);
\end{array}
$$

endtype

type $Interaccion$ **is** $Boolean$

sort $Inter$

opns

$$\_eq\_, \quad \_ne\_ \ : Inter, \quad Inter \ -> Bool$$

endtype

type $Set\_Interacciones$ **is** $Interacciones, Boolean, NaturalNumber$

sort $SetInter$

opns

$$
\begin{array}{llll}
empty & & : & -> SetInter \\
Insert, & Remove & : Inter, SetInter & -> SetInter \\
\_IsIn\_, & \_NotIn\_ & : Inter, SetInter & -> Bool \\
Card & & : SetInter & -> Nat
\end{array}
$$

eqns forall $x, y : Inter, s : SetInter$

ofsort $SetInter$

$$
\begin{array}{ll}
Insert(x,\ Insert(x,s)) & = Insert\ (x,s); \\
Remove(x,\ empty) & = empty; \\
x\ eq\ y\ => & \\
Remove(x,\ Insert(y,s)) & = Remove\ (x,s); \\
x\ ne\ y\ => & \\
Remove(x,\ Insert(y,s)) & = Insert\ (y,\ Remove(x,s));
\end{array}
$$

ofsort $Bool$

$$
\begin{array}{ll}
x\ IsIn\ empty & = false; \\
x\ eq\ y\ => & \\
x\ IsIn\ Insert(y,s) & = true; \\
x\ ne\ y\ => & \\
x\ IsIn\ Insert(y,s) & = x\ IsIn\ s; \\
x\ NotIn\ s & = not(x\ IsIn\ s);
\end{array}
$$

ofsort Nat

$$
\begin{array}{ll}
Card(empty) & = 0; \\
Card(Insert(x,s)) & = Succ(Card(s));
\end{array}
$$

endtype

Apéndice C

Ejemplo de Aplicación de la Métrica de Composición de Eventos

C.1 - Ejemplo de Aplicación de Medidas de Cobertura: La Criba de Eratóstenes

En este apéndice se presenta un estudio de la aplicación de las diferentes definiciones de cobertura aplicadas a un ejemplo real. Se ha elegido la especificación del algoritmo de Eratóstenes para el cálculo de números primos puesto que tiene una serie de características muy interesantes: es de tamaño adecuado, fácil de comprender, representa algo real y su arquitectura dinámica es muy apropiada para mostrar las cualidades y potencia de la definición de cobertura aportada.

C.1.1 - La Criba de Eratóstenes

Eratóstenes (275? - 194 a. de J.C.) fue un matemático y filósofo griego. Trató de las dimensiones de la Tierra, de las constelaciones y de su mitología, de la duplicación del cubo, etc. Compuso también obras de carácter crítico y filosófico y dirigió la biblioteca de Alejandría. Entre sus aportaciones a la Matemática se encuentra la Criba que lleva su nombre, método orientado a determinar números primos.

Básicamente, el método consiste en ir probando todos los números enteros, empezando por el 2 e incrementando de 1 en 1. Existe un conjunto de números, precisamente todos los primos encontrados, contra los que será chequeado el nuevo candidato. Si el candidato no se divisible por ninguno de los primos en el conjunto, entonces es primo; por lo tanto, se añade al conjunto. Si el candidato es divisible al menos por alguno de los primos en el conjunto, se descarta. Y se prueba el siguiente número.

En las implementaciones de la Criba se suele empezar inicializando el conjunto de primos como $\{2\}$ y empezando a contar por 3 y de dos en dos (puesto que ningún número par, excepto el 2, es primo).

C.1.2 - Descripción de la Especificación

La especificación de la Criba de Eratóstenes consta de dos procesos comunicándose entre sí: el primero determina cual es el siguiente número a analizar y será llamado $Generador$.

El otro proceso se encargará de filtrar los números primos. Para comprobar si el número es primo, se intentará dividir por los primos ya encontrados: si es divisible por alguno de los primos el número no será primo. Si todos los primos existentes coinciden en no dividir el número, entonces es primo.

El comportamiento de este segundo proceso es también bastante sencillo. Cada vez que se encuentra un número primo, se creará un filtro que comprobará si el número bajo análisis es primo o no. Cuando un filtro detecta un número que no es divisible por él, lo comunicará al entorno mediante la puerta $good$. Si el filtro encuentra que el número es divisible, lo comunicará por la puerta bad. Todos los filtros sincronizarán en la puerta $good$, de forma que todos ellos tendrán que estar de acuerdo (y sincronizar al mismo tiempo) para decidir que un número es primo. En cuanto uno de los filtros detecte un número no primo y, por tanto, lo presente por la puerta bad los demás no podrán darlo por válido.

La estructura del segundo proceso es un generador de filtros, por lo que lo llamaremos $CreaFiltro$. Este proceso se instancia a si mismo en sincronización en la puerta $good$ con los filtros. Por lo tanto, cada vez que se determine un nuevo número primo, el $CreaFiltro$ sincronizará también, instanciándose con un nuevo filtro (el del número primo encontrado).

Por último, llamaremos $Filtro$ al proceso que se encarga de comprobar el número bajo análisis con cada primo existente.

Una consideración de carácter práctico es que empezaremos a comprobar números primos a partir del 3 y con incremento de dos unidades, para evitar probar todos los números pares.

Por lo tanto, tenemos una estructura recursiva dinámicamente en la que van sincronizando cada vez más procesos. Sin embargo, estos procesos no aportan ejecuciones nuevas cada vez que participan, pues el comportamiento del filtro es exactamente el mismo pero con distinta parametrización.

