UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Sistemas Informáticos y Computación

TESIS DOCTORAL

Semántica de simulación para relaciones de conformidad

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Rafael Martínez Torres

Directores

Luis LLana Díaz
Carlos Gregorio Rodríguez

Madrid, 2016

Semántica de Simulación para Relaciones de Conformidad

Memoria presentada para la obtención del título de doctor por

Rafael Martínez Torres

Trabajo dirigido por

Luis Llana Díaz
Carlos Gregorio Rodríguez

Universidad Complutense de Madrid
Departamento de Sistemas Informáticos y Computación

Tesis doctoral presentada por el doctorando Rafael Martínez Torres en el *Departamento de Sistemas Informáticos y Computación* de la *Universidad Complutense de Madrid* para la obtención del título de doctor en Ingeniería Informática.

Terminada en Madrid a 30 de septiembre de 2015.

Título:

Semántica de simulación para relaciones de conformidad

Doctorando:

Rafael Martínez Torres (rmartine@fdi.ucm.es)
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid
Facultad de Informática

Directores:

Luis Fernando Llana Díaz (llana@sip.ucm.es)
Carlos Gregorio Rodríguez (cgr@sip.ucm.es)

Agradecimientos

A Luis Llana Díaz, Carlos Gregorio Rodríguez y Manuel Núñez García.

Al contribuyente.

Resumen

Esta tesis trata de las llamadas relaciones de conformidad *que pueden darse entre dos sistemas cualesquiera,* especificación *e* implementación*. Como novedad introduce el uso de técnicas coinductivas para la definición de* iocos*, una relación de conformidad capaz de distinguir el contexto local de ejecución de un proceso. Al constituir* iocos *la parte central de una nueva teoría de* Model Based Testing *(***MBT***), se precisa de una descripción formal de los sistemas en juego; esto se lleva a cabo en primera instancia mediante sistemas de transiciones etiquetadas y posteriormente mediante un enfoque más abstracto, un álgebra de procesos.*

Las teorías de testing *tienen por objeto confirmar las relaciones de* conformidad *mediante la ejecución de un conjunto de* tests *sobre un sistema –la* implementación*– cuya estructura interna se desconoce. Particularmente los beneficios de un enfoque* **MBT** *son inmediatos, ya que la generación de los* tests *puede abordarse de manera* sistemática *una vez se disponga de un modelo formal de la* especificación*, lo que contribuye a eliminar el error imputable al factor humano; esto se logra dando un algoritmo generador de* tests *que toma cono entrada una especificación y produce un conjunto de* tests*, posiblemente infinito, suficientemente representativo para asegurar la relación de* conformidad*. Este enfoque inicial, conocido como* testing offline *o* testing estático*, es mejorado para ganar en eficiencia evitando un alto consumo en recursos tanto de tiempo como de memoria mediante la técnica de* testing online *o* testing dinámico*, donde ambos pasos de generación y ejecución se ejecutan de manera alternada.*

La Algoritmia también desempeña un papel importante cuando los modelos de la implementación son disponibles. Basándose en resultados clásicos del procesamiento de grafos, se propone una variante del Generalized Coarsest

Partition Problem, ***GCPP***, *o Problema Generalizado de la Partición más Grande. El planteamiento que subyace a todo ello es simple: adoptando una estrategia de reducción,* iocos *es redefinida en un nuevo entorno bajo unas condiciones que pueden ser computadas eficientemente por el problema* ***GCPP****. Como resultado, se dispone una implementación ejecutable integrada en el marco de la conocida plataforma* ***mCRL2****. También se exponen algunas estadísticas obtenidas a partir de experimentos con casos de estudios de gran tamaño.*

Por último, se proporciona un cálculo *correcto y completo al objeto de* deducir *la conformidad a partir de algunos resultados conocidos. Para lograrlo, es menester definir un sencillo lenguaje de procesos cuya congruencia con respecto a* iocos *es contrastada. Las pruebas pertinentes de corrección y completitud se proporcionan en primer lugar para reglas y fórmulas que contienen procesos finitos; en base a sucesivas aproximaciones finitas se lleva a cabo la corrección de la parte del cálculo que considera procesos infinitos.*

Abstract

This thesis is about conformance *relations you might set up between any two given systems, namely* specification *and* implementation. *As a novelty it introduces coinductive techniques to define* iocos, *a* conformance *relation enabled to track local context of process execution.* iocos *relying at the core a of a new Model Based Testing (***MBT***) Theory, a formal description of the systems is required; this is accomplished, firstly, by means of* labeled transition systems *and then using a more abstract approach, a* process algebra.

Testing theories are designed to state conformance *relations by running a set of tests on an system –the* implementation*–, whose internal details are unknown. In particular, benefits of a* **MBT** *approach are immediate, since test generation can be achieved* systematically *provided a formal model of the* specification*, hence avoiding error-prone human factor; this is achieved by giving a test-generator algorithm accepting an specification as input, and yielding a set of tests, possibly infinite, large enough to state the conformance. Such an initial approach,* testing offline*, is improved to avoid waste of time and space in the form of* testing online*, where both test generation and execution steps are interleaved.*

Algorithms also play an important role when models of implementation are available. Rooted in classic results from graph processing, a variant of the Generalized Coarsest Partition Problem *(**GCPP**) is proposed to solve the question of* iocos conformance. *The rationale behind that is simple: adopting a reduction strategy,* iocos *is redefined on a new environment whose conditions can be efficiently* computed *by **GCPP**. As a result, an executable implementation of* iocos *is available running inside the platform **mCRL2**. Some statistics obtained from experiments involving huge scenarios are commented.*

Lastly, a sound and complete calculus *is provided on the purpose of* deducing

conformance from some known results. In order to achieve that a simple process language is defined, whose congruence respect to iocos *is checked. Proofs of soundness and completeness are provided for those rules involving formulas with finite processes; the ones involving infinite processes are achieved on the basis of arbitrary finite approaches.*

Índice general

Capítulo 1

Introducción

En este primer capítulo ponemos en contexto los contenidos desarrollados a lo largo de los capítulos posteriores que integran esta tesis, describiendo tangencialmente las diversas áreas de la ciencia de la computación en las que se enmarcan. El resultado final ha sido un exhaustivo trabajo capaz de articular diversos aspectos teóricos en torno al estudio de sistemas interactivos que van desde el *testing* formal, hasta la formulación de un cálculo deductivo que permite inquirir si dos sistemas están relacionados conforme un determinado criterio, diseñado *ex profeso*, pasando por tradicionales métodos algorítmicos que permiten, utilizando teorías de transformaciones de modelos, *computar* de modo eficiente esa relación.

Al final de este capítulo, se expone una declaración de los objetivos originales que se pretenden abarcar, añadiendo unas notas relativas a los criterios que hemos seguido a la hora de tratar la enorme cantidad de términos en lengua inglesa, lo que ha de contribuir a completar el sentido del mensaje –necesariamente técnico y especializado– en nuestro idioma.

1.1. Contexto disciplinar

1.1.1. Testing y sistemas interactivos

En su origen, la principal aplicación de la informática era el diseño y posterior implementación de programas que permitieran la resolución automática de problemas de cálculo numérico, dando soporte a otras disciplinas que así

lo requerían como la física en el cálculo de trayectorias, o el procesamiento estadístico de datos por parte de las ciencias sociales.

A partir de la década de 1970 la aparición de los primeros sistemas operativos multitarea, destinados a hacer un uso más eficaz del costoso *hardware*, así como la necesidad de poner en comunicación los diversos sistemas informáticos pone de relieve nuevos problemas que surgen de la diferencia de velocidades entre dispositivos o la latencia de los medios de comunicación empleados.

En general, el estudio de la problemática relativa a este nuevo tipo de sistemas queda englobado bajo el concepto general de *procesos concurrentes*. De modo más específico, cuando queremos hacer énfasis en las capacidades de interacción de los sistemas con el medio que los rodea, ya sea actuando de manera autónoma o como reacción a estímulos presentados desde el exterior, empleamos el término *sistemas interactivos*. Una breve introducción a la forma en que éstos se pueden especificar mediante *sistemas de transiciones etiquetadas LTS (labeled transition systems)* se puede encontrar en el capítulo 2.

De modo más particular, las distintas formas en que podemos percibir el comportamiento de los mismos dan lugar a distintas *semánticas formales*, las cuales aportan un valor añadido al ser susceptibles de automatizarse con suficientes garantías de corrección, aumentando la eficiencia a la hora de someter a escrutinio los sistemas.

La relación semántica en torno a la cual gira esta tesis es iocos, la cual será ampliamente descrita en el capítulo 3. Dicha semántica está inspirada por otra existente, ioco, ampliamente conocida en el mundo del *Testing Basado en Modelos,* **MBT** y de corte lineal, en el sentido de que no permite distinguir el contexto local de ejecución en presencia del indeterminismo. Gracias a las definiciones coinductivas, lograremos un refinamiento de la relación original ioco que sí permita una mayor precisión, más sensible a la contingencia o necesidad con que se produce una acción en un estado concreto dependiendo de la elección indeterminista que haya tenido lugar para llegar a ese estado. Por esta razón el capítulo 3 ocupa un lugar central en la tesis a partir del cual se pueden entender los siguientes.

Con anterioridad hemos aludido al término *testing* para situar los precedentes de nuestra relación objeto, iocos. Bajo cierta perspectiva histórica, y con las precauciones que hay que tomar a la hora de poner en relación disciplinas

tan dispares, diríase que el *testing* supone a la ciencia de la computación lo que la *termodinámica* para la física; en los dos planteamientos prescindimos de la estructura interna de un sistema, y solamente registramos los comportamientos que éste nos muestra de manera autónoma o como reacción a estímulos desde el exterior.

La valoración de los resultados se hace con respecto a una sistema de *referencia* al que conocemos como *especificación*. Ésta, a su vez, servirá como semilla para un algoritmo generador de *tests*, los cuales son correctos en el sentido de que, al arrojar un resultado negativo, tenemos la certeza de que el sistema inspeccionado no tiene una funcionalidad conforme a la especificación. Para certificar la propiedad dual, la de ser correcto con respecto a la especificación, generalmente son necesarios un número infinito de *tests*, lo que hace cierto aquel *adagio* que gobierna las ciencias empíricas: *un experimento sólo detecta errores en la teoría, no la ausencia de los mismos.*

Por todo ello, resulta de particular interés la aplicación de semánticas *branching* o sensibles al contexto local de ejecución a las técnicas de *testing*. Nuestra relación iocos es la respuesta a este reto, lo que supone un incremento de la complejidad con respecto de las más tradicionales semánticas lineales. Todo ello será el objeto del capítulo 4.

1.1.2. Técnicas algorítmicas basadas en grafos

Si bien el incremento de las capacidades de procesamiento disponibles hoy en día hace posible el abordaje de problemas impensables en épocas pretéritas, los más elementales estudios sugieren que confiar todo al potencial de cálculo del *hardware* no es una buena opción si se tiene en cuenta el criterio asintótico que gobierna la complejidad de un problema.

Este es el caso que nos ocupa, pues al tratar con modelos tan elementales como los *sistemas de transiciones etiquetadas* fácilmente los algoritmos degeneran en una complejidad exponencial. Toda tentativa de mejorar la eficiencia del cómputo pasa por considerar estrategias que permitan una *simplificación* de los modelos de entrada. Sobre la base de otros problemas similares, como la *bisimulación* y de manera mas próxima, la *simulación*, se puede plantear una solución basada en el *cómputo de grafos*; mediante este procedimiento los sistemas son transformados en grafos cuyas nodos están formados por clases

de equivalencia –abstrayendo por tanto el carácter individual de cada estado– y cuyas aristas se van constituyendo gracias a un criterio de *estabilidad*, un criterio nada trivial que será esbozado en el capítulo 2 y adaptado para nuestro propósito en el capítulo 5.

En esencia, la estabilidad registra una suerte de *solidaridad* entre las transiciones de los diferentes estados: se empieza considerando un sólo nodo agrupando todos los posibles estados de los sistemas, en la esperanza de que, del mismo modo que otros algoritmos clásicos, como el *quick sort*, las clases de equivalencia –poco numerosas– agrupen muchos elementos por una cuestión estadística, disminuyendo así la complejidad.

1.1.3. Sistemas de deducción

En ocasiones podemos interesarnos no tanto la confirmación de las propiedades de dos sistemas, uno tomado por referencia y el otro del que se espera acreditar su *conformidad* con respecto a aquél, ya sea empíricamente, como señalábamos en la sección 1.1.1, o bien mediante cómputos, descritos en la sección anterior, sino poniendo en conexión resultados sobre configuraciones conocidas para extenderlas a otras nuevas.

Este es el escenario donde entra el papel de la *deducción lógica*, lo que nos exige la introducción de un *lenguaje* para poder referirnos al mundo mediante fórmulas así como la definición sobre él de un *cálculo* adecuado que nos permita inferir nuevas fórmulas a partir de otras dadas. Ambos elementos son el objeto del capítulo 6.

Para el lenguaje nos dotaremos de una sencilla sintaxis pero con una expresividad suficiente para nuestro objetivo, variante de alguna de las tradicionales *álgebras de procesos* que tradicionalmente se han empleado en el área.

En lo relativo al segundo aspecto de nuestro sistema deductivo introduciremos un *cálculo inecuacional*, el cual, mediante acreditados resultados de *corrección* y *completitud*, nos ha de garantizar no sólo la consistencia de las fórmulas inferidas con respecto a las propiedades de los objetos que representan, sino la garantía de que, puestos en relación dos objetos mediante nuestra semántica, se ha de ser capaz de inferir una fórmula que de cuenta de ello basándose únicamente en la aplicación de reglas propias de inferencia a partir de unos axiomas.

Mención especial merece el tratamiento de procesos de naturaleza infinita, situación resuelta al modo tradicional recurriendo al estudio de infinitos casos que se obtienen al considerar sus aproximaciones finitas de arbitraria longitud.

1.1.4. Implementación y experimentos

No se puede dejar pasar por alto uno de los aspectos que más esfuerzo ha consumido a lo largo de este trabajo; pese a su marcado carácter teórico, entroncado con la naturaleza *deductiva* de las matemáticas, la realización práctica de los algoritmos mediante su implementación y ejecución *empírica* en plataformas de curso ordinario puede resultar de gran ayuda para la comunidad de investigadores, a fin de valorar los resultados ya sea procesándolos estadísticamente, o representándolos mediante gráficos que proporcionan una asimilación visual más rápida.

Esta tarea, lejos de resultar trivial, requiere un exhaustivo conocimiento no sólo de lenguajes de programación sino de herramientas complementarias que contribuyen a hacer más *productivo* el proceso –algunas de las mismas requieren *años* de constante práctica– sin las cuales las esperanzas de llevar a la realidad los diseños tan celosamente elaborados desde el punto de vista teórico quedan relegadas a una vaga *quimera*, en una *idea fácil* que no requiere esfuerzo.

Sistema operativo Unix, lenguaje C++, compilador gcc, memoria RAM, memoria swap... La ingente profusión de detalles que dominan este tipo de tecnologías no contribuye a una fácil comprensión de lo *esencial*, razón por la cual se ha optado en los capítulos centrales por el empleo del *pseudo-lenguaje*, más cercano a su comprensión por parte de la inteligencia humana. Por supuesto, el producto final nuestro trabajo está disponible para su descarga electrónica en Internet. A fin de cuentas, su objeto es su procesamiento por computador, en tanto que este texto aspira a ser leído por personas.

1.2. Sobre esta tesis

1.2.1. Alcance y objetivos

Esta tesis pretende profundizar en las relaciones de *conformidad* [59, 78, 19, 25] que se pueden proponer entre una especificación y sus posibles implementaciones desde una nueva óptica, pues en lugar de considerar como pilar básico las trazas de señales, planteamiento de análisis entroncado con la teoría clásica de los autómatas como reconocedores de lenguajes, utiliza una definición más sofisticada basada en técnicas coinductivas. Consecuencia de todo ello será la descripción en el capítulo 3 de una nueva relación de conformidad –iocos– refinamiento de una relación previa existente ioco [83, 82, 79, 80] que describiremos más detalladamente en el capítulo 2, junto con otros conceptos y teorías de las que esta tesis se ha nutrido.

A lo largo del capítulo 4 describiremos cómo dicha relación puede ser detectada mediante técnicas de **MBT**, proponiendo dos variantes: en la primera, un algoritmo generador de *tests* con la propiedad de ser representativo de aquel principio que pone en conexión la noción formal de nuestra relación con *todos* los posibles *tests* aplicables tanto al sistema referencia como al que objeto de inspección; en la segunda, vinculando la generación dinámica de primitivas de *test* al estado de actual del sistema inspeccionado después de someterlo a una primitiva particular. Para el diseño de nuestros algoritmos resultó fundamental la contribución de los trabajos de Abramsky [3] caracterizando la equivalencia observacional, una semántica de corte coinductivo.

Con respecto a los *métodos algorítmicos*, en el capítulo 5 abordaremos un método para el cómputo de nuestra relación de conformidad, para lo cual nos apoyaremos en los trabajos de R. Gentilini [32] y en las siguientes modificaciones al mismo de Rob J. van Glabbek [91]. Al objeto de facilitar la transición a nuestra relación objetivo, plantearemos la solución al problema de la simulación *ready*, la cual, si bien no es propiamente original de esta tesis, reúne unas características intermedias entre la simulación ordinaria y la que es objeto de nuestro estudio, iocos.

Los resultados de la implementación permanecerán disponibles para la comunidad investigadora en Internet en formato de programa fuente para la plataforma Linux, extendiendo el conjunto de herramientas entorno **mCRL2** [21,

55, 42]. Una valoración de los resultados y distintas métricas relativas a considerar con respecto a la *suite* de datos de entrada, **VLTS** [95] nos permitirá contrastar la calidad de nuestra solución con las existentes en la mencionada plataforma **mCRL2**.

Las bases de nuestra teoría originalmente serán formuladas utilizando como modelos los *sistemas etiquetados de transiciones*. Normalmente este tipo de representación tan elemental se caracteriza por la dificultad que plantea la especificación de sistemas complejos, razón por lo que definiremos un lenguaje de alto nivel que nos permita representarlos más cómodamente y que, como es habitual, será dotado de semántica operacional mediante los primeros modelos empleados. Este lenguaje dará cobertura a procesos infinitos, y en base a él se formulará en el capítulo 6 un sistema de axiomas propio capaz de poner en relación iocos dos sistemas arbitrarios especificados en dicho lenguaje mediante procesos de deducción.

Por último en el capítulo 7 haremos una evaluación sobre la consecución de los objetivos que acabamos de proponer y daremos algunas indicaciones sobre aquellos aspectos que por su profundidad escapan al alcance de este trabajo y merecen un estudio detallado aparte en el futuro.

1.2.2. Publicaciones asociadas

El contenido de esta tesis ha dado lugar a publicaciones recogidas en diversos congresos de acreditada reputación (escalas **A**, **B** en el índice de clasificación **CORE**) así como en editoriales especializadas del ramo.

Aunque el material de dichas publicaciones ha sido reorganizado para adecuarlo al formato de esta tesis, la contribución de los capítulos con las publicaciones, sin estar en correspondencia uno a uno, es más o menos la siguiente:

- Los capítulos 3 en su mayoría y 4, en la parte del *testing offline* dieron lugar a una publicación [36] en el congreso FORTE 2013 celebrado en Berlín, Alemania.

- El apartado 3.3 del capítulo 3 es el núcleo de la publicación [61] que tuvo lugar en el encuentro SEFM 2013 en Madrid.

- La parte del capítulo 4 dedicada al *testing online* conjuntamente con los

planteamientos teóricos del capítulo 5 fue reconocida para su publicación [37] entre las actas del congreso FORTE 2014 que tuvo lugar en Florencia, Italia.

- Todo lo relativo a la parte de implementación del capítulo 5 así como el análisis de los resultados presentados fue expuesto en el congreso SAC 2015 celebrado en Salamanca y recogido en forma de publicación en [39].

- Al momento de la edición de esta tesis, el contenido del capítulo 6 es inédito pero con intención de ser propuesto como objeto de publicación de revista especializada.

1.2.3. Cuestiones lingüísticas

Escribir un texto científico en nuestra lengua sobre una materia tan dinámica como la computación, donde el dominio de los países de lengua anglosajona es predominante, no es una tarea trivial. Hay partidarios que sostienen que, puesto que esta tesis se presenta en español se debería hacer un esfuerzo por buscar aquella palabra que, dentro de nuestro léxico, más ajustara a expresar lo que queremos expresar en inglés.

Los resultados de una experiencia tal han resultado frustrantes para aquellos lectores que, siendo hispano-hablantes con acceso a fuentes documentales en inglés, intentan comprender el sentido de aquello que sus compañeros intentan describir. Así los términos *aplicativo, zócalo, octeto, bandera, programador, negocio* resultan ininteligibles en el contexto a las que se refieren sus homónimos en inglés *software, socket, byte, flag, scheduler, bussines*[1]. No obstante, este aspecto no es tan frecuente cuando se trata de escribir sobre materias formales como las matemáticas donde la mayoría de las veces el recurso a las lenguas clásicas como el latín y el griego es más normal, con términos aceptados universalmente como *biyección, homomorfismo, teorema,*....

Las recomendaciones al respecto dictan que siempre que exista un vocablo o término traducción del inglés es preceptivo su empleo, y en aquellos casos

[1]Frecuentemente esta confusión también sucede entre los anglo-parlantes cuando son ajenos a su acepción técnica

en los que se haya adaptado directamente, sin equivalente entre los léxicos de nuestro idioma debe expresarse en *cursiva*.

La opinión que se sostiene en esta tesis es la de seguir en la medida de lo posible las directrices mencionadas al respecto. No obstante, esas recomendaciones se refieren a textos *generalistas*, mientras que efectivamente, el documento que aquí se presenta pertenece a un ámbito científico muy específico, razón por la cual seguir este consejo puede ser contraproducente. Esto puede ocurrir por ejemplo en casos como los siguientes:

ready simulation Resultaría ineficaz la traducción por *simulación preparada, simulación por anticipación*, pese a que ambos términos, en un contexto general, cuentan con vocablos en la familia léxica del español. Optaremos pues, por la expresión mixta: *simulación ready*

branching semantics Al igual que en el caso anterior, optaremos por una solución intermedia, formando la expresión *semántica branching*, en lugar de la propia *semántica ramificada*, en la esperanza de que el lector español pueda evocar los importantes trabajos referentes al tema escritos en lengua inglesa, como [90, 89, 88]

test A no dudarlo es el término más ampliamente referido en esta tesis. Existe una traducción en castellano, *prueba*, pero de nuevo impide su ubicación correcta en el contexto internacional, máxime cuando entronca con el vocablo *proof* en inglés, que denota un matiz distinto sujeto a técnicas de deducción y no de experimentación.

En este documento, pues, se adaptará el siguiente convenio: teniendo en cuenta que el lector ha tenido acceso a las fuentes bibliográficas que se citan, escritas en inglés, sin las cuales es imposible entender el mensaje que se quiere trasladar, se usará el español siempre que aquél término no este tan *ritualizado* entre la comunidad angloparlante que su inclusión en español, conforme a las familias léxicas de nuestro idioma lleve a confusión o dificulte la compresión del mensaje que se quiere citar.

Capítulo 2

Estado del Arte

El hecho de poder distinguir entre dos actividades como *especificar* e *implementar* supone un gran avance en el proceso de desarrollo de *software*, al permitir durante la primera fase la omisión de detalles concretos que pueden ser un obstáculo a la hora de plantear el problema a resolver.

En esta tesis las especificaciones se hacen, en primera instancia, en base a modelos formales muy elementales, los conocidos *sistemas de transiciones etiquetadas*, y con posterioridad mediante *lenguajes algebraicos* que describen aquellas de manera más abstracta. Un repaso a las principales corrientes que se han propuesto históricamente pondrá en contexto el contenido de los capítulos 3 y 6.

El empleo de modelos formales es precisamente lo que caracteriza el llamado *Model Based Testing* (**MBT**), una compilación de técnicas destinada a la evaluación de implementaciones al someterlas a diversos estímulos desde un entorno artificial diseñado conforme a una especificación. Conceptos generales como *relación de conformidad*, *paso de test*, *generación automática de tests* resultan imprescindibles para poder entender el capítulo 4.

Ya a nivel más concreto, en los últimos años alcanzó relevancia una propuesta de **MBT** conocida por ioco –acrónimo de *input ouput conformance*– la cual más tarde se revelará insuficiente desde el punto de vista expresivo ante el concurso del indeterminismo. Para superar esta carencia, la aplicación de los métodos de *testing* requerirá la conexión de esta disciplina con los principios que rigen las conocidas semánticas *branching*, sensibles al contexto local de ejecución. Esto incluye las técnicas de definición coinductivas, de las que histó-

ricamente han surgido relaciones clásicas como la *bisimulación* o la simulación *ready*.

Desde el punto de vista de la algoritmia, el cómputo directo de esta familia de relaciones basadas en simulación es de una complejidad tan alta que en la mayoría de los casos su cálculo se vuelve privativo, razón por la que todo intento por resolver estos problemas pasa por el empleo de estrategias que contemplen una simplificación en el modelo de entrada conservando ciertas propiedades. Se describen las principales tendencias formuladas a salvar este escollo, como el conocido *Problema generalizado de la partición más grande*, **GCPP**, en torno al cual se articula una versión modificada que es el objeto de interés del capítulo 5.

2.1. Formalismos para la descripción de procesos

Desarrollamos a continuación una introducción a los dos tipos de formalismos que se han utilizado para describir sistemas en esta tesis: los llamados *lenguajes de procesos* y los *sistemas de transiciones etiquetadas*, enunciados de mayor a menor abstracción. Ambos enfoques están relacionados, ya que en algún caso particular los segundos contribuyen a dotar de semántica a los primeros, lo cual resulta de particular interés para nuestros futuros planteamientos.

2.1.1. Lenguajes y/o álgebras de procesos

En los albores de la ciencia de la computación, las principales teorías presentaban cálculos cuyo propósito era marcar el límite de las llamadas *funciones computables*, entre las que cabe citar el λ-cálculo y las máquinas de Turing [86]. Desde el punto de vista más pragmático esta época coincide con las primeras aplicaciones de los computadores para el cálculo numérico, criptografía y tratamiento estadístico de la información.

A mediados de los 60, surgió la necesidad de plantear nuevos modelos para razonar sobre otro tipo de fenómenos distintos [65, 64] que incluían la ejecución *concurrente* de programas, la *interacción* que se pudiera dar entre los mismos así como la *comunicación* de valores. Se acuñó el término *proceso* para referirse

al nuevo objeto de estudio relevando al anterior *función*.

Tradicionalmente, tres de las más importantes teorías diseñadas para dar explicación a estos nuevos fenómenos han sido:

- *Calculus of Communicating Processes*
- *Communicating Sequential Processes*
- *Algebra of Communication Process*

de las cuales procedemos a dar una breve reseña histórica.

CCS

Uno los primeros modelos en surgir fue el llamado *Calculus of Communication Systems (**CCS**)* obra de Robin Milner [63, 62]. Incorporaba un lenguaje de procesos cuya sintaxis ya se diferenciaba de la conocida hasta entonces en los lenguajes de programación. Su objeto era la *especificación* de sistemas, frente a aquella cuyo objeto era la *implementación*. A resaltar como una de sus virtudes es que con un conjunto básico de operadores podía sintetizar la variedad de fenómenos observables en torno a un conjunto reducido.

Como todos los lenguajes, éstos quedan determinados al formular unas reglas para la formación de sus términos, *sintaxis*, y una *semántica* para dar una interpretación al término así obtenido. En este último sentido existen varias posibilidades de expresar este concepto: Milner opta por definir una *semántica operacional*, encargada de describir cómo evolucionan los procesos definidos mediante términos y lo hace siguiendo un enfoque estructural en base a unas notas escritas por Gordon D.Plotking [72] en 1981, haciendo uso de los llamados *sistemas de transiciones etiquetadas*, de los que se habla más adelante en el siguiente apartado.

La definición 1 nos marca la sintaxis que rige la formación de términos en el cálculo **CCS**.

Definición 1. Dado un conjunto de nombres de etiquetas $A = \{a, b, c, \dots\}$ que induce el conjunto de conombres $\overline{A} = \{\overline{a}, \overline{b}, \overline{c}, \dots\}$, las acciones de los procesos son elementos del conjunto $Act = A \cup \overline{A}$.

La sintaxis que rige la constitución de los procesos definidos en **CCS** es la siguiente:

- 0 es el proceso inactivo.
- Prefijo de un acción y un proceso, $a.P$
- Elección entre procesos, $P + Q$
- Composición de procesos $P \mid Q$
- Restricción de acciones en un proceso, $P\{l_1, \ldots, l_2\}$
- Reetiquetado de acciones $P[l'_1/l_1, \ldots, l'_n/l_n]$.

CSP

Distinto enfoque seguirá C.A.R Hoare a la hora de publicar su *Communicating Sequential Process,* ***CSP*** [50]. La técnica para dar semántica a los términos obtenidos a partir de su sintaxis es la de asignarles un valor dentro de un *dominio*. Al proceder de esta manera, *denotando* valores, Hoare se permite definir *preórdenes* entre ellos lo que le permite inducir relaciones de equivalencia entre los elementos con gran elegancia.

Cabe señalar que las dos alternativas, el **CCS** operacional de Milner y el **CSP** denotacional de Hoare, lejos de resultar excluyentes, contribuyen a complementarse en su intento por describir la noción de *proceso*.

La figura 2.1 muestra la variedad de operaciones sintácticos propios de **CSP**, junto con una descripción informal del significado de cada uno de ellos.

Notación	**Significado**
$a \to P$	a entonces P
$a \to P \mid b \to Q$	a entonces P en elección con b entonces Q
$x : A \to P(x)$	elección de x en A entonces $P(x)$
$P \mid\mid Q$	P en paralelo con Q
$P \sqcap Q$	P o Q no determinista
$P[]Q$	P en elección con Q

Figura 2.1: Algunos operadores de **CSP**

ACP

Una tercera alternativa de gran relevancia viene dada por el *Algebra of Communicating Processes,* ***ACP*** [5, 10, 9, 11, 6]. Resulta ser la base en torno a la cual se define el lenguaje de una de las herramientas más populares para experimentar con procesos disponible en diversas plataformas, **mCRL2** [55, 21, 53], empleada en los experimentos realizados en el capítulo 5.

Los procesos que considera **ACP** son capaces de ejecutar acciones atómicas $a, b, c \ldots$ Las acciones pueden combinarse y dar lugar a procesos compuestos por las operaciones + y $\cdot$, interpretando que $(a+b)\cdot c$ es el proceso que primero elije entre a o b y, a continuación, ejecuta la acción c después de la cual se acaba. Estás operaciones reciben el nombre de *composición alternativa* y *composición secuencial* y son los constructores básicos de los procesos.

Como el tiempo es orientado, la secuencia resulta no ser conmutativa. No ocurre lo mismo con las acciones que participan en la composición alternativa, cuyas propiedades, *conmutativa*, *asociativa*, e *idempotencia* se pueden sintetizar los siguientes axiomas de lógica encuacional.

$$x + (y + z) = (x + y) + z$$
$$x + y = y + x$$
$$x + x = x$$

Este nueva enfoque de dotar significado a los operadores de un lenguaje recibe el nombre de *semántica axiomática* y jugará un papel importante en el capítulo 6.

2.1.2. Sistemas de Transiciones Etiquetadas

Como decíamos en la sección anterior, en algunos casos –concretamente en el **CCS** de Milner– la semántica *operacional* de un lenguaje algebraico se describe mediante *sistemas de transiciones etiquetadas (labelled transition systems, LTS).*

Los *LTS* suponen un formalismo sencillo e intuitivo para expresar la naturaleza operacional de los procesos. Básicamente constan de un conjunto de

estados y otro de transiciones etiquetadas entre ellos como registra la siguiente definición.

Definición 2. Un sistema de transiciones etiquetadas es una tupla $\langle Q, L, \rightarrow, q_0 \rangle$ donde

- Q es un conjunto no vacío de estados.
- L es un conjunto de etiquetas
- $\rightarrow \subseteq (Q \times L \times Q)$, la relación de transición.
- q_0 es el estado inicial.

Cuando el conjunto de estados es finito, se puede plantear una sencilla representación pictórica mediante grafos en los que estados se asocian a vértices y transiciones a aristas etiquetadas.

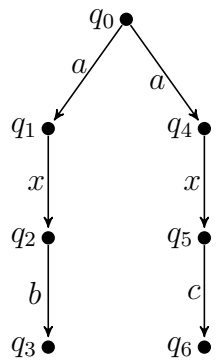


Figura 2.2: Representación de un *LTS* como grafo

En el caso de la figura 2.2 tenemos que:

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
- $L = \{a, b, c, x\}$
- $\rightarrow = \{(q_0, a, q_1), (q_0, a, q_4), (q_1, x, q_2), (q_2, b, q_3), (q_4, x, q_5), (q_5, a, q_5)\}$
- q_0 como estado inicial.

Variantes de *LTS*

A lo largo de la extensa literatura existente sobre los *LTS* se pueden encontrar numerosas variaciones del concepto original, diseñadas para poder expresar diversos matices:

- Mediante la incorporación de una etiqueta especial, τ, denotamos una acción no observable desde el exterior del sistema. Esto da lugar a la consideración de nuevo tipo de transiciones entre estados haciendo abstracción de tal acción , así como la composición de este tipo de transiciones abstractas. Normalmente empleamos para ellas el símbolo $\Rightarrow$, en lugar de $\rightarrow$.

- En ocasiones, el conjunto de acciones L es clasificado en grupos disjuntos de acciones de entrada I y de salida O. Con ello se permite valorar la *autonomía* de una acción por parte un sistema, en el caso de las salidas, o su ejecución *reactiva* a la acción comenzada por otro, cuando tratamos de entradas. Nos referimos a dicha especialización por *LTS interactivos* o de *entrada-salida*.

  Por convenio, se suelen denotar los símbolos de salida con el sufijo ?, y los de salida con !, dando lugar a $a?, b?, x!, y! \ldots$ y a la estructura que lo denota mediante la tupla $LTS = (S, I, O, \rightarrow)$[1].

- En el contexto anterior, para representar la ausencia de acciones de salida se emplea un símbolo especial $\delta!$, denominado *símbolo de quiescencia*. Como veremos, la consideración de este símbolo contribuirá en gran medida a la simplificación de los cálculos en torno a él. En esta tesis consideraremos que $\delta! \in O$.

Trazas y operadores asociados

En el estudio de las propiedades *LTS* resultan de gran interés las secuencias formadas por símbolos de acciones especialmente considerando las transiciones intermedias que determinan cada uno de sus elementos. Esto proporciona nuevas posibilidades de interpretación para el símbolo de transición $\rightarrow$.

[1] El estado inicial q_0 puede omitirse

Definición 3. Sea $p = \langle Q, L, T, q_0 \rangle$ un sistema de transición etiquetado con $q, q' \in Q$, y $\mu \in L$.

Considerando la secuencia $\mu_1 \dots \mu_n = \sigma \in L^*$, podemos extender de manera natural el símbolo $\rightarrow \subseteq Q \times L^* \times Q$ para considerar trazas del modo descrito a continuación:

$$
\begin{array}{ll}
q \xrightarrow{\mu} & \Leftrightarrow_{def} \exists q' (q, \mu, q') \in T \\
q \xrightarrow{\mu_1 \cdots \mu_n} q' & \Leftrightarrow_{def} \exists q_0, \dots, q_n : q = q_0 \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_n} q_n = q' \\
q \xrightarrow{\mu_1 \cdots \mu_n} & \Leftrightarrow_{def} \exists q' q \xrightarrow{\mu_1 \cdots \mu_n} q' \\
q \not\xrightarrow{\mu_1 \cdots \mu_n} & \Leftrightarrow_{def} \nexists q' q \xrightarrow{\mu_1 \cdots \mu_n} q'
\end{array}
$$

A partir de ahí, es común la formulación de algunos operadores útiles en torno a los estados de un LTS reflejando matices como acciones en potencia a partir de un estado (init), trazas o secuencias de acciones alcanzables desde un estado (traces), así como conjunto de acciones observables después de una traza a parir de un estado (outs). En esta tesis seguiremos la notación empleada por Tretmans en [82]

Definición 4. Sea p el estado de un sistema etiquetado de transiciones, y $\sigma \in L^*$.

1. $\mathsf{init}(p) =_{def} \{\mu \in L \mid p \xrightarrow{\mu}\}$
2. $\mathsf{traces}(p) =_{def} \{\sigma \in L^* \mid p \xrightarrow{\sigma}\}$
3. $\mathsf{ins}(q) =_{def} \{x \in I \mid p \xrightarrow{x}\}$
4. $\mathsf{outs}(q) =_{def} \{x \in O \mid p \xrightarrow{x}\}$
5. $\mathsf{outs}(Q) =_{def} \bigcup \{\mathsf{outs}(q) \mid q \in Q\}$
6. $p \,\mathsf{after}\, \sigma =_{def} \{p' \mid p \xrightarrow{\sigma} p'\}$
7. $p \,\mathsf{after}\, \sigma =_{def} \bigcup \{p \,\mathsf{after}\, \sigma \mid p \in P\}$
8. p tiene comportamiento finito si existe algún numero natural n tal que todas sus trazas en $\mathsf{traces}(p)$ tienen longitud menor que n.

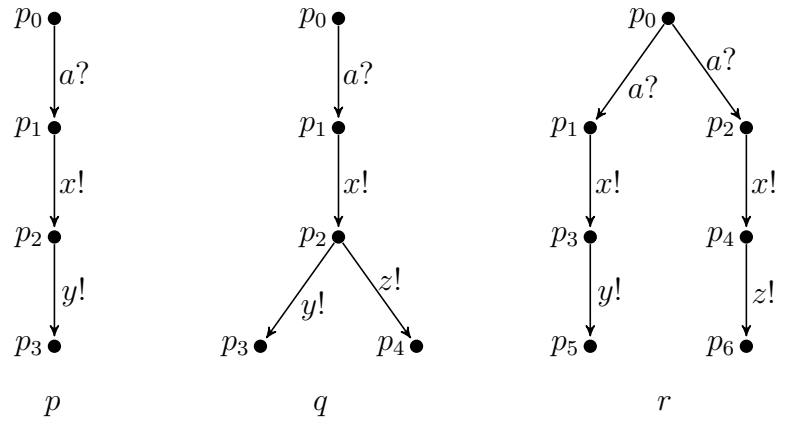


Figura 2.3: Ejemplos de sistemas de transiciones etiquetadas

9. p es determinista si, para todo $\sigma \in L^*$, p after σ tiene como mucho un elemento.

Así definidas, podemos ver que para la figura 2.3 se cumplen las siguientes condiciones:

- el conjunto de acciones iniciales es común a todos

$$\mathsf{init}(p) = \mathsf{init}(q) = \mathsf{init}(r) = \{a?\}$$

- respecto a las trazas potenciales, vemos que tanto r y q son indistinguibles, ya que

$$\mathsf{traces}(p) = \{a?, a?x!, a?y!\}$$
$$\mathsf{traces}(r) = traces(q) = \{a?, a?x!, a?x!y!, a?x!z!\}$$

- análogamente q y r también lo son con respecto a las salidas después de la secuencia $a?x!$, pues

$$\mathsf{outs}(p \text{ after } a?x!) = \{y!\}$$
$$\mathsf{outs}(r \text{ after } a?x!) = \mathsf{outs}(q \text{ after } a?x!) = \{y!, z!\}$$

- los dos primeros sistemas p, q son deterministas, en tanto que el sistema r es indeterminista.

$$(p \text{ after } a?x!) = (q \text{ after } a?x!) = \{p_1\}$$
$$(r \text{ after } a?x!) = \{p_1, p_2\}$$

2.2. Testing basado en modelos

Una de las más importantes contribuciones al estudio del Testing como disciplina científica la encontramos en [46] donde Henessy plantea preórdenes y relaciones de equivalencia entre procesos a partir del conjunto de experimentos que éstos pueden pasar.

Más concretamente, dados dos procesos p, q y un conjunto de experimentos E, distingue tres modalidades o formas de comparación:

semántica may Se dice que $p \sqsubseteq_{MAY} q$, si para cada experimento $e \in E$, $p \,\text{may}\, e$ implica $q \,\text{may}\, e$

semántica must Análogamente , $p \sqsubseteq_{MUST} q$ si para cada experimento $e \in E$, $p \,\text{must}\, e$ implica $q \,\text{must}\, e$

semántica must-may Por último, $p \sqsubseteq q$ si $p \sqsubseteq_{MAY} q$ y $p \sqsubseteq_{MUST} q$

Con posterioridad se han elaborado variaciones que contemplan requisitos no funcionales, como el tiempo [47, 60] y probabilidades [58].

Ya desde un punto de vista más pragmático si se prefiere, el Model Based Testing, **MBT** [81, 78, 82] hace hincapié en la *sistematización* del proceso de generación de *tests* a partir de una especifiación y una relación arbitraria que pone en conexión dos procesos . Esta última acepción es la que se usará en esta tesis y que pasamos a desarrollar con una mayor profundidad.

2.2.1. Descripción general

La figura 2.4, ilustra de forma gráfica las diversas ideas o etapas principales en torno al **MBT**, de las cuales vamos a hacer una pequeña presentación. Estas ideas vienen ampliamente desarrolladas por Tretmans en [82].

El objeto principal de estudio del *testing* como tal es lo que se conoce como **implementación**, y es corriente referirse a él con el acrónimo **IUT** *(Implementation Under Testing)*. Puede comprender desde un dispositivo físico a una aplicación software.

En el seno de la comunidad **MBT**, la mayoría de las veces la **IUT** se considera desde la perspectiva de *black box* en el sentido de cualquier detalle sobre su estructura interna resultará desconocido; la única forma de conocimiento

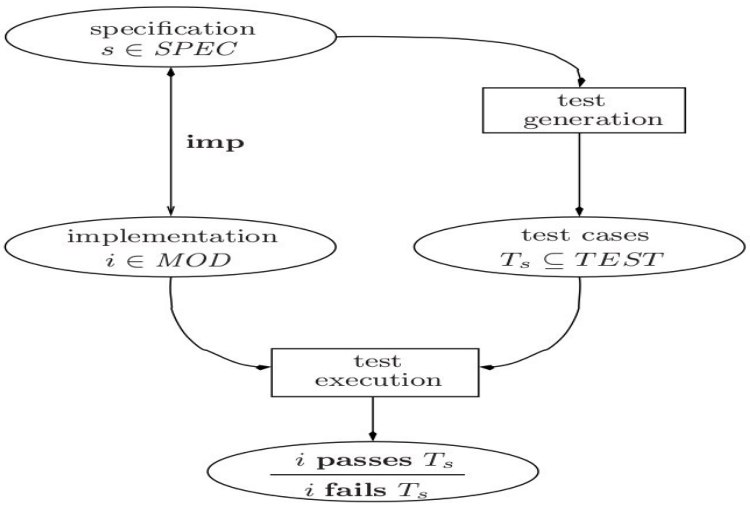


Figura 2.4: Esquema **MBT** tomado de [82]

que podamos tener de él solo se basará en sus interfaces. No obstante, asumimos que ésta puede ser modelada en un dominio concreto. Estas ideas y otras se recogen como supuestos respecto al *test (test assumptions)* en [13, 31].

La **especificación**, sin embargo, es formulada directamente en términos de algún modelo, o indirectamente, en algún lenguaje interpretado en ellos. Cualquier juicio que se emita sobre la idoneidad de una **IUT** se hace siempre respecto a una especificación.

Este último matiz queda formalizado en la relación de **conformidad** dependiendo de la característica funcional, de rendimiento, etc... que se quiera contrastar. Al estar definido formalmente, precisa de su formulación en base a un lenguaje o modelo (sección 2.1) que nos permita contrastar su aplicabilidad.

Como se ha dicho con anterioridad, el único modo que tenemos de conocer el comportamiento de una implementación es mediante experimentos, sometiéndola a una serie de estímulos y registrando sus reacciones. La descripción de ese experimento es lo que se conoce como un ***test* case**, o simplemente *test*, dependiendo del contexto en que nos encontremos.

Para la puesta en marcha y posterior evaluación de los experimentos resulta imprescindible la formulación de una función de **paso de *test*** que nos describa cuándo un experimento, después de practicarlo sobre la **IUT**, ha tenido éxito

y cuándo no. Este concepto se puede extender de modo natural a la totalidad de una **batería de *tests***, afirmando que la implementación supera ésta si pasa cada uno de los *tests* que la integran.

Una de las consecuencias de usar métodos formales es que permite plantear la **generación automática de *tests***, lo que incluye la creación a partir de una especificación de una *batería de* tests bajo los supuestos de *corrección* y *exhaustividad*:

- Por la primera entendemos que cuando un experimento falla, la implementación se revela en efecto como defectuosa con respecto a la especificación.

- La segunda, por lo general más difícil de lograr al requerir un número infinito de *tests*, garantiza como correcta respecto a la relación de **conformidad** una **IUT** que pasa *todos* los *tests*. Expresado de otro modo, equivale a a firmar que ante una **IUT** defectuosa, el sistema encontrará un *test* que lo haga fallar.

Todas estas ideas se abordan en profundidad en [35] y existen numerosos algoritmos de generación de *tests* según la relación de conformidad que consideren en [59, 69].

2.2.2. ioco, una relación de *conformidad*

Uno de los trabajos más ampliamente diseminados en la comunidad **MBT** y punto de partida de nuestro trabajo se puede encontrar en los trabajos de Tretmans [79, 82, 80].

Inspirados por la teoría de la equivalencia entre *tests* y preórdenes ([24, 23]) utiliza como modelo una especialización de sistemas de transiciones etiquetadas, los llamados *LTS* de entrada-salida o interactivos, en los que otorga cierta *direccionalidad* a las acciones observables descritas anteriormente.

Otra de las peculiaridades es la incorporación de un nuevo símbolo δ, usado con un propósito especial: el de denotar la ausencia de acciones autónomas (o de salida) en un estado dado. Dichos estados son denominados *quiescentes* y suponen un ardid matemático que permite tratar este símbolo como uno más de entre los de salida.

Función de paso de test

A la hora de plantear el marco de ejecución de *tests*, Tretmans opta por modelar el objeto de *test* mediante sistemas de transiciones, *LTS*, pero esta vez enriqueciendo el conjunto de estados con dos especiales, **pass**, o en su notación simbólica, ✓, y **fail**, respectivamente ✗, los cuales servirán para decidir el resultado de aplicar el *test* a una eventual **IUT**: cuando el experimento alcanza el estado **pass**, se considera que ha sido ejecutado con éxito; si, por contra, alcanza **fail**, se da por fallado el mismo.

La descripción de la ejecución de un *test* sometido a una implementación se hace mediante un sistema de reglas, de modo que las *premisas* premisas expresan transiciones sobre sistemas que forman parte de una estructura superior. La *conclusión* expresa precisamente las transiciones a este último nivel, siempre que concurran las mencionadas anteriormente en las *premisas*.

En el caso de la teoría ioco̲s, formalmente la *función de paso* queda definida del modo siguiente:

Definición 5. Sea t un *test* e i una **IUT**. Entonces la evolución conjunta de i y t viene dada por:

$$\frac{t \xrightarrow{a} t', i \xrightarrow{a} i'}{t||i \xrightarrow{a} t'||i'} \qquad \frac{t \xrightarrow{\theta} t', i \xrightarrow{\delta} i'}{t||i \xrightarrow{\theta} t'||i'}$$

donde

- a es una acción complementaria entre i y t, esto es, si i se comporta autónomamente con una señal de salida $a!$, entonces t está preparada para recibir la correspondiente señal de entrada $a?$.
- θ es el símbolo que representa la *quiescencia*, desde el punto de vista del experimento
- el símbolo $||$ marca la composición en paralelo de los dos sistemas.

Relación de conformidad

La parte esencial del trabajo de Tretmans es, a no dudarlo, la *relación de conformidad* que propone entre una **IUT** y una especificación.

Intuitivamente, viene a describir que al evaluar sobre la **IUT** cualquier traza aplicable a la especificación, incluyendo símbolos de *quiescencia*, no debería observarse diferencia alguna respecto a las salidas observables en ésta. De modo más formal, teniendo en cuenta los operadores de la definición 4, ioco adopta la siguiente definición:

Definición 6. Sea $LTS = (S, I, O, \rightarrow)$, $i, s \in S$, $\mathsf{ioco} \subseteq S \times S$.

$$i \,\mathsf{ioco}\, s \iff_{def} \forall \sigma \in \mathsf{traces}(s) : \mathsf{outs}(i \,\mathsf{after}\, \sigma) \subseteq \mathsf{outs}(s \,\mathsf{after}\, \sigma)$$

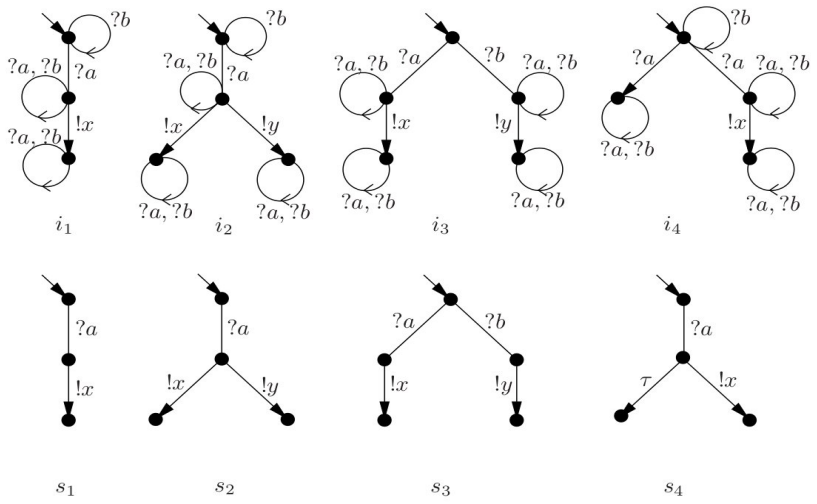


Figura 2.5: Algunos ejemplos según aparecen en [82]

Pasemos a valorar algunos de los ejemplos que aparecen en la figura 2.5

- Vemos que i_2 iocó s_1, pues, al estimular el sistema i_2 con $a?$ produce un conjunto de salidas, $\{x!, y!\}$, que no es subconjunto de las que puede producir s_2, concretamente $\{x!\}$

$$\mathsf{outs}(i_2 \,\mathsf{after}\, a?) = \{x!, y!\} \not\subseteq \{x!\} = \mathsf{outs}(s_1 \,\mathsf{after}\, a?)$$

- No es el mismo caso al valorar i_3 y s_1, para los que se tiene $i_3 \,\mathsf{ioco}\, s_1$, pues, aunque después de estimular con $b?$ a i_3 se produzca $y!$, $b?$ no pertenece

a las trazas de s_1

$$b? \notin \mathsf{traces}(s_1)$$
$$\mathsf{outs}(i_3 \text{ } \mathsf{after} \text{ } b?) = \{y!\} \not\subseteq \varnothing = \mathsf{outs}(s_1 \text{ } \mathsf{after} \text{ } b?)$$

En este último sentido suele decirse que i_3 es *libre* con respecto a la especificación cuando considera señales no especificadas. Más adelante, en el capítulo 3, volveremos a incidir sobre este matiz, esencial para comprender detalles de la relación que forma el núcleo de esta tesis.

- De acuerdo con la definición 4, i_2 constituye una implementación de s_3, puesto que i_2 no tiene definidas acciones después de la sóla ejecución de $b?$.

$$i_2 \text{ } \mathsf{after} \text{ } b? = \varnothing \subseteq \{y!\} = s_3 \text{ } \mathsf{after} \text{ } b?$$

Diferente dominio de definición e indeterminismo

Una observación más detenida de la figura 2.5 nos lleva la conclusión de que las figuras superiores correspondientes pueden haber sido obtenidas a partir de las inferiores mediante procedimientos de *completado* [82], básicamente creando una transición hacia sí mismo para cada estado que no tenga definida alguna para cualquier acción de entrada. Por supuesto, este *completado* no sería necesario si el sistema original ya lo contempla. Véase, por ejemplo el estado inicial de s_3 que no ha requerido transición extra ninguna.

Esta característica de los sistemas de tener definida una transición para cada símbolo de entrada es referida como *input enabled (habilitado para entradas)* y de por sí supone la constatación de un hecho más relevante, a saber, que la relación ioco está definida entre dominios distintos, pues la especificación no requiere el fuerte requisito limitante explicado anteriormente de la habilitación frente a cualquier estímulo.

Esto anula cualquier posibilidad de consideración como preorden, excluyendo una propiedad tan importante como la transitividad a la hora de expresar los refinamientos sucesivos de una implementación a partir de una especificación. Este aspecto se abordará más detenidamente en el capítulo 3.

Así mismo, la circunstancia del *indeterminismo* inexistente en las especificaciones s_n y prácticamente nula en el caso de las implementaciones i_n, salvo

i_4, presupone que ioco, tal como está definida, no suponga una buena relación de conformidad para probar formas más complejas de indeterminismo, como tendremos oportunidad de ver.

Otras variantes de ioco

El mismo autor describe en [82] algunas generalizaciones y variantes de la relación, parametrizando el conjunto de trazas a considerar, y otras que se citan a continuación:

mioco Extensión de ioco para tratar con *múltiples* canales [28]. Cada acción corresponde a un canal de entrada o de salida.

sioco Las acciones son parametrizadas con datos. Para evitar la explosión de estados durante la generación de *tests*, se adopta un enfoque *simbólico* [30].

hioco Sistemas *híbridos* donde acciones discretas y variables continúas desarrollan un papel.

2.2.3. Generación automática de *tests*

Definidos el marco de ejecución de *test* y la relación de conformidad, lo único que resta es definir un procedimiento capaz de generar un conjunto de *tests* a partir de una especificación dada.

En esencia, el algoritmo generador de *tests* crea la estructura de éstos llevando cuenta de la trazas de acciones de entrada (salida) aplicadas (resp. observadas) de la implementación, y de modo consistente a lo que dicta la especificación derivando el *test* hacia estados ✓ para aquellas trazas permitidas, y hacia estados ✗ para el resto de las que no lo son.

En el caso particular de iocos, Tretmans [82] lo consigue mediante la formulación de un algoritmo recursivo indeterminista; partiendo del estado inicial, las transiciones del *test* son derivadas de acuerdo al comportamiento de la especificación y por cada estado generado se procede de manera recursiva, distinguiendo entre aquellos estados del sistema que son reactivos y los que no. *Grosso modo*, ante cada estado de entrada el algoritmo opta por una de las tres posibilidades:

1. Dar el *test* por concluido derivándolo hacia un estado ✓.

2. Someter a la implementación a cualquiera de los estímulos que considera la implementación, valorando la posibilidad de ser interrumpido por cualquier salida de la implementación; caso de que sea una salida no contemplada, derivar el sistema hacia el estado ✗. En otro caso, para cada estado obtenido aplicar recursivamente el algoritmo.

3. Permanecer escuchando las salidas que pueda arrojar la implementación, o concluir que ésta es *quiescente*. Dependiendo de lo que marque la especificación, en uno u otro caso se derivará a ✗ o a otro estado al que continuar aplicando el algoritmo recursivamente.

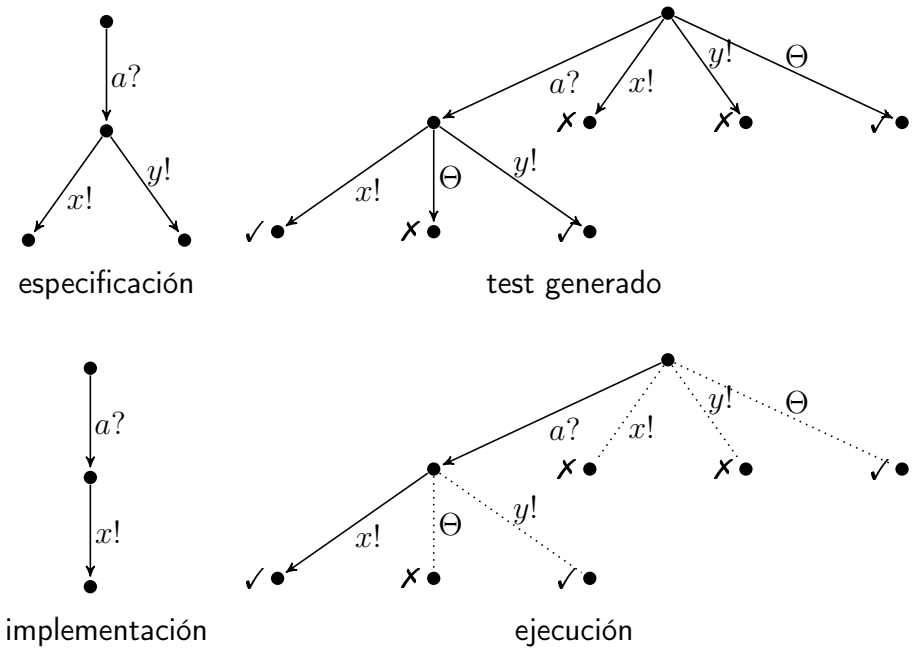


Figura 2.6: Ejemplo de especifiación y *test* generado

La figura 2.6 muestra un pequeño ejemplo ilustrativo de un *test* generado con respecto a una especificación, y la ejecución de una implementación concreta al ser sometido al *test*.

En el mismo artículo [82] se proporcionan pruebas de su corrección y exhaustividad, lo que garantiza que siempre que uno de los *tests* generados resulte en fallo, se tendrá la garantía de que la **IUT** sobre la que se practicó

el experimento no está en relación ioco con la especificación, según la definición 6. Además, para cualquier implementación defectuosa siempre habrá un *test* generado capaz de detectar ese fallo.

Sin embargo, volviendo a la figura 2.6, repárese en el detalle de que, en el caso de la implementación sencilla que nos ocupa, la figura muestra como se ha invertido mucho tiempo y espacio de cómputo en ramas tales que nunca serán inspeccionadas. Tal síntoma de ineficacia será el motivo de la aparición de una nueva modalidad de *testing* explicada en el siguiete apartado: el *testing online*.

En el campo de la implementación están disponibles varias herramientas que permiten poder experimentar con el **MBT**, por ejemplo [7, 84].

2.2.4. Testing online

Como acabamos de ver en la anterior sección, en general la generación de una batería completa de *tests* mediante un algoritmo suele presentar problemas en lo relativo a recursos computacionales como la memoria y el tiempo. Téngase en cuenta que al estar basados en sistemas de transiciones etiquetadas, ya sea directamente o indirectamente a través de un lenguaje, fácilmente se incurre en el conocido problema computacional de explosión de estados, pues todas las posibles respuestas deben ser tenidas en cuenta y representadas en los *tests* generados.

Un modo de evitar este problema es el empleo de técnicas de *testing online*, donde se combina la generación de *primitivas de test* y su ejecución de manera dinámica.

Dichas primitivas incluyen desde la estimulación con posibles acciones de entrada, hasta la detección de acciones de salida esperadas y la posibilidad de detectar la *quiescencia* en determinados estados.

La figura 2.7, tomada de [25], contribuye a comprender este nuevo modo de operar:

1. La estructura principal PRIMER es la responsable de generar todas las primitivas de *test* en cada instante. Dicha entidad tiene capacidad para representar el estado de la especificación teniendo en cuenta las acciones

ejecutadas hasta el momento, lo que incluye posibles respuestas de la **IUT**.

2. Cada vez que el objeto representado DRIVER entra en acción lo hace solicitando una de las posibles *primitivas* a la entidad principal (PRIMER) y ejecutándola sobre la **IUT**. Este proceso puede verse interrumpido por una acción inesperada de la **IUT**.

3. A continuación recoge la salida que ésta pueda ofrecer y se la comunica a la entidad PRIMER, que vuelve a computar el nuevo conjunto de *primitivas* disponible, y el proceso se repite de nuevo.

Como consecuencia de este planteamiento el espacio requerido por muchas herramientas de *test offline* se reduce porque únicamente una parte limitada del espacio necesita ser almacenada en cada punto.

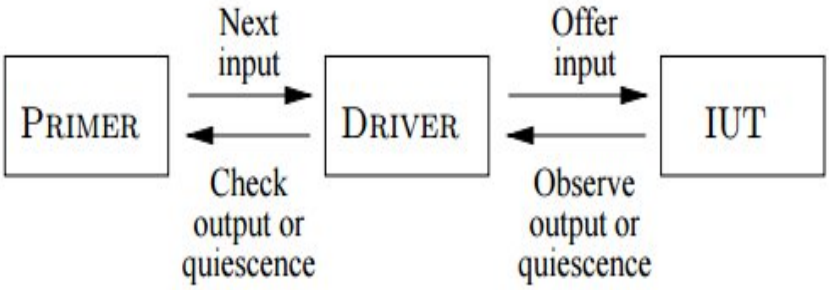


Figura 2.7: Esquema de un sistema de *test-online*

Algoritmos indeterministas

Existe mucha literatura al respecto que permite profundizar sobre las características del *testing online* [94, 48, 54, 56, 57, 67, 25, 19, 49]. De especial interés resulta para nuestro propósito el artículo de Vries y Tretmans [25] en el que se presenta un algoritmo de *testing online* para ioco de naturaleza no determinista y en el que la terminación misma se describe en función de un operador de elección.

También hay que nombrar el trabajo de Larsen [57] donde se describe un algoritmo de *testing online* sobre una variación de la original ioco, denominada por sus autores $\mathsf{rtioco}_\mathsf{e}$, incluyendo información temporal sobre los elementos observables y parametrizada por el tipo de entorno al que se somete a la **IUT**.

Siendo el objeto de la relación de conformidad una semántica lineal o de trazas, el sistema lleva cuenta simultánea de cuantos estados se pueden alcanzar en potencia a partir de uno dado. Para agilizar el cómputo, el algoritmo se apoya en técnicas de computación simbólica para el cálculo de operadores sobre conjuntos. En síntesis, el sistema puede realizar una de estas tres acciones *aleatoriamente*:

- esperar una salida de la **IUT**
- estimular la **IUT** con una señal
- restaurar el proceso de *test* a su estado original. Esta última opción llegar a ser importante porque en base a cierta hipótesis de justicia (*fairness*) el sistema devolverá un veredicto de fallo garantizando que cualquier implementación defectuosa será detectada.

Este trabajo de Larsen [57] sirvió de referencia para la propuesta del algoritmo *online* que se presenta en el capítulo 4.

En lo relativo a implementaciones, destaca la herramienta JTorX [8, 7], con un completo repertorio de pruebas y ejemplos.

2.2.5. Testing y simulación

Hasta la publicación de los trabajos de Abramsky [2], la bisimulación – y la relación de equivalencia que de ella se deriva, conocida en este contexto como *equivalencia observacional*– era considerada como muy restrictiva cuyo poder de discriminación no había podido ser abordado por una teoría de equivalencia de *tests*. Existían trabajos [66] en los que se proponían equivalencias de *testing*, pero ciertamente todas ellas eran semánticas más débiles que la de referencia.

En definitiva, dos sistemas que no eran equivalentes desde el punto de vista observacional no eran detectados cuando se sometían a un experimento de *test*, sino que eran identificados como el mismo. En el horizonte planeaba la cuestión

de ver qué extensiones eran necesarias al marco de *testing* para poder abordar la *equivalencia observacional*.

En el artículo de Abramsky [2], se intenta dar una respuesta diseñando un nuevo marco de *test* al que va incorporando sucesivas construcciones más expresivas, tales como las trazas, *refusals*[38, 70], la copia y el el *testing* global. La figura 2.8 da cuenta de esta sintaxis.

$$t ::= \mathbf{SUCC} \mid \mathbf{FAIL} \mid at \mid \tilde{a}t \mid \epsilon t \mid t_1 \wedge t_2 \mid t_1 \vee t_2 | \forall t | \exists t$$

Figura 2.8: Elementos introducidos en la sintaxis de tests de Abramsky [2]

Su noción de *testing* se formaliza de dos maneras: *operacionalmente*, mediante un sistema de transiciones y *denotacionalmente* por inducción sobre la estructura de *tests* utilizando algunos operadores definidos sobre los dominios potencia de Plotkin [71]. Demuestra que son coincidentes y que la noción de equivalencia inducida por los *tests* coincide con la de bisimulación.

El enfoque denotacional, que se presenta en la figura 2.9 ha servido de inspiración para la formulación del *paso de test* en el capítulo 4 a la hora de plantear la caracterización mediante *tests* de la nueva relación de *conformidad*.

$$\begin{array}{lcl}
O(\mathbf{SUCC}, p) & = & \{\top\} \\
O(\mathbf{FAIL}, p) & = & \{\bot\} \\
O(at, p) & = & \bigcup\{O(t, p') | p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot | p \uparrow\} \cup \{\top | p \stackrel{\epsilon}{\Longrightarrow} p' \& p' \mathbf{ref} a\} \\
O(\tilde{a}t, p) & = & \bigcup\{O(t, p') | p \stackrel{a}{\Longrightarrow} p'\} \cup \{\bot | p \uparrow\} \cup \{\top | p \stackrel{\epsilon}{\Longrightarrow} p' \& p' \mathbf{ref} a\} \\
O(\epsilon, p) & = & \bigcup\{O(t, p') | p \stackrel{e}{\Longrightarrow} p'\} \cup \{\bot | p \uparrow\} \\
O(t_1 \wedge t_2, p) & = & O(t_{,1}, p) \wedge O(t_2, p) \\
O(t_1 \vee t_2, p) & = & O(t_{,1}, p) \vee O(t_2, p) \\
O(\forall t, p) & = & \forall O(t, p) \\
O(\exists t, p) & = & \exists O(t, p)
\end{array}$$

Figura 2.9: Semántica denotacional definida sobre el cojunto de tests

2.3. Simulaciones y algoritmos

Anteriormente hemos abordado la técnica de **MBT** como modo sistemático de abordar mediante *tests* la propiedad de conformidad de un implementación con respecto a una especificación. Debido a la complejidad de la mayor parte de los sistemas, el *testing* solo puede dar cuenta de una parte limitada –finita– de modos de comportamiento, por lo que nunca puede obtenerse una certeza absoluta sobre la satisfactibilidad de una propiedad.

No obstante, cuando se disponemos del modelo del objeto que queremos inspeccionar, este problema puede resolverse con absoluta certeza empleando métodos algorítmicos, a expensas, claro está, de disponer de modelos formales no sólo para la especificación sino también para la implementación.

El mayor inconveniente que se plantea entonces es que, al igual que el *testing*, suele producirse una explosión de estados que hace impracticable el problema; a modo de ejemplo, baste decir que para un sistema de n estados, la complejidad puede llegara a ser del orden de 2^{2^n} según se describe en [4]. Una forma alternativa de proceder en estas situaciones es preprocesar los modelos de entrada de modo que se reduzca su tamaño obteniendo un nuevo sistema en cierto sentido *equivalente* al que originalmente se desea estudiar. Este planteamiento aparece con frecuencia en otras disciplinas como el *model checking* [18, 52].

Una de las formas de superar cierta falta de expresividad en las relaciones de *conformidad* consiste en servirnos de técnicas coinductivas, al modo que lo hacen otras relaciones clásicas pertenecientes al grupo de *semánticas branching*, como puede ser la *bisimulación*, la *simulación ordinaria*, o la *simulación ready*. La búsqueda de algoritmos eficientes tratando de resolver este tipo de problemas ha sido una constante en la investigación de los últimos años, como se puede apreciar en [32, 17, 75].

Los primeros intentos al respecto tienen lugar a la hora de resolver la *bisimulación* en [51], donde se presenta un algoritmo para la minimización del número de estados en un autómata de estados finitos. En años sucesivos, se idearon nuevas técnicas algorítmicas cada vez más eficientes, que, si bien no en todos los casos suponía una mejora en el caso peor, obtenían una mejora sustancial desde una perspectiva estadística. Todas ellas pueden englobarse

bajo la denominación del *Coarsest Partition Problem*, o *Problema de la Mayor Partición*.

Con posterioridad, R. Gentilini en [32], corregido por van Glabbeek y Ploeger en [91] se plantea una generalización del problema que es la que tomamos como base en este trabajo: el *Generalized Coarsest Partition Problem, (**GCPP**)* , o *Problema Generalizado de la Mayor Partición*. Previamente, otros autores, como Bloom [15] también intentaron dar con la solución empleando técnicas de derivación algorítmica, llegando a la conclusión de que se puede llegar a la *simulación ready* sin más que adaptar las condiciones iniciales de un problema que resuelva la *simulación ordinaria*.

En los siguientes apartados haremos un breve repaso por aquellos preórdenes y equivalencias que han conducido a lo largo del tiempo a *perfilar* los algoritmos de relaciones semánticas, ya se trate de equivalencias o preórdenes. Todos ellos son una síntesis de los contenidos que pueden ser accedidos con más profundidad en [33, 32].

2.3.1. Bisimulación

Pese a que la bisimulación como tal no sea parte esencial de esta tesis, merece la pena profundizar un poco en su naturaleza al objeto de poder entender otras relaciones como la simulación, más próximas a nuestro objetivo, la cual constituye una generalización.

Con este propósito nos proponemos describir en qué consiste el *Coarsest Partition Problem*, o *Problema de la Mayor Partición* encargada de computar la *bisimulación*. Para ello necesitaremos traer aquí algunas nociones básicas:

Definición 7. Se dice que una tupla $G = (N, \rightarrow, \Sigma)$ es un *grafo etiquetado* si y sólo si $G^{-} = (N, \rightarrow)$ es un grafo dirigido finito y Σ es una partición sobre N, siendo $\rightarrow \subseteq N \times N$.

La figura 2.10 nos muestra algunos ejemplos de grafos etiquetados. Nótese que los *LTS* presentados en la sección 2.1.2 difieren con respecto a la estructura G descrita anteriormente en que ésta no contempla etiquetas en las transiciones. Más bien el calificativo de *etiquetado* se emplea en otro sentido muy distinto, nombrando clases de equivalencias en el conjunto de vértices. Por

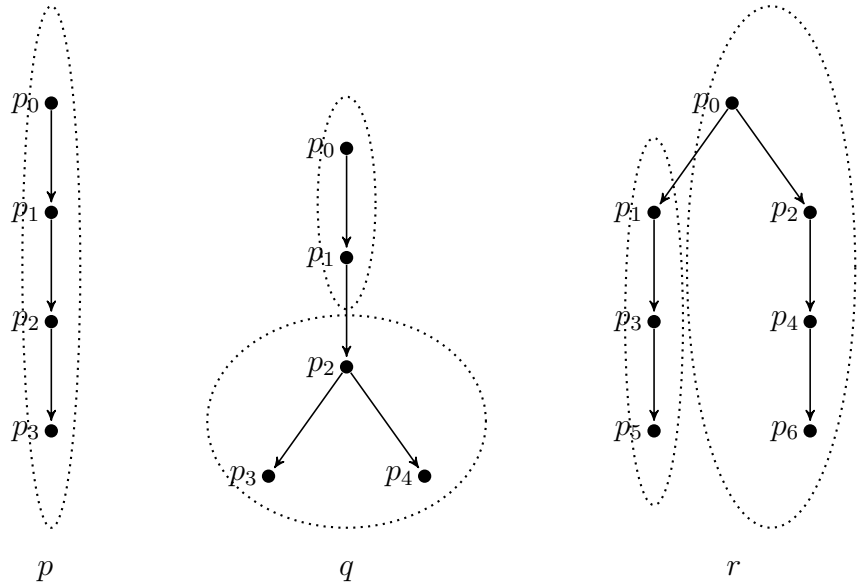


Figura 2.10: Representación de grafos etiquetados $G = (N, \rightarrow, \Sigma)$

razones didácticas, es preferible la versión de la definición 7 para comprender la *esencia* del problema.

Definición 8. Sea $G = (N, \rightarrow, \Sigma)$. Una relación $\asymp \subseteq N \times N$ es una *bisimulación* sobre G si y sólo si para cualquier $a, b, c \in N$

(1) $a \asymp b \Rightarrow [a]_\Sigma = [b]_\Sigma$;

(2) $a \asymp b \wedge a \rightarrow c \Rightarrow \exists d \in N(c \asymp d \wedge b \rightarrow d)$.

(3) $a \asymp b \wedge b \rightarrow d \Rightarrow \exists d \in N(c \asymp d \wedge a \rightarrow c)$.

El *problema de la bisimulación* consiste en calcular la estructura cociente $N/\equiv_b$, donde $\equiv_b$ representa la máxima bisimulación sobre G, es decir, aquella tal que cualquier otra bisimulación $\asymp$ cumpla $\asymp \subseteq \equiv_b$.

El planteamiento de los algoritmos de refinamiento es el cálculo de $\equiv_b$ identificando las clases de equivalencia $N/\equiv_b$ con las particiones sobre el conjunto de nodos de G, es decir N.

Al respecto, nótese que la inclusión de Σ en la definición 8 es una *generalización* de la más conocida de Milner presente en algunos textos clásicos [63],[4] al ser ésta una caso particular cuando $\Sigma = \{N\}$. En efecto, cuando esto ocurre

cualesquiera dos elementos $a, b \in N$ pertenecen a la misma clase de equivalencia, N, por lo que la cláusula 1 de la definición 8 es redundante y se puede prescindir de ella como en la más conocida versión clásica versión clásica.

Estabilidad de una partición

La razón es que esta generalización da las claves para entender el problema de *refinamiento de particiones* en una estructura G, que al iterar un operador llega a transformar la partición original hasta alcanzar una que cumple una condición de *estabilidad*, que pasamos a describir:

Definición 9. Sea $\rightarrow$ una relación binaria sobre N, $\rightarrow^{-1}$ su relación inversa, Σ una partición de N y $\alpha, \gamma \in \Sigma$ clases de equivalencia; por tanto $\rightarrow^{-1}(\gamma)$ denota el subconjunto de N que tiene como imagen en la relación $\rightarrow$ algún elemento $g \in \gamma$.

Se dice que Σ es *estable* con respecto a $\rightarrow$ si y sólo si

$$\alpha, \gamma \in \Sigma, \alpha \subseteq \rightarrow^{-1}(\gamma) \vee \alpha \cap \ \rightarrow^{-1}(\gamma) = \varnothing$$

En términos intuitivos, el sentido de la definición 9 es el siguiente: en un escenario *estable* de bisimulación, si un elemento de una clase realiza una transición hacia otro situado en otra clase entonces *todo* el resto de miembros de esa clase realiza *solidariamente alguna* transición hacia algún elemento de esa clase.

2.3.2. Simulación

El *problema de la simulación* constituirá la piedra angular sobre la que construiremos nuestra propuesta para el cálculo de nuestra nueva relación de *conformidad* en el capítulo 5, al plantear su solución en términos de reducción a dicho problema.

De nuevo, la siguiente definición es una versión generalizada de la tradicional en [63], al emplear clases de equivalencia sobre el conjunto de elementos N, al tiempo que una simplificación en el sentido de que las transiciones son sin etiquetar, facilitando de nuevo la comprensión de la esencia del problema

Definición 10. Sea $G = (N, \rightarrow, \Sigma)$. Una relación $\leq N \times N$ se dice *simulación* sobre G si y sólo si para cualquier $a, b, c \in N$ se cumple:

1. $a \leq b \Rightarrow [a]_\Sigma = [b]_\Sigma$;

2. $a \leq b \wedge a \rightarrow c \Rightarrow \exists d \in N(c \leq d \wedge b \rightarrow d)$.

En el caso de la expresión $a \leq b$ se interpreta que b *simula* a a. Denotamos por $\leq_s$, a la mayor de las relaciones de simulación, o equivalentemente:

$$\leq_s = \bigcup \{R \mid R \subseteq S \times S,\ R \text{ es una simulación}\}$$

De manera análoga a la *bisimulación*, el *problema de la simulación* consiste en encontrar la mayor relación de equivalencia $\equiv_s$ inducida a partir de $\leq_s$ (i.e., $a \equiv_s b$ si y sólo si a $a \leq_s b \wedge b \leq_s a$) sobre la que construir la estructura cociente $N/\equiv_s$, tal que cualquier otra, $\equiv_i$ cumpla que $\equiv_i \subseteq \equiv_b$.

Insuficiencia del criterio de estabilidad

Si se intenta especular con una versión de *estabilidad* que *caracterice* el contenido de la definición 10, pronto se cae en la cuenta de la insuficiencia de la propuesta en la definición 9 en este nuevo escenario más general y abstracto, al contar con una cláusula menos. En efecto, imaginemos un elemento a perteneciente a una clase de equivalencia $[a]_\Sigma \in \Sigma$ hace una transición hacia un elemento c en otro bloque, $[c]_\Sigma \in \Sigma$. Lo que ocurre es que *no necesariamente* todos los elementos de esta clase tienen una transición definida hacia $[c]_\Sigma$. Más bien, lo único que se garantiza es la transición hacia elemento d, cuya clase de equivalencia $[d]_\Sigma$ tiene la propiedad de simular *todos* los elementos de $[c]_\Sigma$, pero no a la inversa.

Este nuevo concepto de *estabilidad*, más abstracto en la medida que la definición 10 lo es comparada con la definición 8, precisa para su formulación de una caracterización de esos *pares* $P \subseteq \Sigma^2$ orientados de *bloques* o clases de equivalencia ($[c]_\Sigma, [d]_\Sigma \in \Sigma$) entre los cuales se da relación de *simulación*. La siguiente definición da cuenta de ello.

Definición 11. Sea $G^- = (N, \rightarrow)$ un grafo finito. Un *par-partición* sobre G es un par $\langle \Sigma, P \rangle$ en el cual Σ es una partición sobre N,y $P \subseteq \Sigma^2$ es una relación reflexiva y acíclica sobre Σ.

El algoritmo encargado de resolver el nuevo problema tomará como condiciones iniciales el *par-partición* $\langle \Sigma, I \rangle$ donde I es la relación identidad sobre

los elementos de Σ e iterarará sobre ella hasta alcanzar un par-partición que sea *estable*, como quiera que este concepto no se haya definido todavía con precisión.

A fin de constatar el progreso del algoritmo en su convergencia hacia el punto fijo, se necesita la definición precisa de un *preorden* sobre el objeto de transformación, los *pares-partición*.

Dados dos *pares-partición*, $\langle \Sigma, P\rangle$ y $\langle \Pi, Q\rangle$, parece claro el sentido de orden entre Σ y Π, sin más que adoptar la *inclusión* de conjuntos entre sus respectivos bloques. Para la relación de orden entre las relaciones de particiones, necesitamos un concepto un poco más elaborado: el de *relación inducida*. Todos estos conceptos son los que se recogen en la siguiente definición

Definición 12. Sean Σ, Π dos particiones cualesquiera

- Decimos que Π es *más fina* que Σ, $\Pi \sqsubseteq \Sigma$ si cada bloque de Π está incluido en *algún* bloque de Σ.
- Sea P una relación binaria $P \subseteq \Sigma \times \Sigma$ y $\Pi \sqsubseteq \Sigma$. Definimos por $P(\Pi)$, o *relación inducida sobre P por* Π, a una relción binaria $P(\Pi) = \Pi \times \Pi$ tal que:

$$\forall \alpha, \beta \in \Pi((\alpha, \beta) \in P(\Pi) \Leftrightarrow \exists \alpha', \beta'((\alpha', \beta') \in P \wedge \alpha \subseteq \alpha' \wedge \beta \subseteq \beta'))$$

- Denotando por $\mathcal{P}(G)$ al conjunto de pares-partición sobre G, el preorden entre pares partición, $\langle \Sigma, P\rangle \sqsubseteq \langle \Pi, Q\rangle$ se establece de la siguiente manera:

$$\langle \Pi, Q\rangle \sqsubseteq \langle \Sigma, P\rangle \text{ si y sólo si } \Pi \subseteq \Sigma \text{ y } Q \sqsubseteq P(\Pi).$$

Al objeto de hacer más legible la notación empleamos las siguientes variantes del símbolo de transición $\rightarrow$ entre clases de equivalencia.

Definición 13. Sea $G^- = (N, \rightarrow)$, y Π una partición de N.

La relación $\exists$-*transition* sobre Π

$$\alpha \rightarrow_\exists \gamma \text{ si y sólo si } \exists a \exists c(a \in \alpha, c \in \gamma \wedge a \rightarrow c)$$

La relación $\forall$-*transition* sobre Π

$$\alpha \rightarrow_\forall \gamma \text{ si y sólo si } \forall a(a \in \alpha \Rightarrow \exists c(c \in \gamma \wedge a \rightarrow c))$$

En estas condiciones, ahora se puede caracterizar el criterio de *estabilidad* que caracteriza la terminación de un algoritmo, haciendo las veces de punto fijo, eventualmente diseñado para resolver el **GCPP**.

Definición 14. Sea $G^{-} = (N, \rightarrow)$. Un par partición $\langle \Sigma, P \rangle$ sobre G^{-1} es *estable* con respecto a la relación $\rightarrow$ si y solo si

$$\forall \alpha, \beta, \gamma \in \Sigma((\alpha, \beta) \in P \wedge \alpha \rightarrow_{\exists} \gamma \Rightarrow \delta \in \Sigma((\gamma, \delta) \in P \wedge \beta \rightarrow_{\forall} \delta))$$

La anterior definición formaliza lo que intuitivamente hemos adelantado a la hora de justificar la insuficiencia del criterio de estabilidad para la bisimulación: dados dos bloques relacionados, si un elemento del primer bloque realiza una transición hacia otro bloque, necesariamente *todos* los elementos del segundo harán una transición hacia un bloque que estará en la situación de poder *simular* a aquél. Como se ha descrito en 12, dos clases de equivalencia con esta propiedad quedan reflejadas mediante relación de preorden establecida entre ellas. La figura 2.11, tomada de[32] sirve para ilustrar el concepto:

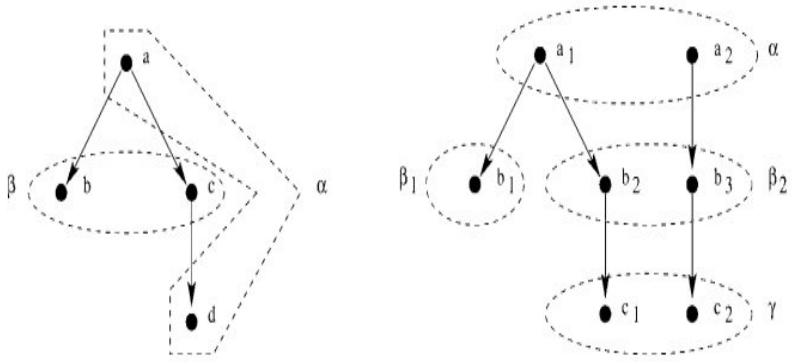


Figura 2.11: G y Σ a la izquierda, G' y Π a la derecha

En el primer caso, el par-partición $\langle \Sigma, I \rangle$ con $\Sigma = \{\alpha, \beta\}$, no se puede considerar *estable* ya que en el caso de β tenemos $\beta \rightarrow_{\exists} \alpha$, y, siendo que sólo $(\alpha, \alpha) \in I$, no ocurre que $\beta \rightarrow_{\forall} \alpha$. En el segundo sí lo es, pues tomando el par-partición $\langle \Pi, P \rangle$ con $\Pi = \{\alpha, \beta_1, \beta_2, \gamma\}$ y $P = \{(\beta_1, \beta_2)\} \cup I\}$ vemos el caso en que $\alpha \rightarrow_{\exists} \beta_1$ mientras que $\alpha \rightarrow_{\forall} \beta_2$ con $(\beta_1, \beta_2) \in P$. Para el caso $\alpha \rightarrow_{\exists} \beta_2$, se tiene de nuevo $\alpha \rightarrow_{\forall} \beta_2$, y $(\beta_1, \beta_2) \in I$. Similar argumento se tiene para $\beta_2 \rightarrow_{\exists} \gamma$, ya que $\beta_2 \rightarrow_{\forall} \gamma$, y $(\gamma, \gamma) \in I$.

Definidos el preorden y el criterio de estabilidad, se puede *especificar* con precisión en que consiste el *Generalized Coarsest Partition Problem* **GCPP**, o problema generalizado del mayor par-partición.

Definición 15. Dado un grafo $G^- = (N, \rightarrow)$ y un par-partición $\langle \Sigma, P \rangle$ sobre G, el problema **GCPP** consiste en hallar el par-partición $\langle S, \leq \rangle$ tal que

(a) $\langle S, \leq \rangle \sqsubseteq \langle \Sigma, P^+ \rangle$

(b) $\langle S, \leq \rangle$ es estable con respecto a $\rightarrow$.

(c) $\langle S, \leq \rangle$ es un par-partición $\sqsubseteq$-maximal satisfaciendo (a) y (b)

El siguiente teorema, cuya demostración puede consultarse en [33] vincula la solución del problema **GCPP** al problema de la simulación.

Teorema 1. *Sean $G = (N, \rightarrow, \Sigma)$ y $\langle S, \leq \rangle$ la solución del problema **GCPP** sobre $G^- = (N, \rightarrow)$ y $\langle \Sigma, I \rangle$ donde I es la relación identidad.*

Entonces S el cociente simulación *de G, esto es, $S = N/\equiv_s$, y $\leq_s$, definida como*

$$a \leq_s b \text{ si sólo si } [a]_s \leq [b]_s$$

es la simulación maximal sobre definida sobre G.

Recordemos que al considerar como condición de entrada el par $\langle \{N\}, I \rangle$ obtenemos la solución tal como se expresa en la formulación clásica de Milner [63], con la salvedad de que no se consideran etiquetas en las transiciones, o equivalentemente, es posible sólo una.

Con esta observación damos fin a este capítulo cuyo objeto ha sido exponer las líneas generales de las teorías que han servido de soporte para esta tesis, desarrollada en los capítulos que siguen a continuación. En la bibliografía citada se pueden encontrar fuentes de información que amplían de manera más detallada todo lo expuesto en torno a los temas descritos.

Capítulo 3

Input-Output Conformance Simulation (iocos)

En este capítulo describimos el concepto fundamental en torno al cual gira esta tesis: una relación de *conformidad* que permitirá determinar si una eventual implementación reúne todos los requisitos para serlo de una especificación propuesta.

En primer lugar haremos una motivación de las principales ideas que nos llevaron a buscar una relación de *conformidad* más satisfactoria que las existentes en la literatura. En particular, buscábamos una relación que tuviese mayor poder de distinción que la clásica relación de conformidad ioco, manteniendo al mismo tiempo muchas de las propiedades que han permitido que dicha relación haya sido estudiada y utilizada de forma extensiva en publicaciones y herramientas [93, 92, 85, 83, 82].

Apoyados en el formalismo clásico de los sistemas de transiciones etiquetadas propondremos una definición formal para la nueva relación, iocos, que toma su acrónimo (*input-output conformance simulation)* del origen que comparte con las denominadas semánticas ramificadas (*branching semantics*). Esta familia de semánticas tiene algunos representantes bien conocidos como son las simulación y la bisimulación.

A lo largo del capítulo estudiaremos las analogías y diferencias claves entre la relación de conformidad clásica de input-output con la nueva relación iocos. Podremos constatar la imposibilidad de ioco, emparentado con las semánticas *lineales*, para poder distinguir el contexto local de ejecución. Es en este aspecto

en el que iocos permite un mayor poder de distinción, refinando estrictamente la clásica relación ioco.

Mostraremos, además, que bajo ciertas hipótesis especiales añadidas, ambas relaciones pueden resultar equivalentes. Esto puede ser especialmente útil, como veremos en el desarrollo de esta tesis, pues la naturaleza *branching* de iocos la hace buena candidata para contar con algoritmos eficientes que la computen, pudiéndose utilizar así como una buena aproximación eficiente a la clásica relación de conformidad ioco.

3.1. Motivación

Desde los primeros tiempos en los que la computación quedó ligada a las ciencias matemáticas mediante la formulación de las *cuestiones abiertas* de Hilbert [29], y una vez resuelta la formulación del concepto clave de *algoritmo*, los científicos de la computación se interesaron por cuestiones como la efectiva *computabilidad* de funciones matemáticas así como la *complejidad* de los algoritmos que las implementan.

Con posterioridad, bien entrada la década de los 70, el tipo de preguntas que se abordaron en la ciencia de la computación dio lugar a un nuevo área de instigación no considerada hasta entonces. Ya no resultaba de tanto interés el *cómputo* de funciones, sino la *reacción, interacción, sincronización, comunicación* que se daba cuando varios procesos se ejecutaban compartiendo algún recurso [26].

El objeto de estudio pasa a ser el *proceso*, remplazando al tradicional *algoritmo*. Sin embargo, ambos conceptos comparten nociones elementales como las de *especificación*, cuyo objeto es la de expresar propiedades de un sistema de manera más o menos abstracta, y la de *implementación*, que pretende materializar un sistema acorde a aquella de manera detallada en un *modelo de cómputo* concreto.

3.1.1. Indeterminismo

Existen diversas maneras de comprender las relaciones entre los procesos. Es lógico pensar que en un escenario *concurrente* la cuestión del *indetermi-*

nismo pase a ocupar uno de los primeros referentes no tan importante en la *algoritmia secuencial*, pues éste pretende *simular* el efecto de las distintas velocidades de los procesos que ocurren. Esto es lo que en algún contexto podemos llamar hipótesis de *entrelazado*.

El indeterminismo y sus consecuencias están en la raíz del problema que queremos resolver mediante nuestra propuesta de nueva relación de *conformidad*, iocos, descrita en profundidad y formalmente en la siguiente sección.

Para mejor entender el sentido de lo que queremos expresar valga el ejemplo ilustrativo de la figura 3.1. Aunque no hayamos descrito propiamente qué tipo de estructura represente, el sentido de la figura es claro: los vértices representan estados o configuraciones de un sistema y los caminos representan secuencias de acciones observables desde el exterior. Una disyuntiva marca las distintas posibilidades alcanzables desde un estado.

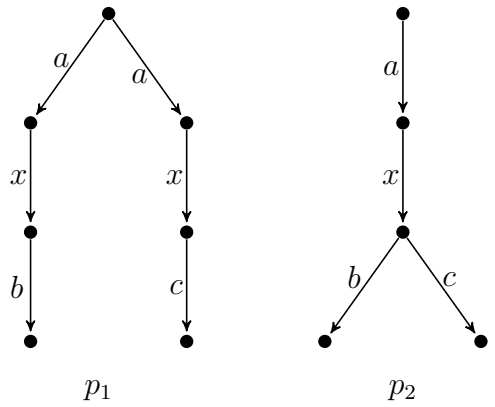


Figura 3.1: Ejemplo ilustrativo del indeterminismo

Bajo cierta perspectiva, existen razones para señalar como indistinguibles ambos sistemas, al estar caracterizados por el mismo conjunto de acciones *potenciales* después de cada secuencia de acciones efectiva a partir del estado inicial.

- Después de la acción a, ambos sistemas están en posición de ejecutar la acción x.

- Del mismo modo, después de la secuencia ax ambos sistemas pueden *eventualmente* realizar acciones b o c.

Fijándonos en el último caso, en otro sentido podría argüirse que *no siempre* los sistemas están en posición de reproducir las mismas acciones

- En el primer sistema, una vez escogido el camino de la izquierda ax se llega a un estado en el que no se tiene la facultad de hacer c, y alternativamente, escogido el camino de la derecha no se llega a un estado por el que se pueda practicar b.
- Por lo que respecta al segundo, al no haber disyuntiva posible a lo largo del camino ax, *siempre* está en disposición de ejecutar cualquier acción concreta, ya sea b o c.

Esta última circunstancia se explica porque en el primer caso el *contexto local* de ejecución exclusiva de cada estado queda *diluido* entre la de los otros estados alcanzados por la misma secuencia de acciones a partir del estado inicial. En el segundo, no sólo mantenemos la perspectiva de las acciones practicadas, sino el estado particular, posiblemente diferente debido a la presencia del *indeterminismo*.

3.1.2. Interacción

En el planteamiento anterior, las acciones se contemplan de manera abstracta, en el sentido de que no se hace distinción sobre su origen. Siendo muy útiles en el análisis de cuestiones como la del *indeterminismo*, su generalidad resulta insuficiente para recoger otros matices que puedan ser interesantes en la observación.

Por ejemplo, puede ser interesante la distinción que se hace entre *comportamiento autónomo* y *comportamiento reactivo*. En el primero, como su nombre indica, es el sistema el que comienza de forma *activa* una acción y el entorno o ambiente sigue. En el segundo, el sistema actúa *reactivamente* a algún estímulo del exterior.

Gráficamente, por convenio suele anotarse con el símbolo de interrogación (?) las acciones reactivas –también denominadas entradas, *inputs*– y con el símbolo de exclamación (!) las acciones autónomas –a la sazón salidas o *outputs*–.

La figura 3.2 marca esta distinción: se trata de un sistema que *reacciona* a una acción $a?$ del exterior llegando a un estado desde el que de manera

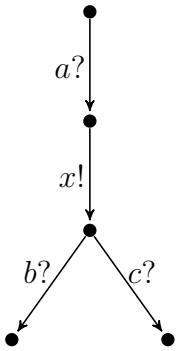


Figura 3.2: Ilustración comportamiento autónomo/reactivo

autónoma se realiza $x!$; el resultado es la transición hacia un estado desde el que se puede reaccionar indistintamente a los eventos del exterior, ya sean $b?$ o $c?$.

Esto será determinante en el capítulo 4, donde se planteará todo un entorno de *testing*. Como veremos, los experimentos de *testing* harán las veces de *entornos* artificiales que son aplicados a una implementación. Las acciones de ambos sistemas serán complementarias, es decir, las acciones de entrada de las implementaciones se corresponderán con las acciones de salida de los experimentos, y viceversa, las acciones de salida de las implementaciones coincidirán con las salidas de los *tests*. La costumbre, no obstante, es referirse a ellas desde el punto de vista de la perspectiva de la implementación.

Ausencia de comportamiento autónomo Atención aparte merecen aquellos estados en los cuales no se produce ninguna acción de salida; dichos estados se denominan *quiescentes*[83, 82, 30]. En ellos el sistema, permanece sin alterarse y reaccionando sólo a una eventual señal o estímulo desde el exterior.

La detección de un estado silencioso conlleva algunas dificultades inherentes que pueden resolverse parcialmente desde el punto de vista técnico con el empleo de un *temporizador* o *timeout*. No obstante, a la hora plantear el diseño de los *tests*, prescindiremos de esta contingencia y asumiremos un símbolo de salida especial, $\delta!$, para poder tratar dicha "ausencia de actividad".

La figura 3.3 muestra la anterior 3.2 completada para marcar este aspecto. En la de la izquierda, utilizando el símbolo $\delta!$ de transición silenciosa. En el de la derecha, empleando el símbolo $\bigcirc$ para sobrecargar menos de simbología el

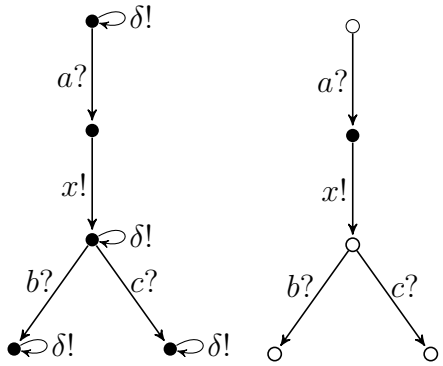


Figura 3.3: Ausencia de comportamiento reactivo

gráfico.

3.1.3. Requisitos y límites en la especificación

El tercer punto sobre el que se apoya nuestra nueva propuesta tiene que ver con los distintos modos en que podemos especificar un sistema. En general, distinguimos dos tendencias:

- Como selección de características mínimo que un sistema *debe* implementar –*carácter de requisito* o aproximación *inicial*–.
- Como selección de características máximo a las características posibles de ejecución –*carácter de límite* o aproximación *final*–.

Ambos enfoques tienen sus inconvenientes y ventajas. De la primera se puede decir que es muy laxa en el sentido de que permite a la implementación *cualquier* comportamiento a partir de unos mínimos. Lo contrario se puede decir de la segunda, pues da como válidas implementaciones más o menos parciales, llegando incluso a la implementación *vacía*.

Éste es precisamente uno de los aspectos en los que se ve afectada la potencia de la relación original ioco, pues cualquier implementación vacía o parcial es correcta con respecto a cualquier especificación.

La figura 3.4 da un ejemplo de lo que queremos decir: el sistema de la izquierda p_1 viene representado por una estructura que es un subárbol estricto de el de la derecha p_2. Por tanto, todo lo relativo a las secuencias de acciones $y!$, $a?x!$, $a?y!$ y $a?z!$ no es tenido en cuenta por la implementación.

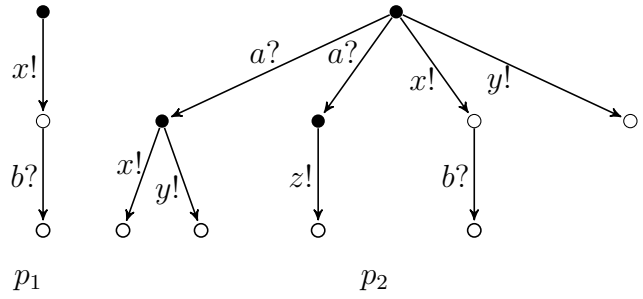


Figura 3.4: Ejemplo de implementación parcial

Nuestro objetivo sería poder reunir lo mejor de ambas aproximaciones: salvando las dificultades de expresar en lenguaje natural, cuestión que será resuelta en la siguiente sección con precisión y claridad gracias a las técnicas coinductivas, podríamos decir que un sistema candidato ha de implementar *al menos* uno de los comportamientos reactivos introducidos por cada acción reactiva que se expresa en la especificación. A partir de ahí, es libre de implementar cuantas acciones reactivas desee.

En efecto, como se puede apreciar en la figura 3.5, el sistema de la izquierda p_1 introduce un nuevo comportamiento reactivo mediante $b?$ y es seguido, de manera excluyente, o por la acción reactiva $a?$ o por la acción autónoma $x!$. En uno y otro caso, las secuencias de acciones $b?a?$, $b?x$ en modo alguno forman parte del sistema de la derecha p_2.

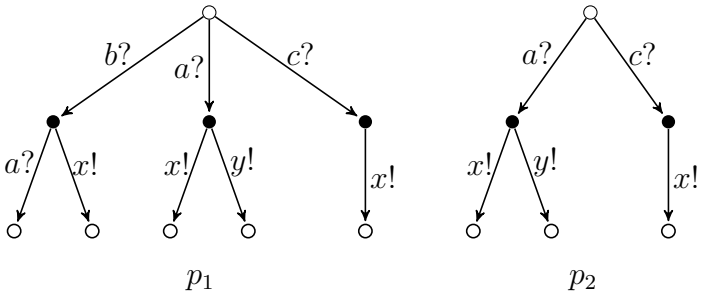


Figura 3.5: Acciones b no especificadas

En lo relativo a los comportamientos autónomos, su único *límite* será el que marca la especificación. Esto significa que en la práctica no hay obligación de implementar las acciones autónomas que estén marcadas por ella. Sin embargo, en lo cualitativo la implementación sólo podrá permanecer *pasiva*, esto es, sin implementar nada, sólo en el caso en que la especificación así lo designe.

Más adelante, veremos que esta distinción se podrá realizar de manera muy natural al considerar el símbolo de *quiescencia* como una acción de salida más, imponiendo una serie de *consistencia* a los estados para los que esté definida este tipo de transición.

3.2. Definiciones

Una vez expuesto el planteamiento y líneas generales de nuestra propuesta, pasamos a exponer de manera más formal y precisa los conceptos utilizados.

Empezaremos repasando los conceptos básicos de los modelos que utilizamos, los llamados *sistemas etiquetados de transiciones*, así como unas definiciones auxiliares que sirven para describir la tradicional relación existente previa a nuestro trabajo: ioco [82].

Basados en las ideas que se han expuesto como motivación, daremos la definición formal de iocos, e ilustraremos mediante ejemplos si se consigue el efecto que perseguimos al comparar su potencia expresiva con la clásica y original ioco. Todo ello quedará demostrado formalmente mediante el teorema 3 que afirma que, en efecto, la relación iocos constituye un refinamiento de ioco.

3.2.1. Modelos

Como se ha citado en el capítulo 2, los sistemas de transiciones etiquetadas, suponen un sencillo mecanismo para describir el comportamiento de un sistema. A lo largo del tiempo se han propuesto muchas variantes capaces de resaltar en menor o mayor grado alguna de las características que, en cada caso, inciden en una propiedad o aspecto de la realidad que se quiere modelar.

En nuestro caso, nuestra especialización requiere que para modelar los sistemas interactivos el conjunto de etiquetas a considerar, L, esté formado por dos clases bien diferenciadas: un conjunto de acciones *autónomas* I y otro de acciones *reactivas* O.

Alternativamente denominadas como acciones de *entrada* o *salida*, el sentido que se les da a estos conjuntos es el que se ha descrito en la anterior sección al hablar de sistemas interactivos. Entonces nos referíamos a la *quiescencia* como un componente esencial en las teorías relativas a dichos sistemas.