C.1.3 - Especificación LOTOS de la Criba de Eratóstenes

A continuación presentamos la especificación LOTOS de la criba de Eratóstenes. Para distinguir entre puertas con el mismo nombre pero distinto ámbito, se ha incluido un identificador entre corchetes antes de la misma.

specification Eratos [good] : **noexit**:

library $Boolean, NaturalNumber$ **endlib**

```
behaviour

    hide bad in
        (   generate [good, bad] (3)
            |[good, bad]|
            mkfilter [good, bad]
        )

where

    process  generate [good, bad] (n: nat): noexit :=
        good !n;
            generate [good, bad] (n+2)
        []
        bad !n;
            generate [good, bad] (n+2)
    endproc

    process mkfilter [good, bad] : noexit :=
    good ?n: nat;
        (   mkfilter [good, bad]
            |[good]|
            filter [good, bad] (n)
        )
    endproc

    process filter [good, bad] (n: nat) : noexit :=
        good ?d: nat [(d % n) <> 0];
            filter [good, bad] (n)
        []
        bad ?d: nat [(d % n) = 0];
            filter [good, bad] (n)
    endproc

endspec
```

C.1.4 - Aplicación de Medida de Cobertura de Puertas

Si aplicamos la cobertura de puertas definida en la sección 4.2.1, se obtienen los siguientes resultados:

Primero se calcula el límite del 100%. Dado que la especificación tiene una puerta visible good, ésta será el objetivo a cubrir.

La batería de pruebas que nos cubre el 100% de la especificación según cobertura de puertas es:

prueba	comportamiento	cobertura $\mathcal{M}(T)$
T_1	good ! 3; **stop**	100%

Como se ve, los resultados de esta Cobertura son muy pobres. La Batería que cubre el 100% de la Puertas no ejercita mas que una mínima parte de la ejecución. De hecho, si se examina estructuralmente la especificación, resulta que no se llega a instanciar un solo filtro. Sólo el número 3 aparece como primo, dejando de lado aspectos tan importantes como la creación de los filtros, la comprobación de las predicados y otros.

C.1.5 - Aplicación de Medida de Cobertura de Patrones de Acción

Esta cobertura no difiere, para este ejemplo, en nada del caso anterior, dado que la única puerta que ofrece eventos tiene siempre el mismo patrón: `good <Nat>`. Por ello, la Batería de pruebas sería la misma. Las conclusiones son las mismas que en el caso anterior: no se captura la semántica ni la estructura de la especificación en ningún caso.

Aplicación de la Cobertura de Guardas y Predicados

Más interesante es esta cobertura, que pone de relieve aspectos fundamentales de la especificación, puesto que la particularidad de la misma reside en los filtros de números.

El límite del 100% lo imponen los dos predicados del proceso $Filter$. Estos predicados son `[(d % n) <> 0]` y `[(d % n) = 0]`. Por lo tanto, se diseñarán pruebas para las cuales cada predicado resulte, respectivamente, $TRUE$ y $FALSE$, en todas las combinaciones posibles. En este sentido, tiene relevancia un aspecto particular de LOTOS: la inexistencia de cláusula **else**. En efecto, si se observan los predicados, se pone de manifiesto la mutua exclusividad del uno sobre el otro. En cualquier caso, esto no debe preocupar al diseñador de pruebas: en el mejor de los casos, se reducirá el tamaño de la Batería de Pruebas. Y su labor es intentar comprobar todos los casos posibles[1].

Por lo tanto, la Batería de Pruebas y su cobertura *a priori* queda como se presenta a continuación:

prueba	comportamiento	cobertura $\mathcal{M}(T)$
T_1	good ! 3; **stop**	0%
T_2	good ! 3; good ! 5; **stop**	50%
T_3	good ! 3; good ! 5; good ! 7; **stop**	50%
T_4	good ! 3; good ! 5; good ! 7; good ! 11; **stop**	100%

En este ejemplo. las coberturas *a priori* de las pruebas están incluidas unas en otras, dado que las pruebas se ven configuradas como continuación de la anterior. Por lo tanto, para obtener una cobertura del 100% en ejecución es suficiente con pasar la prueba T_4.

[1] Por otro lado, siempre se puede haber introducido un error en la operación de desigualdad de enteros <>, o bien = y <> pueden no ser perfectamente complementarios.

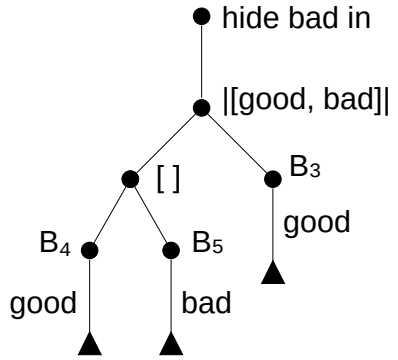
C.1.6 - Aplicación de la Métrica de Composición de Eventos

- **1** Identificación de subexpresiones LOTOS:

$$
\begin{array}{lcl}
B_0 & = & \textbf{hide}\ bad\ \textbf{in}\ B_1 \\
B_1 & = & B_2\ |[good, bad]|\ B_3 \\
B_2 & = & B_4\ []\ B_5 \\
B_4 & = & good\ !n;\ B_2 \\
B_5 & = & bad\ !n;\ B_2 \\
B_3 & = & good\ ?n : nat;\ B_6 \\
B_6 & = & B_3\ |[good]|\ B_7 \\
B_7 & = & B_8\ []\ B_9 \\
B_8 & = & good\ ?d : nat\ [(d\ \%\ n)\ <>\ 0];\ B_7 \\
B_9 & = & bad\ ?d : nat\ [(d\ \%\ n)\ =\ 0];\ B_7
\end{array}
$$

- **2** Inicializamos $PSC = \{\}\ PS = \{\}$. El árbol de sincronización resultante es:

.



.