Al objeto de perseguir la simplicidad y sin pérdida de generalidad (véase por ejemplo [82]), trataremos de forma directa el evento de la *quiescencia* como una acción especial denotada por $\delta!$ en la definición de nuestros modelos.

En síntesis, el objeto de un sistema etiquetado de transiciones es describir el *comportamiento* de un sistema al interactuar con el entorno que le rodea, describiendo su evolución al estimularlo con acciones de entrada y registrando sus acciones de salida, a veces de manera indeterminista, mediante relaciones de transición entre un estado origen y otro destino. Su expresión formal queda recogida en la siguiente definición.

Definición 16. Un *sistema de transiciones etiquetadas con inputs y outputs* es una tupla $LTS = (S, I, O, \rightarrow)$ tal que:

- S es un conjunto de estados o comportamientos $p, q, r \ldots$
- I y O son conjuntos disjuntos de acciones *inputs* $(a?, b?, c? \ldots)$ y *outputs* $(\ldots, x!, y!, z!)$ respectivamente. Definimos $L = I \cup O$ considerando un nuevo símbolo especial $\delta! \in O$ para la *quiescencia*.
- $\rightarrow \subseteq S \times L \times S$.

Como es habitual, regirán los siguientes convenios:

- $p \xrightarrow{a} q$ en lugar de $(p, a, q) \in \rightarrow$
- $p \xrightarrow{a}$, para cualquier $a \in L$, si existe $q \in S$ tal que $p \xrightarrow{a} q$.
- $p \not\xrightarrow{q}$, para cualquier $a \in L$, si no existe q tal que $p \xrightarrow{a} q$.

Además, para poder aceptar sistemas quiescentes con pleno sentido, añadimos también las siguientes restricciones:

- si $p \xrightarrow{\delta!} p'$ entonces $p = p'$. Por lo tanto, una transición *quiescente* es siempre reflexiva.
- si $p \not\xrightarrow{o!}$ para cualquier $o! \in O \setminus \{\delta!\}$, entonces *necesariamente* $p \xrightarrow{\delta!} p$. A un estado sin salidas se le denomina *quiescente*.
- si existe $o! \in O \setminus \{\delta!\}$ tal que $p \xrightarrow{o!}$, entonces $p \not\xrightarrow{\delta!}$. Dicho de otra manera: un estado *quiescente* no puede ejecutar acciones de salida.

El conjunto de todos los sistemas etiquetados de transiciones se designa mediante $\mathcal{LTS}$.

Exponemos a continuación algunas directrices que contribuyen, en algunos casos, a hacer más fácil la notación con la que referirnos a los sistemas de transiciones etiquetadas.

Convenios de notación

Normalmente utilizaremos las letras $p, q, r \ldots$ para designar los estados de sistemas etiquetados de transición, aunque a veces pueden emplearse las letras i, s para enfatizar el papel de implementación y especificación que ambos sistemas juegan en el contexto de una relación de *conformidad*, concepto central en la teoría **MBT** que introdujimos en el capítulo 2.

Unión de sistemas de transiciones etiquetadas

Sin pérdida de generalidad, podemos considerar implementaciones y especificaciones como estados de un mismo sistema. De esta manera, si tuviésemos dos sistemas, siempre podríamos considerar el sistema que resulta de la unión disjunta de los sistemas originales, renombrando los estados de alguno de ellos cuando éstos tengan algún elemento en común.

La razón para proceder así es que algunas de las herramientas utilizadas para el cómputo de simulaciones (capítulo 5), suelen adoptar este convenio a la hora de implementar los algoritmos. Igualmente la definición coinductiva que vamos a presentar (definición 22) se simplifica bastante, así como el razonamiento en las demostraciones.

Definición 17. Sean $(S_1, I_1, O_1, \rightarrow_1), (S_2, I_2, O_2, \rightarrow_2) \in \mathcal{LTS}$ dos sistemas tales que $S_1 \cap S_2 = \varnothing$. La unión de dos sistemas se define como un nuevo $(S, I, O, \rightarrow) \in \mathcal{LTS}$ donde:

- $S = S_1 \cup S_2$
- $I = I_1 \cup I_2$ y $O = O_1 \cup O_2$
- $s_1 \xrightarrow{a} s$ sí y sólo si $s_1 \xrightarrow{a}_1 s_2$ o $s_1 \xrightarrow{a}_2 s_2$

Trazas y operadores asociados

Ya hemos señalado en la sección anterior lo intrincado que resulta dar cuenta en lenguaje natural de las diferencias y semejanzas que introduce la presencia del no determinismo en los sistemas etiquetados de transiciones.

Para formular con una mayor precisión estos conceptos, debemos dotarnos de definiciones de algunos elementos auxiliares e incluir operadores sobre los mismos. Tal es el caso de las llamadas trazas. Las *trazas* juegan un papel importante a la hora de reunir información sobre los comportamientos y se hacen necesarias para la descripción de la relación que queremos plantear.

Definición 18. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$. Teniendo en cuenta que $L = I \cup O$

- Una *traza* σ es una secuencia finita de símbolos de L. El conjunto de trazas se denota por L^*. El símbolo $\epsilon \in L^*$ denota la *traza* vacía.
- Sea $\sigma_1, \sigma_2 \in L^*$. Con la yuxtaposición $\sigma_1\sigma_2$ indicamos la concatenación de *trazas*.

Como habíamos descrito en el capítulo 2 la relación de transición de los sistemas de transiciones etiquetadas según ha sido introducida en la definición 16 puede extenderse de modo natural para contemplar el uso de trazas en lugar de acciones singulares. El resultado es una nueva relación de transición que se define a continuación.

Definición 19. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$, $p, q \in S$ y $\sigma \in L^*$.

Se define inductivamente $\rightarrow \subseteq S \times L^* \times S$ del modo siguiente:

- $p \xrightarrow{\epsilon} p$
- $p \xrightarrow{a\sigma} q$ para $a \in L$, $\sigma \in L^*$ y $p' \in S$ tal que $p \xrightarrow{a} p'$ y $p' \xrightarrow{\sigma} q$.

Por último, se muestran unos operadores que ponen en relación todo lo expuesto hasta aquí: estados, acciones de entrada y salida y trazas.

Definición 20. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$, $p \in S$, y $\sigma \in L^*$. Se define:

1. $\mathsf{init}(p) = \{a \mid a \in L,\ p \xrightarrow{a}\}$, el conjunto de acciones iniciales de p.

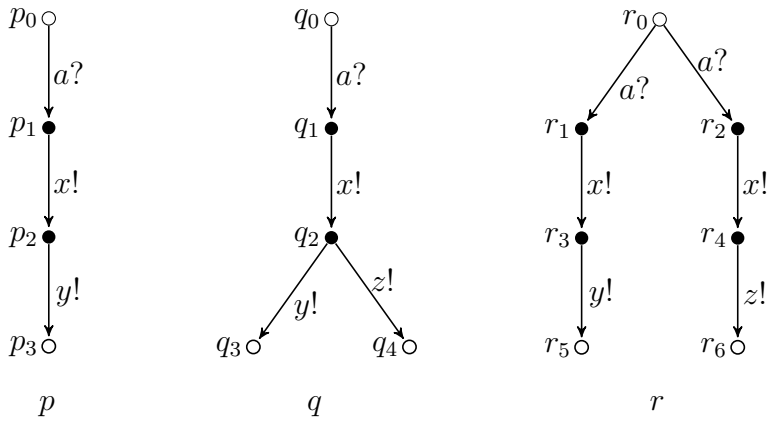


Figura 3.6: Ejemplos de sistemas de transiciones etiquetadas

2. $\mathsf{traces}(p) = \{\sigma \mid \sigma \in L^*,\ p \xrightarrow{\sigma}\}$, el conjunto de trazas desde p.

3. $p\,\mathsf{after}\,\sigma = \{p' \mid p' \in S,\ p \xrightarrow{\sigma} p'\}$, el conjunto de estados alcanzables desde p después de la ejecución de la traza σ.

4. $\mathsf{outs}(p) = \{x \mid x \in O,\ p \xrightarrow{x}\}$, el conjunto de *outputs* de un estado p .

5. $\mathsf{outs}(S') = \bigcup_{p \in S'} \mathsf{outs}(p)$, el conjunto de *outputs* de un conjunto de estados S'.

6. $\mathsf{ins}(p) = \{x? \mid x? \in I,\ p \xrightarrow{x?}\}$, el conjunto de *inputs* de un estado p.

A modo de ilustración, en la figura 3.6, podemos ver que las trazas $a?x!y!$ son alcanzables desde p_0, q_0, r_0 en todos los casos. Así mismo, p_0 after $a?x! = \{p_2\}$, q_0 after $a?x! = \{q_2\}$, y r_0 after $a?x! = \{r_3, r_4\}$. También $\mathsf{outs}(p_0) = \mathsf{outs}(q_0) = \mathsf{outs}(r_0) = \{\delta!\}$, y por último, $\mathsf{outs}(p_0\,\mathsf{after}\,a?x!) = \{y!\}$, $\mathsf{outs}(q_0\,\mathsf{after}\,a?x!) = \{y!, z!\}$, $\mathsf{outs}(r_0$ after $a?x!) = \{y!, z!\}$.

3.2.2. La relación clásica ioco

En el apartado anterior hemos definido con precisión el formalismo que caracteriza los modelos de sistemas que vamos a estudiar: sistemas de transiciones etiquetadas distinguiendo entre capacidades *autónomas* y *reactivas*.

Según se ha descrito en el capítulo 2, disponer de una teoría de modelos resulta fundamental en áreas de aplicación como el *Testing Basado en Modelos*,

MBT. Una de las más ampliamente diseminadas y conocidas teorías, con sus relaciones de *conformidad* y generación automática de *tests* es ioco [82], que precisamente usa como modelos los *sistemas de transiciones etiquetadas* en su versión *interactiva*, esto es, haciendo distinción entre comportamientos autónomos y reactivos.

Restricciones *input enabled* Originalmente, la relación ioco que a continuación vamos a definir impone un requisito semántico adicional a los objetos que identifica como implementaciones formuladas sobre los sistemas de transiciones etiquetadas, según se describe en [82]: allí se cita que éstas deberían ser *input enabled*, es decir, preparadas para reaccionar ante cualquier acción de entrada. En la figura 3.7 podemos ver representada esta condición.

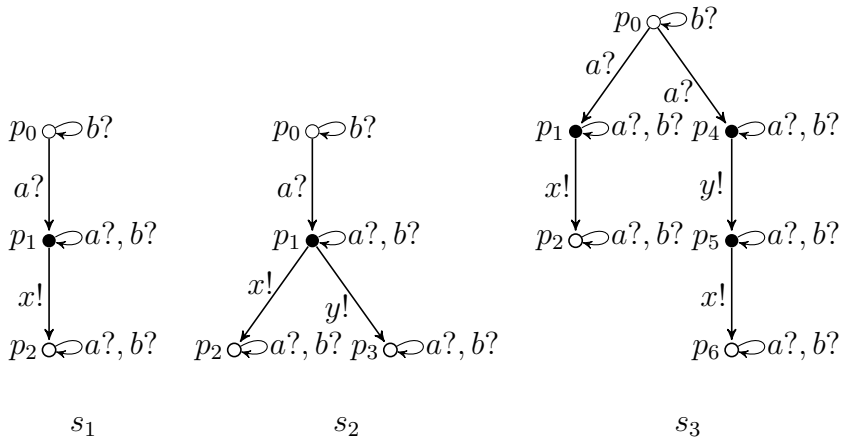


Figura 3.7: Sistemas habilitados para entradas con $I = \{a?, b?\}$

Aunque esta hipótesis puede ser natural en algunos contextos, no lo es en muchos otros. Valgan los siguientes ejemplos:

- En una máquina expendedora, ya sea para una tarjeta de crédito o un billete de aparcamiento, la ranura puede estar sólo habilitada si no está insertada una tarjeta.
- Ya en una plano más virtual, con los controles de los interfaces gráficos de usuario, los programadores no necesitan considerar todos los posibles eventos o acciones que ocurren en una ventana; más bien se encargan de codificar la respuesta de los eventos interesantes en cada momento así

como *deshabilitar* la interacción con los controles según el estado en que se encuentren.

Diferente dominio de definición Hay otro segundo matiz en la teoría ioco que también escapa a la descripción que hemos hecho de nuestros modelos de referencia: mientras que los objetos identificados como implementaciones deben ser *input enabled* las especificaciones no necesitan cumplir este requisito.

La consecuencia de todo esto es que la relación original ioco queda así definida entre dos dominios distintos, sistemas de transiciones etiquetadas interactivos para las especificaciones e *input-enabled* para las implementaciones.

Esta diferencia de dominios impide que la relación ioco pueda ser considerada transitiva y como tal no puede usarse como una relación de refinamiento sucesivo

$$s_n \leq s_{n-1} \leq s_{n-2} \leq \cdots \leq s_2 \leq s_1$$

pues una vez que se obtiene una implementación conforme a la especificación, quedan fijados por ésta todos los comportamientos respecto a las acciones de *input*; hay muy poco margen, cuando éste exista, para continuar refinando el proceso.

En nuestra relación de *conformidad* (definición 22 de la sección 3.2.3) no se requerirá que las implementaciones sean *input enabled*. Como ya es costumbre en otros entornos de *testing*, las especificaciones e implementaciones se expresarán con el mismo formalismo sin añadir condiciones extra. En particular, usaremos los sistemas etiquetados de transiciones tal como aparecen expuestos en la definición 16 tanto para implementaciones como para especificaciones.

Sin embargo, al objeto de comparar la expresividad de la relación original ioco con la relación de *conformidad* que se definirá en la sección siguiente, tendremos que considerar la relación ioco original ligeramente modificada, a fin de estar definida sobre el mismo dominio tanto para implementación como especificación. No obstante, en la sección 3.3, los interesados en ver la relación original en comparación con la propuesta podrán ver teorema de interés en el que, formuladas sobre dominios *input enabled* se afirma que las dos relaciones son indistinguibles.

Hechas estas aclaraciones respecto a las pequeñas divergencias con respecto a su formulación original, exponemos a continuación las definiciones formales

de la relación ioco:

Definición 21. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$, la relación $\mathsf{ioco} \subseteq S \times S$ se define como sigue:

$$p \,\mathsf{ioco}\, q \Leftrightarrow_{def} \forall \sigma \in \mathsf{traces}(s) : \mathsf{outs}(p \,\mathsf{after}\, \sigma) \subseteq \mathsf{outs}(q \,\mathsf{after}\, \sigma)$$

Como es costumbre al formular una relación de *conformidad*, convenimos que cuando dos sistemas son puestos en relación por ella, $p \,\mathsf{ioco}\, q$, el primero p constituye una implementación válida del segundo q. Al respecto, cabe decir que en otros contextos en lugar de expresar $p \,\mathsf{ioco}\, q$ se emplean otras letras más significativas, como en $i \,\mathsf{ioco}\, s$.

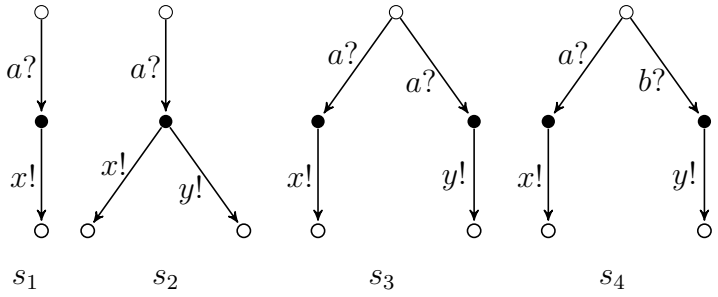


Figura 3.8: Comparación de sistemas parecidos mediante ioco

Los ejemplos de la figura 3.8 ilustran algunos casos a los que se puede aplicar la definición 21. Las relaciones que se dan entre ellos se recogen el cuadro 3.1.

ioco	$s1$	$s2$	$s3$	$s4$
$s1$	✓	✓	✓	✓
$s2$	✗	✓	✓	✗
$s3$	✗	✓	✓	✗
$s4$	✓	✓	✓	✓

Cuadro 3.1: Relaciones ioco entre los sistemas de la figura 3.8

Obsérvese que el primer sistema s_1 es implementado sólo por sí mismo y por s_4 , pues en todos encontramos trazas ($a?$) después de las cuales se llega a un estado en el que el conjunto de *outputs* $\{x!, y!\}$ supera al que se produce en la especificación $\{x!\}$.

Tomando s_2 y s_3 como especificación, resulta que *todos* son válidos como implementación; en efecto, s_1 y s_4 después de la traza (a) producen el subconjunto $\{x!\}$, trivialmente subconjunto de $\{x!, y!\}$. Para s_2 y s_3, conjunto de salidas coincide.

Nótese que en s_3 se ha introducido el *indeterminismo* y éste pasa desapercibido para la relación ioco, al obtener el mismo valor sobre el conjunto $\mathsf{outs}(s_2 \,\mathsf{after}\, a) = \{x!, y!\} = \mathsf{outs}(s_3 \,\mathsf{after}\, a)$. Registrar este hecho resulta fundamental para entender la razón de ser nuestra propuesta que formularemos en el apartado siguiente.

Por último, en el caso de s_4, sólo s_1 figura como implementación válida, pues tanto s_2 como s_3 producen el superconjunto $\{x!, y!\}$ del conjunto obtenido después de la traza $a?$,$\{x!\}$. Llama la atención la traza formada por la acción $b?$. Al no ser aplicable en el resto de los sistemas, s_1,s_2,s_3...el conjunto de salidas asociado a ella es el conjunto vacío, que trivialmente está incluido en cualquiera, en particular, del primero.

3.2.3. iocos, una propuesta basada en simulación

A la vista de los ejemplos de el cuadro 3.1, es evidente que la relación ioco hace pocas discriminaciones entre sistemas de naturaleza muy distinta. Podemos aventurar el siguiente diagnóstico:

- Cierta *laxitud* a la hora de permitir la ausencia de comportamientos reactivos *prescritos* por la especificación e ignorados por las implementaciones.

- Incapacidad de distinguir la acción del *indeterminismo*. La potencialidad de algunos estados queda *diluida* por la presencia de otros a los que se ha llegado mediante la misma traza.

Para enmendar esta situación precisaremos del uso de técnicas matemáticas un poco más complejas que la simple *inclusión* de conjuntos que se usa en la definición 21. Éstas incluyen las definiciones recursivas sobre el conjunto de los estados de un sistema etiquetado de transiciones, para los cuales no siempre hay definido un orden *noetheriano*, pues no es posible encontrar un caso base. Por tanto, no es posible abordarla directamente de *modo inductivo*. La solución

vendrá dada por otro tipo de definición *coinductiva*, empleada por Milner [77] en sus primeros trabajos sobre *bisimulación*.

Definición 22. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$, decimos que una relación $R \subseteq S \times S$ es una iocos-relación si y solo si para cualquier $(p, q) \in R$ se cumplen las siguientes condiciones:

1. $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$
2. Para cualquier $a? \in \mathsf{ins}(q)$ tal que $p \xrightarrow{a?} p'$ existe $q' \in S$ tal que $q \xrightarrow{a?} q'$ y $(p', q') \in R$.
3. Para cualquier $x \in \mathsf{outs}(p)$ tal que $p \xrightarrow{x} p'$ existe $q' \in S$ tal que $q \xrightarrow{x} q'$ y $(p', q') \in R$.

A continuación hacemos unas cuantas observaciones sobre la formulación de la definición anterior:

- La cláusula 22.2 no implica la cláusula 22.1 ni viceversa. El sólo cumplimiento de la primera no garantiza que el sistema p siga recursivamente la especificación marcada por q. Sólo registra el hecho de que al menos, el sistema p tiene que tener una transición definida. Recíprocamente, la segunda por sí misma es aceptable para un sistema p que no tenga transiciones definidas aun cuando q sí tenga.
- La cláusula 22.1 en conjunción con la 22.2 de la definición es la que formaliza el carácter *obligatorio* para la implementación de desarrollar al menos un comportamiento reactivo introducido por la cada acción de entrada en la especificación. Nótese que la condición $a? \in \mathsf{ins}(q)$ –y no $a? \in \mathsf{ins}(p)$, como en otro pariente conocido, la simulación *ready*– resulta clave para permitir otros comportamientos reactivos no expresamente indicados en la especificación.
- La cláusula 22.3, por sí sola *limita* la capacidad de autónoma de la implementación. Teniendo en cuenta que el símbolo de *quiescencia* está integrado en el conjunto de símbolos de salida, es claro que la implementación sólo puede permanecer *pasiva* de acuerdo con la especificación si ésta lo es, pues el conjunto $\{\delta!\}$ sólo puede ser subconjunto de si mismo

($\{\delta!\} \subseteq \{\delta!\}$) dadas las condiciones que garantizan la consistencia de los estados *quiescentes*.

Casos de ejemplo Podemos comprobar que la relación propuesta

$$R_1 = \{(s_1, s'_1), (s_2, s'_2), (s_3, s'_3), (s_4, s'_4)\}$$

sobre los sistemas i y s expuestos en la figura 3.9 no es una iocos-relación conforme a la definición 22.

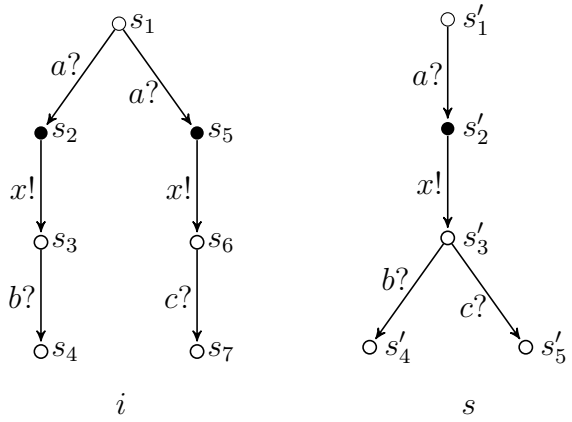


Figura 3.9: R_1 no es iocos-relación

Tomemos el par (s_1, s'_1). Comprobamos que, efectivamente $\mathsf{ins}(s_1) \supseteq \mathsf{ins}(s'_1)$, por lo que se cumple la cláusula 22.1. Vemos que $s_1 \xrightarrow{a?} s_2$ y que $s'_1 \xrightarrow{a?} s'_2$, con $(s_2, s'_2) \in R_1$ (cláusula 22.2). La cláusula 22.3 es cierta al tener ambos estados el mismo conjunto de símbolos de salida, el conjunto unitario $\{\delta!\}$. Tenemos que ver ahora si el nuevo par (s_2, s'_2) cumple la definición; nuevamente $\mathsf{ins}(s_1) = \varnothing \supseteq \mathsf{ins}(s'_1) = \varnothing$, por lo que se cumple la cláusula 1. Al no haber acciones *input*, se cumple trivialmente la cláusula 22.2. Para la única acción *output* tal que $s_2 \xrightarrow{x!} s_3$ se tiene que $s'_2 \xrightarrow{x!} s'_3$, y $(s_3, s'_3) \in R_1$, por lo que de nuevo la cláusula 22.3 se cumple. Al hacer el escrutinio con el par (s_3, s'_3) vemos que $\mathsf{ins}(s_3) = \{b?\} \subset \{b?, c?\}$ por lo que no se cumple la cláusula 22.1 y la relación propuesta no puede considerarse una iocos-relación.

En la figura 3.10 vemos representado un caso en el que

$$R_2 = \{(s_1, s'_1), (s_2, s'_2), (s_3, s'_3), (s_4, s'_4)(s_5, s'_5)\}$$

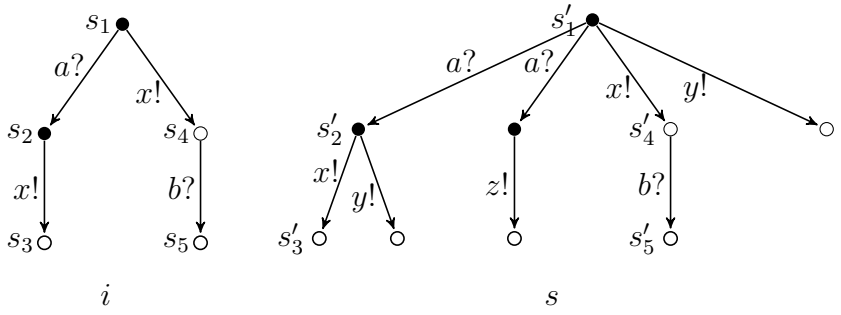


Figura 3.10: R_2 es una iocos-relación.

es una iocos-relación. Partiendo del primer par $(s_1, s'_1) \in R_2$, vemos la cláusula 22.1 se cumple, pues $\mathsf{ins}(s_1) = \{a?\} \supseteq \{a?\} = \mathsf{ins}(s'_1)$. Al tener $s_1 \xrightarrow{a?} s_2$ vemos que $s'_1 \xrightarrow{a?} s'_2$, con $(s_2, s'_2) \in R_2$, conforme a la cláusula 22.2. De nuevo, al considerar $(s_2, s'_2) \in R_2$ tenemos que $s_2 \xrightarrow{x!} s_3$ al tiempo que $s'_2 \xrightarrow{x!} s'_3$, y $(s_3, s'_3) \in R_2$. Al ser *quiescentes*, es decir, $s3 \xrightarrow{\delta!} s3$ y $s3' \xrightarrow{\delta!} s3'$ y sin ninguna *acción* de entrada, trivialmente cumplen la propiedad.

Sólo falta el caso en que $s_1 \xrightarrow{x!} s_4$, partiendo de la configuración $(s_1, s'_1) \in R_2$. En ese caso $s'_1 \xrightarrow{x!} s'_4$, con $(s_4, s'_4) \in R_2$. Como $\mathsf{ins}(s_4) = \{b?\} \supseteq \{b?\} = \mathsf{ins}(s'_4)$, se cumple la cláusula 22.1. Al tener $s_4 \xrightarrow{b?} s_5$, vemos que $s'_4 \xrightarrow{b?} s'_5$, y $(s_5, s'_5) \in R_2$ cumpliéndose la clausula 22.2. Al ser *quiescentes* (s_5, s'_5) cumplen trivialmente los requisitos de la definición 22.

La definición de una iocos-relación es sólo la mitad del proceso de una definición *coinductiva*, que establece que la relación que perseguimos resulta ser la mayor de cuantas cumplen la definición 22. Expresado de modo formal, adopta la siguiente definición.

Definición 23. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ y $p, q \in S$. Decimos que p iocos q , si y sólo si existe una iocos-relación entre ellos, o equivalentemente

$$\mathsf{iocos} = \bigcup \{R \mid R \subseteq S \times S,\ R \text{ es una } \mathsf{iocos}\text{-relation}\}$$

A continuación se exponen algunas valoraciones en torno a cualidades de diversa índole de la relación iocos que se pueden colegir de la anterior definición.

Complejidad

Siguiendo la definición dada, para saber si implementación y especificación cumplen la propiedad iocos, sería necesario analizar todas las posibles relacio-

nes que se pueden dar entre sus estados, 2^{2^n}, hasta buscar alguna que pudiera reunir los requisitos de una iocos-relación, lo cual resulta ser muy ineficiente.

En esta tesis afrontaremos el problema de dos maneras bien distintas; en primer lugar mediante técnicas de *testing* en el capítulo 4, y mediante técnicas de algoritmia de reducción de grafos asociados, lo que será el objeto del capítulo 5.

Preorden

Con anterioridad hemos justificado la eliminación de la hipótesis *input enabled*, o compromiso para reaccionar a todo tipo de señales de entrada, al objeto de permitir en la relación de conformidad iocos la posibilidad de refinamientos sucesivos a partir de una especificación. El concepto matemático que permite describir este hecho es la de *preorden* y el siguiente lema 1, conjuntamente con el teorema 2, permiten describir iocos como preorden sobre el conjunto $\mathcal{LTS}$:

Lema 1. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$. *Las siguientes propiedades se cumplen:*

- *La relación identidad sobre* S *, es decir,* $Id = S \times S$ *es una* iocos*-relación.*
- *Sean* $R, R' \subseteq S \times S$ *dos* iocos*-relaciones, entonces*

$$R \circ R' = \{(p, r) \mid \exists q \in S : \ (p, q) \in R \wedge (q, r) \in R'\}$$

  es una iocos*-relation.*

Demostración. Consideremos la relación identidad $Id \subseteq S \times S$, formada por los pares en los que el primer elemento es igual al segundo, es decir $Id = \{(s_1, s_1), (s_2, s_2), (s_3, s_3) \dots\}$. Para cualquier par (s_i, s_i), al tener $\mathsf{ins}(s_i) = \mathsf{ins}(s_i)$, se cumple que $\mathsf{ins}(s_i) \supseteq \mathsf{ins}(s_i)$, por lo que se cumple la cláusula 22.1. De igual forma, para cada $a? \in \mathsf{ins}(s_i)$, si $s_i \xrightarrow{a?} s_i'$ entonces existirá un $s \in S$ –el mismo s_i'–, tal que $s_i \xrightarrow{a?} s_i'$. Al tratarse de la relación identidad, el par (s_i', s_i') está incluido en el la relación, cumpliéndose la clásula 22.2. Por el mismo razonamiento aplicado a las acciones de salida se obtiene que cumple la cláusula 22.3.

Consideremos ahora la relación $R \circ R'$. Para cualquier par de elementos $(s_i, s_i') \in R \circ R'$ existe, por definición, un elemento s_i'' tal que $(s_i, s_i'') \in R$ y $(s_i'', s_i') \in R'$. Al tratarse de iocos-relaciones, tendremos que $\mathsf{ins}(s_i'') \subseteq \mathsf{ins}(s_i)$ y $\mathsf{ins}(s_i') \subseteq \mathsf{ins}(s_i'')$ por la clásula 22.1. Por la propiedad de la inclusión de los conjuntos tendremos que $\mathsf{ins}(s_i') \subseteq \mathsf{ins}(s_i)$, luego se cumple la clásula 22.1. Si $s_i \xrightarrow{a?} s_j$ con $a? \in \mathsf{ins}(s_i'')$, entonces por ser R una iocos-relación tendremos que $s_i'' \xrightarrow{a?} s_j''$, y $(s_i'', s_j'') \in R$ para algún $s_j'' \in S$. Además, por ser R' una iocos-relación, tenemos que si $s_i'' \xrightarrow{a?} s_j''$ con $a? \in \mathsf{ins}(s_i')$ entonces $s_i' \xrightarrow{a?} s_j'$, con $(s_j'', s_j') \in R'$ para algún $s_j' \in S$. Por la definición de $R \circ R'$, el par $(s_j, s_j') \in R \circ R'$, luego $R \circ R'$ cumple la cláusula 22.2. El mismo razonamiento se aplica cuando consideramos una acción de salida. □

Teorema 2. *La relación* iocos *es un preorden definido sobre el conjunto* $\mathcal{LTS}$.

Demostración. Para demostrar que es un preorden, se necesita demostrar que cumple las propiedades reflexiva y transitiva.

- Por el lema 1, tenemos que la relación $Id \subseteq S \times S$ es una iocos-relación. Al ser iocos la mayor de las iocos-relaciones,(definición 23), tenemos que $Id \subseteq \mathsf{iocos}$, luego $a \,\mathsf{iocos}\, a$ para cualquier a, cumpliendo así la propiedad reflexiva.

- Supongamos que $a \,\mathsf{iocos}\, b$ y $b \,\mathsf{iocos}\, c$. Por la definición 23, existen dos iocos-relaciones R, R', tal que $(a, b) \in R$ y $(b, c) \in R'$. Por la definición de $R \circ R'$, $(a, c) \in R \circ R'$, y por el lema 1 $R \circ R'$ es una iocos relación, luego por la definición 23, $a \,\mathsf{iocos}\, c$, cumpliendo así la propiedad transitiva.

□

3.2.4. iocos como refinamiento de ioco

Una vez que hemos dado una definición formal para iocos, es nuestro objetivo valorar si, efectivamente, hemos conseguido lo que pretendíamos: un mayor grado de refinamiento que la relación clásica de ioco.

El cuadro 3.2 muestra las comparaciones entre los sistemas de la figura 3.8, ahora bajo la perspectiva de la definición 22.

ioco	$s1$	$s2$	$s3$	$s4$
$s1$	✓	✓	✓	✗
$s2$	✗	✓	✗	✗
$s3$	✗	✓	✓	✗
$s4$	✓	✓	✓	✓

Cuadro 3.2: Relaciones $\underline{\text{iocos}}$ entre los sistemas

Excluyendo las relaciones reflexivas, se puede observar que de un total de 12 posibles comparaciones, de las cuales originalmente 2 ni siquiera cumplían los requisitos de la clásica ioco, siguen sin cumplir los criterios, y del resto, 10, un total de 3 han sido discriminadas por el poder expresivo de la nueva relación $\underline{\text{iocos}}$.

Estos datos sugieren la idea de que, en primer lugar, se trata de relaciones comparables, y en segundo, que hemos conseguido una relación de *conformidad* con más poder discriminatorio que la original. Esto encaja perfectamente con lo esperado, pues en diversos estudios [89, 87, 88, 22, 76] se asegura que el poder de las semánticas definidas en base a *trazas*, también conocidas como semánticas *lineales*, a la que pertenece ioco por el modo en que está definida (definición 21) es siempre más laxo que el de aquellas basadas en definiciones coinductivas, caso de la que hemos propuesto, $\underline{\text{iocos}}$ (definición 22.

A continuación, esta –hasta el momento– conjetura queda demostrada mediante el siguiente teorema :

Teorema 3. *La relación* $\underline{\text{iocos}}$ *supone un refinamiento de la relación* ioco*, es decir*

$$\underline{\text{iocos}} \subseteq \text{ioco}$$

Demostración. Consideremos $p, q \in S$ tal que $p \; \underline{\text{iocos}} \; q$. De acuerdo con la definición 21, basta probar

$$\forall \sigma \in \text{traces}(q) : \text{outs}(p \text{ after } \sigma) \subseteq \text{outs}(q \text{ after } \sigma)$$

lo que haremos por inducción sobre la longitud de la traza σ.

Base: $\sigma = \epsilon$**.** En primer lugar, $\text{outs}(p \text{ after } \epsilon) = \text{outs}(p)$ y $\text{outs}(q \text{ after } \epsilon) = \text{outs}(q)$, por la definición del operador after (definición 20). Luego $\text{outs}(p) \subseteq \text{outs}(q)$, ya que $p, q \in \underline{\text{iocos}}$ por hipótesis, y por la cláusula 22.3 .

Inducción: $\sigma = a\sigma', a \in L, \sigma' \in L^*$**.** Asumiendo la hipótesis de inducción sobre σ'

- Si $p \overset{a}{\not\Rightarrow}$ entonces $\sigma \notin \mathsf{traces}(p)$ y $\mathsf{outs}(p \text{ after } \sigma) = \varnothing$. (definición 20 de `traces` y `after`). De manera trivial $\varnothing \subseteq \mathsf{outs}(q \text{ after } \sigma)$
- Consideremos p' tal que $p \xrightarrow{a} p'$. Puesto que, por hipótesis, $\sigma \in \mathsf{traces}(q)$, $q \xrightarrow{a}$. A partir de aquí consideramos dos casos:
  - Si $a \in I$ entonces $a \in \mathsf{ins}(q)$, y por la definición 22.2 existe $q' \in S$ tal que $q \xrightarrow{a} q'$ y $p' \mathsf{iocos}\underline{\ } q'$.
  - Si $a \in O$, entonces por la definición 22.3 existe $q' \in S$ tal que $q \xrightarrow{a} q'$ y $p' \mathsf{iocos}\underline{\ } q'$. Por consiguiente, en cualquier caso existe $q' \in S$ tal que $q \xrightarrow{a} q'$ y $p' \mathsf{iocos}\underline{\ } q'$.

  En ambos casos, por inducción $\mathsf{outs}(p' \text{ after } \sigma') \subseteq \mathsf{outs}(q' \text{ after } \sigma')$. Luego

$$\mathsf{outs}(p \text{ after } \sigma) = \mathsf{outs}(p \text{ after } a\sigma') = \bigcup\{\mathsf{outs}(p' \text{ after } \sigma') \mid p \xrightarrow{a} p'\} \subseteq$$
$$\bigcup\{\mathsf{outs}(q' \text{ after } \sigma') \mid p \xrightarrow{a} p', q \xrightarrow{a} q', p' \mathsf{iocos}\underline{\ } q'\} \subseteq$$
$$\bigcup\{\mathsf{outs}(q' \text{ after } \sigma') \mid q \xrightarrow{a} q'\} = \mathsf{outs}(q \text{ after } a\sigma') = \mathsf{outs}(q \text{ after } \sigma)$$

□

3.3. ioco como simulación

En la sección anterior se ha explicado que originalmente la definición de ioco incluía un requisito muy *fuerte*: las implementaciones debían de ser *input enabled*, esto es, deberían obligatoriamente reaccionar a cualquier estímulo de entrada. Procediendo de esta manera se perdía la importante propiedad transitiva de la relación, al estar definidos implementación y especificación en dominios diferentes.

Llegados a este punto, y demostrado que ambas relaciones ioco e iocos son comparables, siendo iocos un refinamiento de ioco, nos preguntamos qué relación existiría entre ambas en el caso de que ioco contara con el requisito tan fuerte de *input enabled*.

El resultado es que ambas llegan a ser indistinguibles, o dicho de otro modo, equivalentes.

Teorema 4. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$*, con* $p, q \in S$ *tal que* p *reúne las condiciones para ser* input-enabled *y que* $p \,\mathsf{ioco}\, q$*. Entonces* $p \,\mathsf{iocos}\, q$

Demostración. Para probar que $p \,\mathsf{iocos}\, q$, tenemos que encontrar una R que sea iocos-relación tal que $(p, q) \in R$.

Consideremos la siguiente:

$$R = \{(p_1, p_2) \mid \exists \sigma \in traces(s) : \; p_1 \in p \,\mathsf{after}\, \sigma,\, p_2 \in q \,\mathsf{after}\, \sigma\}$$

Es obvio que $(p, s) \in R$ al tomar la traza vacía $\epsilon \in \mathsf{traces}(s)$.

Sólo resta considerar $(p_1, p_2) \in R$ y ver que R reúne efectivamente los requisitos de una iocos-relación (definición 22).

Recordemos que por construcción, siempre tendremos una traza $\sigma \in traces(q)$ tal que $p_1 \in p \,\mathsf{after}\, \sigma$ y $p_2 \in q \,\mathsf{after}\, \sigma$ ya que $(p_1, p_2) \in R$.

- $\mathsf{ins}(p_1) \supseteq \mathsf{ins}(p_2)$. Ya que p es *input enabled* tenemos: $\mathsf{ins}(p_1) = I \supseteq \mathsf{ins}(p_2)$, cláusula 22.1.

- Consideremos $a? \in ins(p_2)$, o lo que es lo mismo, p_2' tal que $p_2 \xrightarrow{a?} p_2'$. Necesariamente existe $P_1 1'$ tal que $p_1 \xrightarrow{a?} p_1'$, por el punto anterior. Bajo estas condiciones, se obtiene $\sigma a? \in traces(q)$, $p_2' \in s \,\mathsf{after}\, \sigma a?$ y $p_1' \in p \,\mathsf{after}\, \sigma a?$. Así $(p_1', p_2') \in R$ por la forma en que está construido R, lo que confirma la cláusula 22.2.

- Supongamos $x! \in O$ de modo que $p_1 \xrightarrow{x!} p_1'$. Entonces $x! \in \mathsf{outs}(p_1) = \mathsf{outs}(p \,\mathsf{after}\, \sigma)$. Como $p \,\mathsf{ioco}\, q$ y $\sigma \in traces(q)$, entonces $\mathsf{outs}(p \,\mathsf{after}\, \sigma) \subseteq \mathsf{outs}(q \,\mathsf{after}\, \sigma)$. Por lo tanto, $x! \in \mathsf{outs}(q \,\mathsf{after}\, \sigma) = \mathsf{outs}(p_2)$, esto es, existe p_2' tal que $p_2 \xrightarrow{x!} p_2'$. De esto se obtiene que $\sigma x! \in traces(q)$. Entonces $p_1' \in p \,\mathsf{after}\, \sigma x$ y $p_2' \in p \,\mathsf{after}\, \sigma x$, luego $(p_1', p_2') \in R$ por la construcción de R.

□

El teorema 4 parece sugerir un hecho en contra de otros estudios [90, 89, 88] que afirman que las relaciones basadas en semánticas lineales son menos expresivas que las llamadas semánticas *branching*, pero esto sólo es de modo aparente, pues la definición original de ioco impone un requisito muy fuerte

como es el poder reaccionar a cada estímulo, imposible de valorar en el estudio general de todas las semánticas.

Por otra parte, esta causa, lejos de ser descartada por nuestro estudio será rescatada en el siguiente capítulo donde introduciremos los *tests*, especie de entornos artificiales creados a partir de un sistema-especificación, para los que, ahora sí, será condición irrenunciable el estar preparados para reaccionar ante cualquier señal de se salida por parte del sistema del que queremos acreditar su relación de *conformidad*.

Capítulo 4

Testing basado en modelos

Una vez expuesto con todo detalle en el capítulo anterior la relación iocos, en el presente nos interesamos por dos metodologías que nos permitirán, mediante *tests*, saber cuándo una propuesta de implementación es correcta respecto a una especificación. Ambas tienen en común que el modelo del objeto a examinar, la *Implementation Under Testing* (**IUT**), se presupone formulado en un dominio del que se desconocen los detalles.

Con la primera técnica, denominada *offline*, dada una especificación conseguiremos de modo *automático* una batería de *tests* con la capacidad de determinar si una implementación es correcta. Para ello será necesario dotarnos de una sintaxis para referirnos a los *tests* así como una caracterización de la relación iocos en función de los mismos. Como no podría ser de otro modo, de ella se comprueba formalmente su *corrección* y *exhaustividad*.

Como es común en este tipo de situaciones, el problema de la *explosión de estados* impone unas limitaciones tanto en memoria como en tiempo, razón por la que se abordará una segunda técnica, denominada *online*, en la que de manera alternativa se sucederán los pasos de *generación* y *ejecución* de test. No obstante, en este nuevo contexto, será preciso asumir unos requisitos extra en la ejecución a la hora de valorar distintas posibilidades en un contexto indeterminista, a los que nos referiremos de manera general como hipótesis de *justicia* (*fairness*).

4.1. Testing *offline*

En ocasiones, la modalidad *testing offline*, o *testing estático*, tiende a confundirse con el término más global, *Model Based Testing*, **MBT**, descrito en líneas generales en el capítulo 2, el cual puede comprender en la actualidad otras modalidades, como el *testing online* o *testing dinámico*, como veremos en la sección 4.2. El origen de esta ambigüedad o imprecisión cabe atribuirla a que históricamente en la literatura apareció primero.

En el marco más concreto que nos ocupa, la relación de *conformidad* iocos entre un sistema respecto a una especificación dada y su posible caracterización por métodos de *testing*, lo primero que debemos abordar es la descripción de los elementos que consideraremos como *test*.

Al optar por referirnos a ellos mediante términos, lo más conveniente es formular una sencilla *sintaxis* que permita mediante definiciones *inductivas* identificar el conjunto de los términos que los identifican. Las particularidades del **MBT** sugerirán, no obstante, ciertas restricciones adicionales sobre este conjunto inicial que impidan aceptar ciertos términos como *válidos*.

Posteriormente, mediante la *semántica* describiremos la evolución de un proceso cuando es expuesto a un *test*. Al haber definido iocos mediante técnicas coinductivas en el capítulo 3, y por tanto quedar integrada nuestra relación dentro de la familia de semánticas *branching*, gran parte de este capítulo está inspirada en el artículo de Abramsky [2] en el que se demuestra que la equivalencia observacional[1] tiene su forma equivalente en el *testing*.

Las dos nociones, la de *conformidad* y la de *testing* quedarán conectadas a través de una *caracterización* de la relación iocos en función de un *preorden* inducido por el paso de *tests*. Puesto que, en principio, este *preorden* se aplica a *todos* los posibles *tests*, a fin de acotar el conjunto de candidatos se propondrá una algoritmo de generación de *tests* lo más ajustado posible al sistema referencia en cada caso. La salida de este algoritmo será representativa de todos los posibles *tests* aplicables, y en su momento procederemos a dar cuenta de su corrección y completitud.

[1] Alternativa para referirse a la bisimulación débil

4.1.1. Sintaxis

Bajo el término *test* nos referiremos en general a los experimentos que realizaremos para examinar el comportamiento de un sistema del que deseamos contrastar su *conformidad* con respecto a una especificación.

En este sentido, al describir la naturaleza de iocos como relación de *conformidad* en el capítulo 3 se hacía una distinción entre comportamientos *autónomos* y *reactivos*. Llevados al mundo del **MBT**, estos conceptos integraban los de **IUT** y *entorno* con el que intercambia acciones. Un *test* representa, *grosso modo*, una especie de entorno o ambiente artificial que es diseñado para la interacción del objeto.

En líneas generales, del resultado de la ejecución del *test* se obtiene el veredicto de *éxito* o el *fallo* del mismo. Para llegar al diagnóstico, el *test* puede *estimular* la implementación con una señal, examinar la salida de ésta, así como extender estas operaciones a un conjunto de señales posibles.

En este último sentido, consideraremos dos variantes del operador de *elección* o *alternativa* en el comportamiento de un *test*:

- El operador $+$, empleado en el sentido clásico que se propone en [82], esto es, estableciendo como requisito para el *éxito* que éste sea obtenido en los *tests* subsiguientes a *todas* las señales aplicadas o recibidas.
- El operador $\oplus$ tomado de [2], que, al modo de la disyunción lógica, sólo requiere obtener *éxito* en *alguno* de los *tests* introducido por una señal.

La aplicación o captación de señales a la **IUT**, la articulación de los diferentes *tests* que pueden seguir a éstas, así como los veredictos de *éxito* y *fallo* que asignamos a la ejecución de los mismos quedarán integrados formalmente mediante una sencilla sintaxis (definición 24) del conjunto que los comprende como elementos.

Definición 24. Consideremos como *test* un término sintáctico definido por la siguiente gramática en forma Backus-Naur:

$$T = \text{✗} \mid \checkmark \mid T_1 \oplus T_2 \mid T_1 + T_2 \mid aT \qquad \text{donde} \quad a \in L$$

Denotaremos el conjunto de tests por $\mathcal{T}$.

Al respecto, nótese que para un *test* la noción de acción de entrada (comportamiento de reacción) o de salida (comportamiento autónomo) es relativa al objeto sobre el que se aplica, por lo que en adelante ha de entenderse que $a?$ representa una acción de *salida* del *test* hacia el objeto de su experimentación; análogamente $x!$ representa una acción de *entrada* al *test* desde dicho objeto.

Tests válidos. Completado de términos

Al margen la sencillez con la que se ha descrito el conjunto $\mathcal{T}$, sucede que en el contexto de **MBT** los ambientes que queremos modelar mediante *tests* tienen unas particularidades inherentes que tenemos que considerar y no están recogidas en la definición 24.

En primer lugar, los *tests* deberían poder reaccionar *siempre* a *cualquier* salida de la implementación. En términos sintácticos, esto quiere decir que algunas expresiones de *test* como

$$a?T_{a?}, \qquad a? \in I$$

no deberían ser aceptadas como *tests* válidos. Más bien aceptamos que deberían ser *completados* con todas las posibles señales de salida que pudieran esperarse de la implementación, tal y como se recoge en el término

$$a?T_{a?} + \sum_{x \in O} xT_x \tag{4.1}$$

donde, bajo el supuesto de que la semántica de los *tests* hace que los operadores $+$ y $\oplus$ sean de naturaleza tanto asociativa como conmutativa, en un intento por facilitar la legibilidad usaremos

$$\sum_{i \in \{1,\ldots,n\}} T_i$$

como abreviatura de $T_1 + \cdots + T_n$

Análogamente, la expresión

$$o!T_{o!}$$

no es un *test* aceptable, sino que en rigor se debería considerar la expresión completada

$$o!T_{o!} + \sum_{x \in O,\ x \neq o!} xT_x \tag{4.2}$$

Nótese que tanto en (4.1) como en (4.2) $\delta! \in O$, por tanto no deja de estar presente el símbolo la *quiescencia*, el cual, tal como fue introducido en el capítulo 3, representaba la ausencia de señales por parte del objeto.

Cuestión aparte –la cual no ha lugar aquí desarrollar– es por qué medios pueda implementarse físicamente la detección de este tipo de eventos. La experiencia sugiere que el empleo de dispositivos *timeout*, capaces de activarse pasado un lapso de tiempo sin que sea emitida una señal, suponen una alternativa válida, en tanto que el intervalo elegido refleje un compromiso entre la eficacia y la fiabilidad: cuanto más grande sea el intervalo de muestreo, más fiable será su diagnóstico sobre ausencia de señal pero también más ineficaz el proceso de *tests* en el caso en que la *quiescencia* sea un comportamiento frecuente en el escenario.

Todas estas cuestiones relativas al *completado* de los *tests* –incluyendo el símbolo de quiescencia– necesario para cumplir las exigencias de un entorno artificial que ha de estar preparado para poder reaccionar a todas las salidas de la **IUT** quedan recogidas en la siguiente definición.

Definición 25. Sea $T \in \mathcal{T}$ un *test*, T es *válido* si y sólo si adopta una de las siguientes formas:

1. $T = \text{✗}$.
2. $T = \text{✓}$.
3. $T = a?T_{a?} + \sum_{x \in O} xT_x$ donde $x \in O$, $a? \in I$, y $T_{a?}$, T_x son *tests válidos*
4. $T = \sum_{x! \in O} x!T_{x!}$ donde $T_{x!}$ denota un test válido para $x! \in O$.
5. $T = T_1 \oplus T_2$ donde T_1 y T_2 son *tests* válidos.

Denotaremos por $\mathcal{T}_v$ al conjunto de *tests* válidos.

4.1.2. Semántica

Una vez establecida una sintaxis para los *tests*, procedemos a continuación a definir cómo estos *tests* interaccionan en base a un comportamiento que será el resultado de la ejecución de ese experimento.

A diferencia de [82], donde Tretmans emplea una definición *operacional*, esto es, en base a un sistema de transiciones entre términos expresados en forma de reglas, nosotros adoptaremos un enfoque *denotacional*, empleando funciones definidas entre dominios claramente establecidos, siguiendo las ideas de Abramsky en [2].

A tal efecto, especificaremos un predicado-función pass (definición 26) sobre pares de elementos; el primero de los elementos será un elemento del conjunto de sistemas de transiciones etiquetadas $\mathcal{LTS}$, categoría que introdujimos en el capítulo 3 para modelar los objetos que queremos inspeccionar, y el segundo un elemento perteneciente al conjunto de tests $\mathcal{T}$, que acabamos de definir en la sección anterior.