- **3a** Hallamos las combinaciones iniciales OSa:

$$OSa = \{good^{\{B_4,B_3\}}_{<nat>}\}$$

- **4a** Como $OSa \neq \emptyset$:

$$OS = OS \cup OSa = \{good^{\{B_4,B_3\}}_{<nat>}\}$$

- **5a** Hallamos el patrón de sincronización $PSCa$:

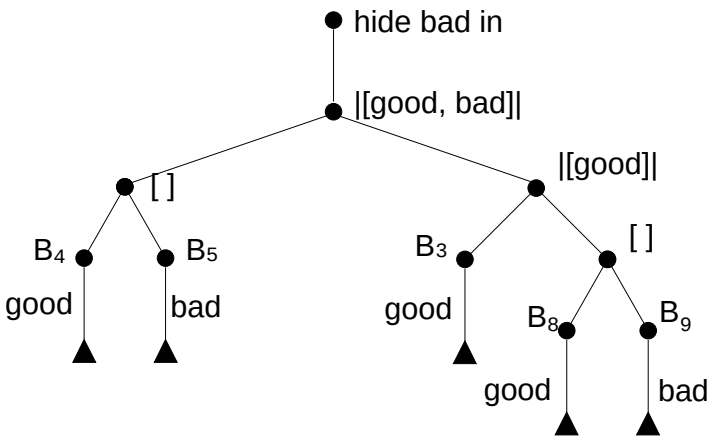
$$PSCa = ((B_4[]B_5)|[good, bad]|B_3)$$

- **6a** Como $PSCa \notin PSC$:

$$PSC = PSC \cup \{PSCa\} = \{((B_4[]B_5)|[good, bad]|B_3)\}$$

- **7a** Para la única acción de OSa, avanzamos la especificación, resultando:

.



.

- **3b** Las nuevas ofertas son:

$$OSa = \{\mathbf{i}^{\{B_5,B_9\}}_{<nat>}, good^{\{B_4,B_3,B_8\}}_{<nat>}\}$$

- **4b** Como $OSa \neq \emptyset$:

$$OS = OS \cup OSa = \{ \begin{array}{l} \mathbf{i}^{\{B_5,B_9\}}_{<nat>}, \\ good^{\{B_4,B_3\}}_{<nat>}, \\ good^{\{B_4,B_3,B_8\}}_{<nat>}\} \end{array}$$

- **5b** El nuevo patrón de sincronización es:

$$PSCa = ((B_4[]B_5)|[good, bad]|(B_3|[good]|(B_8[]B_9)))$$

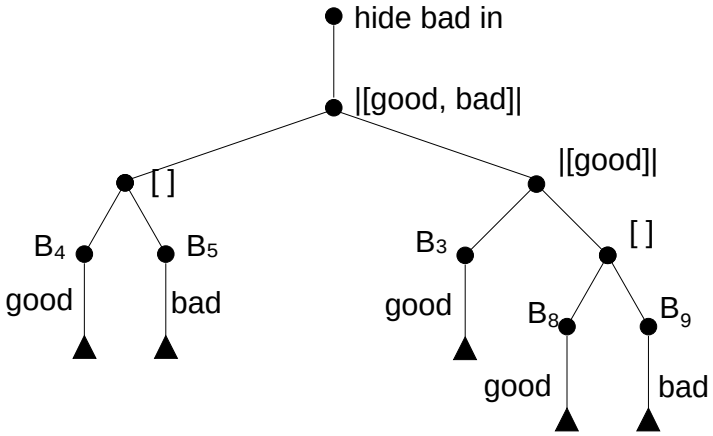
- **6b** Como $PSCa \notin PSC$:

$$PSC = PSC \cup \{PSCa\} =$$
$$\{((B_4[]B_5)|[good, bad]|B_3), ((B_4[]B_5)|[good, bad]|(B_3|[good]|(B_8[]B_9)))\}$$

- **7b1** Tenemos en OSa dos posibles acciones. Para cada una de ellas, avanzamos la especificación y repetimos el proceso.

  Para $\mathbf{i}^{\{B_5,B_9\}}_{<nat>}$, el árbol de sincronización resulta:

.



.

- **3c1** Calculamos ofertas:

$$OSa = \{\mathbf{i}^{\{B_5,B_9\}}_{<nat>}, good^{\{B_4,B_3,B_8\}}_{<nat>}\}$$

- **4c1** Con lo que resulta:

$$OS = OS \cup OSa = \{ \begin{array}{l} \mathbf{i}^{\{B_5,B_9\}}_{<nat>}, \\ good^{\{B_4,B_3\}}_{<nat>}, \\ good^{\{B_4,B_3,B_8\}}_{<nat>}\} \end{array}$$

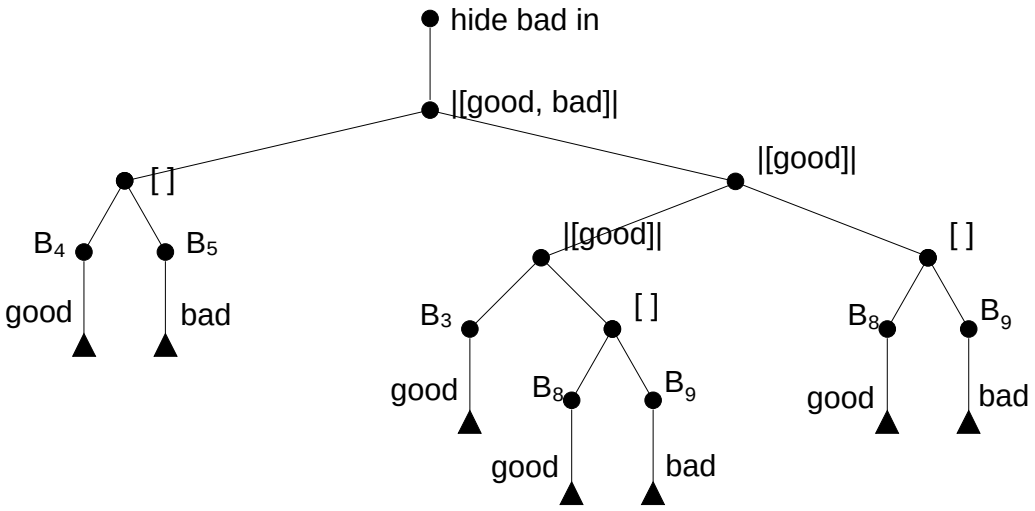
- **5c1** Calculamos $PSCa$:

$$PSCa = ((B_4[]B_5)|[good, bad]|(B_3|[good]|(B_8[]B_9)))$$

- **6c1** Y como $PSCa \in PSC$ terminamos el proceso.