Definición 26. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$. Definimos el predicado

$$\mathsf{pass} \subseteq S \times \mathcal{T}$$

asumiendo $s \in S$, $a? \in I$, y $x! \in O$ del modo siguiente:

$$
\begin{aligned}
& s \ \mathsf{pass} \ \text{✗} = false \\
& s \ \mathsf{pass} \ \text{✓} = true \\
& s \ \mathsf{pass} \ x! T_{x!} = \begin{cases} true & \text{if } x! \notin \mathsf{outs}(s) \\ \bigwedge \{ s' \ \mathsf{pass} \ T_{x!} | s \xrightarrow{x!} s' \} & \text{otherwise} \end{cases} \\
& s \ \mathsf{pass} \ a? T_{a?} = \begin{cases} false & \text{if } a? \notin \mathsf{ins}(s) \\ \bigwedge \{ s' \ pass \ T_{a?} | s \xrightarrow{a?} s' \} & \text{otherwise} \end{cases} \\
& s \ \mathsf{pass} \ T_1 + T_2 = s \ \mathsf{pass} \ T_1 \wedge s \ \mathsf{pass} \ T_2 \\
& s \ \mathsf{pass} \ T_1 \oplus T_2 = s \ \mathsf{pass} \ T_1 \vee s \ \mathsf{pass} \ T_2
\end{aligned}
$$

El resultado de la evaluación sobre estos parámetros produce un valor booleano que ha de indicarnos, como resultado de la aplicación del *test* al sistema, si el resultado ha sido de éxito o fallo.

Nótese que por conveniencia, en la definición 26 el predicado pass está definido sobre el conjunto $\mathcal{T}$, tal como se expresa en la definición 24 al estar definido de manera estructural más simple, pese a que nosotros estaremos interesados sólo en los *tests* válidos $\mathcal{T}_v$.

Sentido intuitivo de la definición En términos generales, la definición suele estar de acuerdo con la intuición que damos a los símbolos ✗, ✓, $+, \oplus \ldots$

Mención aparte que pasamos a comentar merece el operador prefijo para el que, según se trate de un símbolo de entrada o de salida, existe distinta interpretación:

Entradas Se considera no superado el *test*, devolviendo *false* el cómputo de la función *pass* cuando el sistema objeto del *test* no reacciona a la señal propuesta por el test. En otro caso, el resultado del *test* se extiende a *todos* los posibles alcanzables por el sistema.

Salidas En contra de la intuición, si un objeto no emite una señal conforme a la que el *test* está preparado, se considerará el *test* pasado. De nuevo, en caso afirmativo, el cómputo queda relegado a los *tests* realizados por los estados alcanzados por el sistema.

Cabe destacar que este caso es más complejo que el anterior, pues de modo trivial se tiende a pensar que, conforme a lo que se ha descrito anteriormente, un sistema sin ejercer acción autónoma alguna pasaría cualquier forma propuesta de *test*.

Sin embargo, una visión más detenida nos lleva la conclusión de que, de acuerdo a la definición de *test* válido, el *test* ha de estar preparado para *cualquier* símbolo de salida, lo que incluye el símbolo $\delta!$ que denota la *quiescencia*. Como todos ellos y sus subsiguientes *tests* aparecen integrados por el operador + que impone una conjunción de resultados para su satisfacción, concluimos que no basta la simple pasividad de un sistema para pasar un *test* propuesto.

La raíz de este modo diferente de gestionar las señales de entrada y salida radica en la definición 22 del capítulo 3 la cual marca una asimetría a la hora de tratar con los dos tipos de símbolo: es necesario reaccionar a las entradas prescritas, pero contingente la ejecución autónoma de salidas. En uno y otro caso el resultado se extiende a los distintos *tests* que, expresados en el término, marcan la evolución la evolución del experimento después de la ejecución de esas acciones.

Para poder mejor comprender el alcance de la definición daremos unos ejemplos que muestren el proceso de su cómputo sobre valores concretos. Más adelante, como veremos en el apartado 4.1.3, esta definición constituirá la base

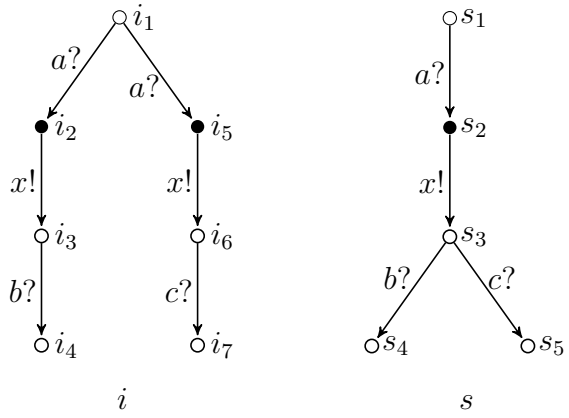


Figura 4.1: i ioco s y i ioco$\not$s s

para la formulación de un *preorden* de estratégica importancia para conectar las nociones de *testing* y *conformidad*.

La figura 4.1 nos presenta dos sistemas i y s, en apariencia indistinguibles para la relación ioco, que vamos a someter a la acción del test

$$T^v = a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) + x!\text{✗}$$

version *válida* y *completada* conforme a la definición 25 del *test* más simple:

$$T = a?x!b?\checkmark$$

De modo informal, un *test* tan sencillo sugiere que se debe pasar siempre que el objeto al que sea aplicado reciba una señal $a?$, emita una salida $x!$ y reciba una señal $b?$.

Se puede observar en el caso del sistema s esto ocurre en todas las ocasiones que esto se plantee, pues después de ejecutar la *traza* $a?x!$, el sistema está presto en s_2 a recibir entradas $b?$ o $c?$. El cómputo de la función pass aplicado al par

(s, T^v) da cuenta de ello, subrayando a cada paso la fórmula reducida.

$$\begin{aligned}
s_1 \text{ pass } T^v &= (s_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \wedge \underline{s_1 \text{ pass } x!\text{✗}} \\
&= (s_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \wedge true \\
&= (s_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \\
&= s_2 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge \underline{s_2 \text{ pass } \delta!\text{✗}} \\
&= s_2 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge true \\
&= s_2 \text{ pass } \underline{x!(b?\checkmark + x!\text{✗} + \delta!\checkmark)} \\
&= s_3 \text{ pass } \underline{b?\checkmark + x!\text{✗} + \delta!\checkmark} \\
&= s_3 \text{ pass } b?\checkmark \wedge \underline{s_3 \text{ pass } x!\text{✗}} \wedge \underline{s_3 \text{ pass } \delta!\checkmark} \\
&= s_3 \text{ pass } \underline{b?\checkmark} \wedge true \wedge true \\
&= s_4 \text{ pass } \checkmark
\end{aligned}$$

Por el contrario, esto *no siempre* tiene lugar en el sistema i, pues si en el transcurso del experimento el sistema y de modo *indeterminado* opta por la rama derecha, no tendrá entre sus opciones disponibles reaccionar a una señal $b?$. El cómputo de la función pass devuelve *false*.

$$\begin{aligned}
i_1 \text{ pass } T^v &= i_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \wedge \underline{i_1 \text{ pass } x!\text{✗}} \\
&= i_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \wedge true \\
&= i_1 \text{ pass } a?(x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗}) \\
&= i_2 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗} \wedge \\
&\quad i_5 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) + \delta!\text{✗} \\
&= i_2 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge \underline{i_2 \text{ pass } \delta!\text{✗}} \wedge \\
&\quad i_5 \text{ pass } x!(b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge \\
&\quad \underline{i_5 \text{ pass } \delta!\text{✗}} \\
&= i_2 \text{ pass } x!b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge true \wedge \\
&\quad i_5 \text{ pass } x!b?\checkmark + x!\text{✗} + \delta!\checkmark) \wedge true \\
&= \underline{i_2 \text{ pass } x!b?\checkmark} \wedge \underline{i_2 \text{ pass } x!\text{✗}} \wedge \underline{i_2 \text{ pass } \delta!\checkmark} \wedge \underline{i_5 \text{ pass } x!b?\checkmark} \wedge \underline{i_5 \text{ pass } x!\text{✗}} \wedge \\
&\quad \underline{i_5 \text{ pass } \delta!\checkmark} \\
&= \underline{i_2 \text{ pass } x!b?\checkmark} \wedge true \wedge true \wedge \underline{i_5 \text{ pass } x!b?\checkmark} \wedge true \wedge true \\
&= \underline{i_2 \text{ pass } x!b?\checkmark} \wedge \underline{i_5 \text{ pass } x!b?\checkmark} \\
&= \underline{i_3 \text{ pass } b?\checkmark} \wedge \underline{i_6 \text{ pass } b?\checkmark} \\
&= \underline{i_3 \text{ pass } \checkmark} \wedge false \\
&= true \wedge false \\
&= false
\end{aligned}$$

4.1.3. Caracterización

En este apartado vamos a demostrar que la relación iocos, definida coinductiva, puede ser caracterizada mediante una relación de *testing*. Esto resulta de interés porque entronca con la principal hipótesis del **MBT** (véase capítulo 2) que asume que al someter un sistema a análisis *se desconoce* el modelo que lo describe, pese a que se de por hecho su existencia. La aplicación directa de la definición de iocos, u otras técnicas basadas en previa reducción de los modelos a examinar, como veremos en el capítulo 5, sólo se puede aplicar cuando contamos con el modelo tanto del sistema a comparar como el que sirve de referencia.

Para ello, lo primero que vamos a hacer es definir un *preorden* basado en *testing* sobre nociones de la sección anterior, a saber, la función de paso de *test* pass (definición 26), que relaciona objetos observados y *tests* en la ejecución de un experimento. De esta manera, declaramos que un comportamiento presenta un comportamiento *mejor* que otro si el primero pasa más *tests* que el último.

Definición 27. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ y $p, q \in S$. Entonces, definimos el preorden

$$p \sqsubseteq_T q \text{ iff } \forall T \in \mathcal{T}_v : \ p \,\mathsf{pass}\, T \implies q \,\mathsf{pass}\, T$$

El objetivo último, que recogerá el teorema 5, es demostrar que el preorden así definido resulta ser precisamente la inversa de la relación iocos

$$\mathsf{iocos} = {\sqsubseteq_T}^{-1}$$

o lo que es lo mismo,

$$p \,\mathsf{iocos}\, q = q \sqsubseteq_T p$$

Intuitivamente se quiere dar a entender que la relación de *conformidad* de iocos entre un sistema cualquiera y una especificación de referencia es cierta cuando *todos* los *tests* pasados por una especificación son pasados por el sistema en consideración. Para una mejor comprensión, la demostración del teorema 5 se ha desglosado en dos partes bien diferenciadas: la proposición 1 para demostrar ${\sqsubseteq_T}^{-1} \subseteq \mathsf{iocos}$ y la proposición 2, que hace lo propio con $\mathsf{iocos} \subseteq {\sqsubseteq_T}^{-1}$.

Mediante la primera, demostramos que si una implementación no está en relación de *conformidad* iocos con una especificación, entonces, necesariamente hemos de encontrar un *test* pasado por la especificación y fallado por la

implementación. Equivalentemente, cuando no es posible encontrar tal *test*, podemos afirmar que ambos están en relación de *conformidad* iocos.

Proposición 1. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p, q \in S$. *Si* $q \sqsubseteq_T p$ *entonces* $p \,\mathsf{iocos}\, q$.

Demostración. Al objeto de probar $p \,\mathsf{iocos}\, q$ debemos encontrar una relación iocos R tal que $(p, q) \in R$. Definamos

$$R = \{(p_1, p_2) \mid p_1, p_2 \in S,\ p_2 \sqsubseteq_T p_1\}$$

Ya que $q \sqsubseteq_T p$, tenemos que $(p, q) \in R$, según está construido R.

De entrada, afirmamos que R es una relación iocos y procedemos por contradicción: suponemos que existe $(p_1, p_2) \in R$ que no satisface alguna de las cláusulas de la definición 22 de una iocos-relación; entonces encontraremos un test $T \in \mathcal{T}_v$ tal que $p_2 \,\mathsf{pass}\, T$ y $p_1 \not\mathsf{pass}\, T$, por lo que $p_2 \not\sqsubseteq_T p_1$ y por tanto $(p_1, p_2) \notin R$, en contra de lo que partíamos.

Distinguiremos los casos de acuerdo con las tres cláusulas obligatorias de definición 22 según las cuales el par (p_1, p_2) deja estar en relación iocos.

(p_1, p_2) *no cumple la cláusula 22.1* . Debería haber un $a? \in \mathsf{ins}(p_2)$ tal que $a? \notin \mathsf{ins}(p_1)$. Consideremos el *test* válido:

$$T = a?\checkmark + \sum_{x! \in O} x!\checkmark$$

Resulta trivial conforme a la definición 26 que $p_2 \,\mathsf{pass}\, T$ pero $p_1 \not\mathsf{pass}\, T$: entradas prescritas no satisfechas hacen *pass* devolver *false*.

(p_1, p_2) *no cumple la cláusula 22.2* Podemos asumir que cláusula 22.1 se cumple, luego existe $a? \in \mathsf{ins}(p_1)$ y $p_1' \in S$ tal que $p_1 \xrightarrow{a?} p_1'$ y $(p_1', p_2') \notin R$ para cualquier p_2' tal que $p_2 \xrightarrow{a?} p_2'$. Consideremos el conjunto

$$P = \{p_2' \mid p_2 \xrightarrow{a?} p_2',\ (p_1', p_2') \notin R\}$$

Puesto que la cláusula 22.1 de la definición se cumple, $P \neq \varnothing$. Por definición de R, para cualquier $p_2' \in P$ existe un *test* $T_{p_2'}$ tal que $p_2' \,\mathsf{pass}\, T$ y $p_1' \not\mathsf{pass}\, T_{p_2'}$. Entonces consideremos el test

$$T = a? \bigoplus_{p_2' \in P} T_{p_2'} + \sum_{x! \in O} x!\checkmark$$

para el que se cumple $p_2 \,\mathsf{pass}\, T$ pero $p_1 \not\mathsf{pass}\, T$.

En efecto, reparemos en el segundo sumando: si p_2 ofrece alguna señal $x! \in O$, entonces pasará el *test* subsiguiente ✓. Si no la ofrece, también acaba en ✓, por ser las señales de salida contingentes en su ejecución. Esto aplica también a p_1.

Como $a? \in \mathsf{ins}(p_2)$, la función de *test* se remonta a la evaluación de $\bigoplus_{p_2' \in P} T_{p_2'}$ para cualquier p_2' tal que $p_2 \xrightarrow{a} p_2'$. Recordemos que todos estos p_2' están considerados en el conjunto P y cada uno al menos pasa uno de los *tests* de la familia $\{T_{p_2'} | p_2' \in P\}$, pues con este fin se han adoptado. La semántica *disyuntiva* de $\oplus$ hace que la evaluación sobre cualquier p_2' devuelva *true* cuando algún *test* ha sido superado. Condición distinta obra para la evaluación sobre cualquier p_1' pues los $\{T_{p_2'} | p_2' \in P\}$ fueron escogidos para no ser superados por p_1', cualquiera que éste fuera.

(p_1, p_2) *no cumple la cláusula 22.3.* Existe $x! \in \mathsf{outs}(p_1)$ tal que $p_1 \xrightarrow{x!} p_1'$ pero $(p_1', p_2') \notin R$ para cualquier p_2 tal que $p_2 \xrightarrow{x!} p_2'$.

Consideremos el conjunto

$$P = \{p_2' \mid p_2 \xrightarrow{x!} p_2',\ (p_1', p_2') \notin R\}$$

En esta ocasión pueden suceder dos casos:

- Si $P = \varnothing$, consideremos el test

$$T = x!✗ + \sum_{x! \neq y!, y! \in O} y!✓$$

  Entonces $p_2 \,\mathsf{pass}\, T$ y $p_1 \not\mathsf{pass}\, T$. Esto es así por la contingencia de las salidas, ya que si $p_2 \xrightarrow{x}$ la función pass devuelve *true*. Sin embargo, como $p_1 \xrightarrow{x} p_1'$, el *test* queda relegado a la evaluación de ✗ que trivialmente da *false*.

- Supongamos que $P \neq \varnothing$. Entonces, como en el caso anterior, para cualquier $p_2' \in P$ existe $T_{p_2'}$ tal que $p_2' \,\mathsf{pass}\, T$ y $p_1' \not\mathsf{pass}\, T_{p_2'}$. Consideremos el test

$$T = x! \bigoplus_{p_2' \in P} T_{p_2'} + \sum_{x! \neq y!, y! \in O} y!✓$$

Entonces p_2 pass T y p_1 pa$\not{\text{ss}}$ T. El razonamiento es el mismo que se ha explicado anteriormente para el símbolo de entrada $a?$ en el anterior caso.

□

La siguiente proposición, recíproca de la segunda, afirma que si una implementación está en relación mediante iocos con una especificación, entonces *cualquier test* que pase la especificación es superado por la implementación. Para poder desarrollarla, necesitamos dar una definición auxiliar que servirá para describir el objeto sobre el que se practicará la inducción.

Definición 28. Sea $T \in \mathcal{T}_v$. Definimos por inducción la función profundidad de un *test* válido

$$prof : \mathcal{T}_v \to N$$

- El caso base comprende dos posibilidades
  - $prof(\checkmark) = 0$
  - $prof(\times) = 0$
- Para el caso recursivo tenemos:
  - $prof(T_1 \oplus T_2) = 1 + prof(T_1) + prof(T_2)$
  - $prof(\sum_{x \in O} xT_x) = 1 + max(prof(T_x))$
  - $prof(\sum_{x \in O} xT_x + a?T_{a?}) = 1 + max(prof(T_x), prof(T_{a?}))$

Proposición 2. *Sea* $LTS = (S, I, O, \to) \in \mathcal{LTS}$ *y* $p, q \in S$. *Si* p iocos q *entonces* $q \sqsubseteq_T p$.

Demostración. Consideremos el *test* $T \in \mathcal{T}_v$ tal que q pass T. Tenemos que probar que p pass T. Lo haremos por inducción estructural sobre la profundidad del *test*

Casos base $n = 0$

$T = \checkmark$ El *test* ✓ es pasado por cualquier $p \in S$.

$T = \boldsymbol{\times}$ En este caso tenemos $q \not\mathsf{pass}\, T$ lo que entra en contradicción con el supuesto de partida, $q \,\mathsf{pass}\, T$.

Casos recursivos Con una profundidad $n > 0$ tomamos en cuenta la hipótesis de inducción formulada sobre *tests* de altura $n' < n$.

$T = T_1 \oplus T_2$ Por definición de $q \,\mathsf{pass}\, T$ tenemos que o bien $q \,\mathsf{pass}\, T_1$ o $q \,\mathsf{pass}\, T_2$.

Asumamos $q \,\mathsf{pass}\, T_1$, ya que el otro caso es simétrico. Por hipótesis de inducción, $p \,\mathsf{pass}\, T_1$, luego $p \,\mathsf{pass}\, T_1 \oplus T_2$.

$\sum_{x!\in O} x!T_{x!}$ Consideremos el conjunto

$$O' = \{x! \mid x! \in O, \exists p' : p \xrightarrow{x!} p'\}$$

Se da la circunstancia de que $O' \neq \varnothing$, porque, en ausencia una acción de salida, presuponemos la acción $\delta!$. Para probar que p pasa dicho *test* tenemos que probar que $p_{x!} \,\mathsf{pass}\, T_{x!}!$ para cualquier $p_{x!} \in S$ tal que $p \xrightarrow{x!} p_{x!}$.

Consideremos cualquiera de estos $p_{x!}$. Por la cláusula 22.3, existe $q_{x!}$ tal que $q \xrightarrow{x!} q_{x!}$ y $p_{x!}\, \mathsf{iocos}\underline{\leq}\, q_{x!}$. Dado que $q \,\mathsf{pass}\, T$, $q_{x!} \,\mathsf{pass}\, T_{x!}!$, según está definida pass. Entonces por hipótesis de inducción, $p_{x!} \,\mathsf{pass}\, T_{x!}!$.

Puesto que esto es cierto para cualquier $x! \in O'$ y $O' \neq \varnothing$, $p \,\mathsf{pass}\, T$.

$T = \sum_{x!\in O} x!T_{x!} + a?T_{a?}$**.** En primer lugar $a? \in \mathsf{ins}(q)$ ya que $q \,\mathsf{pass}\, T$. Por la cláusula 22.1, $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$, luego $a? \in \mathsf{ins}(p)$. Evaluemos la satisfactibilidad de los dos sumandos:

- Respecto al segundo sumando $a?T_{a?}$, consideremos $p_{a?} \in S$ tal que $p \xrightarrow{a?} p_{a?}$. Por cláusula 22.2, existe $q_{a?} \in S$ tal que $q \xrightarrow{a?} q_{a?}$ con $p_{a?}\, \mathsf{iocos}\underline{\leq}\, q_{a?}$. Puesto que $q \,\mathsf{pass}\, T$, $q_{a?} \,\mathsf{pass}\, T_{a?}$, según está definida pass y entonces, por inducción $p_a \,\mathsf{pass}\, T_{a?}$. Añadiendo que $a? \in \mathsf{ins}(p)$ se satisface que $p \,\mathsf{pass}\, a?T_{a?}$.
- Para el primero, consideramos el conjunto

  $$O' = \{x! \mid x! \in O, \exists p' : p \xrightarrow{x!} p'\}$$

  De modo similar, razonando como en el segundo caso recursivo obtenemos que $p_{x!} \,\mathsf{pass}\, T_{x!}$ para cualquier $p_{x!} \in S$ y $x! \in O'$ tal que $p \xrightarrow{x!} p_{x!}$.

En conclusión, satisfecho el primer y el segundo sumando de la expresión, tenemos que $p \,\mathsf{pass}\, T$.

□

Como colofón a este apartado, obtenemos la siguiente *caracterización* de iocos como un preorden sobre un conjunto de tests.

Teorema 5. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p, q \in S$

$$p \,\mathsf{iocos}\, q \text{ si y sólo si } q \sqsubseteq_T p.$$

Demostración. A partir de los resultados anteriores. Por la proposición 1 tenemos que $p \,\mathsf{iocos}\, q$ si $q \sqsubseteq_T p$ y por la proposición 2 el resultado recíproco, es decir, $q \sqsubseteq_T p$ si $p \,\mathsf{iocos}\, q$. □

4.1.4. Generación de tests

El teorema 5 supone un resultado clásico de caracterización mediante *testing* que abre las puertas a implementaciones capaces de detectar si un sistema está en relación de *conformidad* con respecto a otro que se toma por referencia, siempre teniendo las condiciones particulares de la relación iocos.

Una vez legitimado el procedimiento del *testing* como instrumento, una primera aproximación ingenua al problema de determinar si dos sistemas están ligados por el preorden declarado nos llevaría a intentar experimentar con *todos* los posibles *tests* válidos, ya que desconocemos *a priori* qué *tests* sean los que el sistema de referencia pase con éxito. El inconveniente, claro está, es que este conjunto en general es infinito.

De lo que trata este apartado es precisamente de acotar este problema produciendo un conjunto de *tests* lo más representativo de las características que se exponen mediante la especificación. La ventaja de haber formulado la relación de *conformidad* en un contexto **MBT** es que gracias a la formalización de los conceptos, (relación iocos, conjunto $\mathcal{T}$, función $\mathsf{pass}\ldots$) esta tarea puede llevarse a cabo de modo *sistemático*, lo que permite eventualmente su implementación por medios de procesamiento automáticos, ganando en rapidez y fiabilidad frente a otras técnicas de curso manual.

Esta idea ya estaba presente en aquellas alternativas precursoras de nuestra relación iocos. En concreto, podemos hallar en [82] una propuesta para relación ioco de la cual presentamos una variación adaptada a las condiciones propias de una semántica *branching*, superando las limitaciones de la original en lo relativas a la presencia del indeterminismo. En concreto, nuestro algoritmo toma la forma que se describe en la definición 29, la cual es una variación del algoritmo mostrado en [82].

La mayor diferencia, ideada expresamente para salvar esta dificultad, estriba en la inclusión del operador $\oplus$ en los *tests*. Hacemos notar que si tenemos una especificación determinista entonces el operador $\oplus$ se aplica al conjunto unitario (*singleton*) y el resultado es el esperado en la versión original. No obstante, la sola introducción de este operador introducirá alguna complicación no trivial a la hora de demostrar propiedades, como tendremos oportunidad de ver más adelante.

Definición 29. Sea $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ y $p \in S$. Denotaremos por $\mathcal{T}(p)$ al conjunto de *test* válidos de p que resultan de aplicar un número finito de llamadas recursivas a una de los siguientes opciones no-deterministas:

1. $T = \checkmark \in \mathcal{T}(p)$.

2. Si $a? \in \mathsf{ins}(p)$, entonces $T \in \mathcal{T}(p)$ donde

$$T = a? \bigoplus \{T_{p_{a?}} \mid p \xrightarrow{a?} p_{a?}\} + \sum_{\substack{x! \in \mathsf{outs}(p) \\ x! \neq \delta!}} x! \bigoplus \{T_{p_{x!}} \mid p \xrightarrow{x!} p_{x!}\} + \sum_{\substack{x! \in O, x! \neq \delta! \\ x! \notin \mathsf{outs}(p)}} x!\boldsymbol{\times} \; + \; \delta! T_{\delta!}(p)$$

3. Si $\mathsf{ins}(p) = \varnothing$ entonces $T \in \mathcal{T}(p)$ donde

$$T = \sum_{\substack{x! \in \mathsf{outs}(p) \\ x! \neq \delta!}} x! \bigoplus \{T_{p_{x!}} \mid p \xrightarrow{x!} p_{x!}\} + \sum_{\substack{x! \in O, x! \neq \delta! \\ x! \notin \mathsf{outs}(p)}} x!\boldsymbol{\times} + \delta! T_{\delta!}(p)$$

En todos los casos los *tests* T_p se escogen de entre el conjunto $\mathcal{T}(p)$, de modo indeterminista, con la excepción para el símbolo $\delta!$ de quiescencia, que adopta estas formas:

- $T_{\delta!}(p) = \checkmark$ si $p \xrightarrow{\delta!}$,
- $T_{\delta!}(p) = \text{✗}$ en otro caso.

4.1.5. Corrección y exhaustividad

El objetivo esencial del algoritmo es producir una *batería* de *tests* lo más *ajustada* que sea posible para la especificación dada, lo que no excluye que de nuevo el resultado sea un conjunto infinito. Entendemos por ajustada esa característica del conjunto $\mathcal{T}(p)$ que nos permite, con la sola valoración de los *tests* de este conjunto y ni uno más, determinar si cualquier otro sistema q está en relación q $\mathsf{iocos}\underline{\leq}$ p

Gracias a la caracterización que se hizo en el apartado 4.1.3 de la relación $\mathsf{iocos}\underline{\leq}$ (teorema 5), podemos tratar de resolver el problema en base al conjunto $\mathcal{T}(p)$ generado por el algoritmo. Éste el objetivo con el que se enuncian y demuestran las siguientes proposiciones que darán soporte al teorema 8 relacionando el conjunto $\mathcal{T}(p)$ con el preorden $\sqsubseteq_T$.

Abordaremos el objetivo en dos fases diferenciadas: en una primera demostramos que el conjunto de todos los *tests* del conjunto $\mathcal{T}(p)$ resultan correctos respecto a p, o lo que es lo mismo: p pasa todos los *tests* del conjunto $\mathcal{T}(p)$.

Proposición 3. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p \in S$.

$$\forall T \in \mathcal{T}(p) \Rightarrow p \,\mathsf{pass}\, T$$

Demostración. Por inducción estructural sobre la profundidad de los términos del conjunto $\mathcal{T}(p)$.

Casos base, n=0 Si $T = \checkmark$, correspondiente al paso 1 del algoritmo de la definición 29 se cumple trivialmente, ya que p pass $\checkmark$.

Casos recursivos, n>0 Asumiendo la hipótesis de inducción para *tests* válidos de profundidad $n' < n$, tenemos dos casos que valorar.

- El primero, correspondiente al paso 2, adopta el término

$$T = \overbrace{a? \bigoplus T_{p_{a?}}}^{4} + \overbrace{\sum_{\substack{x! \in \mathsf{outs}(p) \\ x! \neq \delta!}} x! \bigoplus T_{p_{x!}}}^{3} + \overbrace{\sum_{\substack{x! \in O, x! \neq \delta! \\ x! \notin \mathsf{outs}(p)}} x!\text{✗}}^{2} + \overbrace{\delta! T_{\delta!}(p)}^{1}$$

Para que p pass T es necesario y suficiente que p pass T' siendo T' cada uno de los escenarios marcados.

1. En efecto,
$$p \text{ pass } \delta!T_{\delta!}(p)$$
si recordamos el modo en que está definido el término $T_{\delta!}(p)$ en definición 29. Si $p \xrightarrow{\delta!}$, entonces $T_{\delta!}(p) = \checkmark$, por lo que se satisface trivialmente. Si $p \not\xrightarrow{\delta!}$, es claro que también pasa, por la forma en que $\delta!$, como un 'simbolo de salida más cuando no está presente, es evaluada por la función pass devolviendo *true*.
2. Si $x \notin \text{outs}(p)$ entonces $p \not\xrightarrow{x!}$, de modo que
$$p \text{ pass } \sum_{\substack{x! \in O \\ x! \notin \text{outs}(p)}} x!\text{✗}$$
de nuevo, por la forma con la que son evaluados los símbolos de salida contingentes.
3. Cuando $x! \in \text{outs}(P)$, $T_{p_{x!}} \in \mathcal{T}(p_{x!})$ para cualquier $p_{x!}$ tal que $p \xrightarrow{x!} p_{x!}$.
Por la hipótesis de inducción $p_{x!}$ pass $T_{p_{x!}}$, luego
$$p \text{ pass } \sum_{x! \in \text{outs}(p)} x! \bigoplus T_{p_{x!}}$$
según está definida la función pass para el operador prefijo y la naturaleza disyuntiva del operador $\oplus$.
4. De modo similar, por la definición del paso 2, $a? \in \text{ins}(p)$ y $T_{p_{a?}} \in \mathcal{T}(p_{a?})$ para cualquier $p_{a?}$ tal que $p \xrightarrow{a?} p_{a?}$.
Por la hipótesis de inducción $p_{a?}$ pass $T_{p_{a?}}$ para cualquier $p_{a?}$, luego
$$p \text{ pass } a? \bigoplus T_{p_{a?}}$$
según está definida la función pass para el operador prefijo y la disyunción del operador $\oplus$, que exige que al menos algún *test* sea pasado por cada $p_{a?}$.

- Para el caso correspondiente al paso 3

$$T = \overbrace{\sum_{\substack{x!\in \mathsf{outs}(p)\\ x!\neq\delta!}} x! \bigoplus T_{p_{x!}}}^{3} + \overbrace{\sum_{\substack{x!\in O, x!\neq\delta!\\ x!\notin \mathsf{outs}(p)}} x!\text{✗}}^{2} + \overbrace{\delta! T_{\delta!}(p)}^{1}$$

Este caso es similar al previo. Sólo tenemos que tener en cuenta que $\mathsf{outs}(p) \neq \varnothing$, siendo igual a los pasos 1, 2, 3 anteriores.

□

La siguiente proposición revela que el conjunto de *tests* generados es suficiente para poder inferir la relación de preorden $\sqsubseteq_T$. A este último resultado se le conoce como *exhaustividad* del conjunto $\mathcal{T}(p)$.

Proposición 4. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p, q \in S$.

$$\forall T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T \Rightarrow p \sqsubseteq_T q$$

Demostración. Probaremos a demostrarlo por contradicción: asumiendo que $p \not\sqsubseteq_T q$, entonces existirá un *test* $T \in \mathcal{T}_v$ tal que $p \,\mathsf{pass}\, T$ y $q \,\not\mathsf{pass}\, T$. En estas circunstancias, debemos describir un *test* $T_p \in \mathcal{T}(p)$ tal que $q \,\not\mathsf{pass}\, T_p$.

De nuevo, procederemos por inducción sobre la profundidad de $T \in \mathcal{T}_v$

Casos base Para $n = 0$

$T =$ ✓ En ese caso, tenemos que $q \,\mathsf{pass}$ ✓, lo que va en contra de lo indicado por la hipótesis.

$T =$ ✗ Ocurre lo mismo, ya que $p \,\not\mathsf{pass}$ ✗, lo cual invalida la hipótesis de partida.

Casos recursivos Para $n > 0$, asumiendo la hipótesis de inducción para *tests*de profundidad $n' < n$

$T = T_1 \oplus T_2$**.** Entonces o bien p pass T_1 o p pass T_2. En ambos casos el resultado se obtiene aplicando la hipótesis de inducción, obteniendo $T_p \in \mathcal{T}(p)$ cumpliendo $p \,\mathsf{pass}\, T_p$ y $q \,\not\mathsf{pass}\, T_p$.

$T = \sum_{x \in O} xT_x$. Supongamos en primer lugar que $p \xrightarrow{\delta!} p$. Por la condición de quiescencia $p \xrightarrow{x!} \not\rightarrow$ para $x! \in O$ y $x! \neq \delta!$ y por la definición de paso de *tests* tenemos que $p \,\mathsf{pass}\, T_{\delta!}$.

Si $q \xrightarrow{\delta!} q$, puesto que $q \,\mathsf{pa\not{s}s}\, T$ es porque $q \,\mathsf{pa\not{s}s}\, T_{\delta!}$ y tendremos el *test* T_p por hipótesis de inducción.

Si $q \xrightarrow{\delta!} \not\rightarrow$ entonces el test

$$T_p = \delta!\checkmark + \sum_{\substack{x! \in \mathsf{outs}(p) \\ x \neq \delta!}} x!\text{✗}$$

verifica las propiedades requeridas: $T_p \in \mathcal{T}(p)$ y $q \,\mathsf{pa\not{s}s}\, T_p$.

Supongamos ahora que $p \xrightarrow{\delta!} \not\rightarrow$, para cualquier $x! \in O$ consideremos un *test* $T'_{x!}$ en base al valor de $x!$

$x! = \delta!$ entonces $T'_{x!} = \text{✗}$

$x! \in \mathsf{outs}(q) \cap \mathsf{outs}(p)$ Dependiendo de $x!$ hay dos casos:

- Si para cualquier $q_{x!}$ tal que $q \xrightarrow{x!} q_{x!}$ tenemos que $q_{x!} \mathsf{pass} T_{x!}$ entonces consideramos $T'_{x!} = \checkmark$.
- En caso contrario existe $q_{x!}$ tal que $q_{x!} \,\mathsf{pa\not{s}s}\, T_{x!}$. Podemos aplicar la hipótesis a cada p' tal que $p \xrightarrow{x!} p'$ para encontrar un test $T_{p'} \in \mathcal{T}(p')$ y $q_{x!} \,\mathsf{pa\not{s}s}\, T_{p'}$. Consideramos entonces el test

  $$T'_{x!} = \bigoplus \{T_{p'} \mid p \xrightarrow{x} p'\}$$

  En este caso, debido a la definición de paso de test tenemos que $q_{x!} \,\mathsf{pa\not{s}s}\, T_{x!}$.

$x \in \mathsf{outs}(q), x \notin \mathsf{outs}(p)$. Tomemos $T'_x = \text{✗}$

$x \notin \mathsf{outs}(q), x \in \mathsf{outs}(p)$. En este caso $T'_x = \checkmark$.

Con todo esto tomamos el *test*

$$T_p = \sum_{x! \in O} xT'_{x!}$$

Así definido T_p adopta una expresión como la de paso 3 del algoritmo generador de la definición 29, por lo tanto $T_p \in \mathcal{T}(p)$.

Veamos ahora que $q \not\mathsf{pass} T_p$. Si $q \xrightarrow{\delta!} q$ entonces $q \not\mathsf{pass} T_p$ puesto que $T_{\delta!} = \boldsymbol{\times}$. Si $q \not\xrightarrow{\delta!}$ y $q \not\mathsf{pass} T$ es porque existe $x! \in \mathsf{outs}(q)$ y $x! \neq \delta!$ tal que $q \xrightarrow{x!} q_{x!}$ y $q_{x!} \not\mathsf{pass} T_{x!}$. Tenemos dos casos

- $x! \notin \mathsf{outs}(p)$. Entonces $T'_{x!} = \boldsymbol{\times}$, por lo que $q_{x!} \not\mathsf{pass} T_{x!}$ y por tanto $q \not\mathsf{pass} T_p$.
- $x! \in \mathsf{outs}(p)$. Entonces –como hemos visto antes en el caso $x! \in \mathsf{outs}(p) \cap \mathsf{outs}(q)$ y $q_{x!} \not\mathsf{pass} T_{x!}$– $q_{x!} \not\mathsf{pass} T_{x!}$ y por tanto $q \not\mathsf{pass} T_p$.

$T = a?T_{a?} + \sum_{x!\in O} x!T_{x!}$. Puesto que $p \,\mathsf{pass}\, T$, $a? \in \mathsf{ins}(p)$. Vamos a considerar el *test* $T'_{a?}$ como sigue:

$a? \in \mathsf{ins}(q)$. $T'_{a?}$ se construye en un modo similar al del caso previo cuando $x! \in \mathsf{outs}(q) \cap \mathsf{outs}(p)$.

$a? \notin \mathsf{ins}(q)$. $T'_{a?} = \checkmark$.

Consideremos $T_1 = \sum_{x!\in O} x!T_{x!}$. Debido a la definición de paso de test tenemos que $p \,\mathsf{pass}\, T_1$. Si $p \,\mathsf{pass}\, T_1$ consideremos el test

$$T'_1 = \sum_{x!\in O} x!\checkmark$$

En caso contrario el test T'_1 es el *test* T_p que se construye como en el apartado anterior.Entonces el test

$$T_p = a?T'_{a?} + T_1$$

cumple que $T_p \in \mathcal{T}(p)$ y $q \not\mathsf{pass} T_p$.

□

A partir de las proposiciones 3 y 4 podemos enunciar el siguiente resultado de *completitud*[2] relacionando el conjunto $\mathcal{T}(p)$ con el preorden $\sqsubseteq_T$.

Teorema 6. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p, q \in S$

$$\forall T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T \Leftrightarrow p \sqsubseteq_T q$$

[2]El significado que adquiere la palabra *completitud* aquí empleada no coincide con la que se le otorga al término en en otros contextos afines, como la Lógica, donde la completitud se identifica con la segunda de las propiedades, la exhaustividad.

Demostración. Hagamos la demostración en los dos sentidos:

- Por la proposición 4, tenemos directamente que $\forall T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T \Rightarrow p \sqsubseteq_T q$.
- Para demostrar $p \sqsubseteq_T q \Rightarrow \forall T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T$, supongamos que existe $T \in \mathcal{T}(p)$ tal que $q \not\mathsf{pass}\, T$. Por la proposición 3 sabemos que $p \,\mathsf{pass}\, T$, luego $p \not\sqsubseteq_T q$.

□

Por último, basándose en la carcaterización de la relación iocos con el preorden $\sqsubseteq_T$ obtenemos el siguiente corolario.

Corolario 1. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ *y* $p, q \in S$

$$\forall T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T \Leftrightarrow p \,\mathsf{iocos}\, q$$

Demostración. Por la caracterización de la relación iocos demostrada en el teorema 5, vinculando la relación iocos con el preorden $\sqsubseteq_T$ y el teorema 6, que relaciona éste con el conjunto $\mathcal{T}(p)$. □

El corolario marca una propiedad deseable en todo sistema automático de generación de *tests*, pues nos da la garantía de obtener una **IUT** conforme a la especificación únicamente en el caso en que ésta pase *todos los tests* que integran el conjunto $\mathcal{T}(p)$.

En la realidad, cuando el conjunto $\mathcal{T}(p)$ es infinito se sacrifica esta propiedad seleccionando un subconjunto finito y por tanto reduciéndola al objetivo más modesto de saber que una **IUT** no resulta conforme a la especificación al dar con *algún test* no superado por aquella. Nada se puede asegurar cuando se superan todos los *tests* de dicho subconjunto. La práctica industrial no basada en métodos formales llega incluso a sacrificar éste último aspecto, con los llamados *falsos positivos*, esto es, *tests* que siendo fallados no necesariamente marquen como defectuosa la **IUT**.

Cuestión aparte es la eficiencia tanto en espacio como en memoria cuya resolución abordamos en la siguiente sección mediante una técnica más sofisticada en la que los *tests* se generan dinámicamente.

4.2. Testing *online*

En la anterior sección hemos descrito un algoritmo generador de *tests* indeterminista cuyo resultado a partir de una especificación dada era una *suite* de *tests*.

La principal ventaja, sintetizada en el teorema 6, es que podemos considerar un conjunto representativo de todos los posibles *tests* capaces de distinguir la relación iocos entre una implementación y una especificación. Este resultado es tan importante en el contexto de la teoría de **MBT** (capítulo 2) que en ocasiones se llega a identificar esta modalidad, la estática u *offline*, como la única posible de abordar el *testing* en el proceso del que es parte, refiriéndose a ella por **MBT**.

4.2.1. Planteamiento general

Suponiendo el *testing offline* una reducción importante del número de *tests* a ejecutar, cabe todavía una objeción a esta técnica: esencialmente, en la generación de *tests* se tienen en cuenta todos los posibles compartimientos ulteriores a partir de un estado dado. La figura arbórea así resultante, requiere una cantidad de recursos computacionales tan elevada, ya sea en términos de memoria o de tiempo, que puede hacer inviable su realización práctica. A este problema se le conoce comúnmente como *problema de la explosión de estados*.

El problema es más acuciante al observar que muchas de las ramas así precomputadas no son exploradas desde el momento que otra rama paralela, en el transcurso del experimento, alcanza un veredicto antes de que otras hayan comenzado siquiera a entrar en el cómputo. Como conclusión, una gran parte de ellas es pre-computada en vano.

En esta sección trataremos de disminuir este efecto desde un enfoque distinto, integrando en un único algoritmo la ejecución del experimento, descrita en la sección anterior mediante la función pass (definición 26), con la generación de *primitivas* llevada a cabo por el algoritmo generador (definición 29).

Bajo esta nueva perspectiva, la generación de las *primitivas* será relativa al estado actual en que se encuentre la implementación, resultando por ello innecesario el cómputo de *potenciales* estados accesibles desde él. Según se resuelva la aplicación de la primitiva, tanto la implementación como la especi-

ficación progresarán hacia un nuevo estado, y en ese momento tendrá de nuevo lugar un nuevo cálculo de primitivas. Éstas comprenden el siguiente conjunto de acciones:

- el estímulo de la implementación mediante señales de entrada
- la observación de acciones de salida de manera autónoma
- la capacidad de devolver implementación a sus estados iniciales en cualquier momento, a la que nos referiremos como *reset*.

La primera y la segunda están presentes en la definición de la gramática en la que se describen los *tests offline* (definición 24) ($a?T$, $x!T$). La tercera, novedad en el *testing online*, resultará capital para asegurar la completitud del algoritmo bajo ciertos supuestos de ejecución *fairness*. Al respecto del resto de operadores presentes en la gramática, tales como $T_1 + T_2, T_1 \oplus T_2$, su efecto será reproducido de manera implícita mediante las estructuras de control que regirán la evolución del experimento según sea el resultado de haber aplicado las anteriores primitivas.

Existen muchos trabajos que profundizan en la técnica del *testing online*. Véase por ejemplo [94, 57] tratando el tema concierta generalidad. Como fuentes más especializadas considerando el tiempo como variable a tener en cuenta tenemos [48, 54, 56, 67, 19, 49] y en relativo a implementaciones o experimentos con herramientas ejecutables [25, 8, 7].

Para nuestro trabajo, adquieren especial relevancia dos trabajos esenciales que describen algoritmos de similares características a nuestra propuesta: en [57] se emplea la primitiva *reset*, y siguiendo la misma idea que [25], se indica que la detección de la *quiescencia*, tratada desde el punto de vista matemático como un símbolo de salida más, puede implementarse gracias a la acción de un *timeout*, consistente en un dispositivo capaz de detectar cuando en un determinado tiempo no se ha producido señal por parte de la implementación.

En la sección 4.2.2 describiremos una propuesta de algoritmo que toma su inspiración en los trabajos anteriormente citados. La corrección y completitud del algoritmo será el objeto de la sección 4.2.3.

4.2.2. Propuesta de algoritmo de *testing online*

La figura 4.2 nos presenta un esquema que puede darnos una idea del nuevo modo de operación: un *actuador/sensor* emite/recoge las primitivas del objeto examinado y deriva los resultados del mismo hacia un proceso *conductor*. Éste, en función del resultado y de la naturaleza de la especificación, ordena al *actuador/sensor* nuevas primitivas a exponer al objeto. El proceso se repite hasta que o bien se detecta un comportamiento inesperado o se alcanza una cota en el número de acciones realizadas.

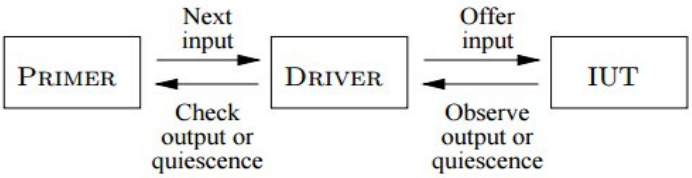


Figura 4.2: Arquitectura *Online tester* según aparece en [25]

Este proceso adopta la forma concreta del algoritmo 1, dependiendo de tres parámetros:

la especificación s**.** Formulada en términos de un modelo le son aplicables todos los operadores auxiliares presentados en 20 del capítulo 3. De esta manera, s evolucionará conforme a su estado actual y a los valores que a cada paso sean aplicados al objeto de *test* o recibidos de ella.

el estado de la implementación iut**.** Como veremos, será necesario un artificio de *copia* del estado al objeto de mantener la semántica correcta respecto a la relación de *conformidad*, tal como se sugiere en el trabajo pionero de Abramsky [2].

el número finito de iteraciones $maxIter$**.** Al igual que se hace tradicionalmente en otros campos de la computación al experimentar con funciones semidecidibles, este parámetro marca un límite en el número máximo de primitivas a experimentar. La probabilidad de encontrar un fallo en el comportamiento de la implementación se eleva conforme más alto es este número, en conjunción con la hipótesis de *justicia* (*fairness*).

Algorithm 1 Algoritmo de testing *online* para iocos

```
1: function TE(s, iut, maxIter)
2:     continue ← ✓
3:     numIter ← maxIter
4:     while numIter > 0 ∧ continue == ✓ do
5:         continue, numIter ← TE_REC(s, iut, numIter)
6:         if continue == ✓ then
7:             reset iut
8:     return continue
9: function TE_REC(s, iut, numIter)
10:    if numIter = 0 then
11:        return ✓, numIter
12:    else
13:        choice
14:            case action do          ▷ Offers an input to the implementation
15:                choice a ∈ ins(s)
16:                if a? is not enabled in iut then
17:                    return ✗, numIter
18:                send a? to iut
19:                iut_0 ← copy(iut)
20:                for s' ∈ s after a? do
21:                    iut ← copy(iut_0)
22:                    continue, numIter ← TE_REC(s', iut, numIter − 1)
23:                    if continue == ✓ then
24:                        return ✓, numIter
25:                return ✗, numIter
26:            case wait do    ▷ Waits for an output from the implementation
27:                wait o! from iut
28:                if s after o! = ∅ then
29:                    return ✗, T
30:                iut_0 ← copy(iut)
31:                for s' ∈ s after o! do
32:                    iut ← copy(iut_0)
33:                    continue, numIter ← TE_REC(s', iut, numIter − 1)
34:                    if continue == ✓ then
35:                        return ✓, numIter
36:                return ✗, numIter
37:            case reset do              ▷ Resets implementation and restart
38:                return ✓, maxIter
39:
```

Como se puede observar, la propuesta recogida en el algoritmo 1 queda dividida en dos partes bien diferenciadas, que pasamos a describir a continuación.

Función TE

Una primera, denotada por la función TE, actúa como conductora del proceso principal, encargada de *restaurar* la implementación a requerimiento de la segunda. En esencia, el proceso de *testing* continuará hasta que se produzca un veredicto de fallo, indicando que la implementación i no es conforme a s, o hasta que el número de iteraciones se haya alcanzado sin detectar un comportamiento defectuoso en la implementación.

Función TE_{REC}

Respecto a la segunda, TE_{REC}, está orientada a la exploración del espacio de estados que se genera a partir de sucesivas aplicaciones de las primitivas de *test* a la implementación de manera indeterminada con objeto de detectar una situación de fallo lo antes posible.

En el texto principal de TE_{REC}, vemos que la función esta definida inductivamente con respecto al número de iteraciones, devolviendo un diagnóstico de ✓ al alcanzar el caso base, lo que se interpreta como la imposibilidad de haber detectado un fallo a lo largo de exploración del espacio de estados.

El caso recursivo, organizado en torno al operador de elección **choice** de naturaleza indeterminista –como lo era el algoritmo generador de *tests* de la sección anterior– expone los tres tipos de primitivas descritas en la sección anterior:

aplicación de señales a la implementación El proceso debería de devolver ✗ en caso de que una entrada prescrita fuera rechazada por la **IUT**. Esta situación se resuelve en el primer condicional dentro de la primera sentencia **case** de la función TE_{REC} (líneas 16 y 17). Nótese que comprobar las acciones disponibles en la **IUT** (línea 16), es equivalente a comprobar la condición en la cláusula 22.1 de la definición de iocos.

En otro caso, si la acción que el *test* elige ofrecer está disponible en la **IUT**, el estímulo se envía a la implementación (línea 18) y entonces la

condición de la cláusula 22.2 de la definición de iocos debe ser comprobada: al menos uno de los *tests* restantes debe ser pasado por el estado calculado a partir del actual después de haber aplicado la señal a la implementación, ya que en otro caso la sentencia de vuelta de fuera del bucle propagaría el veredicto ✗.

La cláusula *copy* (línea 19) –esencial desde el punto de vista teórico, véase [2, 74]– se usa para comprobar cada *test* descendiente contra el estado original de la implementación, pues este podría haber sido modificado a lo largo de la experimentación de sucesivos *tests*[3].

recepción de señales desde la implementación De igual manera, si nos concentramos en las acciones de salida, el condicional de las líneas 28 y 29 detecta salidas inesperadas de la implementación. Cuando esto no ocurre, esto es, cuando una señal recibida de la **IUT** es aceptable, el algoritmo continúa y comprueba la condición de la cláusula 22.3 de la definición de iocos similar a la que se ha descrito anteriormente para el caso de las entradas.

restauración de implementación a valores iniciales Las dos opciones anteriores, junto con el mecanismo de recursión, son las responsables de una búsqueda en anchura en el espacio de estados de la **IUT**, lo que en ocasiones puede llevar unilateralmente al proceso a explorar ramas en las que no se detecte, o se haga tarde, un comportamiento anómalo. Para evitar este tipo de conducta, algunos dispositivos ofrecen la posibilidad de *resetear* el equipo a sus valores iniciales, dando la oportunidad de exploración a otras ramas donde encontrar comportamiento anómalo.