- **5b2** Queda una segunda acción, $\mathbf{i}^{\{B_5,B_9\}}_{<nat>}$, para la cual el árbol de sincronización resulta:

.



.

- **3c2** Y, tras esta segunda acción, las combinaciones son:

$$OSa = \{\mathbf{i}^{\{B_5,B_9\}}_{<nat>}, good^{\{B_4,B_3,B_8,B_8\}}_{<nat>}\} \quad = \{\mathbf{i}^{\{B_5,B_9\}}_{<nat>}, good^{\{B_4,B_3,B_8\}}_{<nat>}\}$$

- **4c2** Que no modifica el conjunto actual de combinaciones.
- **5c2** Y el patrón de sincronización de comportamiento:

$$PSCa = ((B_4[]B_5)|[good, bad]|((B_3|[good]|(B_8[]B_9))|[good]|(B_8[]B_9)))$$

- **6c2** Y, dado que $PSCa \in PSC$, terminamos el proceso.

Límite del 100% Las posibles acciones y sus combinaciones de la especificación son:

1)	$good^{\{B_4,B_3\}}_{<nat>}$
2)	$good^{\{B_4,B_3,B_8\}}_{<nat>}$
3)	$\mathbf{i}^{\{B_5,B_9\}}_{<nat>}$

En definitiva, consideramos 3 acciones distintas que nos delimitan el 100% de cobertura.

Batería de Pruebas A continuación se presenta la Batería de Pruebas obtenida para alcanzar el 100% de cobertura.

Prueba	Comportamiento	Cobertura $\mathcal{M}(T)$
T_1	good ! 3; **stop**	33.33%
T_2	good ! 3; good ! 5; **stop**	66.66%
T_3	good ! 3; good ! 5; good ! 7; **stop**	66.66%
T_4	good ! 3; good ! 5; good ! 7; i (* bad ! 9 *); good ! 11; **stop**	100%

Donde, evidentemente, la acción interna debida al número 9 no será observable en la ejecución de un producto que implemente el algoritmo de Eratóstenes. Por lo tanto, no se podrá asignar cobertura *a posteriori* hasta observar la acción siguiente, $good$!11. Recordemos que se define la medida de cobertura para prueba cuyas trazas terminantes sean *identificables* en la especificación.

Resultados del prototipo El prototipo desarrollado para calcular las posibles combinaciones obtiene, como era de esperar, los mismos resultados, con la única diferencia en los identificadores asignados a los comportamientos. Esto diferencia es debida únicamente a detalles de implementación que provocan una numeración en distinto orden asignado.

Las combinaciones resultantes son:

```
COMBINATIONS ON SPECIFICATION eratos:
good <nat> {15,2,8}
bad <nat> {17,10}
good <nat> {2,8}
```

y el árbol de combinaciones es:

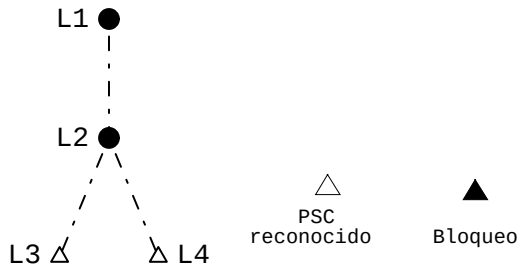


Figura C.1: Árbol de Sincronización de la especificación Eratos, expandido hasta localización de PSCs repetidos

Cuya leyenda es:

L1:
L2: good ! 3
L3: i (* bad ! 5 [(d % n) = 0] *)
L4: good ! 5 [((d % n) ne 0) = true]

C.2 - Ejemplo de Aplicación de Medidas de Cobertura: El Protocolo Abracadabra

En esta sección se presentan los resultados de aplicar el prototipo de la herramienta de cálculo de combinaciones a un ejemplo real. Hemos elegido el protocolo Abracadabra [ISO88], puesto que contempla los aspectos más realistas de las comunicaciones, entre las que se encuentran: numeración de secuencias (por Bit Alternante), Retransmisión en los Vencimientos de Plazos, Asentimientos, Conexión y Desconexión.

El ejemplo es lo suficientemente complicado, largo y tedioso como para desarrollar el algoritmo paso a paso, de forma que decidimos elaborar un prototipo que determinara las posibles combinaciones, para posteriormente integrarlo en unas herramientas que calcularan las medidas *a priori* y *a posteriori* de una batería de pruebas. En el primer caso, nos movemos en el mundo formal en el que tanto la especificación como la batería de pruebas están escritas en LOTOS. El segundo caso debe tratar, además, con la IUT, elemento informal en la que necesitamos un estudio sistemático de los PCOs de la IUT para poder integrarla con el sistema de ejecución de pruebas, así como el evaluador de cobertura *a priori*.

Veremos una breve descripción del protocolo, para a continuación detallar las tareas llevadas a cabo y los resultados de las herramientas.

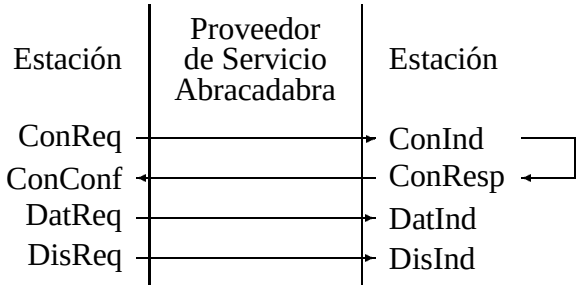
C.2.1 - El Protocolo Abracadabra

Descripción del Servicio

El protocolo Abracadabra opera entre un par de estaciones, denominadas como `A` y `B`. Se supone que cada estación tiene un interfaz local con la entidad del protocolo. El servicio ofrecido es fiable, orientado a conexión, entre un par de usuarios. Las primitivas del servicio soportadas son:

ConReq, ConInd – Connection Request, Indication
ConResp, ConConf – Connection Response, Confirmation
DatReq, DatInd – Data Request, Indication
DisReq, DisInd – Disconnection Request, Indication

Las únicas primitivas que transportan datos son `DatReq` y `DatInd`. Este parámetro es una unidad de datos del servicio, `SDU`. Las primitivas del servicio están relacionadas según:



Se puede establecer una conexión por parte de cualquiera de las dos estaciones. Para más detalles, ver [Tur93b].