4.2.3. Corrección y exhaustividad

En esta sección demostraremos que el algoritmo 1 es completo, esto es, correcto y exhaustivo. Para ello, siguiendo el planteamiento de [25, 57], asumiremos ciertas hipótesis de *justicia* (fairness) en los pasos de elección no

[3] Esta circunstancia puede ser reproducida fácilmente en el caso del software e incluso en ciertos tipos de sistemas integrados

determinista. A todo esto hay que añadir la *hipótesis de test* [82] asumiendo que cualquier **IUT** puede ser modelada en el dominio de los sistemas de transiciones etiquetadas, pero sin ser necesariamente conocido su modelo.

Corrección

El siguiente teorema demuestra la corrección del algoritmo en el sentido de que, al devolver ✗ como veredicto, tenemos la certeza de que nos encontramos ante una implementación defectuosa.

Teorema 7. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$ $i, s \in S$, *y* $n \in \mathbb{N}$. *Si la llamada a la función* $\mathrm{TE}_{\mathrm{REC}}(s, i, n)$ *del algoritmo 1 devuelve* ✗, *entonces* i $\mathsf{ioco\acute{s}}$ s.

Demostración. La demostración se realiza por inducción sobre el parámetro n de $\mathrm{TE}_{\mathrm{REC}}(s, i, n)$.

$n = 0$ En este caso, el algoritmo devuelve como veredicto ✓, por lo que, al ser falsa la condición, el enunciado resulta cierto

$n > 0$ Supongamos por hipótesis de inducción que tal propiedad se cumple para cierto k. Tenemos que comprobar que se cumple para $k + 1$.

Recordemos que el algoritmo sólo puede devolver este resultado en contados sitios:

- A la hora de someter a la implementación a *estímulos* de entrada, en la rama *action* del operador **choice** de elección. Esto puede ser deberse a dos razones:
  - La primera, porque la implementación no está habilitada para esa *acción* $a? \in \mathsf{ins}(s)$. Entonces, no se cumple que $\mathsf{ins}(i) \supseteq \mathsf{ins}(s)$, luego i $\mathsf{ioco\acute{s}}$ s.
  - En caso de que sí lo estuviese, solamente se devuelve fallo cuando la llamada al algoritmo $\mathrm{TE}_{\mathrm{REC}}(s', iut, k)$ para *todos* los posibles estados s' tales que $s \xrightarrow{a?} s'$ devuelve ✗.

  Según la hipótesis de inducción esto marcaría que i' $\mathsf{ioco\acute{s}}$ s', donde i' es el estado al que ha llegado *iut* al someterla a la acción $a?$, para cualquier s' tal que $s \xrightarrow{a?} s'$, lo que determina que efectivamente, i $\mathsf{ioco\acute{s}}$ s.

- Al esperar una señal por parte de la *iut*. El razonamiento es similar, pues sólo hay dos sitios donde esto puede pasar:
  - cuando la salida no es registrada por la especificación, es decir, $s \text{ after } o! = \varnothing$
  - cuando, aún siendo registrada, la llamada al algoritmo con $\text{TE}_{\text{REC}}(s', iut, k)$, siendo s' e iut, los estados posteriores a los que se llega a través de esa señal, devuelve ✗.

  Por la hipótesis de inducción esto marcaría que $i' \not\text{ioco}\, s'$, y en consecuencia, $i \not\text{ioco}\, s$.

□

Exhaustividad e hipótesis de justicia

Asegurar la *exhaustividad* implica que, para cualquier implementación defectuosa con respecto a una especificación s, el proceso de *testing online* del algoritmo 1 debería finalmente producir un resultado de ✗ o lo que es lo mismo, el algoritmo debería dirigir el cómputo a través de un conjunto de primitivas que fuera pasado por la especificación y fallado por la implementación.

Notemos que un punto clave para demostrar la completitud es la *certeza* de que *todos* los estados en la especificación y en la implementación deberían poder ser inspeccionados. Esto se formula más precisamente bajo el requisito de *fairness* o *justicia* en la ejecución: toda rama del árbol de estados infinitamente elegible ha de ser ejecutada en algún momento.

Para poder asegurar este requisito una de las posibilidades del algoritmo es reiniciar la **IUT** y consecuentemente el algoritmo (líneas 37 y 38). Sin embargo, asumir la hipótesis de justicia implica también que el número de veces que se requiere para inspeccionar todos los estados es desconocido *a priori*. Por lo tanto, el parámetro que registra este valor, $maxIter$, no puede ser sino especulado por el experimentador antes de la ejecución del *test*. De lo que se puede estar cierto, no obstante, es que un tal número existe. Tal es el objeto de la siguiente proposición.

Proposición 5. *Sea* $LTS = (S, I, O, \rightarrow) \in \mathcal{LTS}$, $i, s \in S$, *y* $T \in \mathcal{T}(s)$ *un* test *tal que* $s \text{ pass } T$ *y* $i \not\text{pass}\, T$. *Entonces existe* $n, n' \in \mathbb{N}$ *tal que* $\text{TE}_{\text{REC}}(s, i, n)$ *devuelve el par* $(✗,\ n')$.

Demostración. La demostración se hace por inducción sobre la profundidad de T.

$prof(T) = 0$ En este caso, o bien $T = \checkmark$ o $T = \text{✗}$.

- En el primer caso, se obtiene que i pass T en contra de la hipótesis y por lo tanto haciendo correcta la proposición.
- En el segundo, se repite el razonamiento, con la circunstancia de que s pa$\not$ss T.

$prof(T) > 0$ Asumiendo la hipótesis de inducción, de acuerdo con la definición 29, tenemos dos casos.

- Vamos a considerar el test

$$T = a?\bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\} + \sum_{o!\in \mathsf{outs}(s)} o!\bigoplus\{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\} \; + \sum_{\substack{o!\in O \\ o!\notin \mathsf{outs}(s)}} o!\text{✗}$$

  Dado que i pa$\not$ss T, hay tres posibilidades de hacer que el *test* falle:

  i pa$\not$ss $a?\bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$

  Puesto que estamos asumiendo la hipótesis *fairness*, al elegir un n arbitrariamente grande la elección en la línea 15 del algoritmo tomará la acción $a?$.

  Si $a?$ no está disponible en i, entonces el algoritmo devuelve $(\text{✗}, numIter)$ en la línea 17.

  En otro caso el algoritmo envía $a?$ a la implementación. De nuevo, por la *hipótesis de justicia* la implementación terminará finalmente en un estado i' tal que i' pa$\not$ss $\bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$.

  Llamaremos $n_{i'}$ al número de iteraciones que se necesitan para llegar a este estado.

  Por otra parte, como s pass T, s' pass $\bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$ para cualquier $s' \in s$ after $a?$. Luego por la hipótesis de inducción para cualquier $s' \in s$ after $a?$ existirá un $n_{s'} \in \mathbb{N}$ de modo que $\mathrm{TE}_{\mathrm{REC}}(s', i', n_{s'})$ devuelve ✗.

  Por consiguiente, al elegir $n_0 \geq n_{i'} + \sum_{s'\in s\text{after}a?} n_{s'}$, existirá un n' tal que $\mathrm{TE}_{\mathrm{REC}}(s, i, n_0)$ devuelva $(\text{✗},\ n')$.

$i \not{\mathsf{pass}} \sum_{o! \in \mathsf{outs}(s)} o! \bigoplus \{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$

En este caso existe $o! \in \mathsf{outs}(s)$ tal que i produce una señal de salida $o!$ y pasa a un estado i' tal que $i' \not{\mathsf{pass}} \bigoplus \{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$. De nuevo por la *hipótesis de justicia* la implementación emitirá una señal de salida y acabará en un estado i'. Tomemos $n_{i'}$ el número de iteraciones necesarias para alcanzar este estado.
Por otro lado, $s' \mathsf{pass} \bigoplus \{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$ para cualquier $s' \in s\mathsf{after}\, o!$. Luego por hipótesis de inducción para cualquier $s' \in s\mathsf{after}o!$ existirá $n_{s'} \in \mathbb{N}$ tal que $\mathrm{TE}_{\mathrm{REC}}(s', i, n_{s'})$ devuelve $(\text{✗}, n'_s)$.
Al elegir $n_0 \geq n_{i'} + \sum_{s' \in s\mathsf{after}o!} n_{s'}$, existirá n' tal que $\mathrm{TE}_{\mathrm{REC}}(s, i, n_0)$ devuelva $(\text{✗},\ n')$.

$i \not{\mathsf{pass}} \sum_{\substack{o! \in O \\ o! \notin \mathsf{outs}(s)}} o!\text{✗}$

En ese caso existe $o! \in O$ tal que i produce una salida $o!$. Por la *hipótesis de justicia* la implementación producirá una salida en el algoritmo devolverá $(\text{✗}, numIter)$ en la línea 29.

- El otro caso es incluso más simple que éste, a saber,

$$T = \sum_{o! \in \mathsf{outs}(s)} o! \bigoplus \{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\} \ + \sum_{\substack{o! \in O \\ o! \notin \mathsf{outs}(s)}} o!\text{✗}$$

pues no contempla la posibilidad de que la implementación reciba señales del exterior. Este caso es como el segundo de los casos anteriores.

□

Puesto que el cometido de la función principal del algoritmo es hacer llamadas al algoritmo recursivo como corolario obtenemos el siguiente resultado:

Teorema 8. *Sea* $LTS = (S, I, O, \rightarrow)$, $i, s \in S$. *Si* $i \not{\mathsf{ioco}} s$ *entonces existe* $n \in \mathbb{N}$ *tal que la llamada a la función* $\mathrm{TE}(s, i, n)$ *en el algoritmo 1 devolverá* ✗.

Demostración. Por la proposición 5, existe un $n' \in \mathbb{N}$ que hace que $\mathrm{TE}_{\mathrm{REC}}(s, i, n')$ devuelva un par $(\text{✗}, n)$. El algoritmo TE registra estas variables en *continue* y $numIter$, lo que hace que el bucle termine, devolviendo el resultado de ✗. □

El teorema 8 da por concluido este capítulo en el que hemos expuesto dos tecnologías de *testing* capaces de determinar la relación de conformidad entre una eventual implementación y una especificación. En la primera, conocida como *testing offline*, se propuso un algoritmo para generar un conjunto de *tests* a partir de una especificación. En la segunda, pensada para superar los problemas de complejidad en memoria que presenta la anterior, el *test* en curso se va generando dinámicamente conforme a la actividad que muestra la implementación. A esta última solemos referirnos por la expresión *testing online*. Un factor común a ambas es que en ellas la especificación está formulada en algún lenguaje formal, siendo desconocido el modelo que subyace a la implementación.

En el siguiente capítulo abordaremos el problema desde una perspectiva computacional, para lo cual será necesario conocer los modelos tanto de la especificación como de la implementación. Para ello haremos uso de otra técnica: la resolución de problemas a partir de otros cuya solución ya se conoce.

Capítulo 5

Algoritmos para iocos

Aunque la mayoría de las veces no contamos con el modelo de la implementación, cuando esto ocurre podemos emplear métodos algorítmicos para resolver la cuestión de la idoneidad de una implementación con respecto a una especificación. Recurriendo a la literatura existente, vemos que se han resuelto problemas parecidos estableciendo equivalencias con otras áreas, como la del *procesamiento de grafos*: identificando las particiones del conjunto de nodos con clases de equivalencia, formulando las correspondientes estructuras cociente, el algoritmo que los resuelve gira en torno al concepto clave de *estabilidad*, que marca las condiciones del *punto fijo* que identificará la solución que se persigue.

En este capítulo se presenta una solución al problema de la relación de *conformance* iocos basada en dichas técnicas. No obstante, las particularidades especiales de iocos hacen que la aplicación de éstas no sea tan directa o trivial. Siguiendo estrategias conocidas en computabilidad, como la *reducción* de un problema a otro conocido de cuya solución se dispone, basaremos nuestro planteamiento en el llamado *Generalized Coarsest Partition Problem* o **GCPP**, del cual se ha demostrado su corrección en el contexto de otra semántica más general, la simulación. Para facilitar la explicación del proceso de reducción nos apoyaremos en una relación de preorden clásica, la simulación *ready*, de características intermedias entre las dos.

Una vez planteada la solución desde el punto de vista teórico abordamos su integración en el seno de una de las herramientas más reputadas disponibles para su ejecución: el entorno **mCRL2**. Se describen los experimentos realizados

con sistemas de gran tamaño correspondientes a la suite **VLTS** [41], conjunto significativo de casos extraídos del mundo académico e industrial.

5.1. Simulación ready

La conocida *simulación ready* [12, 14], supone el primer intento por establecer una comparación de procesos más débil que la *bisimulación*. De todo el espectro de semánticas, constituye la más restrictiva por debajo de ella [88]. Construida sobre la base de la definición del preorden de simulación, restringe ésta al exigir al objeto simulado al menos una transición por cada transición presente en el objeto simulador.

Considerando como modelos los *grafos etiquetados* G, la llamada *simulación ready* adopta el siguiente perfil:

Definición 30. Sea $G = (N, \rightarrow, \Sigma)$. Una relación $\leq N \times N$ se dice *simulación ready* sobre G si y solo si para cualquier $a, b, c \in N$

1. $a \leq b \Rightarrow [a]_\Sigma = [b]_\Sigma$;
2. $a \leq b \wedge b \rightarrow \quad \Rightarrow \; a \rightarrow$
3. $a \leq b \wedge a \rightarrow c \Rightarrow \exists d \in N(c \leq d \wedge b \rightarrow d)$

La llamada relación *ready*, $\leq_{RS}$, es la mayor de todas simulaciones *ready*, esto es,

$$\leq_{RS} = \bigcup \{R \mid R \text{ es una simulación } \textit{ready}\}$$

Por tanto, para cualquier simulación *ready* $\leq$ se tiene que $\leq \subseteq \leq_{RS}$.

Transiciones sin etiquetar Conviene resaltar, como ya hicimos en la sección 2.3, que en la anterior definición las transiciones entre nodos aparecen sin etiquetar, cuestión que quedará resuelta en la siguiente sección donde se describirá un procedimiento para evitar la pérdida de esta información manteniéndonos en el mismo dominio de una estructura $G = (N, \rightarrow, \Sigma)$.

La importancia para nuestro trabajo de la *simulación ready* radica en que ya Bloom [15] plantea una solución para su cómputo sobre la misma base del

algoritmo encargado de resolver la *simulación ordinaria*, descrita con anterioridad (definición 10). Esencialmente se trata de modificar las condiciones *iniciales* del algoritmo que resuelve ésta. Si bien aborda el problema desde un enfoque totalmente distinto –siguiendo un enfoque transformacional de derivación estilo *Dijkstra*– sus ideas, unidas a la formulación del problema **GCPP** (definición 15) parametrizado por un par-partición $\langle \Sigma, P \rangle$, son las que, con algunos ajustes, nos hacen albergar cierta esperanza de resolver el problema objeto de esta tesis: iocos.

Relaciones inducidas

La siguiente definición pone en conexión una relación definida entre un conjunto N y la relación que se establece entre las clases de equivalencia inducidas por la relación original.

Definición 31. Sea una relación $\leq$: $N \times N$. Denotando por $\equiv$ a la relación de equivalencia inducida por $\leq$, esto es

$$a \equiv b \iff a \leq b \text{ y } b \leq a$$

- Sobre $N/\equiv$ se puede definir $\precsim$ como la relación *natural* de la siguiente manera. Para cada $n_1, n_2 \in N$ se obtiene

$$[n_1]_{N/\equiv} \precsim [n_2]_{N/\equiv} \iff n_1 \leq n_2$$

A la inversa, partiendo de una relación entre clases de equivalencia, se describe una relación inducida entre los elementos de las clases.

Definición 32. Sea Σ un conjunto de clases de equivalencia definidas sobre N, y la relación $\precsim$: $\Sigma \times \Sigma$. Queda definida la relación $\precsim (N) : N \times N$ de la siguiente manera:

$$a \precsim (N) b \iff [a]_\Sigma \precsim [b]_\Sigma$$

Propuesta para el cálculo de la simulación ready

Como adelantábamos con anterioridad, en el caso de la *simulación ready* básicamente se trata de variar las condiciones iniciales del problema **GCPP** destinado a resolver la *simulación ordinaria*, haciendo un preprocesamiento

para refinar el par-partición de entrada $\langle \Sigma, P \rangle$ hacia otro $\langle \Omega, \preccurlyeq \rangle$ que restrinja más la solución final para poder la propiedad (2) de la definición 30.

Definición 33. Sea $G = (N, \rightarrow, \Sigma)$ un grafo etiquetado. El par-partición inicial $\langle \Omega, \preccurlyeq \rangle$ queda definido de la siguiente manera

- $\Omega = \{\alpha_1, \alpha_2, \ldots \alpha_n\} \subseteq \mathcal{P}(N)$ es la mayor partición más fina que Σ tal que:
$$(n_1, n_2) \in \alpha_i \iff n_1 \rightarrow \text{ si y solo si } n_2 \rightarrow$$
- Siendo $I \subseteq \Sigma \times \Sigma$ la relación identidad, $\preccurlyeq \subseteq \Omega \times \Omega$ define como la máxima relación subconjunto de $I(\Omega)$ que cumple lo siguiente:
$$[n_1]_\Omega \preccurlyeq [n_2]_\Omega \iff n_2 \rightarrow \text{ solo si } n_1 \rightarrow$$

Nota Nótese por tanto que si $[n_1]_\Omega \preccurlyeq [n_2]_\Omega$ entonces $[n_1]_\Sigma = [n_2]_\Sigma$

Carácter *ready*

Al igual que en una estructura $G = (N, \rightarrow, \Sigma)$ las relaciones sobre N que son simulaciones inducen pares-partición con la propiedad de la estabilidad (véase sección 2.3.2), las relaciones definidas con propiedades análogas a las de la definición 33, a las que nos referiremos como *ready*, están en correspondencia con la siguiente propiedad muy útil a la hora de dar un procedimiento que resuelva el cómputo de $\langle \Omega, \preccurlyeq \rangle$.

Definición 34. Sea $G^- = (N, \rightarrow)$. Un par-partición $\langle \Sigma', P' \rangle$ es *ready* si y solo si:
$$\forall \alpha', \beta' \in \Sigma',\ (\alpha, \beta) \in P', \beta' \rightarrow_\exists \ \Rightarrow\ \alpha' \rightarrow_\forall$$

Recordemos que con $\beta' \rightarrow_\exists$ queremos decir que *algún* elemento de la partición β' tiene alguna transición indefinida. Igualmente $\alpha' \rightarrow_\forall$, donde *todos* los elementos tienen alguna.

A continuación pasamos a definir cual es el par-partición que buscamos para con posterioridad poder identificarlo con las propiedades que caracterizan a la solución del problema **GCPP**.

Definición 35. Sea $G = (N, \rightarrow, \Sigma)$ un grafo etiquetado y $\leq_{RS}$ la mayor relación que cumple las propiedades de la definición 30. Siendo $\equiv_{RS}$ a la relación de equivalencia inducida por $\leq_{RS}$, es decir,

$$a \equiv_{RS} b \iff a \leq_{RS} b \text{ y } b \leq_{RS} a$$

definimos el par-partición $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$ de la siguiente manera:

- Al ser $\equiv_{RS}$ una relación de equivalencia, se induce una partición cociente $N/\equiv_{RS}$.
- Sobre esta partición $N/\equiv_{RS}$ se puede definir $\precsim_{RS}$ como la relación *natural* inducida por $\leq_{RS}$ en $N/\equiv_{RS}$, o más explícitamente, para cada $n_1, n_2 \in N$,

$$[n_1]_{\equiv_{RS}} \precsim_{RS} [n_2]_{\equiv_{RS}} \iff n_1 \leq_{RS} n_2$$

Nuestra propuesta es el cálculo de $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$, y por tanto la *simulación ready* inducida sobre el conjunto N, encontrando el mayor par-partición estable conforme a la definición 14 que sea refinamiento de $\langle \Omega, \preccurlyeq\rangle$.

Según la definición 15 esta es la solución del problema **GCPP** aplicada al par-partición $\langle \Omega, \preccurlyeq\rangle$. El par-partición $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$, pues, representa nuestro objeto de estudio por comparación con la solución al **GCPP**; los siguientes lemas 2, 3, 4 y 5 lo identifican con dicha solución al asegurar que en efecto el par-partición $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$ es un refinamiento de $\langle \Omega, \preccurlyeq\rangle$, es estable y maximal con respecto a estas dos propiedades respectivamente.

Refinamiento

Lema 2. *Sea $G = (N, \rightarrow, \Sigma)$ un grafo transformado. Se tiene entonces que $\langle N/\equiv_{RS}, \precsim_{RS}\rangle \sqsubseteq \langle \Omega, \preccurlyeq\rangle$, es decir, se trata de un refinamiento de $\langle \Omega, \precsim\rangle$.*

Demostración. Se procede por separado con los dos elementos del par-partición:

- $N/\equiv_{RS} \sqsubseteq \Omega$, es decir, para cualquier $\alpha \in N/\equiv_{RS}$ debe existir $\alpha' \in \Omega$, tal que $\alpha \subseteq \alpha'$.

  Supongamos efectivamente que $\alpha \in N/\equiv_{RS}$. Para dos cualesquiera elementos $s_1, s_2 \in \alpha$ tendremos que, por la definición de $N/\equiv_{RS}$, se cumple

que $s_1 \leq_{RS} s_2$ y $s_2 \leq_{RS} s_1$. Por la definición 30 esto quiere decir que, además de $[s_1]_\Sigma = [s_2]_\Sigma$, se tiene que

$$s_1 \rightarrow \text{ si y solo si } s_2 \rightarrow$$

luego debe existir $\alpha' \in \Omega$, tal que $s_1, s_2 \in \alpha'$, según está definido Ω.

- Ahora supongamos dos clases $[n_1]_{\equiv_{RS}}$ y $[n_2]_{\equiv_{RS}}$ tal que

$$[n_1]_{\equiv_{RS}} \precsim_{RS} [n_2]_{\equiv_{RS}}$$

Por la definición 35 de $\precsim_{RS}$, para cualesquiera elementos $n_1 \in [n_1]_{\equiv_{RS}}$, $n_2 \in [n_2]_{\equiv_{RS}}$ se obtiene que $n_1 \leq_{RS} n_2$, lo que en particular quiere decir $[n_1]_\Sigma = [n_2]_\Sigma$ y

$$n_2 \rightarrow \text{ solo si } n_1 \rightarrow$$

Según está definido $\preccurlyeq \subseteq \Omega \times \Omega$, (definición 33), esto ocurre si y solo si $[n_1]_\Omega \precsim [n_2]_\Omega$.

□

Estable

Demostraremos ahora que $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$ supone un par-partición estable. Consideremos las clases de equivalencia $[n_1]_{N/\equiv_{RS}}, [n_2]_{N/\equiv_{RS}}, \gamma \in N/\equiv_{RS}$ tal que $[n_1]_{N/\equiv_{RS}} \precsim_{RS} [n_2]_{N/\equiv_{RS}}$ y $[n_1]_{N/\equiv_{RS}} \rightarrow_\exists \gamma$. Deberemos probar que existe una clase $[n_2']$, tal que $[n_2] \rightarrow_\forall [n_2']$ y $[n_1']_{N/\equiv_{RS}} \precsim_{RS} [n_2']_{N/\equiv_{RS}}$, de acuerdo con la definición 14 de estabilidad.

Lema 3. *Sea $G = (N, \rightarrow, \Sigma)$ un grafo transformado. Entonces el par partición $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$ es estable con respecto a $\rightarrow$.*

Demostración. Si $[n_1]_{N/\equiv_{RS}} \rightarrow_\exists \gamma$ entonces existe $n_1' \in \gamma$ tal que $n_1 \rightarrow n_1'$. Como $n_1 \leq_{RS} n_2$ existe n_2' tal que $n_2 \rightarrow n_2'$ y $[n_1']_{N/\equiv_{RS}} \precsim_{RS} [n_2']_{N/\equiv_{RS}}$. Consideremos n_2' un elemento *maximal* con respecto a $\leq_{RS}$, es decir, para cualquier n_i' tal que $n_1 \leq_{RS} n_i'$ tenemos $n_i' \leq_{RS} n_2'$.

A continuación probaremos que $[n_2] \rightarrow_\forall [n_2']$: considérese $n_3 \in [n_2]$. Dado que ambos pertenecen a la misma clase de equivalencia $\equiv_{RS}$, el núcleo de la equivalencia hace cierto $n_2 \leq_{RS} n_3$. Por consiguiente existirá n_3' tal que

$n_3 \rightarrow n_3'$ y $n_2' \leq_{RS} n_3'$. Por la misma razón $n_3 \leq_{RS} n_2$, y ha de existir n_2'' tal que $n_2 \rightarrow n_2''$ y $n_3' \leq_{RS}, n_2''$. Dado que $\leq_{RS}$ es transitiva obtendremos $n_2' \leq_{RS} n_2''$. Ya que n_2' es un elemento maximal entonces $n_2' = n_2''$, luego $n_3' \in [n_2']$. □

Maximalidad

Ahora tenemos que describir $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$ como el mayor par-partición estable y refinamiento de $\langle \Omega, \preccurlyeq\rangle$. Este paso lo haremos de manera indirecta valorando las propiedades de la relación inducida por ésta sobre los nodos del grafo, que resulta ser una simulación *ready*. De esto da cuenta el siguiente lema auxiliar.

Lema 4. *Sea $G = (N, \rightarrow, \Sigma)$ un grafo transformado y sea $\langle S, \precsim\rangle$ un refinamiento de $\langle \Omega, \preccurlyeq\rangle$ tal que es estable con respecto a $\rightarrow$. En estas circunstancias la relación $\precsim(N)$ es una simulación ready, esto es $\precsim(N) \subseteq \leq_{RS}$.*

Demostración. Consideremos que $n_1 \precsim(N) n_2$.

- Dado que $\langle S, \precsim\rangle$ es un refinamiento de $\langle \Omega, \preccurlyeq\rangle$, tendremos $[n_1]_\Omega \preccurlyeq [n_2]_\Omega$. Ya que, por la definición 33 de $\preccurlyeq \subseteq \Omega \times \Omega$, $\preccurlyeq \subseteq I(\Sigma)$, se tiene que $[n_1]_\Sigma = [n_2]_\Sigma$, cumpliendo la cláusula 30.1. Además $n_2 \rightarrow$ sólo si $n_1 \rightarrow$, por lo tanto cumpliendo la cláusula 30.2

- Consideremos ahora $n_1' \in N$ tal que $n_1 \rightarrow n_1'$. Al considerar todos esos términos en el seno de su clase de equivalencia en S obtenemos $[n_1]_S \precsim [n_2]_S$ y $[n_1]_S \rightarrow_\exists [n_1']_S$.

  Dado que $\langle S, \precsim\rangle$ es estable existe $\delta \in S$ tal que $[n_2]_S \rightarrow_\forall \delta$ con $[n_1']_S \precsim \delta$. Revirtiendo las clases a sus elementos discretos esto quiere decir que existe $n_2' \in \delta$ tal que $n_2 \rightarrow n_2'$ y $[n_1']_S \precsim [n_2']_S$, lo que significa que $n_1' \precsim(N) n_2'$, conforme a la cláusula 30.3.

□

Lema 5. *Sea $G = (N, \rightarrow, \Sigma)$ un grafo estructurado. Cualquier par-partición $\langle S, \precsim\rangle$ refinamiento de $\langle \Omega, \preccurlyeq\rangle$ y estable es un refinamiento de $\langle N/\equiv_{RS}, \precsim_{RS}\rangle$, esto es, $\langle S, \precsim\rangle \sqsubseteq \langle N_{\equiv_{RS}}, \precsim_{RS}\rangle$.*

Demostración. Procedemos por separado cada uno de los elementos del par-partición

- Consideremos en primer lugar $\alpha \in S$ y $a, b \in \alpha$. Ya que $\precsim$ es reflexiva tenemos $\alpha \precsim \alpha$ y por consiguiente $a \precsim(N) b$ y $b \precsim(N) a$. Ahora bien, por el lema 4 sabemos que $\precsim(N) \subseteq \leq_{RS}$ por lo que tenemos $a \equiv_{RS} b$. Por consiguiente todos los elementos de α están en la misma clase de equivalencia en $N/\equiv_{RS}$, es decir, S es un refinamiento de $N/\equiv_{RS}$.

- A continuación consideremos $\alpha \precsim \beta$, y $a, b \in N$ tal que $[a]_S = \alpha$ y $[b]_S = \beta$. Ya que por el lema 4 tenemos que $\precsim(N) \subseteq \leq_{RS}$, tenemos $a \leq_{RS} b$ y consecuentemente $[a]_{RS} \precsim_{RS} [b]_{RS}$. Puesto que $[a]_S \subseteq [a]_{RS}$ y $[b]_S \subseteq [b]_{RS}$, tenemos que $[a]_S \precsim_{RS} (S)[b]_S$.

□

Por último, antes de presentar el teorema que relaciona la solución al problema **GCPP** con el cálculo de la simulación *ready*, la siguiente proposición certifica, como exige dicho problema, que en el par $\langle \Omega, \preccurlyeq \rangle$ la relación $\preccurlyeq$ es reflexiva y acíclica.

Proposición 6. *Sea el par partición $\langle \Omega, \preccurlyeq \rangle$ según la definición 33. $\preccurlyeq$ es reflexiva y acíclica*

Demostración. En efecto

- $\preccurlyeq$ es reflexiva, ya que para cualquier $n \in N$ se cumple trivialmente que $n \rightarrow$ solo si $n \rightarrow$.

- Por contradicción, supongamos que existe un ciclo, esto es, $[n_1]_\Omega \preccurlyeq [n_2]_\Omega \preccurlyeq \ldots [n_i] \preccurlyeq [n_1]_\Omega$. Esto implica que $n_1 \precsim(N) n_2 \precsim(N) \ldots \precsim(N) n_i \precsim (N) n_1$, y como $\precsim(N)$ es un preorden, por la transitiva se obtiene que $n_1 \precsim(N) n_i$. En consecuencia, n_1, n_i pertenecen a la misma clase de equivalencia, y por tanto todos los elementos que aparecen en la lista son en realidad la misma clase de equivalencia.

□

Este teorema central da cuenta de la vinculación entre la simulación *ready* y el problema **GCPP**

Teorema 9. *Sea* $G = (N, \rightarrow, \Sigma)$ *y sea* $\langle RS, \precsim \rangle$ *la solución del problema* ***GCPP*** *sobre* $G^- = \langle N, \rightarrow \rangle$ *y* $\langle \Omega, \preccurlyeq \rangle$*, siendo* $\langle \Omega, \preccurlyeq \rangle$ *el mayor par-partición* ready *sobre* G*. Entonces* RS *el* cociente simulación ready *de* G*, esto es,* $RS = N/\equiv_{RS}$*, y la relación* $\leq_{RS}$ *verifica*

$$a \leq_{RS} b \text{ si sólo si } [a]_{RS} \precsim [b]_{RS}$$

Demostración. Este teorema es consequencia de los lemas 2, 3 y 5 que identifican el par $\langle N/\equiv_{RS}, \precsim_{RS} \rangle$ con las propiedades de la solución al problema **GCPP**. □

Nuestra definición iocos comparte ciertas similitudes con la simulación *ready*, aunque añade cierta complejidad con respecto a la definición 30 al tratar con etiquetas en las transiciones, y no de manera uniforme, pues hace distinciones entre las transiciones referidas a comportamientos *autónomos*, o de entrada, y *reactivos*, de salida. Esto será el objeto de estudio de la siguiente sección.

5.2. Reducción de iocos al problema GCPP

El objeto de esta sección es el planteamiento de un método de un método algorítmico –por oposición al del capítulo 4 basado en *testing*– para resolver el problema de ver si, dados dos *sistemas de transiciones etiquetadas*, entre ellos se da la relación iocos descrita en el capítulo 3. Ya entonces se hacía notar que un enfoque ingenuo, ciñéndose a la definición original, hace caer en costes privativos tanto en espacio como en tiempo.

A fin de tomar ventaja de las técnicas ya desarrolladas por otros autores en torno a relaciones semánticas *branching* como las descritas en la sección 2.3, una buena elección es adoptar las estrategias del problema **GCPP**. Como veremos, no sólo la literatura disponible en torno a él [32, 91] sino además la existencia de herramientas [21] que implementan los algoritmos en plataformas UNIX son una garantía en el propósito de nuestra búsqueda.

En primera instancia, dos son los problemas que nos impiden la aplicación del problema **GCPP** tal como se ha expuesto en la sección anterior para resolver iocos.

- En lo relativo a los modelos, la teoría iocos aparece formulada sobre estructuras $LTS = (S, I, O, \rightarrow)$ (definición 22) mientras que el problema **GCPP** se formula en base a *grafos etiquetados* $G = (N, \rightarrow, \Sigma)$ (definición 7).

- Las condiciones de la definición 22 de iocos, como quiera que está planteada en un modelo diferente, tratan de distinta manera las transiciones de entrada y las de salida, lo que hace suponer una reformulación del concepto de *estabilidad* (definición 14).

Desarrollar una nueva versión adaptada del problema **GCPP** a las características de iocos conllevaría un enorme esfuerzo, ya que al redefinir la *estabilidad* quedan sin efecto los teoremas que se encargaban de tomar la existencia y unicidad de par-partición estable, así como la caracterización de la simulación como problema **GCPP**, trabajo recogido por otros autores en [33, 32].

Una alternativa más elegante, abstracta y económica consiste en plantear la solución mediante técnicas de *reducción*: sobre la base de una solución contrastada al problema conocido, trataremos de reformular el problema objetivo adaptándolo al entorno de las condiciones del original, no sin antes asegurarnos de lo que al operar de este modo obtendremos una *caracterización* conservando cierta propiedad semántica: la propiedad de estar relacionado mediante iocos.

Partiendo de este contexto ya conocido, la aplicación del problema **GCPP** arroja una solución óptima en algún sentido –el mayor par-partición estable– que es reinterpretada en virtud de la anterior caracterización y por tanto, con total *legitimidad* en el plano original con que iocos es formulada en estructuras $LTS = (S, I, O, \rightarrow)$.

Así pues, se plantean los siguientes objetivos:

- Plantear un algoritmo que nos permita la obtención de un *grafo transformado* $G = (N, \rightarrow, \Sigma)$ a partir de un sistema $LTS = (S, I, O, \rightarrow)$, sin pérdida de información con respecto al original.

- *Caracterización* de una propiedad-relación en el nuevo escenario que se demuestre equivalente a iocos en el contexto original.

- Adecuación del problema **GCPP** para el cálculo de la nueva relación obtenida.

Los siguientes apartados serán los encargados de describir, por orden, cada una de las etapas que acabamos de mencionar.

5.2.1. Transformación de un *LTS* a una estructura *G*

El problema **GCPP**, tal como se describe en [32], queda definido en términos de estructuras de *grafos etiquetados* $G = (N, \rightarrow, \Sigma)$.

Estos grafos no tienen etiquetas en las aristas, o lo que es lo mismo, las transiciones asociadas entre sus nodos no tienen asociada una etiqueta, o la tienen única, según se prefiera considerar.

Por otro lado, la estructura $LTS = (S, I, O, \rightarrow)$ sobre la que se asienta la definición 22 no tiene definida ninguna partición entre el conjunto S de sus nodos, o equivalentemente, la partición coincide con la llamada partición *universal*.

Valor implícito de Σ

Denotaremos por $\mathbb{T}(LTS) = (N', \Rightarrow)$ a la obtención del nuevo grafo transformado a partir de la estructura original $LTS = (S, I, O, \rightarrow)$, el cual comprende tanto al conjunto de *estados* (N') a considerar como las aristas o *transiciones* que se dan entre ellos ($\Rightarrow$).

Para simplificar la notación y los cálculos, consideraremos en lo sucesivo definido implícitamente el valor $\Sigma = N'$ para $\mathbb{T}(LTS) = (N', \Rightarrow)$. De esta manera, para cualesquiera elementos $n'_1, n'_2 \in N'$ tendremos se cumplirá trivialmente $[n'_1]_\Sigma = [n'_2]_\Sigma$ y no será necesario recargar innecesariamente las definiciones con éstas cláusulas como se hacía en las definiciones 30 y 33.

Transiciones sin etiquetar

Para evitar la pérdida de información de las transiciones que tienen lugar en un *LTS* se puede adoptar la siguiente estrategia: en cada estado del nuevo conjunto-dominio de vértices N' se codificará la acción misma asociada a una transición hacia ese estado en el grafo, de modo que los estados del grafo asociado, de un $LTS = (S, I, O, \rightarrow)$ sean pares del nuevo dominio

$$N' \subseteq S \times L \cup \{\cdot\}$$

donde el primer componente del par se refiere al estado del *LTS* original mientras que el segundo componente es la acción que se necesita para alcanzar este estado. El símbolo $(\cdot)$ se ha añadido como para denotar las acciones de aquellos estados que no tienen acciones para llegar a él.

Como consecuencia es previsible que encontremos en N' más elementos que en N, puesto que por cada estado en el que incidan n transiciones encontraremos exactamente n estados en N', lo que motivará una actualización consistente de las nuevas transiciones en la nueva estructura $G' = (N', \Rightarrow)$. La siguiente regla expresa más precisamente el procedimiento que considera en G' todas las transiciones originalmente presentes en el *LTS*

$$\frac{s \xrightarrow{y} s' \quad s, s' \in S}{(s, x) \Rightarrow (s', y) \quad x, y \in L \cup \{\cdot\}}$$

De nuevo nos encontramos con que existirán más transiciones en $\Rightarrow$ que en $\rightarrow$, aunque la ampliación sólo se ha efectuado teniendo en cuenta que por cada transición en *G'* existe una en el sistema original. Por sí solo esto es garantía de que el cambio de dominio no es motivo para perder información respecto a las transiciones existentes en el sistema original. Como veremos a continuación, aún es necesario extender más tanto el número de estados debido a otros factores.

El estado *mágico* y transiciones virtuales

Las particularidades de la definición 22 (ioco$\underline{s}$) sugieren que el objeto a *simular* permite introducir un comportamiento no especificado susceptible de no ser tenido en cuenta por el objeto simulador. Por lo tanto, sería de esperar que no tuviéramos que tener en cuenta esos estados alcanzados cuando las acciones de entrada no están presentes en el objeto simulador. Esto entra en contradicción con la definición 10 de simulación, donde *cualquier* compartimiento debe ser simulado, y por tanto con el planteamiento del problema **GCPP**, cuya noción de *estabilidad* recoge esta circunstancia.

Para solucionar esta situación en el sistema transformado, vamos a emplear el siguiente ardid: dado que es posible que el objeto de comparación efectúe movimientos que no han de ser simulados, derivaremos entonces el sistema simulador hacia una suerte de *estado mágico*, que denotaremos con el símbolo $*$.

De nuevo, el nuevo dominio queda extendido a

$$N' \subseteq (S \cup \{*\}) \times (L \cup \{\cdot\})$$

Ese estado tendrá la propiedad de ser capaz de simular *cualquier posible estado*. Por tanto, la regla que describe las *transiciones virtuales* hacia este nuevo estado por parte del objeto simulador ante acciones inicialmente no presentes adquiere esta forma:

$$\frac{s \overset{a?}{\not\longrightarrow}}{(s,x) \Rightarrow (*,a?)} \quad \begin{array}{l} s, s' \in S \\ a? \in I \end{array}$$

Esta modificación se completa haciendo que el estado *mágico* siga recursivamente simulando a todos aquellos estados alcanzables desde la primera acción no presente en el objeto simulado, volviendo a completar con transiciones virtuales,-esta vez, para cualquier tipo de acción- desde el estado *mágico* hacia sí mismo.

$$\frac{}{(*,x) \Rightarrow (*,y)}, \qquad x, y \in L \cup \{\cdot\},\ a? \in I$$

La definición 36 sintetiza el planteamiento anteriormente explicado en forma de algoritmo basado en reglas.

Definición 36. Sea $LTS = (S, I, O, \rightarrow)$ un sistema de transición etiquetado. Definiremos su grafo transformado como $\mathbb{T}(LTS) = (N, \Rightarrow)$ donde

- $N = (S \cup \{*\}) \times (L \cup \{\cdot\})$
- $\Rightarrow$ queda definido por las reglas de la figura 5.1.

$$\frac{s \overset{y}{\longrightarrow} s'}{(s,x) \Rightarrow (s',y)}, \quad \frac{s \overset{a?}{\not\longrightarrow}}{(s,x) \Rightarrow (*,a?)}, \quad \frac{}{(*,x) \Rightarrow (*,y)}, \quad \begin{array}{l} s, s' \in S \\ x, y \in L \cup \{\cdot\},\ a? \in I \end{array}$$

Figura 5.1: Reglas de transición en sistema transformado

5.2.2. Redefinición de la relación iocos

La idea del método de resolución de un problema por *reducción* lleva asociado, además de la adaptación del problema original al nuevo marco, una formulación de cierta propiedad que sea caracterización de la que se daba primitivamente. En este apartado vamos a definir una propiedad, g-iocos, (iocos aplicado a estructuras de grafos etiquetados) que resultará fundamental a la hora de considerar nuestro problema, el cálculo de la relación iocos en la nueva estructura anteriormente definida G.

Actualización del operador ins En el momento de introducir la relación iocos en el capítulo 3, veíamos como característica fundamental el distinto trato que recibían las acciones según estas fueran o *entrada* o de *salida*. La definición 22 de iocos marcaba esta distinción mediante el operador ins(definición 20), que indicaba el conjunto de acciones de *entrada* asociadas a un estado. Todo el esfuerzo de la sección anterior ha tenido por objetivo hacer indistinguibles las acciones, a fin de adaptarse al perfil de los algoritmos existentes que resuelven el problema **GCPP**.

No obstante, al haber introducido el *estado mágico* y las llamadas *transiciones virtuales*, necesitamos actualizar el operador ins para que pueda reflejar el nuevo valor consistentemente. Así, dado un estado en $G' = (N', \Rightarrow)$, que tiene la forma (n, x), es preciso que se le atribuyan *sólo* aquellas acciones de entrada presentes en el original. En el caso del *estado mágico*, al no disponer de representación en el original, es preciso que *no le sea asignado acción alguna*, y ello de en virtud de aquel principio que motivó su aparición afirmando que, puesto que cualquier estado debería estar en relación con él, el objeto simulado tuviese al menos tantas acciones de entrada (cláusula 22.1) como el objeto simulador; el único valor capaz de satisfacer esta circunstancia, aún cuando el objeto simulado no disponga de acciones de entrada originales viene dado por el conjunto vacío. Esto motiva la siguiente redefinición:

Definición 37. Sea $(s, x) \in N \subseteq (S \cup \{*\}) \times (L \cup \{\cdot\})$. El operador ins de la definición 20 se extiende al nuevo dominio de la siguiente manera:

$$\mathsf{ins}(s, x) \begin{cases} \varnothing & \text{si } s = * \\ \mathsf{ins}(s) & \text{si } s \in S \end{cases}$$

Después de haber definido en el apartado anterior cómo obtener un grafo transformado $\mathbb{T}(LTS) = (N, \Rightarrow)$ a partir de un sistema de transiciones $LTS = (S, I, O, \rightarrow)$, y habiendo adaptado el operador ins al nuevo dominio, estamos preparados para formular la relación g-iocos̲ que nos permitirá caracterizar la relación original en este nuevo contexto.

Convenio de escritura A fin de facilitar la notación, haremos uso de un sencillo convenio de escritura consistente en anotar las transiciones $\Rightarrow$ sobre los nodos $\mathbf{n} \in N$ con la acción codificada en el nodo destino, escribiendo

$$\mathbf{n_1} \stackrel{x_2}{\Longrightarrow} \mathbf{n_2}$$

siempre que exista $\mathbf{n_1} \Rightarrow \mathbf{n_2}$ con $\mathbf{n_1} = (n_1, x_1)$ y $\mathbf{n_2} = (n_2, x_2)$.

De modo similar a como planteamos en el capítulo 3 al definir iocos̲, plantearemos en dos fases el nuevo concepto de relación, g-iocos̲, verdadero objeto de nuestro interés.

Definición 38. Sea $\mathbb{T}(LTS) = (N, \Rightarrow)$ el grafo transformado de una estructura $LTS = (S, I, O, \rightarrow)$.

Una relación $R \subseteq N \times N$ es g-iocos̲ *simulación* si y solo sí para cualquier $\mathbf{n_1}, \mathbf{n_2} \in N,\ (\mathbf{n_1}, \mathbf{n_2}) \in R$ las siguientes condiciones se cumplen:

1. $\mathsf{ins}(\mathbf{n_2}) \subseteq \mathsf{ins}(\mathbf{n_1})$
2. para cualquier $a \in L$, si $\mathbf{n_1} \stackrel{a}{\Longrightarrow} \mathbf{n_1}'$ entonces existe $\mathbf{n_2}'$ tal que $\mathbf{n_2} \stackrel{a}{\Longrightarrow} \mathbf{n_2}'$ y $\mathbf{n_1}' R\ \mathbf{n_2}'$.

Definición 39. Sea $\mathbb{T}(LTS) = (N, \Rightarrow)$ y $\mathbf{n_1}, \mathbf{n_2} \in N$. Decimos que

$$\mathbf{n_1}\ \text{g-iocos̲}\ \mathbf{n_2}$$

si y sólo si existe una g-iocos̲-relación entre ellos, o equivalentemente

$$\text{g-iocos̲} = \bigcup \{R \mid R \subseteq S \times S,\ R \text{ es una g-iocos̲ relación}\} \tag{5.1}$$

A primera vista, la definición 38 se asemeja a una *especie* de *simulación ready* según se describe en 5.1, pues gracias al proceso de completar a base del estado mágico y transiciones virtuales resulta ya indiferente la distinción entre

acciones *input* y *output*, sin posibilidad de encontrar acciones que aún siendo válidas desde la relación no deben ser simuladas, lo cual suponía el problema más acuciante a la hora de tratarlo como una simulación ordinaria. Cierto que la cláusula de potencialidad de acciones 38.1 sí hace una distinción marcando sólo las acciones *input*, pero este es precisamente el el papel que tomará el par-partición *inicial* de entrada al problema **GCPP** que describiremos más adelante en la siguiente sección.

La intención era precisamente ésa, pues según se explicaba en la sección 5.1, el hecho de poder contar con una solución para la *simulación ready* en términos del problema **GCPP** de contrastada corrección y eficiencia hacía tener albergar razonables esperanzas de hallar una solución similar para iocos.

Reescritura de relaciones entre distintos dominios Una vez expuesta la nueva relación g-iocos, lo que corresponde ahora es dotarnos de procedimientos que nos permitan reescribir posibles relaciones desde sus respectivos dominios hacia los alternativos. Así, mediante la proposición 7 se describe cómo transformar una iocos-simulación concreta en una g-iocos-simulación, mientras que la proposición 8 presenta la transformación recíproca. Esto resulta fundamental para lograr una caracterización de la relación iocos en la nueva estructura del grafo obtenido según recogerá el teorema 10.

Proposición 7. *Sea* $LTS = (S, I, O, \rightarrow)$ *un sistema de transiciones etiquetado,* $\mathbb{T}(LTS) = (N, \Rightarrow)$ *su grafo transformado,* $R \subseteq S \times S$ *una* iocos *relación, y definamos* $R' \subseteq N \times N$ *como*

$$R' = \big\{((i,a),(s,a) | (i,s) \in R, \forall a \in L \cup \{\cdot\}\big\} \cup \big\{(i,a)(*,a) |\ a \in L \cup \{\cdot\}, i \in S \cup \{*\}\big\}$$

Entonces R' *es una* g-iocos *simulación.*

Demostración. Tomemos $(n_1, n_2) \in R'$ y comprobemos que R' cumple las cláusulas de la definición 38. Procediendo por casos:

$n_2 = (*, a)$**.** Esto es, estamos en el segundo subconjunto de R'. Trivialmente, tenemos $\mathsf{ins}(n_1) \supseteq \varnothing = \mathsf{ins}(n_2)$ (cláusula 38.1).

Como $(*, x) \Rightarrow (*, y)$ para cualquier $x, y \in L$, según se ha descrito en la transformación de un grafo, y $(n'_1, (*, a)) \in R'$ para cualquier n'_1, (por construcción de R') la cláusula 38.2 se cumple.

$n_1 = (*, a)$**.** Según R' está definida, $n_2 = (*, a)$, se resuelve como en el caso anterior, puesto que en el primer subconjunto nada se dice sobre pares en los que interviene el *estado mágico* $*$.

$n_1 \neq (*, a)$ **y** $n_2 \neq (*, a)$**.** Supongamos que $n_1 = (i, a)$ y $n_2 = (s, a)$. Dado que R es una $\underline{\text{iocos}}$ simulación, tenemos

$$\mathsf{ins}(n_1) = \mathsf{ins}((i, a)) = \mathsf{ins}(i) \supseteq \mathsf{ins}(s) = \mathsf{ins}((s, a)) = \mathsf{ins}(n_2)$$

por la cláusula 38.1 y la definición 37 del operador ins.

Ahora tomaremos la acción $x \in L$ tal que $n_1 \stackrel{x}{\Longrightarrow} n_1'$.

- Si $x \in O$ entonces tendremos que existe un i' tal que $i \stackrel{x}{\longrightarrow} i'$ y $n_1' = (i', x)$. Dado que R es una $\underline{\text{iocos}}$ simulación, entonces existirá s' tal que $s \stackrel{a}{\longrightarrow} s'$ y $(i', s') \in R$. De nuevo, por la forma en que están construidas las transiciones $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (s', x) = n_2'$ y $(n_1', n_2') \in R'$ por la definición de R', primer subconjunto.
- Si $x \in I$ existen dos posibilidades: o bien $x \notin \mathsf{ins}(i)$ o $x \in \mathsf{ins}(i)$.
  - En el primer caso se tiene que $(i, a) \stackrel{x}{\longrightarrow} (*, a) = n_1'$ por el modo en que las transiciones están definidas. Dado que R es una $\underline{\text{iocos}}$ simulación $\mathsf{ins}(s) \subseteq \mathsf{ins}(i)$ y consecuentemente $x \notin \mathsf{ins}(s)$. Entonces $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (*, x) = n_2'$ y $(n_1', n_2') \in R'$, por el modo en que está definida R', segundo subconjunto.
  - En el segundo caso existe i' tal que $i \stackrel{x}{\longrightarrow} i'$.
    - Si $x \in \mathsf{ins}(s)$, ya que R es una $\underline{\text{iocos}}$ simulación existirá s' tal que $s \stackrel{x}{\longrightarrow} s'$ y $(s, s') \in R$. Por la construcción de las transiciones $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (s', x) = n_2'$ y $(n_1', n_2') \in R'$ por la definición de R', segundo subconjunto.
    - Si $x \notin \mathsf{ins}(s)$, entonces $(s, a) \stackrel{x}{\Longrightarrow} (*, x) = n_2'$. Por consiguiente $(n_1', n_2') \in R'$ por la definición de R', primer subconjunto.