Descripción del Protocolo

El protocolo opera sobre un medio de comunicación no fiable, full-duplex. Las dos estaciones se comunican mediante el intercambio de Unidades de Datos de Protocolo, PDUs. La comunicación proporcionada es por el protocolo es bidireccional y simultánea, simétrica y fiable. Esta comunicación se divide en cuatro fases: conexión, transferencia de datos, desconexión y tratamiento de errores.

Sólo se permiten las siguientes PDUs:

PDU	significado	primitiva del servicio correspondiente
CR	Connection Request	ConReq, ConInd
CC	Connection Confirmation	ConResp, ConConf
DT	Data Transfer	DatReq, DatInd
AK	Acknowledgment	—
DR	Disconnection Request	DisReq, DisInd
DC	Disconnection Confirmation	—

Las únicas PDUs que transportan parámetros son DT y AK: ambas llevan un número de secuencia de un bit. Además, DT transporta una SDU. Cada SDU del Abracadabra es transportada en una única DT.

El protocolo Abracadabra esta parametrizado por dos constantes: N $(N > 0)$ que define el máximo número de intentos de transmitir una PDU sin recibir el asentimiento correspondiente y P (que debe ser mayor que el retardo de tránsito) que define el periodo de tiempo a esperar antes de intentar una retransmisión.

Para ver en detalle las fases del protocolo y su comportamiento, ver [Tur93a].

Servicio de Comunicaciones del Medio El Servicio de Comunicaciones del Medio opera entre dos estaciones denominadas como A y B. Este Servicio es no orientado a conexión, y se accede a el mediante las Primitivas del Servicio UnitReq y UnitInd (Petición e Indicación). Las Primitivas del Servicio que llevan SDUs corresponden a las

PDUs del protocolo Abracadabra. El medio de comunicación es full-duplex y transparente, pero no garantiza entrega. Los mensajes puede perderse, pero no se desordenarán, ni se corromperán ni se inventaran o duplicarán. Las primitivas del Servicio son:

C.2.2 - Interfaz de la Especificación

La especificación tiene dos puertas externas que son la interfaz del sistema con el entorno: `upper` para intercambiar primitivas de servicio y `lower` para intercambiar PDUs.

Se han especificados tres grandes bloques de tipos de datos: Protocol Data Units (entre entidades), Upper Service Data (a intercambiar con el Usuario del Servicio de Abracadabra) y Lower Service Data (a intercambiar con el proveedor del Servicio de Comunicaciones).

El siguiente cuadro detalla los patrones de acción de las ofertas en las puertas externas:

1)	upper	ASP		
2)	upper	ASP		mess
3)	lower	PDU		
4)	lower	PDU	bit	
5)	lower	PDU	bit	mess

El primer caso es para Primitivas de Servicio sin parámetros (conexión y desconexión). El segundo para primitivas de servicio con parámetros (`DatReq` y `DatInd`).

El tercer caso es para PDUs sin datos. El cuarto para asentimientos de datos (sólo necesita el bit de secuencia). El último caso es para PDUs con datos (bit de secuencia y datos a transportar).

La especificación está parametrizada por dos valores: el número de retransmisiones (`N`) y el periodo de tiempo entre retransmisiones (`P` en unidades de tiempo). Se usan números naturales para modelar ambas variables.

Para obtener más información sobre la especificación LOTOS del protocolo Abracadabra, se recomienda [TF93, MF93].

C.2.3 - Cálculo de Posibles Combinaciones

Para asignar una métrica *a priori* a una prueba dada, existe una herramienta llamada `cober` que, dada una especificación, calcula todas las posibles combinaciones. Además,

esta herramienta proporciona información suficiente para generar el árbol de sincronización mediante un simple preprocesado. También se ha construido un prototipo para representar gráficamente éste árbol.

Dado que la herramienta permite cortar el árbol por profundidad y el protocolo es bastante grande, vamos a presentar los resultados obtenidos de analizar la fase de conexión.

Las combinaciones ofrecidas son:

```
EXIT <> {32,171}
lower <CMSP, PDU> {33}
upper <SP> {34}
upper <SP> {35}
EXIT <> {42,171}
EXIT <> {42}
EXIT <> {29,171}
upper <SP> {30}
EXIT <> {40,171}
EXIT <> {40}
lower <CMSP, PDU> {51}
ERROR <> {54,170}
lower <CMSP, PDU> {43}
lower <CMSP, PDU, bit, MESS> {47}
lower <CMSP, PDU> {49}
lower <CMSP, PDU, bit> {45}
lower <CMSP, PDU> {41}
retry <> {199,52}
kill <> {202,55}
lower <CMSP, PDU> {26}
start <> {2,27}
upper <SP> {28}
lower <CMSP, PDU> {38}
lower <CMSP, PDU> {36}
```

El resultado del prototipo que nos da la jerarquía de acciones que en definitiva constituye el LTS de la especificación se presenta en la figura C.2.