□

Como hemos apuntado con anterioridad, la proposición 8 nos muestra el procedimiento recíproco.

Proposición 8. *Sea* $LTS = (S, I, O, \rightarrow)$ *un sistema de transiciones etiquetado,* $\mathbb{T}(LTS) = (N, \Rightarrow)$ *su grafo transformado,* $R' \subseteq N \times N$ *una* g-iocos *relación, y definamos* $R \subseteq S \times S$ *como*

$$R = \{(i,s) | \exists a \in L \cup \{\cdot\},\ ((i,a),(s,a)) \in R' \wedge i, s \in S\}$$

Entonces R *es una* iocos *simulación.*

Demostración. Sea $(i,s) \in R$

- Por la propia construcción de R, existe $a \in L \cup \{\cdot\}$ y $n_1 = (i,a)$ y $n_2 = (s,a)$ tal que $(n_1, n_2) \in R'$. Dado que R' es una g-iocos simulación, tenemos que

  $$\mathsf{ins}(i) = \mathsf{ins}((i,a)) = \mathsf{ins}(n_1) \supseteq \mathsf{ins}(n_2) = \mathsf{ins}((s,a)) = \mathsf{ins}(s)$$

  de acuerdo con la cláusula 22.1.

- Consideremos $b? \in \mathsf{ins}(s)$ tal que $i \xrightarrow{b?} i'$, de modo que $(i,a) \stackrel{b?}{\Longrightarrow} (i', b?)$. Ya que R' es una g-iocos simulación existirá $n_2' = (s', b?)$ tal que $n_2 \stackrel{b?}{\Longrightarrow} n_2'$. Dado que $b? \in \mathsf{ins}(s)$, se tiene que $s' \neq *$ por la definición del operador ins, luego $s' \in S$. Finalmente, por la construcción de R obtenemos $(i', s') \in R$, conforme a la cláusula 22.2.

- El caso en que $o! \in \mathsf{outs}(i)$ y $i \xrightarrow{o!} i'$ es similar al anterior. En efecto, $s' \neq *$ ya que $o! \notin I$ y por la reglas al estado mágico $*$ sólo se llega desde una estado diferente mediante un símbolo de entrada. Luego $s \in S$ e $(i', s') \in R$, cumpliendo 22.3.

□

Finalmente, el teorema 10 es un corolario de los dos resultados previos mostrando iocos y g-iocos como dos formulaciones distintas de la misma semántica en los dos dominios diferentes.

Teorema 10. *Sea* $LTS = (S, I, O, \rightarrow)$ *sistema de transición etiquetado y* $\mathbb{T}(LTS) = (N, \Rightarrow)$ *su grafo transformado. Entonces, para cualquier* $s_1, s_2 \in S$

$$(s_1, s_2) \in \mathsf{iocos} \iff ((s_1, x), (s_2, x)) \in \mathsf{g\text{-}iocos}\ \ \forall x \in L \tag{5.2}$$

Demostración. A partir de las proposiciones 7 y 8. □

5.2.3. giocos como un problema GCPP

Llegados a este punto, disponemos de una relación, g-iocos, formulada en términos de un modelo de grafos etiquetados $G = (N, \rightarrow, \Sigma)$ y un problema, **GCPP**, que a partir de una par-partición de entrada nos garantiza como solución el mayor par-partición refinamiento de aquella que reúne las condiciones de estabilidad.

Recordemos que los pares-partición $\langle \Sigma, P \rangle$ de un grafo estructurado representaban clases de equivalencia definidas en el conjunto de nodos N así como un preorden establecido entre ellas respectivamente (definición 12).

El objeto de este apartado –capital para entender la legitimidad del proceso de reducción del problema– es identificar precisamente el par-partición *final* que resulta de la solución al **GCPP** bajo ciertas condiciones *iniciales* con las clases de equivalencia y el preorden *inducido* por la relación g-iocos.

La definición 40 indica cuáles son las condiciones iniciales o de entrada para el problema **GCPP** en nuestro intento de resolver iocos. Nótese el parecido que guarda esta definición con la correspondiente a la definición de las condiciones iniciales para resolver la simulación *ready* (definición 33).

Definición 40. Sea $LTS = (S, I, O, \rightarrow)$ un sistema de transición etiquetado y $\mathbb{T}(LTS) = (N, \Rightarrow)$ su grafo transformado. La *partición inicial* $\langle \Omega, \preccurlyeq \rangle$ se define del modo siguiente:

- $\Omega = \{\alpha_1, \alpha_2, \ldots \alpha_n\} \subseteq \mathcal{P}(N)$ es mayor la partición definida del modo siguiente:

$$(n_1, n_2) \in \alpha_i \iff n_1 = (s_1, x), n_2 = (s_2, y) \ , \ \mathsf{ins}(s_2) = \mathsf{ins}(s_1)$$

- $\preccurlyeq \subseteq \Omega \times \Omega$ como la mayor relación subconjunto de $I(\Omega)$ entre clases de la partición inicial que cumplen además lo siguiente:

$$[n_1]_\Omega \preccurlyeq [n_2]_\Omega \iff n_1 = (s_1, x), n_2 = (s_2, y), \mathsf{ins}(s_2) \subseteq \mathsf{ins}(s_1)$$

Así mismo en la definición 41 se definen las clases de equivalencia inducidas por la relación g-iocos, la estructura cociente y eventualmente una relación de preorden entre dichas clases.

Definición 41. Sea $LTS = (S, I, O, \rightarrow)$ un sistema de transición etiquetado y $\mathbb{T}(LTS) = (N, \Rightarrow)$ su grafo transformado.

- Denotaremos por $\equiv_{\mathsf{g\text{-}iocos}}$ el kernel de $\mathsf{g\text{-}iocos}$, esto es,

$$(a, b) \in \equiv_{\mathsf{g\text{-}iocos}} \iff (a, b) \in \mathsf{g\text{-}iocos} \text{ y } (b, a) \in \mathsf{g\text{-}iocos}$$

  Al ser $\equiv_{\mathsf{g\text{-}iocos}}$ una relación de equivalencia, se induce una partición cociente $N/\equiv_{\mathsf{g\text{-}iocos}}$.

- En esta partición $N/\equiv_{\mathsf{g\text{-}iocos}}$ se puede definir $\precsim_{\mathsf{g\text{-}iocos}}$ como la relación *natural* inducida por $\mathsf{g\text{-}iocos}$ en $N/\equiv_{\mathsf{g\text{-}iocos}}$, o más explícitamente, para cada $n_1, n_2 \in N$,

$$([n_1]_{\mathsf{g\text{-}iocos}}, [n_2]_{\mathsf{g\text{-}iocos}}) \in \precsim_{\mathsf{g\text{-}iocos}} \iff (n_1, n_2) \in \mathsf{g\text{-}iocos}$$

Proposición 9. *$\precsim_{\mathsf{g\text{-}iocos}}$ es reflexiva y acíclica en $N/\equiv_{\mathsf{g\text{-}iocos}}$.*

Demostración. En efecto

- $\precsim_{\mathsf{g\text{-}iocos}}$ es reflexiva, ya que para cualquier $n \in N$ se cumple trivialmente que $\mathsf{ins}(n) = \mathsf{ins}(n)$.

- Por contradicción, supongamos que existe un ciclo, esto es, $[n_1]_{\mathsf{g\text{-}iocos}} \preccurlyeq [n_2]_{\mathsf{g\text{-}iocos}} \preccurlyeq \ldots [n_i] \preccurlyeq [n_1]_{\mathsf{g\text{-}iocos}}$. Esto implica que $(n_1, n_2) \in \mathsf{g\text{-}iocos}$, $(n_i, n_1) \in \mathsf{g\text{-}iocos}$, y como $\mathsf{g\text{-}iocos}$ es un preorden, (por ser una caracterización de iocos que también lo es) por la transitiva se obtiene que $(n_1, n_i) \in \mathsf{g\text{-}iocos}$. En consecuencia, $[n_1]_{\mathsf{g\text{-}iocos}} = [n_i]_{\mathsf{g\text{-}iocos}}$ pertenecen a la misma clase de equivalencia. El razonamiento se extiende al resto de los elementos $n_2 \ldots$.

□

Debido a la proposición 9 podemos afirmar que $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$ se puede considerar un par-partición conforme a la definición 11.

Lo que resta por hacer es identificar el objeto-solución del problema **GCPP** con el par-partición $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$ inducido por la relación $\mathsf{g\text{-}iocos}$ que acabamos de definir. Aquel objeto quedaba caracterizado por ciertas propiedades, expuestas en la definición 15, las cuales reproducimos a continuación en el teorema 11 en forma de propiedades a demostrar, formulándolas sobre el par-partición $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$ (en lugar de $\langle \Sigma, P \rangle$) a partir del par-partición de entrada $\langle \Omega, \preccurlyeq \rangle$ (respectivamente $\langle S, \leq \rangle$).

Teorema 11. *Sea* $LTS = (S, I, O, \rightarrow)$ *un sistema de transición etiquetado y* $\mathbb{T}(LTS) = (N, \Rightarrow)$ *su grafo transformado. Entonces el par-partición*

$$\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle$$

reúne las siguientes propiedades:

(a) $\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle \sqsubseteq \langle \Omega, \preccurlyeq \rangle$ *(Refinamiento)*

(b) $\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle$ *es estable con respecto a* $\Rightarrow$*. (Estabilidad)*

(c) $\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle$ *es un par-partición* $\sqsubseteq$*-maximal satisfaciendo (a) y (b) (Maximalidad)*

Demostración. A partir de los Lemas 6,7,8 y 9 que se demuestran a continuación. □

Refinamiento

Mediante el siguiente lema, aseguramos que, efectivamente, el par-partición $\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle$ es un *refinamiento* de $\langle \Omega, \preccurlyeq \rangle$.

Lema 6. *Sea* $LTS = (S, I, O, \rightarrow)$ *un sistema de transición etiquetado y su grafo transformado* $\mathbb{T}(LTS) = (N, \Rightarrow)$*. Entonces* $\langle N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}, \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} \rangle \sqsubseteq \langle \Omega, \preccurlyeq \rangle$.

Demostración. Se procede por separado con los dos elementos del par-partición:

- $N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}} \sqsubseteq \Omega$, es decir, para cualquier $\alpha \in N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}$ debe existir $\alpha' \in \Omega$, tal que $\alpha \subseteq \alpha'$.

  Supongamos efectivamente que $\alpha \in N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}$. Para dos cualesquiera elementos $s_1, s_2 \in N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}$ tendremos que, por la definición de $N/ \equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}$,

  $$\mathsf{ins}(s_1) = \mathsf{ins}(n_1, x) = \mathsf{ins}(s_2) = \mathsf{ins}(n_2, y)$$

  luego debe existir $\alpha' \in \Omega$, tal que $s_1, s_2 \in \alpha'$, según está definido Ω.

- Ahora supongamos dos clases $[n_1]_{N/\equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}}$ y $[n_2]_{N/\equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}}$ tal que

  $$[n_1]_{N/\equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}} \precsim_{\mathsf{g\text{-}iocos}_{\underline{s}}} [n_2]_{N/\equiv_{\mathsf{g\text{-}iocos}_{\underline{s}}}}$$

Por la definición de $\precsim_{\text{g-iocos}}$, para cualesquiera elementos $n_1 \in [n_1]_{N/\equiv_{\text{g-iocos}}}$, $n_2 \in [n_2]_{N/\equiv_{\text{g-iocos}}}$ se obtiene que $(n_1, n_2) \in \text{g-iocos}$, lo que en particular quiere decir que $\text{ins}(n_1) = \text{ins}(s_1, x) \supseteq \text{ins}(s_2, y) = \text{ins}(n_2)$. Según está definido $\preccurlyeq \subseteq \Omega \times \Omega$, (definición 41), esto ocurre si y sólo si $[n_1]_\Omega \precsim [n_2]_\Omega$.

□

Estabilidad

La estabilidad, según se presentó en definición 14, era aquella cualidad que nos permitía distinguir cuándo el proceso de refinamiento del par-partición reunía las condiciones que hacen recursivamente aplicable las condiciones formuladas sobre el preorden de las clases de equivalencia asociadas a la partición. El lema 7 confiere al par-partición $\langle N/\equiv_{\text{g-iocos}}, \precsim_{\text{g-iocos}}\rangle$ esta propiedad.

Lema 7. *Sea $LTS = (S, I, O, \rightarrow)$ sistema de transición etiquetado y su grafo transformado, $\mathbb{T}(LTS) = (N, \Rightarrow)$; entonces el par partición $\langle N/\equiv_{\text{g-iocos}}, \precsim_{\text{g-iocos}}\rangle$ es estable con respecto a $\Rightarrow$.*

Demostración. Consideremos las clases de equivalencia $[n_1]_{\text{g-iocos}}, [n_2]_{\text{g-iocos}}, \gamma \in N/\equiv_{\text{g-iocos}}$ tal que $([n_1]_{\text{g-iocos}}, [n_2]_{\text{g-iocos}}) \in \precsim_{\text{g-iocos}}$ y $[n_1]_{\text{g-iocos}} \Rightarrow_\exists \gamma$. Deberemos probar que existe una clase $[n_2']$, tal que $[n_2] \Rightarrow_\forall [n_2']$ y $([n_1']_{\text{g-iocos}}, [n_2']_{\text{g-iocos}}) \in \precsim_{\text{g-iocos}}$, de acuerdo con la definición 14 de estabilidad.

Si $[n_1]_{\text{g-iocos}} \Rightarrow_\exists \gamma$ entonces existen $n_1' \in \gamma$ y $x \in L$ tal que $n_1 \stackrel{x}{\Longrightarrow} n_1'$. Como $(n_1, n_2) \in \text{g-iocos}$ existe n_2' tal que $n_2 \stackrel{x}{\Longrightarrow} n_2'$ y $([n_1']_{\text{g-iocos}}, [n_2']_{\text{g-iocos}}) \in \precsim_{\text{g-iocos}}$. Consideremos n_2' un elemento *maximal* con respecto a g-iocos, es decir, para cualquier n_i' tal que $(n_1, n_i') \in \text{g-iocos}$ tenemos $(n_i', n_2') \in \text{g-iocos}$.

A continuación probaremos que $[n_2] \Rightarrow_\forall [n_2']$: considérese $n_3 \in [n_2]$. Dado que ambos pertenecen a la misma clase de equivalencia $\equiv_{\text{g-iocos}}$, el núcleo de la equivalencia hace cierto $(n_2, n_3) \in \text{g-iocos}$. Por consiguiente existirá n_3' tal que $n_3 \stackrel{x}{\Longrightarrow} n_3'$ y $(n_2', n_3') \in \text{g-iocos}$. Por la misma razón $(n_3, n_2) \in \text{g-iocos}$, y ha de existir n_2'' tal que $n_2 \stackrel{x}{\Longrightarrow} n_2''$ y $(n_3', n_2'') \in \text{g-iocos}$. Dado que g-iocos es transitiva obtendremos $(n_2', n_2'') \in \text{g-iocos}$. Ya que n_2' es un elemento maximal entonces $n_2' = n_2''$, luego $n_3' \in [n_2']$. □

Maximalidad

También necesitamos demostrar que $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}}\rangle$ es *maximal* con respecto a las dos primeras cláusulas del **GCPP** (a) y (b).

Para proceder, notemos primero que cualquier par-partición $\langle S, \precsim\rangle$ sobre una estructura $G^- = (N, \rightarrow)$ induce una relación natural en N, denotada por $\precsim(N)$, recogida en la definición 42.

Definición 42. Sea $LTS = (S, I, O, \rightarrow)$ sistema de transición etiquetado y $\mathbb{T}(LTS) = (N, \Rightarrow)$ su grafo transformado, y $\langle S, \precsim\rangle$ un par partición. Se define la relación $\precsim(N) \subseteq N \times N$:

$$(n_1, n_2) \in \precsim(N) \iff ([n_1]_S, [n_2]_S) \in \precsim$$

A continuación nuestra estrategia para demostrar la maximalidad se apoya en dos lemas. El primero de ellos, lema 8, muestra que *cualquier* relación así inducida por un par-partición $\langle S, \precsim\rangle$ que sea *refinamiento* de $\langle \Omega, \preccurlyeq\rangle$ y estable con respecto a $\Rightarrow$ es una $\mathsf{g\text{-}iocos}_{\underline{s}}$ simulación.

Lema 8. *Sea $LTS = (S, I, O, \rightarrow)$ un sistema de transición etiquetado, $\mathbb{T}(LTS) = (N, \Rightarrow)$ su grafo transformado y sea $\langle S, \precsim\rangle$ un refinamiento de $\langle \Omega, \preccurlyeq\rangle$ tal que es estable con respecto a $\Rightarrow$. Entonces $\precsim(N)$ es una $\mathsf{g\text{-}iocos}_{\underline{s}}$ simulación.*

Demostración. Consideremos que $(n_1, n_2) \in \precsim(N)$.

- Dado que $\langle S, \precsim\rangle$ es un refinamiento de $\langle \Omega, \preccurlyeq\rangle$, tendremos $([n_1]_\Omega, [n_2]_\Omega) \in \preccurlyeq$. Entonces n_1, n_2 son de la forma (s_1, x), (s_2, y) respectivamente, y $\mathsf{ins}(n_2) \subseteq \mathsf{ins}(n_1)$ (por la definición 40 de $\preccurlyeq \subseteq \Omega \times \Omega$), por lo tanto cumpliendo la cláusula 38.1.

- Consideremos ahora que $x \in L$ y $n_1' \in N$ tal que $n_1 \stackrel{x}{\Longrightarrow} n_1'$. Al considerar todos esos términos en el seno de su clase de equivalencia en S obtenemos $([n_1]_S, [n_2]_S) \in \precsim$ y $[n_1]_S \Rightarrow_\exists [n_1']_S$.

  Dado que $\langle S, \precsim\rangle$ es estable existe $\delta \in S$ tal que $[n_2]_S \Rightarrow_\forall \delta$ y $([n_1']_S, \delta) \in \precsim$. Revirtiendo las clases a elementos puntuales esto quiere decir que existe $n_2' \in \delta$ tal que $n_2 \Rightarrow n_2'$.

  Ya que $\langle S, \precsim\rangle$ es un refinamiento de $\langle \Omega, \preccurlyeq\rangle$,tenemos $([n_1']_\Omega, [n_2']_\Omega) \in \preccurlyeq$. Puesto que $n_1 \stackrel{x}{\Longrightarrow} n_1'$, n_1' será de la forma (s_1', x). Análogamente, por la

definición 40 de $\preccurlyeq \subseteq \Omega \times \Omega$), n_2' sólo puede ser de la forma (s_2', x). Por consiguiente $n_2 \stackrel{x}{\Longrightarrow} n_2'$. Como $([n_1']_\Omega, [n_2']_\Omega) \in \precsim$, tenemos que $(n_1', n_2') \in \precsim (N)$, conforme a la cláusula 38.2.

□

Por último, por el lema 9, en el caso en que $\langle S, \precsim \rangle$ sea un refinamiento estable de $\langle \Omega, \preccurlyeq \rangle$, y que, por el lema 8 $\precsim(N)$ sea una g-iocos simulación, entonces es un refinamiento del par-partición $\langle N/\equiv_{\text{g-iocos}}, \precsim_{\text{g-iocos}} \rangle$. En otros términos, éste último par-partición es *maximal* con respecto a los criterios (a) y (b).

Lema 9. *Sea* $\mathbb{T}(LTS) = (N, \Rightarrow)$ *el grafo transformado de* $LTS = (S, I, O, \rightarrow)$ *y sea* $\langle S, \precsim \rangle$ *un par-partición tal que* $\precsim(N)$ *es una* g-iocos *simulación. Entonces* $\langle S, \precsim \rangle \sqsubseteq \langle N_{\equiv_{\text{g-iocos}}}, \precsim_{\text{g-iocos}} \rangle$.

Demostración. Procedemos por separado cada uno de los elementos del par-partición

- Consideremos en primer lugar $\alpha \in S$ y $a, b \in \alpha$. Ya que $\precsim$ es reflexiva tenemos $(\alpha, \alpha) \in \precsim$ y por consiguiente $(a, b) \in \precsim(N)$ y $(b, a) \in \precsim(N)$. Ahora bien, considerando que $\precsim(N) \subseteq$ g-iocos tenemos $(a, b) \in \equiv_{\text{g-iocos}}$. Por consiguiente todos los elementos de α están en la misma clase de equivalencia en $N/\equiv_{\text{g-iocos}}$, es decir, S es un refinamiento de $N/\equiv_{\text{g-iocos}}$.

- A continuación consideremos $\alpha, \beta \in \precsim$, y $a, b \in N$ tal que $[a]_S = \alpha$ y $[b]_S = \beta$. Ya que $\precsim(N) \subseteq$ g-iocos, obtendremos $(a, b) \in$ g-iocos y $([a]_{\text{g-iocos}}, [b]_{\text{g-iocos}}) \in \precsim_{\text{g-iocos}}$. Puesto que $[a]_S \subseteq [a]_{\text{g-iocos}}$ y $[b]_S \subseteq [b]_{\text{g-iocos}}$, tenemos que $([a]_S, [b]_S) \in \precsim_{\text{g-iocos}} (S)$.

□

5.3. Implementación

En las secciones precedentes hemos propuesto un método para el cómputo de la relación iocos siempre que contemos con un modelo que describa la implementación y la especificación. Con la garantía que dan las pruebas de

corrección, el reto que nos planteamos es su implementación en una plataforma *operativa* al alcance de cualquier usuario que desee contrastar la teoría con ejemplos o casos de uso que pudieran ser de interés.

Abordar la codificación de un algoritmo como el formulado en la sección 2.3.2, de por sí tan complejo, pero al que hay que añadir consideraciones no menores como la representación de estructuras de datos, operaciones sobre los mismos, así como todo el conjunto de infraestructura necesaria para los procesos de entrada/salida que cualquier sistema real soporta, tales como ficheros, sistemas de *log*... puede llevar más tiempo que el esfuerzo dedicado con anterioridad a *diseñar* una posible solución con garantías de corrección.

En este sentido, la estrategia elegida para resolver el problema ha sido la de *extender* las capacidades de una de las herramientas *software* más reputadas en el mundo especializadas en el cómputo de relaciones entre procesos: el entorno **mCRL2** [21, 42, 40], que describiremos brevemente en el siguiente apartado antes de explicar –a un nivel abstracto, sin demasiado detalle técnico– los algoritmos que forman la base de dicha extensión.

5.3.1. El entorno mCRL2

El entorno **mCRL2** [21, 42, 40] fue comenzado como proyecto por Jan Friso Groote y es actualmente desarrollado en la Technische Universiteit Eindhoven, en colaboración con LaQuSo, CWI y la Universidad de Twente. Se distribuye en formato de software de libre disposición, concretamente licenciado por *Boost Software Licence 1.0, (BSL-1.0)* [1].

La plataforma **mCRL2** consiste en conjunto de herramientas que incluyen funcionalidad para aplicar a una amplia variedad de técnicas formales que van desde el modelado a la validación y verificación de sistemas concurrentes y protocolos. Una perspectiva general de las herramientas se puede observar en la figura 5.2.

En su planteamiento original, **mCRL2** está planteado para operar sobre especificaciones de sistemas formuladas *algebraicamente*, cuestión que abordaremos en nuestro caso en el capítulo 6. Más concretamente, **mCRL2** viene a ser una variante del lenguaje de álgebra de procesos ACP_τ, o *Algebra of Communicating Processes*[9]. El sistema se plantea la normalización de la especificación a una forma canónica linear equivalente en la que se prescinde de

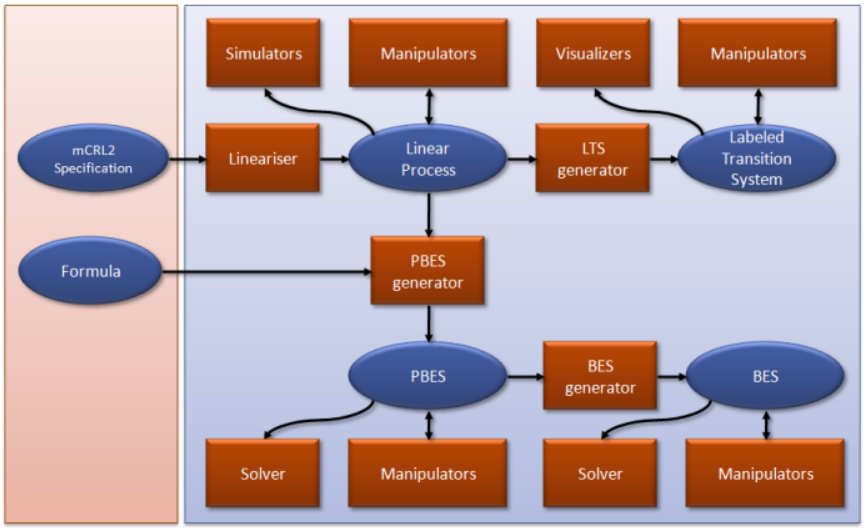


Figura 5.2: Esquema de la arquitectura de herramientas de mCRL [21]

operadores de paralelismo, consiguiendo una representación abstracta y finita de un espacio de estados potencialmente infinito. Esta forma resulta ideal para contrastar fórmulas propuestas sobre el comportamiento de un proceso.

En principio, ninguna de estas características nos resulta de provecho para nuestro objetivo inmediato, el cálculo de la solución **GCPP** a un par-partición dado. Más bien nuestro interés recae sobre aquellas herramientas que previamente contando con sistemas en forma de *sistema de transiciones etiquetadas* $LTS = (S, I, O, \rightarrow)$ nos permiten compararlas de acuerdo con algún criterio de equivalencia o preorden semántico. Entre estas equivalencias están la bisimulación, las inducidas por la simulación ordinaria, descritas en el capítulo 2. Otro tanto sucede para los preórdenes, como la misma simulación ordinaria que acabamos de mencionar.

Nuestro objetivo consistirá, pues, en extender la herramienta incorporando dos nuevos preórdenes: la simulación *ready*, descrita en la sección 2.3 y la que es objeto principal de esta tesis, iocos. La razón para la implementación de la primera es que, aparte de no estar incluida en la distribución oficial[1] supone un paso intermedio hacia la obtención de la segunda.

Al objeto de no abrumar con la exposición directa de los fuentes del proyecto, conteniendo cientos de ficheros, nos limitaremos a decir que todo el

[1] A fecha de 2013

sistema está codificado en C++. Básicamente, la clase objeto de nuestro trabajo es la denominada `sim_partitioner`, y su método más significativo es el `partition_algorithm`, que como su nombre sugiere, implementa el algoritmo de punto fijo de partición **GCPP**. En un estilo propio de *programación orientado a objetos*, el valor inicial de las variables, y por tanto la inicialización del bucle, se realiza en el constructor.

```
template <class LTS_TYPE>
class sim_partitioner
{
  public:
    sim_partitioner(LTS_TYPE& l);

    void partitioning_algorithm();

    std::vector < mcrl2::lts::transition> get_transitions() const;

    size_t num_eq_classes() const;

    size_t get_eq_class(size_t s) const;

    bool in_preorder(size_t s,size_t t) const;

    LTS_TYPE& aut;
    mcrl2::lts::outgoing_transitions_per_state_action_t trans_index;
    size_t s_Sigma;
    size_t s_Pi;
    std::vector<bool> state_touched;
    std::vector<bool> block_touched;
    std::vector<state_bucket> state_buckets;
    std::vector<size_t> block_Pi;
    std::vector<size_t> parent;
    ...
    hash_table3* exists;
    hash_table3* forall;
    std::vector< std::vector<size_t> > pre_exists;
    std::vector< std::vector<size_t> > pre_forall;
    std::vector< std::vector<bool> > P;
    std::vector< std::vector<bool> > Q;

    /* auxiliary variables */
    std::vector<size_t> touched_blocks;
    std::vector<size_t> contents;

    virtual void initialise_datastructures();
    void refine(bool& change);
    void update();
    ...
}
```

Figura 5.3: Muestra parcial de la cabecera del archivo `sim_part.h`

Las principales variables del algoritmo (bloques, relaciones y otras abstracciones como las $\forall$ y $\exists$ estructuras-cociente) aparecen como atributos privados de clase y codificadas en base a otras estructuras de bajo nivel como *arrays* y variables escalares, presentes en cualquier entorno imperativo. Se facilitan

rutinas abstractas para que el programador pueda manejar el conjunto de transiciones asociadas a un grafo y las transiciones salientes asociadas a un nodo con relativa facilidad.

Resulta de especial interés un método, `in_preorder`, que comprueba si dos estados están relacionados al observar si sus respectivas clases de equivalencia –o los `blocks` a los que pertenecen, terminología con la que se refiere a éstas en el código– lo están, a la manera que sugiere la definición 42 del apartado anterior.

5.3.2. Algoritmos de *completado*

Como hemos visto en la definición 36, para poder tratar el problema en términos del problema **GCPP**, tuvimos que *completar* el grafo original con un *estado mágico* y una serie de *transiciones virtuales* que conservaran la semántica del planteamiento inicial de *estabilidad*.

Transiciones etiquetadas en mCRL2 Llegados a este punto, nos encontramos que la implementación que hace del problema **GCPP** se hace en términos de sistemas de transiciones etiquetadas $LTS = (S, I, O, \rightarrow)$ lugar de grafos estructurados $G = (N, \rightarrow, \Sigma)$, con $\Sigma = N$ como valor inicial, tal como se describía en la propuesta original [32]. Esto supone que en nuestro planteamiento sobra la recodificación de los nodos en términos de pares de elementos, destino-transición, sino que puede abordarse directamente. En este sentido, podemos actuar del mismo modo que surge la notación abreviada, tal como se menciona en la sección anterior cuando describimos el procedimiento para transformar una estructura $LTS = (S, I, O, \rightarrow)$ en otra $G = (N, \rightarrow, \Sigma)$.

Lo que no deja de ser necesario es el proceso por el que se añaden las *transiciones* virtuales hacia el *estado mágico*. El algoritmo 2 expone en pseudocódigo el procedimiento a seguir puesto que su codificación en el lenguaje de programación C++ resulta considerablemente más compleja de seguir.

En él la variable S registra el conjunto de estados en la estructura, a la vez que T las transiciones vinculadas a la relación $\rightarrow$ en el seno de la estructura $LTS = (S, I, O, \rightarrow)$.

Algorithm 2 Procedimiento de completado

```
1: S := S ∪ {∗}
2: for l ∈ I ∪ O do
3:     T := T ∪ {(∗, l, ∗)}
4: for i ∈ I do
5:     for s ∈ S do
6:         if i ∉ ins(s) then
7:             T := T ∪ {(s, i, ∗)}
```

5.3.3. El subproblema de la partición inicial

En la definición 40 se especificó el par-partición inicial $\langle \Omega, \preccurlyeq \rangle$ conforme a una serie de propiedades tales que la aplicación del problema **GCPP** al mismo daba con la solución perseguida $\langle N_{\equiv_{\mathsf{g\text{-}iocos}}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$.

Nuestra propuesta es ahora dar un algoritmo –de nuevo, en pseudo-código– para obtener dicha partición $\langle \Omega, \preccurlyeq \rangle$ a partir de la partición universal, $\langle \{N\}, Id \rangle$. Básicamente, lo abordaremos en dos pasos:

- Ω es la mayor partición refinamiento de $\Pi = \{N\}$ que, para $i \in I$ verifica

$$\forall \alpha \in \Omega,\ \alpha \xrightarrow{i}_{\exists} \Rightarrow \alpha \xrightarrow{i}_{\forall} \tag{5.3}$$

- $\preccurlyeq$ es la mayor relación $\preccurlyeq \subseteq \Omega \times \Omega$ que satisface $\preccurlyeq \subseteq Id(\Omega)$ y

$$(\alpha, \beta) \in \preccurlyeq \wedge \beta \xrightarrow{i}_{\exists} \Rightarrow \alpha \xrightarrow{i}_{\forall} \tag{5.4}$$

El algoritmo 3 cumple con la primera de las premisas. La demostración de su corrección se expone en la proposición 10.

Proposición 10. *El algoritmo 3 computa la mayor partición refinamiento de* $\{N\}$ *que satisface 5.3.*

Demostración. En primer lugar, notemos que la condición 5.3 puede redefinirse equivalentemente como

$$\forall \alpha \in \Omega, \alpha \not\rightarrow_{\exists} N \vee \alpha \rightarrow_{\forall} N$$

Algorithm 3 Algoritmo de refinamiento de la partición universal

1: $\Omega := \{N\}$
2: **for** $i \in I$ **do**
3: **for** $\alpha \in \Omega, \alpha \xrightarrow{i}_{\exists}$ **do**
4: $\alpha_1 := \{a \mid a \in \alpha \wedge \exists b \in N \; a \xrightarrow{i} b\}$
5: $\alpha_2 := \alpha \setminus \alpha_1$
6: $\Omega := \Omega \setminus \{\alpha\}$
7: $\Omega := \Omega \cup \{\alpha_1\}$
8: **if** $\alpha_2 \neq \varnothing$ **then**
9: $\Omega := \Omega \cup \{\alpha_2\}$

Seguidamente consideremos el invariante generalización de la anterior expresión

$$\forall i \in I_l, \alpha \in \Omega, \alpha \not\xrightarrow{i}_{\exists} N \vee \alpha \rightarrow_{\forall} N$$

donde I_l denota el conjunto de etiquetas que han sido inspeccionadas por el bucle.

Dicho invariante se cumple trivialmente al principio, cuando el conjunto de etiquetas inspeccionadas es vacío. Trataremos de ver que es, en efecto, una propiedad invariante. De hecho, con cada partición $\alpha \in \Omega$ en cada vuelta del ciclo interior se opera del modo siguiente:

- Si $\alpha \not\xrightarrow{i}_{\exists}$ no hay necesidad de partir y el invariante se cumple al considerar la nueva etiqueta i.

- Si $\alpha \xrightarrow{i}_{\exists}$ entonces el cómputo de α_1 nos garantiza el *mayor* subconjunto de α cuyos elementos a tienen definida *alguna* transición, es decir, $a \xrightarrow{i}$. Recíprocamente, por la diferencia de conjuntos α_2 contiene el *mayor* subconjunto cuyos elementos a no tienen definida *ninguna* transición, o alternativamente $a \not\xrightarrow{i}$.

  Previamente eliminando α de la partición Ω ambos subconjuntos α_1, α_2 son añadidos a la nueva partición, salvo que $\alpha_2 = \varnothing$, en cuyo caso $\alpha_1 = \alpha$, no hizo falta partir, y se razona igual que el caso en que $\alpha \not\xrightarrow{i}_{\exists}$.

  En el caso en que $\alpha_2 \neq \varnothing$, ambos subconjuntos son los mayores que cumplen la propiedad para la nueva etiqueta. Dado que $\alpha \subseteq N$ ya era

el mayor subconjunto que lo cumplía para las etiquetas hasta entonces inspeccionadas, por el lema 10 α_1, α_2 también lo cumplen para ellas.

□

Lema 10. *Sea $\alpha \subseteq N$ e I un conjunto de etiquetas, tal que, para cada etiqueta $i \in I$, $\alpha \overset{i}{\not\rightarrow}_\exists \vee \alpha \overset{i}{\rightarrow}_\forall$ Entonces cualquier subconjunto $\alpha' \subseteq \alpha$, cumplirá que para cada etiqueta $i \in I$, $\alpha' \overset{i}{\not\rightarrow}_\exists \vee \alpha' \overset{i}{\rightarrow}_\forall$*

Demostración. Por hipótesis para cualquier etiqueta que $i \in I$ que consideremos, se cumple de manera excluyente que, o *ningún* elemento $a \in \alpha$ tiene definida una transición $a \overset{i}{\rightarrow}$, o *cualquier* elemento $a \in \alpha$ tiene definida una transición así. Como $\alpha' \subseteq \alpha$, el razonamiento se extiende de forma natural a los elementos de α' que forman parte de α. □

En segundo lugar, necesitaremos actualizar la relación entre particiones inducida por la nueva partición, $Id(\Omega)$ (definición 12). Lo único que tenemos que hacer excluir todos aquellos pares que no cumplen (5.4), es decir, aquellos para los que se tiene

$$(\alpha, \beta) \in Q, l \in L, (\beta \overset{l}{\rightarrow}_\exists) \wedge (\alpha \overset{l}{\not\rightarrow}_\forall)$$

De nuevo, el algoritmo 4 marca esta idea. El propio texto del contenido sugiere como se lleva a caso el proceso sin necesidad de formular alguna proposición al respecto.

Algorithm 4 Actualización de la relación *ready*

```
1: Q := Id(Π)
2: for l ∈ L do
3:     for (α, β) ∈ Q, β --l-->_∃ do
4:         if α --l-/->_∀ then
5:             Q := Q \ {(α, β)}
```

5.3.4. Medios de cómputo y casos de uso

En esta sección mostraremos y comentaremos los resultados que obtuvimos al ejecutar iocos. Los experimentos se ejecutaron en una plataforma convencional dotada de una CPU i7-4770 a 3.40Ghz y 16GB de memoria RAM, ejecutando el kernel Linux de la serie `3.2.0` y el sistema operativo Ubuntu-12.04

server que integra la correspondiente distribución de **mCRL2** disponible en repositorios ordinarios.

Benchmark Se optó por utilizar las descripciones de sistemas disponibles de la suite de *benchmark* **VLTS** [41], un proyecto conjunto desarrollado con equipos de investigación pertenecientes al CWI y al INRIA.

El interés en usar la suite **VLTS** tiene dos propósitos.

- El acrónimo **VLTS** abrevia *Very Large Transition Systems* y convierte a esta suite en un candidato idóneo para realizar pruebas sobre la implementación de la teoría iocos que describimos en este capítulo.
- Los *sistemas de transiciones etiquetadas* **VLTS** se han obtenido a partir de varios caso de estudio de protocolos de comunicación y sistemas concurrentes. Se da la circunstancia de que muchos de estos casos de estudio están presentes en la vida real, en sistemas industriales.

Cuadro 5.1: Especificaciones de algunos de los benchmark **VLTS**

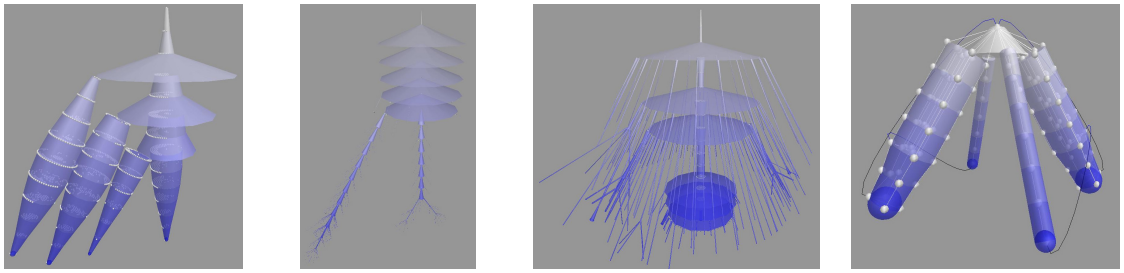
Name	States	Transtitions	Labels	Branching factor	Size
vasy_8_24.aut	8,879	24,411	11	2.75	460K
vasy_10_56.aut	10,849	56,156	12	5.18	1.3M
vasy_52_318.aut	52.268	318.126	17	6.09	10M
vasy_157_297.aut	157,604	297,000	235	1.88	5.5M
cwi_214_684.aut	214,202	684,419	5	3.20	15M
cwi_371_641.aut	371,804	641,565	61	1.73	17M
vasy_386_1171.aut	386,496	1,171,872	73	3.03	23M
cwi_566_3984.aut	566,640	3,984,157	11	7.03	77M
vasy_574_13561.aut	574,057	13,561,040	141	63.62	264M
vasy_1112_5290.aut	1,112,490	5,290,860	23	4.76	176M

En la Tabla 5.1 se muestra alguna información sobre los sistemas que hemos utilizado, y a continuación se dan unas pequeñas orientaciones que ayuden a comprender su significado:

- El número de estados varía desde 10^3 hasta 10^6, marcado en la columna States.

- Bajo la columna Transitions el número de transiciones se extiende desde 10^4 hasta 10^7.
- La columna Labels indica el cardinal del conjunto L de etiquetas.
- El valor Branching Factor da un promedio estadístico del número de transiciones que se dan por cada nodo considerado.
- Finalmente, la columna Size de la Tabla 5.1 indica el tamaño del fichero que contiene la descripción del sistema.

También se proporcionan en la figura 5.4 algunas representaciones pictóricas de los sistemas que hemos considerado *después* de haber practicado los experimentos sobre dichos sistemas.



`vasy_10_56` simulation | `vasy_52_318` ready simulation | `vasy_157_297` iocos 25 | `vasy_386_1171` iocos 50

Figura 5.4: Grafos reducidos indicando la semántica considerada

Experimentos Las implementaciones de las semánticas tanto iocos como las de simulación *ready* pueden abordarse de diferentes maneras. La más inmediata sería comprobar mediante un comando las dos descripciones de sistemas respecto a los preórdenes asociados y sus equivalencias. El estudio estadístico se hace tomando la medida del tiempo que ha llevado al sistema resolver la *comparación*.

Otra práctica común frecuentemente empleada en otras áreas de estudio como el *Model Checking*[20, 43, 68, 34] consiste en la *minimización* de un sistema *módulo-equivalencia de estados*, al objeto de encontrar la más pequeña

de las representaciones equivalentes a la original. En este último caso podremos registrar, además del tiempo, la calidad de la reducción en términos de porcentaje sobre el número de estados y transiciones.

Hablando en términos computacionales no hay diferencia esencial entre los problemas *a priori* diferentes entre comparar las descripciones de dos sistemas y la de minimizar o reducir uno dado. De hecho los cómputos son muy parecidos en ambos casos:

Comparación Al comparar dos sistemas, en primer lugar éstos se integran en uno más grande, y el algoritmo **GCPP** se aplica para encontrar el mayor par partición estable sobre el sistema unificado; entonces, si los dos estados iniciales de los sistemas originales pertenecen a la misma partición final, o a dos particiones relacionadas entre si, la comparación responde positivamente.

Minimización Dado un sistema, el algoritmo **GCPP** se aplica para encontrar el mayor par-partición estable; esto determina una estructura de grafo cociente entre los estados de los sistemas, y a partir de la relación entre las particiones finales se eliminan ciertas aristas consideradas redundantes con respecto al preorden considerado. El subgrafo así obtenido a partir de la clase del estado inicial determina un representante minimal.

5.3.5. Valoración estadística de los experimentos

La Tabla 5.2 muestra los resultados de los experimentos. Para una mayor compresión a continuación desglosamos la información que nos proporciona cada columna.

- Bajo la columna System aparece el nombre de los sistemas del benchmark **VLTS** que se han usado, como el número de estados y transiciones, referidas en la Tabla 5.1.

- En la columna Semantics, para cada sistema, se dispone de cinco filas, tres de las cuales están relacionadas con iocos, a las que se ha añadido un número (25,50, y 75), del que pasamos a explicar su significado.

  La herramienta **mCRL2** no distingue entre acciones de entrada y de salidas, pero admite cualquier identificador en las transiciones etiquetas,

Cuadro 5.2: Resultados sobre la minimización de grafos

System	Semantics	Time	Mem	States	Reduction	Trans	Reduction
vasy_8_24	simulation	0.86	0.01Gb	408	95.40 %	1102	95.49 %
	ready	0.99	0.01Gb	414	95.34 %	1140	95.33 %
	iocos 25	0.96	0.01Gb	412	95.36 %	1420	94.18 %
	iocos 50	1.00	0.01Gb	412	95.36 %	1420	94.18 %
	iocos 75	1.05	0.01Gb	398	95.52 %	1703	93.02 %
vasy_10_56	simulation	5.23	0.02Gb	2112	80.53 %	11372	79.75 %
	ready	5.06	0.02Gb	2112	80.53 %	11372	79.75 %
	iocos 25	15.65	0.16Gb	2113	80.52 %	18117	67.74 %
	iocos 50	15.92	0.13Gb	2113	80.52 %	22107	60.63 %
	iocos 75	13.41	0.04Gb	2113	80.52 %	24060	57.16 %
vasy_52_318	simulation	53.48	0.13Gb	5476	89.52 %	28643	91.00 %
	ready	35.59	0.10Gb	8126	84.45 %	46007	85.54 %
	iocos 25	75.64	0.21Gb	801	98.47 %	6648	97.91 %
	iocos 50	103.25	0.36Gb	1656	96.83 %	19348	93.92 %
	iocos 75	127.84	0.52Gb	1656	96.83 %	24254	92.38 %
vasy_157_297	simulation	490.64	0.09Gb	4289	97.28 %	13642	95.41 %
	ready	475.18	0.10Gb	4289	97.28 %	13642	95.41 %
	iocos 25	1011.26	2.24Gb	4290	97.28 %	263302	11.35 %
	iocos 50	1539.06	3.85Gb	4290	97.28 %	512889	-72.69 %
	iocos 75	2038.13	5.37Gb	4290	97.28 %	759343	-155.67 %
cwi_214_684	simulation	19150.54	2.13Gb	46037	78.51 %	100446	85.32 %
	ready	19547.16	2.02Gb	49785	76.76 %	113606	83.40 %
	iocos 25	29055.69	2.28Gb	48247	77.48 %	150204	78.05 %
	iocos 50	39104.20	2.35Gb	50012	76.65 %	202763	70.37 %
	iocos 75	943.64	1.92Gb	3766	98.24 %	18997	97.22 %
cwi_371_641	simulation	2277.64	0.68Gb	31714	91.47 %	68925	89.26 %
	ready	2310.25	0.73Gb	31714	91.47 %	69125	89.23 %
	iocos 25	13860.30	12.49Gb	25467	93.15 %	431247	32.78 %
	iocos 50	14168.89	13.21Gb	17297	95.35 %	564830	11.96 %
	iocos 75	4634.98	5.61Gb	6067	98.37 %	288177	55.08 %
vasy_386_1171	simulation	175.85	0.13Gb	113	99.97 %	150	99.99 %
	ready	155.12	0.20Gb	113	99.97 %	150	99.99 %
	iocos 25	207.14	1.27Gb	114	99.97 %	2229	99.81 %
	iocos 50	256.16	2.37Gb	114	99.97 %	4343	99.63 %
	iocos 75	260.61	3.40Gb	114	99.97 %	6308	99.46 %
cwi_566_3984	simulation	406.11	0.60Gb	8322	98.53 %	35809	99.10 %
	ready	380.83	0.76Gb	10291	98.18 %	45222	98.86 %
	iocos 25	400.65	0.87Gb	8322	98.53 %	35809	99.10 %
	iocos 50	579.99	1.57Gb	8323	98.53 %	74893	98.12 %
	iocos 75	410.62	0.87Gb	8322	98.53 %	35809	99.10 %
vasy_574_13561	simulation	720.83	1.28Gb	3577	99.38 %	16168	99.88 %
	ready	582.59	2.17Gb	3577	99.38 %	16168	99.88 %
	iocos 25	853.87	5.13Gb	3578	99.38 %	137753	98.98 %
	iocos 50	913.20	7.68Gb	3578	99.38 %	262074	98.07 %
	iocos 75	1101.97	7.70Gb	3578	99.38 %	383332	97.17 %
vasy_1112_5290	simulation	197.23	0.54Gb	265	99.98 %	1300	99.98 %
	ready	131.80	0.86Gb	265	99.98 %	1300	99.98 %
	iocos 25	205.19	1.50Gb	266	99.98 %	2265	99.96 %
	iocos 50	176.85	2.34Gb	266	99.98 %	3510	99.93 %
	iocos 75	206.56	3.16Gb	266	99.98 %	4740	99.91 %

lo que es aprovechado por nuestra de iocos para distinguir entre acciones de entrada y salida, marcadas por los sufijos '?' y '!' respectivamente. Este último aspecto resulta determinante porque uno de los aspectos que caracterizan la relación iocos es el diferente tratamiento de las entradas y salidas.

Igualmente, las descripciones de sistemas **VLTS** no hacen distinción entre entradas y salidas, y en consecuencia se han modificado las etiquetas según el protocolo descrito anteriormente para ajustarlas a iocos. El índice que se añade al nombre de iocos marca un factor de proporcionalidad para cada tipo de acciones: 25 para el 25 % de acciones de entrada, 50 para 50 % y 75 para el 75 % .

- Las columnas Time y Mem de la tabla se expresan en segundos y gigabytes respectivamente e indican el *total* del tiempo y memoria invertidos en ejecutar la reducción.

  Hay que destacar que estas medidas incluyen todos los pasos necesarios para el cómputo de iocos y la simulación *ready*, es decir, el cómputo de la partición inicial 5.3.3, y el tiempo necesario para completar con transiciones virtuales, en el caso de la relación iocos.

- Las dos columnas siguientes States/Reduction muestran el número final de estados del sistema reducido y el porcentaje de la reducción con respecto al número de estados en el sistema original. Una información análoga se recoge en las dos últimas columnas Trans/Reducción con respecto a las transiciones del sistema reducido y el original.

Valoraciones sobre la reducción Un hecho destacado a tener en cuenta en los datos de la Tabla 5.2 es la gran reducción en el número de estados de los sistemas reducidos, que va desde el 76 % a un sorprendente 99.98 % en el casi de sistemas muy grandes `vasy_1112_5290` en el cual más de un millón de estados se reducen a poco más de doscientos cincuenta, con independencia de la semántica testeada.

Al considerar la reducción en las transiciones, los experimentos con `vasy_157_297` resultan de interés particular, pues al considerar los grafos reducidos mediante iocos, la reducción es pequeña (11 % para iocos 25) o incluso el número de

transiciones es mayor que en el original (en el caso de iocos 50 y 75). Este caso resulta perfectamente asumible por el proceso de completado de transiciones virtuales comentado en la sección 5.3.2. En la Tabla 5.1 vemos que el número de etiquetas de este sistema es alto y el factor de ramificación bajo, lo que en consecuencia quiere significar que hay que invertir mucho en la fase de completado de transiciones virtuales.