Cuya leyenda es:

L1:
L2: upper ! ConReq
L3: i (* start *)
L4: lower ! UnitReq ! CR
L5: i (* kill *)
L6: i (* ERROR *)
L7: i (* retry *)
L8: lower ! UnitReq ! CR
L9: lower ! UnitInd ! CR

```
L10: EXIT
L11: upper ! ConConf
L12: EXIT
L13: EXIT
L14: upper ! ConConf
L15: lower ! UnitInd ! AK ? : bit
L16: lower ! UnitInd ! DC
L17: lower ! UnitInd ! DT ? : bit ? : MESS
L18: lower ! UnitInd ! CC
L19: EXIT
L20: EXIT
L21: lower ! UnitInd ? : PDU [not((isCR(PDU) or isDR(PDU))) = true]
L22: lower ! UnitInd ! CR
L23: upper ! ConInd
L24: upper ! ConResp
L25: lower ! UnitReq ! CC
L26: EXIT
```

Este árbol de sincronización puede facilitar enormemente la tarea de elaboración y extracción de trazas orientadas a cubrir el 100% de cobertura *a priori*.

Se ha pasado la herramienta a toda la especificación, obteniéndose un total de 192 posibles combinaciones, para una profundidad máxima de exploración de 40 acciones. Se han explorado 3371 estados expandidos (lo que explica la imposibilidad de presentar aquí el árbol), en un tiempo aproximado de 1 hora con una Sparc 10 (Sun Os 4.1.3) descargada con 32M de memoria principal y 210 M de espacio de swap.

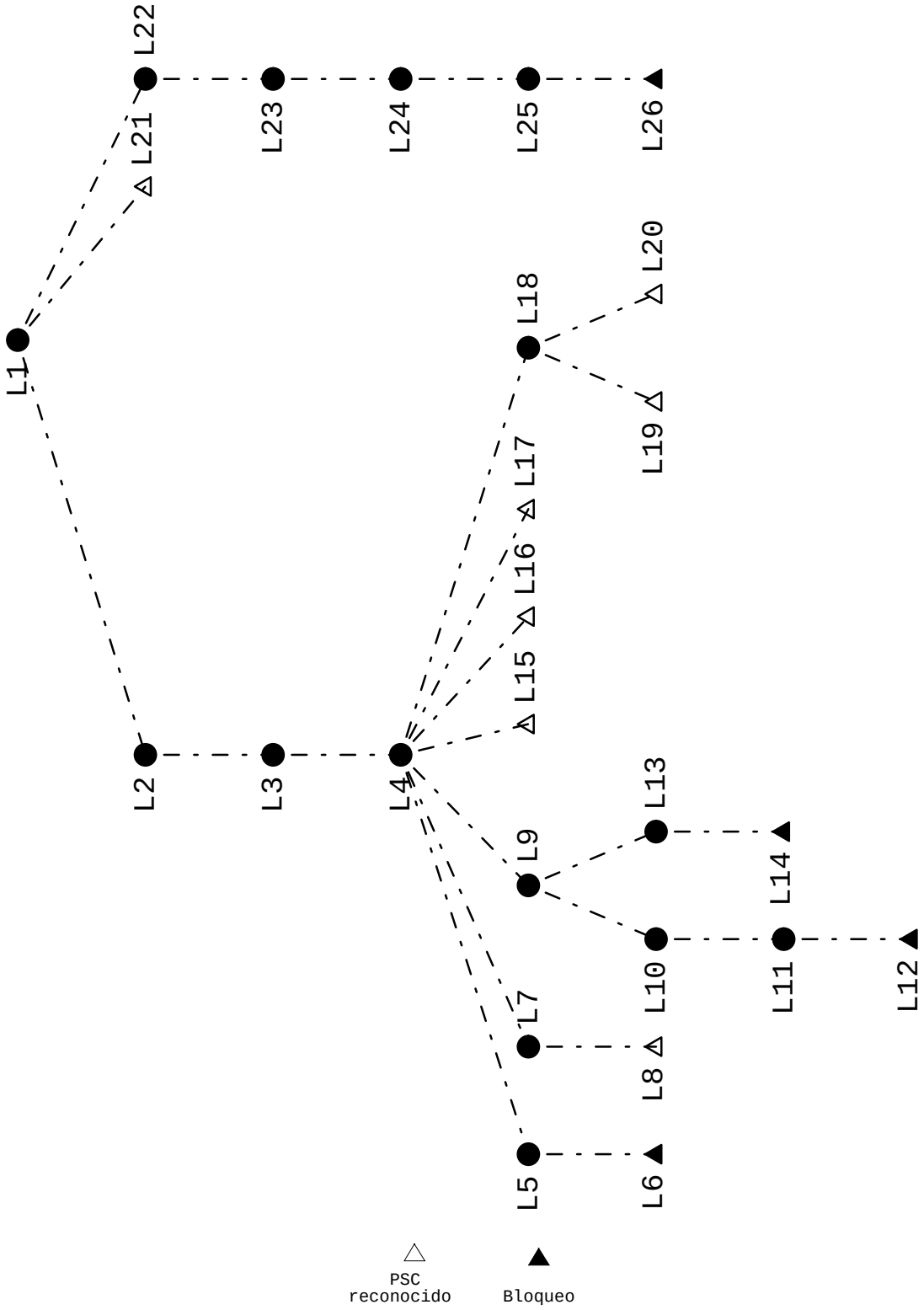


Figura C.2: Árbol de Sincronización del protocolo Abracadabra, expandido hasta localización de PSCs repetidos

Bibliografía

[Ald90] Rudie Alderden. COOPER: The Compositional Construction of a Canonical Tester. In Vuong [Vuo90], pages 13–17. Proceedings FORTE'89, 5–8 December, 1989.

[AR86] A. Alabau and J. Riera, editors. *Teleinformática y Redes de Computadores*. Mundo Electrónico. Marcombo, Boixareu editores, 2 edition, 1986.

[BAL+91] Ed Brinksma, Rudie Alderden, Rom Langerak, Jeroen van de Lagemaat, and Jan Tretmans. A Formal Approach to Conformance Testing. In Jan Kroon, editor, *4th Intl. Workshop on Protocol Test Systems*, pages 349–363, The Hague (NL), October 1991. IFIP.