No obstante, no es una idea que se pueda generalizar puesto que hay otros sistemas con características similares –bajo factor de ramificación y alto número de etiquetas– como `vasy_386_1171`, para los que sus resultados son por completo diferentes y la reducción por la transición es cercana al 99 % en el caso de iocos.

Valoraciones sobre el tiempo Al valorar el tiempo empleado en las reducciones, resulta interesante advertir que no hay una clara correlación entre el tamaño de un sistema (en términos de estados y transiciones) y el tiempo invertido para ello.

Se dan casos de grandes sistemas como `vasy_1112_5290` o `cwi_566_3984` que se reducen en unos pocos minutos, mientras que otros considerablemente menores, como `cwi_214_684` pueden llevar horas. Se podría atribuir este efecto a la naturaleza *intrínseca* de los sistemas (factor de ramificación, número de etiquetas, redundancia, estructura interna...)

De hecho, en el caso de `cwi_214_684` se destaca que la naturaleza de las entradas y las salidas puede tener un tremendo impacto en el tiempo requerido para el cómputo de la reducción inicial, al punto de ver como iocos 75 es llevado a cabo hasta 40 veces más rápido que iocos 50.

En conclusión, hemos sido capaces de implementar en la práctica una herramienta en base al diseño razonado y expuesto en la sección 5.2 para las semánticas de simulación *ready* e iocos, no presentes originalmente en la plataforma **mCRL2**. Los resultados en la mayor parte de los casos revelan unos resultados similares a los de la simulación ordinaria cuando no superiores, y todo ello teniendo en cuenta el procesamiento extra que hay que realizar en las nuevas semánticas, debido a la mayor complejidad que conlleva su definición.

Capítulo 6

Caracterización axiomática de iocos

En los capítulos anteriores hemos conseguido caracterizar la relación de *conformidad* iocos mediante procedimientos que incluían tanto técnicas de *testing* –capítulo 4– como de reducción a problemas conocidos resueltos mediante algoritmos de procesamiento sobre grafos –capítulo 5–. Ambos comparten en común que los modelos de los sistemas a tener en cuenta se formulaban en base a sistemas de transiciones etiquetadas.

En esta última parte de la tesis mostraremos la resolución de la relación de *conformidad* iocos mediante deducción lógica, lo que nos exigirá la formulación de un sencillo cálculo inecuacional. A diferencia de los procedimientos anteriores, necesitaremos formular los sistemas a tener en cuenta mediante un lenguaje de alto nivel de naturaleza algebraica, que nos permitirá manipular con facilidad las expresiones referidas a ellos en el proceso de *deducción* que tenga lugar en cada caso. Algunos sencillos ejemplos de deducción nos servirán para ilustrar el proceso.

El núcleo del capítulo corresponde a la demostración de la corrección y completitud del cálculo presentado. Este último aspecto lo abordaremos en dos fases: en primer lugar, con procesos de dimensión finita, y en segundo, extendiéndolo a procesos infinitos, los cuales basarán sus propiedades en base a la aproximaciones finitas de los mismos.

6.1. Un lenguaje algebraico

6.1.1. Sintaxis

Como hemos visto en los capítulos anteriores, los *sistemas de transiciones etiquetadas* constituyen un modelo operacional que nos permite representar sistemas y razonar sobre posibles relaciones que se puedan dar entre ellos, como es el caso de nuestra relación de conformidad iocos.

Mediante los *sistemas de transiciones etiquetadas* una representación gráfica resulta inmediata al identificar las transiciones con las aristas de un grafo cuyos nodos representan los estados del sistema. A menudo sucede que los sistemas realistas objeto de estudio están formados por millones de estados, de modo que su representación, o la simple enumeración resulta poco factible desde el punto de vista práctico.

La razón por la que en esta sección nos alejamos de un sistema operacional como el tenido en cuenta hasta aquí es que para la deducción lógica –verdadero objeto de estudio de este capítulo– no es posible la formulación de un cálculo sin un lenguaje que permite la cómoda manipulación de los términos que lo componen.

Tradicionalmente nos referimos a estos lenguajes como *lenguajes o álgebras de procesos*, y a lo largo de la literatura ha habido muchos que han arraigado en mayor o menor medida, la mayoría de los cuales aparecen como variantes de tres fundamentales: **CCS**, **CSP** y **ACP** (véase la sección 2.1).

La notación que aquí emplearemos es una simplificación de **CCS**, sin perder nada de su expresividad original, puesto que los operadores ausentes pueden ser definidos en términos de combinaciones presentes. Nos referiremos a ella como **BCCS**, acrónimo de "Basic **CCS**".

Definición 43. Dado un conjunto de acciones $L = I \cup O$, el conjunto de procesos **BCCS** queda definido por la siguiente expresión BNF:

$$p ::= \mathbf{0} \quad | \quad x \quad | \quad ap \quad | \quad p + q \quad | \quad rec\ x.p$$

donde $a \in L = I \cup O$, y $x, y \ldots$ representan variables. Como de costumbre, en el caso de las etiquetas indicaremos mediante los símbolos $!, ?$ la cualidad de las mismas en tanto que símbolos de entrada o salida; de esta manera $a? \in I$ y $o! \in O$.

Al respecto de la gramática generada por esta expresión BNF debemos tener en cuenta dos aspectos que restringen el conjunto de términos válidos:

Términos cerrados No hay variables libres, esto es, todas las apariciones de una variable lo hacen bajo el alcance del operador *rec*.

Variables guardadas Todas las apariciones de variables han de hacerlo precedidas por un prefijo en el árbol sintáctico del término que lo contiene.

De acuerdo con lo anterior, son términos no válidos $x, x+y$ los cuales aparecen como subtérminos en los términos válidos $rec\ x.ax, rec\ x.rec\ y.a(x+y)$

Hechas estas restricciones, notemos que cuando formulemos la semántica operacional será necesario considerar los términos con variables libres.

Intuición de los operadores. Sistemas quiescentes y el proceso nulo

Como viene siendo común en los lenguajes de procesos (capítulo 2), cada operador expresa informalmente los siguientes comportamientos intuitivos:

- el operador prefijo **ap** la ejecución de una acción en secuencia
- $\mathbf{p} + \mathbf{q}$ es el operador de elección de dos procesos dados
- mediante el operador recursivo **rec** nos permite la declaración de procesos infinitos. En efecto, mediante el término

  $$rec\ x.a!x$$

  denotamos el proceso que indefinidamente emite la señal de salida $a!$.
- al respecto del empleo de los símbolos de variable, $x, y \ldots$, éstos carecen de interpretación propia y sólo cobran sentido integradas dentro de una expresión recursiva, como en el caso de la anterior.
- Atención especial requiere el proceso *nulo*, **0**, tradicionalmente identificado en los lenguajes tradicionales con el proceso *inactivo* carente de transiciones. Para simplificar los términos, las apariciones finales del proceso nulo se eliminarán, así en lugar de escribir $a?x!0$ escribiremos tan solo $a?x!$.

Como veremos en la siguiente sección, una de las particularidades que puede distinguir este lenguaje respecto de otros similares [63] es que tanto el proceso inactivo como el secuencial precedidos de una acción de entrada tienen definida una transición implícita correspondiente al comportamiento *quiescente* –ausencia de comportamiento autónomo– sintetizado por el símbolo de transición $\delta!$, lo que facilitará mucho la demostración de algunos teoremas posteriores.

6.1.2. Semántica operacional

El siguiente paso en la descripción de nuestro lenguaje es la formulación de una semántica operacional. En esencia, consiste en describir como evolucionan cada uno de los estados denotados por los términos del lenguaje mediante reglas de transiciones, a la manera de una semántica operacional estructurada [73].

Definición 44. Sea el conjunto de reglas de transciones definido en la figura 6.1. La semántica operacional de los términos de **BCCS** consiste en un sistema de transiciones $LTS = (S, I, O, \rightarrow)$ definido de la siguiente manera:

- El conjunto de estados S está formado por los términos **BCCS**.
- Dados $p, q \in$ **BCCS** y $a \in L$, existe una transición marcada por a, es decir, $p \xrightarrow{a} q$, si una tal transición se puede deducir de las reglas de la figura 6.1

Consistencia del símbolo de quiescencia

Hemos de notar que esta semántica operacional es consistente con las propiedades que en el capítulo 3 requeríamos a los *sistemas de transiciones etiquetadas* respecto de la oportunidad del símbolo de quiescencia $\delta!$, tal como demuestran las siguientes proposiciones:

Lema 11. *Sea $p \in$ **BCCS**. Si p es quiescente, entonces la única transición posible desde el operador $\delta!$ es hacia sí mismo.*

Demostración. Trivial según está definido el conjunto de reglas de la figura 6.1, pues vemos que las transiciones con símbolos $\delta!$ sólo se hacen a procesos similares al original. □

$$\frac{}{0 \xrightarrow{\delta!} 0}$$

$$\frac{}{a?p \xrightarrow{a?} p} \quad \frac{}{a!p \xrightarrow{a!} p} \quad \frac{}{a?p \xrightarrow{\delta!} a?p}$$

$$\frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} a \in L\backslash\{\delta!\} \quad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'} a \in L\backslash\{\delta!\} \quad \frac{p \xrightarrow{\delta!} p, q \xrightarrow{\delta!} q}{p+q \xrightarrow{\delta!} p+q}$$

$$\frac{p \xrightarrow{a} p'}{rec\ x.p \xrightarrow{a} p'[x/rec\ x.p]} a \in L\backslash\{\delta!\} \quad \frac{p \xrightarrow{\delta!} p}{rec\ x.p \xrightarrow{\delta!} rec\ x.p}$$

Figura 6.1: Reglas de la semántica operacional de **BCCS**

Proposición 11. *Sea $p \in$ **BCCS**. p realiza acciones de salida si y solo si no es* quiescente*, es decir,*

$$a! \in O \setminus \{\delta!\}\ p \xrightarrow{a} p' \Leftrightarrow p \not\xrightarrow{\delta!}$$

Demostración. Procederemos por inducción estructural sobre el término $p \in$ **BCCS**.

Base Según se aprecia en la figura 6.1, para el caso en que $p = 0$ no hay definido ningún símbolo de salida distinto de $\delta!$ y sí una transición para $\delta!$, por lo que se cumple trivialmente.

Recursivo Tenemos los siguientes casos:

a?p En la reglas se puede apreciar que no tienen definidas transiciones para símbolos de salida y sí –de manera incondicional– para $\delta!$, por lo que se cumple de manera trivial la proposición.

a!p Inversamente al modo anterior, las reglas registran transiciones para símbolos de de salida y ninguna para $\delta!$.

p+q Consideremos los dos sentidos de la proposición:

- Supongamos que $p + q \xrightarrow{a}$. Entonces, por la semántica operacional esto pasa si $p \xrightarrow{a}$. Por la hipótesis de inducción esto implica que $p \not\xrightarrow{\delta!}$, por lo que, de acuerdo a las reglas, $p+q \not\xrightarrow{\delta!}$.

- Parecido razonamiento si $p+q\overset{\delta!}{\not\longrightarrow}$. Entonces, es claro que o bien $p\overset{\delta!}{\not\longrightarrow}$ o bien $q\overset{\delta!}{\not\longrightarrow}$. Tomando el primer caso, por la hipótesis de inducción tenemos que $p\overset{a}{\longrightarrow}$ para algún símbolo de salida, luego por las reglas se obtiene que $p+q\overset{a}{\longrightarrow}$.

rec x.p Razonando de modo similar al anterior, en los dos sentidos de la proposición tenemos:

- Supongamos que $rec\ x.p\overset{a}{\longrightarrow}$. Según la reglas, esto sólo es posible si $p\overset{a}{\longrightarrow}$. Por la hipótesis de inducción, esto implica que $p\overset{\delta!}{\not\longrightarrow}$ y por las reglas se tiene que $rec\ x.p\overset{\delta!}{\not\longrightarrow}$.
- Si se tiene que $rec\ x.p\overset{\delta!}{\not\longrightarrow}$ esto ocurre porque $p\overset{\delta!}{\not\longrightarrow}$ según las reglas, y por la hipótesis de inducción $p\overset{a}{\longrightarrow}$. Entonces, por las reglas se tiene que $rec\ x.p\overset{a}{\longrightarrow}$

□

Corolario Hemos obtenido un conjunto de términos coherentes conforme a las condiciones que exigíamos a los *sistemas de transiciones etiquetadas* del capítulo 3.

Con ello es posible una extensión natural de la definición 22 adaptándola al nuevo dominio de los términos de **BCCS**, sin más que actualizar adecuadamente los operadores que sean necesarios. La semántica operacional definida sobre los términos sintácticos nos induce un sistema de transiciones etiquetadas; por tanto –acorde con la definición 4– tenemos definidas las funciones $\mathsf{ins}(p)$ y $\mathsf{outs}(p)$ para cada $p \in \mathbf{BCCS}$.

No obstante esas funciones se pueden definir de forma recursiva a partir de los términos sintácticos de la siguiente forma:

$$\mathsf{ins}(p) = \begin{cases} \varnothing & \text{si } p = 0 \\ a & \text{si } p = ap' \wedge a \in I \\ \varnothing & \text{si } p = ap' \wedge a \notin I \\ \mathsf{ins}(q) \cup \mathsf{ins}(r) & \text{si } p = q + r \\ \mathsf{ins}(p_1) & \text{si } p = rec\ x.p_1 \end{cases}$$

$$\mathsf{outs}(p) = \begin{cases} \varnothing & \text{si } p = 0 \\ a & \text{si } p = ap' \wedge a \in O \\ \varnothing & \text{si } p = ap' \wedge a \notin O \\ \mathsf{outs}(q) \cup \mathsf{outs}(r) & \text{si } p = q + r \\ \mathsf{outs}(p_1) & \text{si } p = rec\ x.p_1 \end{cases}$$

Mediante una simple inducción estructural es fácil comprobar que esta última definición es equivalente a la referida en la definición 4.

Por otro lado, la introducción del nuevo dominio **BCCS** en la relación iocos adquiere una especial relevancia a la hora de considerar una propiedad, la precongruencia, que resultará fundamental para establecer la corrección y completitud del cálculo que presentaremos en la sección 6.2. Este es el objeto del siguiente apartado.

6.1.3. Precongruencia del lenguaje

Demostrar la *precongruencia* de iocos respecto a los operadores equivale a afirmar que la relación iocos se mantiene sobre dos términos al aplicarles cualesquiera de los operadores del lenguaje. Técnicamente esto quiere decir que las referencias a la definición 22 serán muy frecuentes, por lo que adoptaremos el siguiente convenio a la hora de referirnos a sus cláusulas: por cláusula 22.1 entenderemos que nos referimos a la cláusula 1 de la defínción 22, cláusula 22.2 para referirse a cláusula 2 de la defínción 22, etc.

El operador prefijo

En el lema 12 exponemos que si dos sistemas están *a priori* en relación iocos entonces los sistemas que se obtienen al prefijarlos con una acción mediante el operador prefijo continúan estándolo también.

Lema 12. *Si* $p\ \mathsf{iocos}\ q$ *entonces* $ap\ \mathsf{iocos}\ aq$*, con* $a \in I \cup O$*.*

Demostración. Sea el siguiente conjunto

$$R = \{(ap, aq) \mid p, q \in \mathsf{iocos}\}$$

El objeto es demostrar que el conjunto

$$R \cup \mathsf{iocos}$$

es una iocos-simulación a la que pertence el par (ap, aq), por lo que, por la definición 23 estarán en relación iocos.

Pueden darse dos casos, dependiendo si se trata de un símbolo de salida o de entrada.

$a! \in O$ Tratándose de símbolo de salida, incluyendo la *quiescencia*, tenemos:

- $\mathsf{ins}(x!p) = \varnothing \supseteq \varnothing = \mathsf{ins}(x!q)$, luego la cláusula 22.1 se cumple.
- La cláusula 22.2 también, puesto que no hay acciones de entrada que considerar.
- Vemos que $outs(x!p) = \{x!\}$, luego si $x!p \xrightarrow{x!} p$ entonces $x!q \xrightarrow{x!} q$, por la semántica operacional.

  Como $(p, q) \in \mathsf{iocos}$ por hipótesis, tenemos que $(p, a) \in R \cup \mathsf{iocos}$, luego 22.3 se cumple.

$a \in I$ Si $b?$ es un símbolo de salida entonces

- $\mathsf{ins}(b?p) = \{b?\} \supseteq \{b?\} = \mathsf{ins}(b?q)$, luego la cláusula 22.1 se cumple.
- Como $b? \in ins(b?q)$ se tiene que $b?p \xrightarrow{b?} p$ y $b?q \xrightarrow{b?} q$, por la semántica operacional.

  Como por hipótesis tenemos que $(p, q) \in \mathsf{iocos}$ tenemos que tenemos que $(p, q) \in R \cup \mathsf{iocos}$, luego se cumple la cláusula 22.2.
- Al no haber salidas que valorar, trivialmente se cumple la cláusula 22.3

Hemos demostrado que la relación $R \cup \mathsf{iocos}$ es una iocos relación, por lo que cabe afirmar que $ap\ \mathsf{iocos}\ aq$. □

El operador suma

A diferencia del operado de prefijo, el operador de suma no es congruente con respecto al preorden iocos en general. El siguiente contraejemplo muestra este hecho.

Ejemplo 1. *Consideremos* $p = a?0 + b?0$ *y* $q = b?0$. *Tenemos que* p $\mathsf{iocos}\underline{\leq}$ q *debido a que* p *puede añadir acciones de entrada no especificadas o libres.*

Supongamos ahora $r = a?x!$. *Tenemos*

$$p + r \not\mathsf{ioco}\acute{\mathsf{s}}\, q + r$$

debido a que la acción $a?$ *ahora sí está especificada en* $q + r$, $a?x!$, *y por tanto no hay margen para* $p + r$ *para poder realizar* $a?0$, *pues la acción* $a?$ *ha dejado de ser libre y tiene un comportamiento limitado por* $a?x!$.

Sin embargo este problema se puede solucionar estudiando en profundidad las causas que rompen la congruencia del operador $+$: éstas quedan reflejadas en el ejemplo anterior.

En efecto, si una acción estaba *libre* en una implementación, esa acción no puede quedar *ligada* en el comportamiento que se añade mediante el operador $+$. En el ejemplo anterior la $a?$ que aparece en p no estaba a sujeto a lo que indicaba la especificación q. Sin embargo al añadirle el comportamiento de r esto ya no se puede sostener.

Hay dos formas de asegurar que el operador $+$ se comporte bien con respecto a la relación $\mathsf{iocos}\underline{\leq}$ que son las que a aparece en el lema 13. Intuitivamente lo que significan esas condiciones son:

- Que el comportamiento de la implementación original (p en el ejemplo anterior) no presente acciones de entrada no contempladas por la especificación (respectivamente q).
- Que las entradas libres de la implementación (la acción $a?$ de p en el ejemplo anterior) no se vean comprometidas–interferidas por las acciones de entrada del comportamiento que se añade (respectivamente r).

El resultado de la anterior disquisición queda recogido en el siguiente lema:

Lema 13. *Sea* $p, q, r \in \boldsymbol{BCCS}$, *con* p $\mathsf{iocos}\underline{\leq}$ q. *Entonces*

$$p + r \;\mathsf{iocos}\underline{\leq}\; q + r$$

si se dan una de las dos circunstancias siguientes:

- $\mathsf{ins}(p) \subseteq \mathsf{ins}(q)$

- $\mathsf{ins}(r) \cap \mathsf{ins}(p) = \varnothing$

Demostración. De modo análogo al lema anterior, definiendo

$$R = \{(p + r, q + r) \mid (p, q) \in \mathsf{iocos}\}$$

tenemos que valorar que la relación

$$R \cup \mathsf{iocos}$$

es una iocos-simulación a la que pertence el par $(p + r, q + r)$, por lo que, por la definición 23 estarán en relación iocos.

Valoremos si se cumplen las cláusulas de la definición 22.

- Ya que $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$ por hipótesis, se tiene que $\mathsf{ins}(q + r) = \mathsf{ins}(q) \cup \mathsf{ins}(r) \subseteq \mathsf{ins}(p) \cup \mathsf{ins}(r)$, luego la cláusula 22.1 se cumple.

- Supongamos que $p + r \xrightarrow{a?} p'$, con $a \in I$. Entonces, o bien $p \xrightarrow{a?} p'$ o bien $r \xrightarrow{a?} p'$, por la semántica operacional.

  Sabemos que $a? \in \mathsf{ins}(q+r)$, de lo contrario carece de interés la transición.

  - En el caso en que $r \xrightarrow{a?} p'$, $q+r \xrightarrow{a?} p'$, por la semántica operacional y $(p', p') \in \mathsf{iocos}$, ya que iocos es un preorden, y en consecuencia $(p', p') \in R \cup \mathsf{iocos}$
  - Si $p \xrightarrow{a?} p'$ entonces urge demostrar que $a \in \mathsf{ins}(q)$, lo cual sucede de modo trivial si $\mathsf{ins}(p) \subseteq \mathsf{ins}(q)$. Si no sucede esto, ya que $a \in \mathsf{ins}(p)$, sucede que $a \notin \mathsf{ins}(r)$, puesto que $\mathsf{ins}(r) \cap \mathsf{ins}(p) = \varnothing$, y dado que $a? \in \mathsf{ins}(q + r) = \mathsf{ins}(q) \cup \mathsf{ins}(r)$, necesariamente $a \in \mathsf{ins}(q)$.
    Por tanto, existe q' tal que $q \xrightarrow{a?} q'$ y, como por hipótesis $p \,\mathsf{iocos}\, q$, sucede que $(p', q') \in \mathsf{iocos}$ y por tanto $(p', q') \in R \cup \mathsf{iocos}$

  Por todo lo anterior se puede afirmar que la cláusula 22.2 se cumple.

- Considerar el caso de las salidas es mucho más sencillo que las entradas. En efecto la cláusula 22.3 se cumple ya que:

  - En el caso en que $r \xrightarrow{o!} p'$, se procede del mismo modo, ya que $q + r \xrightarrow{o!} p'$ y $(p', p') \in \mathsf{iocos}$.

- Si $p \xrightarrow{o!} p'$, como por hipótesis $p \,\mathsf{iocos}_{\underline{s}}\, q$, tenemos que $q \xrightarrow{o!} q'$, con $(p', q') \in \mathsf{iocos}_{\underline{s}}$.

□

Por último, es conveniente notar que la equivalencia inducida por $\mathsf{iocos}_{\underline{s}}$ –el *kernel* del preorden– implica que el conjunto de acciones de entrada de los procesos relacionados son iguales. Como consecuencia los procesos relacionados cumplen la primera de las condiciones mencionadas del lema 13, haciendo que el operador + sea una congruencia para la relación de equivalencia inducida por $\mathsf{iocos}_{\underline{s}}$. Esto queda recogido en el siguiente lema:

Lema 14. *Sea $p, q, r \in \boldsymbol{BCCS}$, con $p \equiv_{\mathsf{iocos}_{\underline{s}}} q$. Entonces $p + r \equiv_{\mathsf{iocos}_{\underline{s}}} q + r$.*

Demostración. Puesto que $p \equiv_{\mathsf{iocos}_{\underline{s}}} q$ tenemos $\mathsf{ins}(p) = \mathsf{ins}(q)$ y en particular $\mathsf{ins}(p) \subseteq \mathsf{ins}(q)$ y $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$. Por tanto aplicando el lemma 13 tenemos $p+r \,\mathsf{iocos}_{\underline{s}}\, q+r$ y $q+r \,\mathsf{iocos}_{\underline{s}}\, p+r$, lo que por definición supone $p+r \equiv_{\mathsf{iocos}_{\underline{s}}} q+r$. □

6.2. Un cálculo para iocos

Aunque el objeto de este capítulo es la presentación de un cálculo para la relación de conformidad de $\mathsf{iocos}_{\underline{s}}$, una motivación de la misma nos llevará a introducir algunos conceptos generales relacionados con la semántica denotacional que sólo de manera tangencial se usarán para motivar el empleo de la lógica, verdadero fin de la semántica axiomática que queremos introducir.

Especificaciones algebraicas

Al definir un conjunto de operaciones sobre un tipo de valores lo que obtenemos propiamente es un *álgebra*. En el caso de las *álgebras de procesos*, todas estas operaciones vienen dadas por la interpretación que damos a los símbolos de la *signatura* que hemos expuesto en la definición 43.

Sucede sin embargo que para cada signatura propuesta, en general existen muchas álgebras que son puedan ser interpretación de ella. Si tenemos en cuenta que cada valor es interpretación de un término legalmente formado con los símbolos de la signatura, una manera de diferenciar *álgebras* es estudiar su

conjunto cociente, esto es, las clases de *equivalencia* que resultan de considerar aquellos términos que, siendo diferentes, denotan un mismo valor.

Para determinar estas clases de equivalencia se emplean *especificaciones* en lógica ecuacional, proponiendo igualdades entre términos que denotan valores en el álgebra. Cuanto más axiomas se proponen más términos quedan igualados y el número de clases de equivalencia disminuye, siendo el caso extremo aquel en que, sin axiomas o ecuaciones, cada término por si mismo denota un valor, y por tanto, una clase de equivalencia. Como referencia, en el capítulo 2 hablábamos de la *semántica axiomática* como medio de dotar de significado a los términos en el lenguaje de álgebra **ACP**.

Cada forma de interpretar un álgebra denota una semántica, y en el contexto concreto que nos ocupa, las *álgebras de procesos*, la *bisimulación* comprende una de las semánticas más restrictivas que se puedan obtener a la hora de estimar iguales dos procesos, puesto que prácticamente se limita a considerar pequeñas variaciones en el orden de los operandos y el concurso de la idempotencia en los mismos.

Cálculo inecuacional

En ocasiones, como la que nos concierne, no estamos interesados en la especificación de equivalencias, sino en la formulación de relaciones de preorden, de tal forma que su equivalencia quede definida de manera indirecta al considerar ésta inducida por un preorden y su relación inversa. Es por ello que introducimos el símbolo $\sqsubseteq$ en torno al que se definen las fórmulas que integran el marco de la lógica *inecuacional*. De esta manera, mediante la fórmula

$$a?p \sqsubseteq a?p + a?q$$

lo que realmente queremos dar a entender es que el sistema de lado izquierdo está puesto en relación iocos con el sistema del de la derecha.

Al emplear este símbolo en lugar de la forma hasta ahora utilizada, el símbolo iocos en forma infija, se destaca que la fórmula afirma es obtenido mediante un procedimiento de *deducción*, el cual a la postre habrá que demostrar correcto y completo a fin de ver que tal "sustitución" no se hace de modo gratuito.

Como es conocido, el símbolo = se emplea para denotar igualdades. Éstas pueden en realidad considerarse como la conjunción de dos inecuaciones, la segunda obtenida cambiando los términos derecho e izquierdo de la igualdad. Por ejemplo el axioma de la asociatividad

$$p + (q + r) = (p + q) + r$$

en realidad está indicando que tenemos dos axiomas:

$$p + (q + r) \sqsubseteq (p + q) + r$$
$$(p + q) + r \sqsubseteq p + (q + r)$$

En nuestro intento por dotar de un cálculo axiomático a nuestra relación iocos, haremos una exposición progresiva en tres fases:

- En la primera, se exponen los axiomas que caracterizan desde el punto de vista axiomático
- A continuación, veremos algunos ejemplos para ilustrar el cálculo.
- Por último demostraremos que el cálculo es correcto y completo.

6.2.1. Reglas del cálculo

Como se apuntaba con anterioridad, la figura 6.2 sintetiza una serie de propiedades muy comunes a otros preórdenes semánticos. Para demostrar su corrección nos remontaremos algunos lemas y teoremas ya expuestos con anterioridad (lemas 1 12, 13 y 14).

Aunque por sí mismas no constituyen un conjunto completo, estas reglas resultan necesarias para poder demostrar la corrección y completitud de aquellas que son propias de la relación que estamos considerando, iocos, las cuales, como se ha dicho, se mostrarán en los apartados 6.3.1 y 6.3.2.

- Un primer conjunto, (P1) y (P2), servirá para reflejar el carácter de la relación iocos como *preorden*, capaz de articular bajo el cálculo las propiedades reflexiva y transitiva.

$$p \sqsubseteq p \tag{P1}$$

$$\frac{p \sqsubseteq q \;\; q \sqsubseteq r}{p \sqsubseteq r} \tag{P2}$$

$$\frac{p \sqsubseteq q}{ap \sqsubseteq aq} \tag{P3}$$

$$\frac{p \sqsubseteq q, \mathsf{ins}(p) \subseteq \mathsf{ins}(q)}{p + r \sqsubseteq q + r} \tag{P4a}$$

$$\frac{p \sqsubseteq q, \mathsf{ins}(p) \cap \mathsf{ins}(r) = \varnothing}{p + r \sqsubseteq q + r} \tag{P4b}$$

$$\frac{p = q}{p + r = q + r} \tag{P5}$$

Figura 6.2: Reglas de preorden y congruencia

- El segundo conjunto, (P3), (P4a) y (P4b) pretende caracterizar la relación $\underline{\mathsf{iocos}}$ como *precongruencia* ante algunos operadores del lenguaje en el proceso de deducción.

La figura 6.3 expresa mediante axiomas todas aquellas propiedades propias y características de la naturaleza coinductiva de la relación a considerar, $\underline{\mathsf{iocos}}$.

Un primer grupo, que incluye (B1), (B2), (B3), (B4), (I), (II) y (III) se refiere a procesos de naturaleza finita, y es el que ocupa esta sección. Por último, la regla (IV) captura el comportamiento de los procesos infinitos.

A continuación damos una breve justificación intuitiva de lo que cada axioma quiere expresar:

- Mediante el axioma (B1) declaramos la *asociatividad* o más concretamente, la posibilidad de declarar indistinguible mediante $\underline{\mathsf{iocos}}$ la elección de más de dos procesos a la vez.

$$p + (q + r) = (p + q) + r \tag{B1}$$

$$p + q = q + p \tag{B2}$$

$$p + p = p \tag{B3}$$

$$p + 0 = p \tag{B4}$$

$$a!p \sqsubseteq a!p + b!q \tag{I}$$

$$a?p \sqsubseteq 0 \tag{II}$$

$$a?p \sqsubseteq a?p + a?q \tag{III}$$

$$\frac{\forall n \in \mathbb{N},\ p \downarrow_n \sqsubseteq q \downarrow_n}{p \sqsubseteq q} \tag{IV}$$

Figura 6.3: Axiomas del cálculo para iocos

- El axioma (B2), conocida como *conmutativa*, declara indistinguible el orden de dos procesos cualesquiera en la elección.
- Por la *idempotencia*, axioma (B3), podemos simplificar un sistema puesto en elección consigo mismo eliminando repeticiones.
- Mediante la axioma (B4) elección de un proceso con el proceso nulo es indistinguible del proceso mismo.
- La axioma (I) se corresponde con la necesidad por la relación iocos de limitar los comportamientos autónomos de una eventual implementación a los marcados por una especificación, tal como marcaba la cláusula 22.3.

- La axioma (II) marca la libertad por parte de un sistema para implementar comportamientos reactivos no indicados en una especificación; tal el era el objetivo de la cláusula 22.1 de la definición22 de iocos.

- La axioma (III) nos indica que no es necesario implementar *todos* los comportamientos reactivos marcados por una especificación relativos a una señal particular de entrada, pero sí al menos uno. Este era el objeto de la cláusula 22.2 de la definición 22 de iocos.

- Por último, la regla (IV) nos indica que para poder relacionar dos procesos infinitos debemos comparar todas sus aproximaciones finitas. Intuitivamente una aproximación finita de profundidad n se consigue expandiendo el operador de recursión todo lo que sea necesario para luego aplicar una poda de forma que no haya trazas de longitud mayor que n.

Definición 45. Sean $p, q \in \mathbf{BCCS}$, decimos que $\vdash p \sqsubseteq q$ si y solo si $p \sqsubseteq q$ se puede deducir en un número finito de pasos a partir de las reglas y axiomas que aparecen en las figuras 6.2 y 6.3.

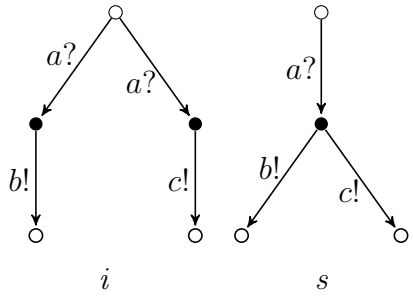
6.2.2. Ejemplos de deducción

A continuación ilustraremos la capacidad del cálculo para deducir la relación existente entre dos procesos descritos mediante el lenguaje algebraico; para apoyar la visión intuitiva, daremos la representación visual de los procesos puestos en conexión, los cuales han aparecido en capítulos anteriores.

Al tratarse de un instrumento formal, la forma de las demostraciones adoptará la de secuencia de fórmulas, donde cada una de ellas, o bien es un axioma o bien se deriva mediante un proceso de inferencia lógica a partir de las fórmulas anteriores.

Ejemplo 2.
Implementación indeterminista con respecto a señal de entrada $a?$ y comportamiento posterior limitado en salida.

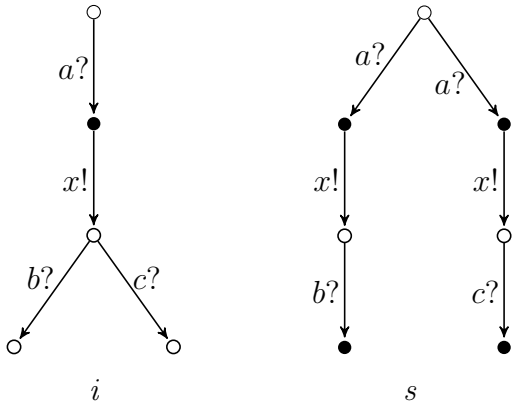
$$\vdash a?b! + a?c! \sqsubseteq a?(b! + c!)$$



$$\vdash b! \sqsubseteq b! + c! \quad \text{(I)}$$
$$\vdash c! \sqsubseteq b! + c! \quad \text{(I)}$$
$$\vdash a?b! \sqsubseteq a?(b! + c!) \quad \text{(P3)}$$
$$\vdash a?c! \sqsubseteq a?(b! + c!) \quad \text{(P3)}$$
$$\vdash a?b! + a?c \sqsubseteq a?(b! + c!) + a?c \quad \text{(P4a)}$$
$$\vdash a?(b! + c!) + a?c \sqsubseteq a?(b! + c!) + a?(b! + c!) \quad \text{(P4a)}$$
$$\vdash a?(b! + c!) + a?(b! + c!) = a?(b! + c!) \quad \text{(B3)}$$
$$\vdash a?b! + a?c \sqsubseteq a?(b! + c!) \quad \text{(P2)}$$

Ejemplo 3.
Efecto de especificación indeterminista e implementación determinista con libertad para reaccionar ante señales no especificadas.

$$a?x!(b? + c?) \sqsubseteq a?x!b? + a?x!c?$$

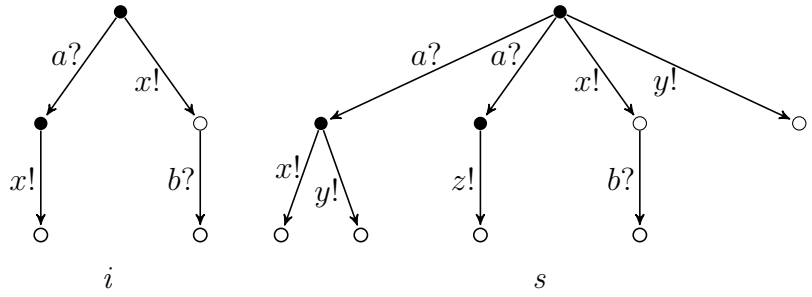


$$\vdash b? \sqsubseteq b? \quad \text{(P1)}$$
$$\vdash c? \sqsubseteq 0 \quad \text{(II)}$$
$$\vdash b? + c? \sqsubseteq b? + 0 \quad \text{(P4a)}$$
$$\vdash b? + 0 \sqsubseteq b? \quad \text{(B4)}$$
$$\vdash b? + c? \sqsubseteq b? \quad \text{(P2)}$$
$$\vdash c? + b? \sqsubseteq c? + 0 \quad \text{(P4a)}$$

$$\vdash c? + 0 \sqsubseteq c? \quad \text{(B4)}$$
$$\vdash b? + c? \sqsubseteq c? \quad \text{(P2)}$$
$$\vdash a? \sqsubseteq a? \quad \text{(P1)}$$
$$\vdash x! \sqsubseteq x! \quad \text{(P1)}$$
$$\vdash x!(b? + c?) \sqsubseteq x!b? \quad \text{(P3)}$$
$$\vdash a?x!(b? + c?) \sqsubseteq a?x!b? \quad \text{(P3)}$$
$$\vdash x!(b? + c?) \sqsubseteq x!c? \quad \text{(P3)}$$
$$\vdash a?x!(b? + c?) \sqsubseteq a?x!c? \quad \text{(P3)}$$
$$\vdash a?x!(b? + c?) + a?x!(b? + c?) \sqsubseteq a?x!(b? + c?) + a?x!c? \quad \text{(P4a)}$$
$$\vdash a?x!(b? + c?) + a?x!c? \sqsubseteq a?x!b? + a?x!c? \quad \text{(P4a)}$$
$$\vdash a?x!(b? + c?) + a?x!(b? + c?) \sqsubseteq a?x!b? + a?x!c? \quad \text{(P2)}$$
$$\vdash a?x!(b? + c?) \sqsubseteq a?x!(b? + c?) + a?x!(b? + c?) \quad \text{(B3)}$$
$$\vdash a?x!(b? + c?) \sqsubseteq a?x!b? + a?x!c? \quad \text{(P2)}$$

Ejemplo 4.
Especificación indeterminista en señal $a?$ *e implementación* obligada *a desarrollar al menos de las opciones*

$$a?x! + x!b? \sqsubseteq a?(x! + y!) + a?z! + x!b? + y!$$



$\vdash x! \sqsubseteq x! + y!$ (I)

$\vdash x!b? \sqsubseteq x!b?$ (P1)

$\vdash a?x! \sqsubseteq a?(x! + y!)$ (P3)

$\vdash a?x! \sqsubseteq a?(x! + y!) + a?z!$ (III)

$\vdash a?x! + x!b? \sqsubseteq a?(x! + y!) + a?z! + x!b?$ (P4a)

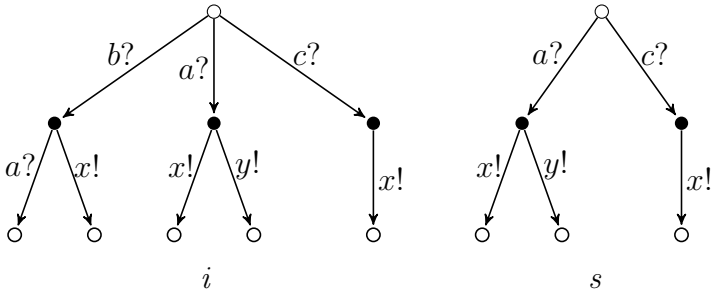
$\vdash a?(x! + y!) + a?z! + x!b? \sqsubseteq a?(x! + y!) + a?z! + x!b? + y!$ (P4a)

$\vdash a?x! + x!b? \sqsubseteq a?(x! + y!) + a?z! + x!b? + y!$ (P2)

Ejemplo 5.

Implementación con margen para implementar comportamientos reactivos no presentes en la especificación

$$b?(a? + x!) + a?(x! + y!) + c?x! \sqsubseteq a?(x! + y!) + c?x!$$



$\vdash b?(a? + x!) \sqsubseteq 0$ (II)

$\vdash a?(x! + y!) + c?x! + 0 \sqsubseteq a?(x! + y!) + c?x!$ (B4)

$\vdash a?(x! + y!) + c?x! + b?(a? + x!) \sqsubseteq a?(x! + y!) + c?x! + 0$ (P4b)

$\vdash a?(x! + y!) + c?x! + b?(a? + x!) \sqsubseteq a?(x! + y!) + c?x!$ (P2)

6.3. Corrección y Completitud del cálculo

En todas las fases nuestro objetivo será la vinculación de la noción semántica de iocos tal como aparece en la definición 22 con los cálculos de las figuras 6.3 y 6.2, recurriendo a dos conceptos tradicionales a la hora de estudiar un cálculo lógico: *corrección* y *completitud*.

Corrección Al obtener una desigualdad a partir de las reglas de inferencia, necesariamente debe cumplirse que los sistemas designados por los términos en consideración están en relación iocos.

Completitud Partiendo de dos procesos en relación iocos, ha de ser posible hallar una fórmula que lo acredite mediante un proceso deductivo a partir de los axiomas expresados en las figuras 6.2,6.3.

En síntesis, nuestro objetivo es conectar las nociones del capítulo 3 con el cálculo que vamos a desarrollar, lo que expresado de modo más formal podemos formular del siguiente modo:

$$p \;\mathsf{iocos}\; q \text{ si y sólo si } \vdash p \sqsubseteq q.$$

En primer lugar demostraremos la corrección de los axiomas correspondientes a la congruencia de los operadores.

Lema 15. *El conjunto de reglas* (P1), (P2) *es correcto con respecto a la relación* iocos.

Demostración. Por el lema 1 del capítulo 3, que asegura que la relación iocos es un *preorden*, y por tanto cumple las propiedades reflexiva y transitiva. □

Lema 16. *La regla* (P3)*, de la precongruencia del operador prefijo, es correcta con respecto a la relación* iocos.

Demostración. Por el lema 12, el cual demuestra que dos sistemas p y q en relación iocos continúan estándolo cuando se les antepone la misma acción $a \in L$. □

Lema 17. *Las reglas* (P4a) *y* (P4b)*, relativas al operador de elección, son correctas con respecto a la relación* iocos.

Demostración. Por el lema 13 que expresa la congruencia del operador + con respecto a la relación iocos. Nótese que la segunda premisa del axioma (P4a), $\mathsf{ins}(p) \subseteq \mathsf{ins}(q)$, coincide con la primera de las condiciones que exige el lema 13 para su cumplimiento; de modo similar, la segunda premisa del axioma (P4b), $\mathsf{ins}(p) \cap \mathsf{ins}(r) = \varnothing$, es la segunda de las condiciones de dicho lema. □

Como señalábamos con anterioridad, el caso de la regla (P5) supone una excepción a todo lo que expresa el resto, ya que su contenido se refiere a toda una relación de equivalencia, $\equiv_{\mathsf{iocos}}$, el *kernel* inducido por el preorden iocos.

Lema 18. *La regla* (P5) *es correcto respecto de la relacción* $\equiv_{\mathsf{iocos}}$

Demostración. De $p = q$ se sigue lo siguiente:

- $\mathsf{ins}(p) = \mathsf{ins}(q)$, y más concretamente, $\mathsf{ins}(p) \subseteq \mathsf{ins}(q)$
- $p \sqsubseteq q$

En virtud de la regla (P4a) podemos deducir que $p + q \sqsubseteq q + r$. Invocando el mismo razonamiento cambiando p por q llegamos a la conclusión de que $q + r \sqsubseteq p + r$. Por el lema 17 esto equivale a decir que $p + q \;\mathsf{iocos}\; q + r$ y $q + r \;\mathsf{iocos}\; p + r$, o lo que es lo mismo, $p + q \equiv_{\mathsf{iocos}} q + r$. □

6.3.1. Procesos finitos

En un primer paso demostraremos la corrección y completitud del cálculo para procesos finitos. Un proceso finito es aquel en el que no aparece el operador *rec*.

Corrección

Las siguientes proposiciones tienen por objetivo demostrar que cada una de las reglas a las que se refieren son correctas. En conjunto, todas ellas sirven al teorema 12 que garantiza que el cálculo presentado es correcto siempre en lo relativo a procesos de dimensión finita.

Proposición 12. *Las reglas* (B1), (B2), (B3) *referidas a las propiedades asociativa, conmutativa e idempotencia respectivamente son correctas.*

Demostración. En todas ellas se procede de un modo similar. Escogemos el caso (B1) en particular.

Sea $s = p+(q+r)$ y $t = (p+q)+r$, siendo $p, q, r \in \mathbf{BCCS}$. Lo que en realidad vamos a demostrar es el desdoblamiento de (B1) en sus dos inecuaciones $s \sqsubseteq t$ y $t \sqsubseteq s$. Abordaremos únicamente la primera, ya que el razonamiento para la segunda es análogo.

Obsérvese que

$$s \xrightarrow{a} s' \Leftrightarrow t \xrightarrow{a} s' \tag{6.1}$$

ya que, si $s \xrightarrow{a} s'$, entonces por la semántica operacional se tiene que o $p \xrightarrow{a} s'$, o o $q + r \xrightarrow{a} s'$. En este último caso $q \xrightarrow{a} s'$ o $r \xrightarrow{a} s'$. Por la misma razón, que o bien $p + r \xrightarrow{a} s'$ o $r \xrightarrow{a} s'$, lo que en definitiva implica que $t \xrightarrow{a} s'$.

La condición 6.1 expresa una condición más fuerte que la que pone en relación dos procesos iocos, puesto que de esta condición se deriva que $\mathsf{ins}(s) = \mathsf{ins}(t)$, y puesto que $t \xrightarrow{a} s'$, sucede que $(s', s') \in \mathsf{iocos}$ al tratarse de un preorden con la propiedad reflexiva. □

Proposición 13. *La regla* (B4)*, marcando un proceso indistinguible al ser elegido con el proceso nulo, es correcta.*

Demostración. Al igual que en al proposición 12, indicaremos la demostración para una de las dos igualdades necesarias $x + 0 \sqsubseteq x$. La recíproca se obtiene con un planteamiento parecido.

En principio, formulamos el siguiente conjunto

$$R = \{(x + 0, x)\} \cup \{(x, x)\}$$

y demostramos que efectivamente, es una iocos-simulación con

- El proceso 0 no aporta nuevos símbolos de entrada, por lo que $\mathsf{ins}(x+0) = \mathsf{ins}(x)$.

- Supongamos que $x+0 \xrightarrow{\delta!} x'$. En este caso, por la semántica operacional y las proposiciones relativas a la *quiescencia* $x' = x + 0$ y $x \xrightarrow{\delta!} x$ y $0 \xrightarrow{\delta!} 0$, por lo que de nuevo el par $(x+0, x)$ está incluido en el conjunto R.

- En el caso en que $x + 0 \xrightarrow{a} x'$, con $a \neq \delta!$. Por la semántica operacional esto sólo tiene lugar si $x \xrightarrow{a} x'$, y el par (x', x') pertenece a la relación R por la forma en que está construido.

El razonamiento se extiende al conjunto $\{(x', x')\}$, al tratarse iocos de un preorden, por lo que R forma una iocos-relación con $(x + 0, x) \in R$. En consecuencia, $x + 0 \,\mathsf{iocos}\, x$. □

Las reglas (B1), (B2), (B3) y (B4) bien podrían considerarse comunes a cualquier tipo de relación coinductiva. El grupo siguiente es el que distingue con propiedad nuestro relación iocos.

Proposición 14. *La regla* (I)*, referida a los límites del comportamiento reactivo de un sistema, es correcta.*

Demostración. Consideremos las cláusulas de la definición 22 de iocos y la relación definida

$$R = \{(a!p, a!p + b!q)\} \cup \{(x, x)\}$$

- Tenemos que $\mathsf{ins}(a!p) = \varnothing = \mathsf{ins}(a!p + b!q)$, luego se cumple la cláusula 22.1.
- Al no tener acciones de entrada por considerar, se cumple trivialmente la cláusula 22.2.
- Para las acciones de salida, cuando $a!p \xrightarrow{a!} p$, tenemos, por la semántica operacional que $a!p + b!q \xrightarrow{a!} p$, y el par (p, p) pertenece a la relación R por la forma en que está construido. Luego se cumple la cláusula 22.3.

El razonamiento se extiende al conjunto $\{(x, x)\}$ por ser iocos un preorden. En consecuencia R es una iocos-simulación con $(a!p, a!p + b!q) \in R$. De todo esto se sigue que $a!p \,\mathsf{iocos}\, a!p + b!q$. □

Proposición 15. *La regla* (II)*, referida a la libertad de acciones reactivas no especificadas, es correcta.*

Demostración. Consideremos las cláusulas de la definición 22 de iocos y la relación definida

$$R = \{(a?p, 0)\}$$

- Obsérvese que $\mathsf{ins}(a?p) \supseteq \varnothing = \mathsf{ins}(0)$, por lo que se cumple la cláusula 22.1.

- Por lo que se refiere a la transición de entrada $a?p \xrightarrow{a?} p$, ocurre que $a \notin \mathsf{ins}(0)$, por lo que no se tiene en cuenta, y trivialmente se cumple 22.2.

- Como el proceso $a?p$ no tiene transiciones de salida, a excepción de la quiescencia $\delta!$, sólo puede ocurrir que $a?p \xrightarrow{\delta!} a?p$ y $0 \xrightarrow{\delta!} 0$ y trivialmente se cumple la cláusula 22.3.

Por tanto R es una iocos simulación con $(a?p, 0) \in R$, luego $a?p \,\mathsf{iocos}\, \mathbf{0}$. □

Proposición 16. *La regla* (III)*, referente a las acciones reactivas especificadas, es correcta.*

Demostración. Consideremos la relación

$$R = \{(a?p, a?p + a?q)\} \cup \{(x, x)\}$$

y las cláusulas de la definición 22 de iocos.

- Se tiene que $\mathsf{ins}(a?p+a?q) = \{a\} \subseteq \{a\} = \mathsf{ins}(a?p)$, luego 22.1 se cumple.

- Con respecto a 22.2, tenemos que $a?p \xrightarrow{a?} p$ así como $a?p + a?q \xrightarrow{a?} p$ según la semántica operacional, y $(p, p) \in R$, por el modo en que se ha construido R.

- Puesto que, a excepción del símbolo de quiescencia, no hay acciones de salida, se tiene que $a?p \xrightarrow{\delta!} a?p$ y $a?p + a?q \xrightarrow{\delta!} a?p + a?q$, por lo que 22.3 se cumple trivialmente.

En consecuencia, R es una iocos-simulación que contiene al par $(a?p, a?p+a?q)$, luego $a?p \,\mathsf{iocos}\, a?p + a?q$. □

A la vista de las anteriores proposiciones, tomando siempre en cuenta procesos de naturaleza finita, cualquier resultado obtenido por inferencia a partir de los axiomas parece ser correcto con respecto a la relación iocos. Esto es lo que viene sintetizado por el siguiente teorema de corrección del cálculo.