[BD87] Stephan Budkowski and P. Dembinski. An introduction to estelle: A specification language for distributed systems. *Computer Networks and ISDN Systems*, 14(1):3–23, 1987.

[Bei83] Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold Co., New York, 1983.

[BH89] Ferenc Belina and Dieter Hoegrefe. The CCITT–Specification and Description Language SDL. *Computer Networks and ISDN Systems*, 16(4):311–341, Mar 1989.

[BHS91] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with Applications from Protocol Specification*. Prentice Hall Int'l, Englewood Cliffs, N.J., 1991.

[Blo92] John Bloomer. *Power programming with RPC*. O'Reilly & Associates, Inc., 1992.

[Bri85] Ed Brinksma. A Tutorial on LOTOS. In M. Diaz, editor, *Protocol Specification, Testing, and Verification V*, pages 171–194. IFIP, Elsevier Science B.V. (North-Holland), June 1985.

[Bri87a] Ed. Brinksma. On the existence of canonical testers. Technical Report INF-87-5, Tech. Hogeschool Twente, January 1987.

[Bri87b] Ed Brinksma. On the Existence of Canonical Testers. Memorandum INF-87-5, Univ. Twente, Enschede, The Netherlands, Dec 1987.

[CCI88] CCITT. Specification and Description Language (SDL), volume Blue Book X.1 of Recommandation Z.100. Technical Report Z.100, CCITT, 1988.

[Dil91] F. Dilonardo. A Comparison Between Conformance, Interoperability Performance, Development Testing. In CEN/CENELEC & ETSI, editor, *Conformance Testing and Certification in Information Technology and Telecommunications*, pages 377–384. IOS Press, 1991. Proc. of the European Conf., Brussels, 13–15 June, 1990.

[dNH84] Rocco de Nicola and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Fra87] John B. Fraleigh. *Algebra Abstracta*. Addison-Wesley Iberoamericana, Wilmington, Delaware, U.S.A., 1987.

[Hoa94] C.A.R. Hoare. Unified theories of programming. July 1994.

[Hor94] Horgan, Joseph R. and Londoan, Saul and Lyu, Michael R. Achieving Software Quality with Testing Coverage Measures. *Computer*, 27(9):60–69, September 1994.

[ISO83] ISO. Programming Language - Pascal. First Edition. IS 7185, ISO, 1983.

[ISO84] ISO. Information technology - open systems interconnection - basic reference model. IS 7498, ISO, 1984.

[ISO88] ISO. *Working Draft for the Guidelines for Application of Estelle, LOTOS and SDL*. ISO/IEC JTC 1/SC 21/N2549. International Standards Organization, March 1988.

[ISO89a] ISO. Information processing systems - Open systems Interconnections - ESTELLE, a Formal Description Technique based on an Extended State Transition Model. IS 9074, ISO, 1989. [published 15/jul/1989].

[ISO89b] ISO. Information processing systems - Open systems Interconnections - LOTOS a Formal Description Technique based on the Temporal Ordering of Observational Behaviour. IS 8807, ISO, 1989.

[ISO91a] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework. IS 9646, ISO, 1991.

[ISO91b] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 1: General Concepts. IS 9646, ISO, 1991.

[ISO91c] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 2: Abstract Test Suite Specification. IS 9646, ISO, 1991.

[ISO91d] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. IS 9646, ISO, 1991.

[ISO91e] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 4: Test Realization. IS 9646, ISO, 1991.

[ISO91f] ISO. Information Processing Systems - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 5: Requirements on Test laboratories nad Clients for the Conformance Assessment Process. IS 9646, ISO, 1991.

[Lin86] Richard J. Linn. Testing to Assure Interworking of ISO/OSI Protocols. *Computer Networks and ISDN Systems*, 11(4):277–286, April 1986.

[Lin88] Richard J. Linn. An Overview of Conformance Testing Methodology. In Sudhir Aggarwal and Krishan Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, Atlantic City, New Jersey, USA, June 1988. IFIP, Elsevier Science B.V. (North-Holland).

[Lin89] Richard J. Linn. Conformance Evaluation Methodology and Protocol Testing. *IEEE Jou. on Selected Areas in Communications*, 7(7):1143–1158, September 1989.

[LJ87] Richard J. Linn Jr. Tutorial on the features and facilities of estelle. Technical report, National bureau of Standards, Gaithersburg, MD 20899, U.S.A., August 1987.

[Mad91] Eric Madelaine. Lotos to basic lotos projections. Technical Report Lo/WP2/T2.2/INRIA/N0004, Esprit Project Lotosphere, 1991.

[Mad92] Eric Madelaine. Batch transformation tool. Technical Report Lo/WP2/T2.2/INRIA/N0008v2, Esprit Project Lotosphere, 1992.

[MdM89] José A. Mañas and Tomás de Miguel. From LOTOS to C. In Ken J. Turner, editor, *Formal Description Techniques, I*, pages 79–84, Stirling, Scotland, UK, 1989. IFIP, North-Holland. Proceedings FORTE'88, 6–9 September, 1988.

[MdMSA93] José A. Mañas, Tomás de Miguel, Joaquín Salvachua, and Arturo Azcorra. Tool Support to Implement LOTOS Formal Specifications. *Computer Networks and ISDN Systems*, 25(7):815–839, February 1993.

[MF93] José A. Mañas and Fernando Fournón. Abracadabra Protocol: Formal Description in LOTOS. In Turner [Tur93c], chapter 9.4, pages 290–315.

[MH85] Jan de Meer and Klaus Peter Hasler. Tutorial on act one. Technical report, Jun 1985.

[Mn88a] J.A. Mañas. A Tutorial on ADT semantics for LOTOS users. Part I: Fundamental Concepts. Technical Report , Dpt. Telemática, UPM, Nov 1988.

[Mn88b] J.A. Mañas. A Tutorial on ADT semantics for LOTOS users. Part II: Operations on Types. Technical Report , Dpt. Telemática, UPM, Nov 1988.

[MN91] Carlos Miguel Nieto. *Técnicas de Descripción Formal Aplicadas a la Evaluación de Prestaciones de Sistemas de Telecomunicación*. PhD thesis, Universidad Politécnica de Madrid, Madrid, 1991.

[Mor90] Juan José Morán. Implementación de un Generador de Pruebas para LOTOS. Master's thesis, Dpt. Ingeniería Telemática, ETSITM, Tech. Univ., Madrid (ES), 1990. In Spanish.

[Mye79] Glenford J. Myers. *The Art of Software Testing*. Wiley, New York, 1979. [En Español: El Ateneo, 1983, Madrid (ES)].

[N6294] ISO/IEC JTC1/SC21 N6201. Information retrieval, transfer and management for osi, formal methods in conformance testing, working draft. Technical report, Southampton Output, ISO, February 1994.

[NC88] C. J. Nix and B. P. Collins. The Use of Software Engineering, Including the Z Notation, in the Development of CICS. *Quality Assurance*, pages 103–110, Sep 1988.

[Plo81] Gordon D. Plotkin. A structutal approach to operational semantics. *Number DAIMI FN-1 Computing Science Departments, Aarhu*, September 1981.

[QFM87] Juan Quemada, Angel Fernández, and José A. Mañas. LOLA: Design and Verification of Protocols Using LOTOS. In *IBERIAN Conference on Data Communications*, Lisbon, May 1987. Also in Computer Communication Systems A. Cerveira (ed) North-Holland (1988).

[QMV91] Juan Quemada, José A. Mañas, and Enrique Vázquez, editors. *Formal Description Techniques, III*, Madrid (ES), 1991. IFIP, Elsevier Science B.V. (North-Holland). Proceedings FORTE'90, 5–8 November, 1990.

[Rei93] Gotzhein Reinhard. *Open Distributed Systems: On Concepts, Methods, and Design from a Logical Point of View*. PhD thesis, University of Hamburg, Hamburg, April 1993.

[Rob91] Tomás Robles. *Contribución al Tratamiento Formal de la Fase de Pruebas del Ciclo Software en Ingeniería de Protocolos*. PhD thesis, Dpt. Ingeniería Telemática, ETSITM, Tech. Univ., Madrid (ES), 1991. In Spanish.

[Sal93] Joaquín Salvachúa. *Arquitectura de Programas a Partir de Especificaciones Formales*. PhD thesis, Universidad Politécnica de Madrid, Madrid, 1993.

[Spi89] J. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Hemel Hempstead (UK), 1989.

[Tar73] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, September 1973.

[TF93] Alistair J. Tocher and Fernando M. Fournón. Abracadabra Service: Formal Description in Lotos. In Turner [Tur93c], chapter 8.4, pages 242–257.

[Tre90] Jan Tretmans. Test Case Derivation from LOTOS Specifications. In Vuong [Vuo90], pages 345–359. Proceedings FORTE'89, 5–8 December, 1989.

[Tre92] Jan Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, Enschede, Holland, December 1992.

[Tur93a] Ken J. Turner. Abracadabra Protocol. In Turner [Tur93c], chapter 9, pages 267–327.

[Tur93b] Ken J. Turner. Abracadabra Service. In Turner [Tur93c], chapter 8, pages 231–266.

[Tur93c] Kenneth J. Turner, editor. *Using Formal Description Techniques*. John Wiley & Sons, 1993.

[vEE91] Peter van Eijk and Henk Eertink. Design of the LotoSphere Symbolic LOTOS Simulator. In Quemada et al. [QMV91], pages 577–580. Proceedings FORTE'90, 5–8 November, 1990.

[Vei93] Marcelino Veiga. GLAD: General Language to Annotate Data using TOPO. Technical report, Dpt. Telematics Engineering Technical Univ. Madrid, Ciudad Universitaria, E-28040 Madrid, Spain, July 1993. Version 3R2.

[Vuo90] Son T. Vuong, editor. *Formal Description Techniques, II*, Vancouver (CA), 1990. IFIP, Elsevier Science B.V. (North-Holland). Proceedings FORTE'89, 5–8 December, 1989.

[WBL91] Clazien D. Wezeman, S. Batley, and J. A. Lynch. Formal Methods to Assist Conformance Testing. A Case Study. In Quemada et al. [QMV91], pages 157–174. Proceedings FORTE'90, 5–8 November, 1990.

[Wez88] Clazien Wezeman. The “CO-OP method”: a method for compositional derivations of canonical testers. Master’s thesis, Faculteit del Informatica, Univ. Twente, Enschede, The Netherlands, August 1988.

[Wez89] Clazien D. Wezeman. The CO-OP Method for Compositional Derivation of Conformance Testers. In Ed Brinksma, Giuseppe Scollo, and Chris A. Vissers, editors, *Protocol Specification, Testing, and Verification IX*, pages 145–158, Enschede (NL), June 1989. IFIP, Elsevier Science B.V. (North-Holland).

[Wir71] Niklaus Wirth. The programming language pascal. *Acta Informatica*, 1:35–63, 1971.

Curriculum Vitae

Ocubre 88 - Octubre 89:	Proyecto Fin de Carrera: *Especificación y Validación del Protocolo SLAP para Columbus*. Calificación: Matrícula de Honor. Becado por el Dpto. de Ingeniería de Sistemas Telemáticos de la E.T.S.I.T.M. y por la empresa CRISA.
Octubre 89:	Título de Ingeniero de Telecomunicación por la Universidad Politécnica de Madrid. Calificación: NOTABLE.
Octubre 89-Febrero 90:	Ingeniero de Desarrollo en la empresa CRISA.
Febrero 90- :	Investigador adscrito al Departamento de Ingeniería de Sistemas Telemáticos en la E.T.S.I.T..
Septiembre 94- :	Profesor Titular de la Universidad Alfonso X el Sabio.