Teorema 12. . *Sean $p, q \in \boldsymbol{BCCS}$ dos procesos de naturaleza finita. Entonces*

$$Si \ \vdash p \sqsubseteq q \ entonces \ p \,\mathsf{iocos}\underline{\mathsf{s}}\, q$$

Demostración. Dado que las reglas son correctas según las anteriores proposiciones, la demostración se hace trivialmente por inducción sobre los pasos de la deducción. □

Completitud

Una cuestión distinta es saber si el espacio de fórmulas obtenido por deducción mediante reglas en su interpretación como relaciones $\mathsf{iocos}\underline{\mathsf{s}}$ coincide con éste último precisamente.

El teorema 13, notablemente más sofisticado, demuestra la *completitud* del cálculo expuesto para procesos finitos. No obstante, antes de abordarlo, nos permitiremos definir una suerte de forma *prenormal* cuyo propósito no es otro que el de la simplificación de su demostración.

Definición 46. Un proceso $p \in \mathbf{BCCS}$ se dice en forma prenormal si adopta la forma

$$p = \sum_{i \in I} a_i ? p_i^1 + \sum_{j \in J} a_j ! p_j^2$$

Donde I y J son conjuntos finitos de índices. Es importante tener en cuenta que en la notación anterior es posible que $a_i = a_j$ para $i \neq j$.

Cuando el conjunto de señales de entrada en la notación anterior es vacío, es decir, $I = \varnothing$ lo que en realidad caracterizamos es el proceso que solo consta de acciones de salida. Si $J = \varnothing$ nos encontramos con un término quiescente. Si ambos índices son vacíos tenemos el término $0 + 0$, que por el axioma (B3) es equivalente a 0.

Adoptando esta manera, podemos separar las posibles acciones de un proceso en torno a conjunto de señales de entrada I, y de salida J, lo que ha de facilitar posteriores razonamientos, como tendremos ocasión de ver en la proposición

Lema 19. *Sea $p \in \boldsymbol{BCCS}$. Existe p_n en forma prenormal tal que $\vdash p_n = p$*

Demostración. Sea $p \in \mathbf{BCCS}$. Para obtener la forma normal p_n se procede a reordenar los elementos merced a las propiedades asociativa y conmutativa (B1),(B2) a la vez que quitar aquellos que se presumen innecesarios siguiendo las leyes de la idempotencia (B3) o elemento neutro (B4). □

Provistos de una forma normal, el siguiente teorema da cuenta de la completitud de dos procesos de naturaleza finita.

Teorema 13. *Sean $p', q' \in \boldsymbol{BCCS}$ de naturaleza finita. Si p' iocos q' entonces $\vdash p' \sqsubseteq q'$*

Demostración. Consideraremos las fórmulas prenormales p, q respectivas de los procesos p', q', las cuales existen en virtud del lema 19 . A continuación se procederemos por inducción sobre la profundidad del término de la izquierda p.

n=0 Si $p = 0$, entonces necesariamente q no puede tener entradas ya que por hipótesis p iocos q y por tanto $\mathsf{ins}(p) = \varnothing \supseteq \varnothing = \mathsf{ins}(q)$. Tampoco puede tener salidas, –a excepción del símbolo $\delta!$–, ya que $0 \xrightarrow{\delta!}$. En estas circunstancias, podemos escribir $0 \sqsubseteq 0$.

n>0 Valoremos ahora dos procesos en forma prenormal:

$$p = \sum_{i \in I} a_i ? p_i^1 + \sum_{j \in J} a_j ! p_j^2$$
$$q = \sum_{i \in I'} a_i ? q_i^1 + \sum_{j \in J'} a_j ! q_j^2$$

A continuación, y teniendo en cuenta que por hipótesis p iocos q, hagamos una interpretación de las cláusulas de la definición 22 con respecto a los conjuntos I, J, I', J'.

Supongamos que $I' \neq \varnothing$. Entonces tenemos:

- $I' \subseteq I$, por la cláusula 22.1
- Por la cláusula 22.2 y la hipótesis de inducción sobre procesos de menor profundidad obtenemos que el conjunto de índices I' se puede dividir en dos partes disjuntas $I' = I_1 \cup I_2$ tales que:

- $I_1 \subseteq I$.
- Para cualquier $i \in I$ existe un índice $j \in I_1$ tal que $a_i? = a_j?$ y $p_i^1 \sqsubseteq q_i^1$. Ese índice j lo vamos a denotar con $I(i)$.
- Para cada $k \in I_2$ existe $l \in I_1$ tal que $a_k? = a_l?$

▪ De forma análoga, debido a la cláusula 22.3 y la hipótesis de inducción, el conjunto de índices J' se puede dividir en dos partes disjuntas $J' = J_1 \cup J_2$ tales que:

- $J_1 \subseteq J$.
- Para cualquier $i \in J$ existe $j \in J_1$ tales que $a_i! = a_j!$ y $p_i^2 \sqsubseteq q_j^2$. Ese índice j lo vamos a denotar por $J(i)$.

Así, si $I' \neq \varnothing$, q se puede escribir de la forma

$$q = \sum_{i \in I_1} a_i? q_i^1 + \sum_{i \in I_2} a_i? q_i^1 + \sum_{j \in J_1} a_j! q_j^2 + \sum_{j \in J_2} a_j! q_j^2$$

Usando el axioma de idempotencia (B3), podemos deducir

$$\vdash q = \overbrace{\sum_{i \in I} a_i? q_{I(i)}^1 + \sum_{i \in I_2'} a_i? q_i^1 + \sum_{i \in J} a_i! q_{J(i)}^2 + \sum_{i \in J_2} a_i! q_i^2}^{q'}$$

Ese segundo término lo vamos a llamar q'. Puesto que para cada $i \in I$ tenemos y puesto que

$$p_i^1 \sqsubseteq q_{I(i)}^1$$

aplicando la regla (P3)

$$\vdash a_i? p_i^1 \sqsubseteq a_i? q_{I(i)}^1$$

Aplicando reiteradamente la regla (P4a) obtenemos

$$\vdash \sum_{i \in I} a_i? p_i^1 \sqsubseteq \sum_{i \in I} a_i? q_{I(i)}^1$$

Aplicando la regla (III) a cada $i \in I_2$ obtenemos

$$\vdash \sum_{i \in I} a_i? q_{I(i)}^1 \sqsubseteq \sum_{i \in I} a_i? q_{I(i)}^1 + \sum_{i \in I_2} a_i? q_i^1$$

Aplicando la regla de la transitividad obtenemos:

$$\vdash \sum_{i\in I} a_i?p_i^1 \sqsubseteq \sum_{i\in I} a_i?q_{I(i)}^1 + \sum_{i\in I_2} a_i?q_i^1$$

Ahora, si $J = \varnothing$ tenemos que $p \xrightarrow{\delta!} p$, por la condición 22.3 tenemos que $q \xrightarrow{\delta!} q$ y por la propiedad 11 relativa a la consistencia del símbolo de quiescencia, obtenemos $J' = \varnothing$. En este caso ya hemos acabado puesto que el término de la derecha de la expresión anterior es p y el de la izquierda es q' que habíamos probado equivalente a q.

Supongamos ahora que $J \neq \varnothing$. Para cualquier $i \in J$

$$p_i^2 \sqsubseteq q_{J(i)}^2$$

se tiene

$$\vdash a_i!p_i^2 \sqsubseteq a_i!q_{J(i)}^2$$

y por tanto

$$\vdash \sum_{i\in J} a_i!p_i^2 \sqsubseteq \sum_{i\in J} a_i!q_{J(i)}^2$$

Ya que $J \neq \varnothing$, podemos aplicar la regla (I) obteniendo

$$\vdash \sum_{i\in J} a_i!p_j^2 \sqsubseteq \sum_{i\in J} a_j!q_{J(i)}^2 + \sum_{i\in J_2} a_i!q_i^2$$

Aplicando la regla (P4b) obtenemos:

$$\vdash \overbrace{\sum_{i\in I} a_i?p_i^2 + \sum_{j\in I} a_j!p_j^2}^{p} \sqsubseteq \sum_{i\in I} a_i?q_{I(i)}^1 + \sum_{i\in I_2} a_i?q_i^1 + \sum_{j\in J} a_j!p_j^2$$

Y aplicando otra vez la regla (P4b)

$$\vdash \sum_{i\in I_1'} a_i?q_i^1 + \sum_{i\in I_2'} a_i?q_i^1 + \sum_{j\in J} a_j!p_j^2 \sqsubseteq \underbrace{\sum_{i\in I_1'} a_i?q_i^1 + \sum_{i\in I_2'} a_i?q_i^1 + \sum_{j\in J} a_j!q_j^2 + \sum_{j\in J_2'} a_j!q_j^2}_{q'}$$

Y hemos visto antes que se puede deducir $\vdash q = q'$.

Supongamos que $I' = \varnothing$. Aplicando el axioma (II) obtenemos

$$\vdash \sum_{i \in I} a_i ? p_i^1 \sqsubseteq 0$$

Al igual que antes si $J' = \varnothing$ entonces tendríamos que $J = \varnothing$. En este caso $q = 0$ y p se corresponde con el término de la izquierda de la expresión anterior, y por tanto ya hemos acabado.

Si $J' \neq \varnothing$ podemos hacer la división del índice J como en el caso anterior en los índices J_1 y J_2. Aplicando el axioma de idempotencia (B3) como en el caso anterior podemos deducir

$$\vdash q = \overbrace{\sum_{i \in J} a_i ! q^2_{J(i)} + \sum_{i \in J_2} a_i ! q_i^2}^{q'}$$

Al igual que en el caso anterior podemos deducir

$$\vdash \sum_{i \in J} a_i ! p_j^2 \sqsubseteq \underbrace{\sum_{i \in J} a_j ! q^2_{J(i)} + \sum_{i \in J_2} a_i ! q_i^2}_{q'}$$

Ahora, aplicando la regla (P4b) y el axioma (B4) tenemos

$$\vdash \sum_{i \in I} a_i ? p_i^1 + \sum_{i \in J} a_i ! p_j^2 \sqsubseteq 0 + \sum_{i \in J} a_i ! p_j^2 = \sum_{i \in J} a_i ! p_j^2 \sqsubseteq q'$$

Y como hemos visto antes $\vdash q = q'$.

□

Una vez demostradas por separado la *corrección* y *completitud* del cálculo, podemos enunciar el siguiente teorema con respecto al cálculo enunciado en la sección anterior:

Teorema 14. *Sea p, q dos procesos de naturaleza finita.*

$$p \,\mathsf{iocos}\, q \text{ si y sólo si } \vdash p \sqsubseteq q.$$

Demostración. A partir de los teoremas 12 y 13 □

6.3.2. Procesos infinitos

Los axiomas demostrados en la sección anterior, (B1),(B2),(B3), (B4), (I), (II) y(III) sólo garantizan la corrección y completitud cuando se consideran procesos finitos. Sin embargo, cuando se toman en cuenta procesos infinitos, construidos a través del operador de recursión, como en

$$p = rec\ x.a?x$$

el cálculo presentado en la sección anterior resulta incompleto.

En el intento por extender el cálculo para dar cuenta de esta nueva situación, se empleará una estrategia muy recurrente que consiste en razonar en base a los infinitos procesos de naturaleza finita que están "contenidos" en un proceso de naturaleza infinita. En este contexto solemos referirnos a ellos como *aproximaciones finitas* y son el objeto de la definición 47.

Definición 47. Sea p un proceso. La aproximación finita de altura n queda definida del siguiente modo.

$n = 0$ En este caso, $p \downarrow_0 = \mathbf{0}$, para cualquier p.

$n > 0$ Dependiendo de la forma de p, tenemos los siguientes casos:

- Para el proceso nulo, $\mathbf{0} \downarrow_n = \mathbf{0}$
- Aunque la aparición de variables sólo tiene lugar en un contexto bien marcado, necesitamos considerar $\mathbf{x} \downarrow_n = \mathbf{x}$
- Considerando el operador prefijo $(ap) \downarrow_n = ap \downarrow_{n-1}$
- Para el operador elección, tenemos que $(p + q) \downarrow_n = p \downarrow_n + q \downarrow_n$
- Con el operador de recursión $rec\ x.p \downarrow_n = p_1$ donde p_1 es el resultado de sustituir en $p \downarrow_n$ todas las apariciones de x que aparecen a una altura k por $rec\ x.p \downarrow_{n-k}$. Esto es así puesto que la recursión está guardada a un nivel $k > 0$.

La figura 6.4 permite dar un idea intuitiva de como es la aproximación finita cuando consideramos un término definido mediante el operador de recursión.

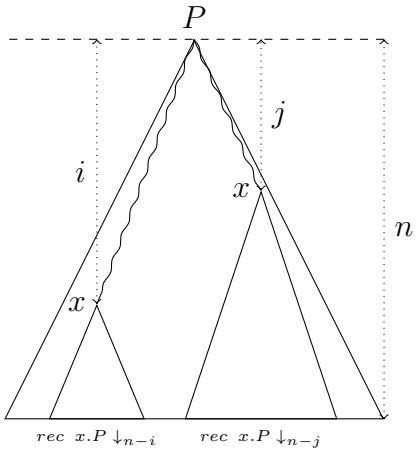


Figura 6.4: Resultado de la aproximación finita en $rec\ x.p \downarrow_n$

Nueva regla para procesos infinitos Una vez que hemos dado la definición de aproximación finita, estamos preparados para abordar la corrección y el cálculo de los procesos de naturaleza infinita. Enunciado de modo informal, la idea que queremos expresar viene a decir que, siendo p y q dos procesos infinitos, siempre que podamos inferir fórmulas para cualquier aproximación finita de los mismos podremos inferir una fórmula que los ponga en relación. Expresado en forma de regla formalmente, se corresponde con la regla (IV) que aparece en la figura 6.3 y que volvemos a exponer separadamente a continuación:

$$\frac{\forall n \in \mathbb{N} p \downarrow_n \sqsubseteq q \downarrow_n}{p \sqsubseteq q} \qquad \text{(IV)}$$

Corrección

Para demostrar la corrección de esta nueva regla, necesitaremos poner en relación la semántica operacional de un proceso infinito con la de una cualquiera de sus aproximaciones finitas. De esto se encargan los lemas 20 y 21 que enunciamos y demostramos a continuación.

Lema 20. *Sea p un proceso infinito y $a \in L \setminus \{\delta!\}$. Se cumple*

$$p \xrightarrow{a} p_a \Rightarrow p\downarrow_n \xrightarrow{a} p_a\downarrow_{n-1}$$

Demostración. Procederemos a demostrar por inducción sobre la estructura de los términos expresados en la definición 43.

Caso base Supongamos el proceso 0. Claramente $0 \overset{a}{\not\longrightarrow}$, por lo que trivialmente se cumple. Lo mismo cabe decir para el caso x, ya que $x \overset{a}{\not\longrightarrow}$.

Caso recursivo Existen tres posibilidades.

- Sea ap_a. En este caso $(ap_a)\downarrow_n = ap_a\downarrow_{n-1}$ y por la semántica operacional $ap_a\downarrow_{n-1} \xrightarrow{a} p_a\downarrow_{n-1}$.
- Sea $p+q$. Entonces, por la semántica operacional, o bien $p \xrightarrow{a} p_a$ o $q \xrightarrow{a} q_a$. Tomemos en cuenta el primer caso. Por hipótesis de inducción, tenemos que $p\downarrow_n \xrightarrow{a} p_a\downarrow_{n-1}$, y puesto que $(p+q)\downarrow_n = p\downarrow_n + q\downarrow_n$, nuevamente por la semántica operacional tenemos que $p\downarrow_n + q\downarrow_n \xrightarrow{a} p_a\downarrow_{n-1}$. El razonamiento es igual escogiendo $q \xrightarrow{a} q_a$
- Sea $rec\ x.p$. Si, por hipótesis, tenemos que $rec\ x.p \xrightarrow{a} p_a$, entonces por la semántica operacional, $p \xrightarrow{a} p'$, y además $p'[x/rec\ x.p] = p_a$. Podemos suponer que $rec\ x.p\downarrow_n \xrightarrow{a} p''$, donde p'' sustituye en p' las apariciones de profundidad k, con $k < n$, por $rec\ x.p\downarrow_{n-k-1}$. La cuestión entonces queda relegada a verificar que:

  $$p'' = p'[x/rec\ x.p]\downarrow_{n-1}$$

  Para ello, nos fijamos en el término del lado derecho en el que se han practicado una serie de sustituciones y posteriores aproximaciones finitas. Analizando la profundidad de las apariciones de la variable x, vemos que pueden pasar dos casos:

  - x aparece en p' una profundidad mayor que $n-1$ (figura 6.5), por lo que no aparece en el término final cuando se toma $p'\downarrow_{n-1}$. Tampoco aparecían en p'', pues éste era de profundidad $n-1$.
  - x aparece a una profundidad menor que $n-1$ (figura 6.5). Inicialmente se practica la sustitución completa en x por $rec\ x.p$

, pero al tomar $p'[x/rec\ x.p] \downarrow_{n-1}$, se recorta la parte de $rex\ x.p$ a una altura de $n-1-k$, ya que, como se puede ver en la figura 6.5, al estar el nodo original x a una altura k, es la forma por la que $p'[x/rec\ x.p] \downarrow_{n-1}$ tendrá una altura exacta de $n-1$.

Los dos casos descritos describen exactamente p'', por lo que como consecuencia de lo anterior tenemos que, en efecto, $rec\ x.p \downarrow n \xrightarrow{a} p_a \downarrow_{n-1}$.

□

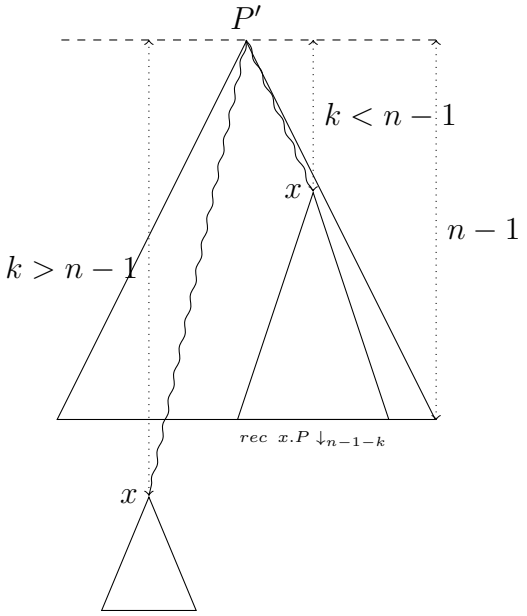


Figura 6.5: Ilustración de sustitución y posterior poda finita

Lema 21. *Sea p un proceso infinito y $a \in L \setminus \{\delta!\}$. Se cumple*

$$Si\ p \downarrow_n \xrightarrow{a} p'\ entonces\ p \xrightarrow{a} p_1\ y\ p' = p_1 \downarrow_{n-1}$$

Demostración. Procederemos por inducción estructural sobre p

Base En los casos en los que $p = 0$ y $p = x$ se cumple trivialmente, pues según la definición de aproximación finita $0 \downarrow_n = 0$ y $x \downarrow_n = x$ y por la semántica operacional $0 \xrightarrow{a}\!\!\!\!\!/\ \rightarrow$ y $x \xrightarrow{a}\!\!\!\!\!/\ \rightarrow$, para $a \in L \setminus \{\delta!\}$

Recursión Vamos a distinguir tres casos:

- Supongamos que $p = ap_1$. Es claro que por la semántica operacional $p \xrightarrow{a} p_1$. Al mismo tiempo por la definición de aproximación finita $p \downarrow_n a = ap_1 \downarrow_{n-1}$ y resulta trivial de nuevo por la semántica operacional que $p' = p_1 \downarrow_{n-1}$
- Sea $p = p_1 + p_2$. Por la definición de aproximación finita, tenemos que $p \downarrow_n = p_1 \downarrow_n + p_2 \downarrow_n$. Vamos a suponer que $p_1 \downarrow_n \xrightarrow{a} p'$. Entonces, por hipótesis de inducción tenemos que $p_1 \xrightarrow{a} p'_1$ y que $p' = p_1 \downarrow_{n-1}$. Y por la semántica operacional esto quiere decir que $p1 + p2 \xrightarrow{a} p'_1$. El razonamiento es similar si se considera $p_2 \xrightarrow{a} p'_2$.
- Vamos a valorar el caso $p = rec\ x.p_1$. Supongamos que $(rec\ x.p_1) \downarrow_n \xrightarrow{a} p'$. Esto es lo mismo que plantear que $p_1 \downarrow_n \xrightarrow{a} p''$ de forma que p' es sustituir en p'' cada aparición de x por $p \downarrow_{n-k-1}$ siendo k la altura de cada aparición de x, como se puede ver en la figura 6.5. Ahora bien, por hipótesis de inducción $p_1 \xrightarrow{a} p_a$ y $p'' = p_a \downarrow_{n-1}$. Por otra parte, tenemos que $p \xrightarrow{a} p_a[x/rec\ x.p]$, cumpliendo que $p_a[x/rec\ x.p] \downarrow_{n-1} = p'$ según se ha definido la aproximación para p_a.

□

El objeto de todos los lemas demostrados con anterioridad ha sido dar la necesaria cobertura al siguiente teorema que pone en conexión las transiciones de procesos infinitos con sus aproximaciones finitas mediante el siguiente teorema:

Teorema 15. *Sean* $p \in \boldsymbol{BCCS}$ *y* $a \in L \setminus \{\delta!\}$

$$p \xrightarrow{a} p_a \Leftrightarrow p \downarrow_n \xrightarrow{a} p_a \downarrow_{n-1}$$

Demostración. Este teorema es la simple yuxtaposición del lema 20, que demuestra la implicación de derecha a izquierda, y el lema 21 que garantiza la implicación de izquierda a derecha. □

Los anteriores lemas no han considerado las transiciones con símbolos $\delta!$. Este será el objeto del lema 22. Para el símbolo $\delta!$ tenemos un resultado pa-

recido, lo único que cambia es que en lugar de ir a la aproximación de nivel $n-1$ se que da en la misma aproximación.

Lema 22. *Sea p un proceso infinito. Entonces*

$$p\xrightarrow{\delta!} \Leftrightarrow p\downarrow_n \xrightarrow{\delta!} p\downarrow_n$$

Sustentados por el teorema 15, estamos en disposición de proceder a la demostración de la corrección del axioma (IV), para la cual adoptaremos la siguiente estrategia: por la corrección de procesos finitos, sabemos que cualquier fórmula inferida sobre procesos finitos es correcta respecto a la relación iocos; por tanto, se puede abordar la cuestión de la corrección de la regla (IV) probando que dos procesos infinitos están en relación si lo están todas sus aproximaciones finitas. Esta es la idea recogida en la proposición 17. No obstante, antes necesitamos el siguiente lema

Lema 23. *Sean p, q dos procesos y $n > 0$ un número natural, si $p\downarrow_n \mathsf{iocos}\ q\downarrow_n$ entonces $p\downarrow_{n-1} \mathsf{iocos}\ q\downarrow_{n-1}$.*

Demostración. La demostración se hace por inducción sobre n. Si $n = 1$ el resultado es trivial puesto que $p\downarrow_{n-1} = q\downarrow_{n-1} = 0$. Supongamos entonces $n > 1$ y demostremos que $p\downarrow_{n-1} \mathsf{iocos}\ q\downarrow_{n-1}$. Veremos que se cumplen las condiciones de la definición de iocos (definición 22).

1. En primer lugar veamos que $\mathsf{ins}(q\downarrow_{n-1}) \subseteq \mathsf{ins}(p\downarrow_{n-1})$. Tomemos $a? \in \mathsf{ins}(p\downarrow_{n-1})$, y por el teorema 15, tenemos que $q\xrightarrow{a?} q_1$ y $q\downarrow_n \xrightarrow{a?} q_1\downarrow n-1$. Puesto que $p\downarrow_n \mathsf{iocos}\ q\downarrow_n$ tenemos que $a? \in \mathsf{ins}(p\downarrow_n)$. Aplicando otra vez el teorema 15, tenemos que $p\xrightarrow{a?} p_1$ y $p\downarrow_{n-1} \xrightarrow{a?} p_1\downarrow n-2$. Por tanto tenemos que $a?\mathsf{ins}(p)$

2. Tomemos $a? \in \mathsf{ins}(q\downarrow_{n-1})$ tal que $p\downarrow_{n-1} \xrightarrow{a?} p'$. Aplicando otra vez el teorema 15 tenemos que $p\xrightarrow{a?} p''$, $p' = p''\downarrow_{n-1}$ y $p\downarrow_n \xrightarrow{a?} p''\downarrow_{n-1}$. Por otro lado, puesto que $a? \in \mathsf{ins}(q\downarrow_{n-1})$ y aplicando el teorema 15, tenemos que $a? \in \mathsf{ins}(q)$ y que $a? \in \mathsf{ins}(q\downarrow_n)$. Dado que $p\downarrow_n \mathsf{iocos}\ q\downarrow_n$ podemos deducir que existe q' tal que $q\downarrow_{n-1} \xrightarrow{a?} q'$ y que $p''\downarrow_{n-1} \mathsf{iocos}\ q'$. Por el teorema 15 tenemos que existe q'' tal que $q\xrightarrow{a?} q''$ y que $q' = q''\downarrow_{n-1}$. Aplicando la hipótesis de inducción tenemos que $p''\downarrow_{n-1} \mathsf{iocos}\ q''\downarrow_{n-1}$.

3. Supongamos ahora $o! \in O \setminus \{\delta!\}$ y $p \downarrow_{n-1} \xrightarrow{o'!} p'$. Para encontrar q' tal que $q \downarrow_{n-1} \xrightarrow{o'!} q'$ tal que $p' \,\mathsf{iocos}\underline{\leq}\, q'$ se procede de forma análoga al caso anterior.

4. Si $p \downarrow_{n-1} \xrightarrow{\delta!} p'$ entonces $p' = \downarrow_{n-1}$ (lema 11). Por el lema 22 tenemos que $p \xrightarrow{\delta!} p$ y también $p \downarrow_n \xrightarrow{\delta!} \downarrow_n$. Puesto que $p \downarrow_n \mathsf{iocos}\underline{\leq} q \downarrow_n$ tenemos que $q \downarrow_n \xrightarrow{\delta!} q \downarrow_n$. Aplicando otra vez el lema 22 tenemos $q \downarrow_{n-1} \xrightarrow{\delta!} q \downarrow_{n-1}$.

□

Proposición 17. *Sean $p, q \in \boldsymbol{BCCS}$ dos procesos de naturaleza infinita. Entonces*

$$\forall n \quad p \downarrow_n \mathsf{iocos}\underline{\leq}\, q \downarrow_n \Rightarrow p \,\mathsf{iocos}\underline{\leq}\, q$$

Demostración. Proponemos la relación

$$R = \{(r, s) \mid \forall n > 0 \quad r \downarrow_n \mathsf{iocos}\underline{\leq}\, s \downarrow_n\}$$

Por hipótesis, tenemos que $(p, q) \in R$, al tener todas sus aproximaciones finitas relacionadas mediante $\mathsf{iocos}\underline{\leq}$. Veamos si se cumplen las tres cláusulas de la defínción 22, tomemos $(r, s) \in R$.

1. Merced al lema 20 y la definición de $\mathsf{iocos}\underline{\leq}$, tomando $n > 0$, tenemos:

$$\mathsf{ins}(p) = \mathsf{ins}(p \downarrow_n) \supseteq \mathsf{ins}(q \downarrow_n) = \mathsf{ins}(q)$$

2. Consideremos ahora $a? \in \mathsf{ins}(s)$ tal que $r \xrightarrow{a?} r'$. Por el teorema 15, tenemos que $r \downarrow_n \xrightarrow{a?} r' \downarrow_{n-1}$ para cada $n > 0$. Puesto que $r \downarrow_n \mathsf{iocos}\underline{\leq}\, s \downarrow_n$ y $a? \in \mathsf{ins}(s \downarrow_n)$ para cada $n > 0$, tenemos que existe $s_{1,n}$ tal que $s \downarrow_n \xrightarrow{a?} s_{1,n}$ y $r' \downarrow_{n-1} \mathsf{iocos}\underline{\leq}\, s_{1,n}$. Por el teorema 15, para cada $n > 0$ existe $s_{2,n}$ tal que $s \xrightarrow{a?} s_{2,n}$ y $s_{1,n} = s_{2,n} \downarrow_{n-1}$. Ahora el conjunto $A = \{s' \mid s \xrightarrow{a?} s'\}$ es finito. Por tanto debe existir $s_2 \in A$ tal que para cada k existe $l \geq k$ tal que $s_2 = s_{2,n}$, por tanto $r \downarrow_l \mathsf{iocos}\underline{\leq}\, s_2 \downarrow_l$. Por el lema anterior obtenemos que $r \downarrow_j \mathsf{iocos}\underline{\leq}\, s_2 \downarrow_j$ para $j \leq l$ y por tanto podemos deducir $r \downarrow_n \mathsf{iocos}\underline{\leq}\, s_2 \downarrow_n$ para cada $n > 0$. Esto último significa que $(r, s) \in R$.

3. Si consideramos $o! \in O$ tal que $r \xrightarrow{o!} r'$, podemos usar un razonamiento parecido al anterior para deducir que existe s' tal que $s \xrightarrow{o!} s'$ y $r' \,\mathsf{iocos}\underline{\leq}\, s'$.

□

Completitud

Valorar la completitud del cálculo para procesos infinitos lleva a plantearse la reflexión de si, dados dos procesos infinitos relacionados por $\mathsf{iocos}_{\underline{\leq}}$, podemos encontrar una fórmula del cálculo que los acredite como tal.

De nuevo plantearemos la cuestión desde el punto de vista de la relación $\mathsf{iocos}_{\underline{\leq}}$, remontándonos a los resultados que ya conocemos sobre la completitud relativa a procesos finitos y se vio en la sección anterior 6.3.1. De esta manera, la completitud del cálculo relativo a los procesos infinitos se reduce a la posibilidad de poder relacionar todas sus posibles aproximaciones finitas mediante $\mathsf{iocos}_{\underline{\leq}}$. De ello da cuenta la proposición 18 que enunciamos a continuación

Proposición 18. *Sean p, q dos procesos, potencialmente infinitos,*

$$p \,\mathsf{iocos}_{\underline{\leq}}\, q \Rightarrow p\downarrow_n \,\mathsf{iocos}_{\underline{\leq}}\, q\downarrow_n$$

Demostración. Consideremos la relación

$$R = \{(r\downarrow_n, s\downarrow_n) \mid r \,\mathsf{iocos}_{\underline{\leq}}\, s\}$$

que toma todos los pares de aproximaciones finitas de cualesquiera procesos en relación $\mathsf{iocos}_{\underline{\leq}}$.

Por hipótesis, y por la construcción de R, tenemos que todos los pares de aproximaciones finitas $(p\downarrow_n, q\downarrow_n) \in R$. Para valorar que todos los elementos de R cumplen las condiciones de una $\mathsf{iocos}_{\underline{\leq}}$-simulación, teniendo en cuenta las cláusulas de la definición 22 de $\mathsf{iocos}_{\underline{\leq}}$, procederemos a considerar dos casos por separado:

$n = 0$ Ya que, por la definición de aproximaciones finitas, $p\downarrow_0 = \mathbf{0}$, $q\downarrow_0 = \mathbf{0}$ nos encontramos que ambos procesos no tienen entradas. Esto hace que las cláusulas 22.1 y 22.2 se cumplan trivialmente.

En el caso de la salidas, solamente está presente el símbolo de la *quiescencia*, disponible en los estados como transición hacia sí mismos, presentes ya en la relación R, por lo que se cumple la cláusula 22.3.

$n > 0$ En este caso sabemos que pueden existir transiciones definidas para $p\downarrow_n$. Por el teorema 15 y por la hipótesis, $p \,\mathsf{iocos}_{\underline{\leq}}\, q$, tenemos que

$$\mathsf{ins}(p\downarrow_n) = \mathsf{ins}(p) \supseteq \mathsf{ins}(q) = \mathsf{ins}(q\downarrow_n)$$

y por tanto se cumple la cláusula 22.1 para cualquier par de aproximaciones finitas.

Pasemos a considerar los distintos tipos de transiciones que pueden darse:

- Si $p \downarrow_n \xrightarrow{\delta!} p \downarrow_n$, entonces $q \downarrow_n \xrightarrow{\delta!} q \downarrow_n$, de forma que $(p \downarrow_n, q \downarrow_n) \in R$. Al tratarse el símbolo $\delta!$ como un símbolo de salida más, esta condición basta para cumplir la cláusula 22.3.
- Si $p \downarrow_n \xrightarrow{a} p_a \downarrow_n$ con $a \in O$, entonces, por el teorema 15, se tiene que $p \xrightarrow{a} p_a$. Ya que por hipótesis, $p \,\mathsf{iocos}\underline{\mathsf{s}}\, q$, sucede que $q \xrightarrow{a} q_a$, con $p_a \,\mathsf{iocos}\underline{\mathsf{s}}\, q_a$ y por el teorema 15 $q \downarrow_n \xrightarrow{a} q_a \downarrow_{n-1}$. Al darse $p_a \,\mathsf{iocos}\underline{\mathsf{s}}\, q_a$, y dado que $n > 0$, sucede que el par $p_a \downarrow_{n-1}, q_a \downarrow_{n-1} \in R$, por la forma en que está definido R, dando cumplimiento a la cláusula 22.3 que debe cumplir una $\mathsf{iocos}\underline{\mathsf{s}}$-relación.
- En el caso en que $a \in I$, pueden ocurrir dos casos:
  - o bien, $a \notin \mathsf{ins}(q \downarrow_n)$, en cuyo caso no hay nada que valorar y la cláusula 22.2 se cumple de modo trivial
  - o bien $a \in \mathsf{ins}(q \downarrow_n)$; en este último caso $p \downarrow_n \xrightarrow{a} p_a \downarrow_{n-1}$, ya que debido a que $p \,\mathsf{iocos}\underline{\mathsf{s}}\, q$ y por el teorema 15 tenemos

    $$\mathsf{ins}(p \downarrow_n) = \mathsf{ins}(p) \supseteq \mathsf{ins}(q) = \mathsf{ins}(q \downarrow_n)$$

    Además, también por el teorema 15 , tendremos que $p \xrightarrow{a} p_a$, y por hipótesis, $q \xrightarrow{a} q_a$, con $p_a \sqsubseteq q_a$. De nuevo por el teorema 15 se tiene que $q \downarrow_n \xrightarrow{a} q_a \downarrow_{n-1}$. Dado que $n > 0$, por la definición de R tenemos que $(p_a \downarrow_{n-1}, q_a \downarrow_{n-1}) \in R$, cumpliendo así la cláusula 22.2 relativa a los símbolos de entrada en una relación $\mathsf{iocos}\underline{\mathsf{s}}$.

□

Ahora sí, merced al lema 18 estamos en condiciones de formular el teorema 16 que se encarga de asegurar la completitud del cálculo referido a procesos infinitos, basándose en resultados anteriores ciertos para cualquier proceso finito.

Teorema 16. *Sean p y q dos procesos infinitos. Entonces*

$$p \,\mathsf{iocos}_{\underline{\leq}}\, q \Rightarrow p \sqsubseteq q$$

Demostración. Por el lema 18 tenemos que $p \downarrow_n \mathsf{iocos}_{\underline{\leq}} q \downarrow_n$ para cualquier aproximación finita de p y q. Al tratarse de procesos finitos, por la completitud demostrada en el teorema 13 anteriormente es posible deducir la fórmula $p \downarrow_n \sqsubseteq q \downarrow_n$ para cualesquiera $p \downarrow_n, q \downarrow_n$. Y esto, en virtud de la regla (IV) implica que $p \sqsubseteq q$ □

El teorema 16 es el encargado de cerrar este capítulo en el que hemos abordado la axiomatización de $\mathsf{iocos}_{\underline{\leq}}$. Para ello hemos tenido que proponer un sencillo lenguaje de naturaleza algebraica y especular un cálculo que hemos demostrado correcto y completo con respecto a la relación $\mathsf{iocos}_{\underline{\leq}}$. La demostración se ha efectuado en dos etapas, considerando en primer lugar aquellos procesos de naturaleza finita, esto es, definidos sin el operador de recursión, y después extendiendo el razonamiento al resto de procesos infinitos.

Con ello se cierra además la serie de capítulos centrales y originales de esta tesis. En el siguiente capítulo haremos una valoración del trabajo desarrollado apuntando nuevas líneas de investigación futura relacionadas que, por la profundidad que requieren, no han podido ser incluidas en esta tesis.

Capítulo 7

Conclusiones

Termina esta tesis con una valoración del trabajo desarrollado teniendo en cuenta las perspectivas que se generaron al principio de la misma, valorando el efectivo cumplimiento de determinados objetivos en algunos casos y en otros mostrando líneas de investigación futura que, por su sola envergadura, no pudieron llevarse a cabo y merecen una atención más allá del alcance de esta tesis.

El punto de partida de la investigación era la mejora de aspectos semánticos de las relaciones de conformidad existentes hasta el momento dentro del área de *Model Based Testing*. Además de las caracterizaciones propias de un marco de *testing*, la aplicación de planteamientos coinductivos propia de las semánticas de simulación como nueva relación de conformidad llevó a plantear su resolución mediante técnicas algorítmicas basadas en grafos empleadas en escenarios similares, como en de verificación y *model checking*.

En este último contexto se sitúa el decisivo algoritmo **GCPP**, presente en una de las plataformas especializadas en relaciones semánticas, **mCRL2**, que conseguimos adaptar al objeto de nuestro estudio.

Después del *testing* y la *algoritmia*, nos propusimos el reto de abordar la relación de conformidad mediante la *deducción lógica*, para lo cual fue necesario la *axiomatización* de la misma empleando un lenguaje algebraico.

Quedan como futuras lineas de investigación a seguir la ampliación del estudio de la relación iocos a sistemas donde se consideran acciones internas no perceptibles desde el exterior. Desde un punto de vista de la implementación, la paralelización de los algoritmos de *testing* y su despliegue en plataformas de

computación grid suponen una oportunidad para aprovecharse del incremento de la capacidad de cómputo que estas tecnologías proporcionan.

7.1. La integración de técnicas simulación y el *testing*

Antes de comenzar nuestro trabajo, el principal problema con que se encontraban las relaciones semánticas de *conformidad* presentes en el *Model Based Testing*, mayormente basadas en *trazas*, era su incapacidad para distinguir los contextos locales en los que tenía lugar el indeterminismo, circunstancia que emerge constantemente en aquellos escenarios donde la relativa velocidad de los procesos que intervienen es desconocida y puede variar.

En nuestra empresa por superar este *deficit*, hemos utilizado unos conocimientos teóricos interesantes por parte de Abramsky [2] en los que se indicaba que la potencia expresiva de una relación próxima a nuestro objetivo como la bisimulación, o como el se refiere a ella, la *equivalencia observacional*, puede ser discriminada mediante métodos de *testing*.

Gracias a sus planteamientos, se ha conseguido la formulación de una nueva relación de *conformidad*, iocos, que aplicada al mundo de los sistemas de entrada-salida salva con éxito los problemas que plantea la presencia del indeterminismo. Y todo ello inspirándose en los trabajos de Tretmans [83, 82, 30] los cuales en la definición de su relación de *conformidad* ioco, tratan de manera distinta los eventos autónomos por parte del sistema y aquellos que operan como respuesta a la acción de agentes externos: para los primeros se establece un límite marcado por la especificación, mientras que para los segundos, se otorga cierto margen de contingencia sobre la base de unos mínimos que resultan ser obligatorios. Parte de estas líneas de diseño ya se recogían en la relación original, pero fueron mejoradas con objeto de evitar aquellos casos en los que sistemas nulos o casi nulos podían acreditarse como implementaciones válidas frente a la referencia.

Cabe resaltar que la inclusión de un símbolo especial, la *quiescencia*, simplificó notablemente el tratamiento de aquellos sistemas que carecían de comportamientos autónomos observables desde el exterior.

Integrada en el mundo del *Model Based Testing*, iocos fue el núcleo de una nueva teoría en torno a la cual se elaboró todo un marco capaz de caracterizar mediante *tests* esta nueva relación semántica basada en simulación. El hecho de tratar los conceptos formalmente nos dio la oportunidad de organizar todo un sistema automatizado del cual no sólo se ha demostrado la corrección y completitud del algoritmo que selectivamente genera los *tests* con respecto a la especificación, modalidad que se enmarca bajo la categoría del *testing offline*, sino que se han superado algunos inconvenientes propios relativos a la alta complejidad de tiempo y memoria, formulando algoritmos capaces de generar las primitivas al tiempo que se ejecutan contra el elemento que se evalúa, versión esta última conocida por *testing online*.

Extensión hacia otras variantes de relación Nuestro trabajo ha establecido las bases para el tratamiento de relaciones de conformidad que incorporan técnicas de simulación. No obstante, se podría extender este planteamiento al resto de variantes de la relación ioco. Por ejemplo, la relación multi-ioco [45], que extiende ioco con múltiples canales, es un buen candidato para este ejercicio que puede marcar un futuro trabajo en esta dirección.

Lo mismo ocurre con la variante de real-time ioco [57, 56, 16] en la que se incorporan periodos de tiempo especificando los tipos de señales que deben ocurrir. Una aplicación de los principios de simulación a estos escenarios contribuiría a dotar de una precisión semántica en estas relaciones, como ha ocurrido con la relación matriz.

7.2. Verificación mediante algoritmos

Las anteriores técnicas de *testing* resultan de aplicabilidad cuando no se dispone de la descripción o modelo del objeto que se quiere evaluar. A veces, incluso contando con ella, el *testing* resulta un modo eficaz de evaluar la conformidad de un sistema con respecto a una, debido a la gran complejidad que puede llevar el cómputo directo de la relación.

Distintas son las circunstancias cuando podemos contar con un algoritmo eficaz, puesto que en este caso, podemos salvar el problema de la *completitud*, insalvable en la mayoría de los casos *testing* al requerir un número infinito de

tests. A lo largo del capítulo 5 hicimos factible esta propuesta del mismo modo que se lleva a cabo en otras disciplinas como el *model checking*: preprocesando el problema de modo que razonemos a partir de un sistema equivalente pero notablemente de menor tamaño.

A la hora de realizar el diseño, fue capital la contribución de autores como R.Gentilini [32], y van Glabeck [91],los cuales habían desarrollado un sofisticado algoritmo, **GCPP**, planteado inicialmente para la resolución de problemas como la simulación ordinaria.

Mediante *técnicas de reducción*, conseguimos expresar nuestra relación objeto, iocos, dentro del dominio original donde se plantea **GCPP**, no sin antes certificar una equivalencia semántica entre la nueva relación, que llamamos oportunamente g-iocos, y la original. Básicamente esta reducción persigue describir el nuevo problema, iocos, en términos que hagan factible su solución mediante la aplicación directa del **GCPP**. Esto lo conseguimos operando el sistema original de partida al considerar un estado con propiedades especiales al que son derivados con transiciones *virtuales* el resto de estados en circunstancias previstas por la definición.

Implementación Pese a su naturaleza teórica, esta tesis ha podido llevar a implementación algunos de los algoritmos propuestos basados en procesamiento de grafos, identificándose con esa corriente que valora a la *ciencia de la computación* como una disciplina que pretender explicar los fundamentos de la praxis al tiempo que se nutre de ésta última al demandar de ella nuevos servicios que contribuyan a facilitar su objetivo final.

Desde el primer momento en que se valoró la posibilidad de una implementación, pronto se cayó en la cuenta de que ello no sería factible sin contar con una base sólida que diera soporte a un mínimo conjunto de abstracciones sin las cuales el trabajo del programador se convertiría en una empresa quimérica, ya sea con servicios próximos a la programación de sistemas, como en todo lo relativo a gestión de entrada-salida, o con representaciones de tipos abstractos de datos lejos de ser triviales, como pueden ser los sistemas de transiciones, estructuras de grafos y bloques, etc... Esto se hace aún más evidente aún cuando se tiene acceso a los códigos fuentes de la plataforma elegida, **mCRL2**. La ingente profusión de detalles que se puede observar en ellos sugirió la separación

de ambas tareas, diseño y codificación, a fin de poder ser más eficaz en cada una de ellas.

Los resultados, tal como se muestran en el capítulo 5, superaron las expectativas al obtener unos resultados de tiempo la mayor parte de las veces del mismo orden que las de los relativos a problemas ya conocidos de la simulación ordinaria, pese a ser más compleja la formulación de la relación de conformidad objeto, distinguiendo entre la naturaleza de las acciones, ya sean de entrada o de salida.

Grid y tecnologías de cómputo en paralelo Relacionado con la implementación, pero más propio de la tecnología anteriormente citada, una línea de investigación abierta son los algoritmos de *testing*, los cuales suponen un significativo reto en el futuro para plantear su implementación en tecnologías `grid`, pues la complejidad de los *tests* que requieren contrastar la viabilidad de *todas* las ramas dadas a partir de un nodo las hace candidatas ideales para el empleo de esquemas de programación paralela, como los conocidos trabajadores replicados. En este tipo de esquemas un sólo nodo computacional es el encargado de distribuir las tareas hacia otros subordinados en paralelo, a la vez que recoge los resultados de todos ellos, los procesa y emite un veredicto.

7.3. Sistemas de deducción

Además de los procedimientos ordinarios de *testing*, que proporcionan una experiencia *empírica* sin que sea necesario tener acceso al modelo que subyace a la implementación, contar con una cálculo *deductivo* puede ser una ventaja en la medida que gracias a su axiomatización podemos compararla con otras relaciones existentes, como se puede ver en las taxonomías llevadas a cabo por van Glabbeek [90, 89, 88].

Para ello fue menester la formulación de un sencillo lenguaje algebraico, y la demostración de una serie de propiedades que hacían al lenguaje *congruente* en presencia de objetos relacionados mediante iocos. Sin esta demostración no podrían abordarse dos aspectos comunes a todo cálculo que se formule: la corrección y la completitud.

Notablemente más difícil de lograr la completitud, en una primera fase pu-

dimos plantear un escenario de procesos finitos para después abordar aquellos de profundidad infinita en base a infinitas aproximaciones de profundidad finita, para lo cual tuvimos que relacionar los pasos de ejecución en uno y otro contexto.

Sistemas con acciones internas Por último, y al margen de todo sistema algebraico, cabe mencionar una extensión del estudio de la relación de conformidad al considerar un tipo de transiciones internas que tienen lugar en los sistemas sin ser detectadas en el exterior. Normalmente, a estas variantes se les conoce como versiones *weak* se la semántica a considerar, incorporando las conocidas acciones τ y razonando sobre ellas y viendo si aquellos presupuestos de refinamiento que hicimos de iocos$\underline{\text{s}}$ sobre ioco se mantienen, así como los principios de la congruencia del lenguaje. Factores todos ellos que por falta de tiempo requieren la apertura de un proyecto de investigación continuación del que se ha presentado a lo largo de estas páginas.

Bibliografía

[1] Boost software licence 1.0, (bsl-1.0). `http://www.boost.org/users/license.html`, August 2003.

[2] S. Abramsky. Observational equivalence as a testing equivalence. *Theoretical Computer Science*, 53(3):225–241, 1987.

[3] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, 1991.

[4] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba. *Reactive systems: modelling, specification and verification.* Cambridge University Press, 2007.

[5] J.C.M. Baeten and R. J van Glabbeek. Abstraction and empty process in process algebra. *Fundamenta Informaticae*, 12(2):221–241, 1989.

[6] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Computer Science. Cambridge University Press, 1990.

[7] A. Belinfante. Jtorx: A tool for on-line model-driven test derivation and execution. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 266–270. Springer, 2010.

[8] A. Belinfante. Jtorx: A tool for on-line model-driven test derivation and execution. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 266–270. Springer, 2010.

[9] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical computer science*, 37:77–121, 1985.

[10] J. A. Bergstra and J.W. Klop. Process algebra: specification and verification in bisimulation semantics. *Math. & Comp. Sci. II*, 4, 1986.

[11] J.A. Bergstra and J.W. Klop. $\mathrm{Act}_{\mathrm{tau}}$: A universal axiom system for process specification. In Martin Wirsing and J. A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications [papers from a workshop in Passau, Germany, June 9-11, 1987]*, volume 394 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 1987.

[12] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[13] G. Bernot, M. C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, 1991.

[14] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[15] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comput. Program.*, 24(3):189–220, 1995.

[16] L. B. Briones and E. Brinksma. *A test generation framework for quiescent real-time systems*. Springer, 2005.

[17] P. E. Bulychev, T. Chatain, A. David, and K. G. Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation. In Joël Ouaknine and Frits W. Vaandrager, editors, *FORMATS*, volume 5813 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009.

[18] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Trans. Comput. Logic*, 4(2):181–206, April 2003.

[19] R. Cardell-Oliver. Conformance testing of real-time systems with timed automata. In *Nordic Workshop on Programming Theory*, 1999.

[20] E. M. Clarke Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[21] S. Cranen, J. Groote, J.J.A. Keiren, F.P.M. Stappers, ErikP. Vink, Wieger Wesselink, and TimA.C. Willemse. An overview of the mcrl2 toolset and its recent advances. In Nir Piterman and ScottA. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin Heidelberg, 2013.

[22] D. de Frutos-Escrig, C. Gregorio-Rodríguez, M. Palomino, and D. Romero-Hernández. Unifying the linear time-branching time spectrum of process semantics. *Logical Methods in Computer Science*, 9(2:11):1–74, 2013.

[23] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24(2):211–237, 1987.

[24] R. De Nicola and M. Hennessy. Testing equivalence for processes. In *ICALP'83 – 10th International Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 548–560. Springer, 1983.

[25] R. G. de Vries and J. Tretmans. On-the-fly conformance testing using spin. *STTT*, 2(4):382–393, 2000.

[26] E. W. Dijkstra. Cooperating sequential processes. Published as [27], 1965.

[27] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968. More easily available in [44].

[28] N. Ellis. Vocabulary acquisition: The implicit ins and outs of explicit cognitive mediation. *Implicit and explicit learning of languages*, pages 211–282, 1994.

[29] W. B. Ewald. *From Kant to Hilbert Volume 1: A Source Book in the Foundations of Mathematics: A Source Book in the Foundations of Mathematics*, volume 1. Oxford University Press, 2005.

[30] L. Frantzen, J. Tretmans, and T. AC Willemse. *Test generation based on symbolic specifications*. Springer, 2005.

[31] M. C. Gaudel. Testing can be formal, too. *TAPSOFT'95: Theory and Practice of Software Development*, pages 82–96, 1995.

[32] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[33] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *RR 12-2003,Dep. Of Computer Sicence, University of Udine,Italy*, 2003.

[34] P. Godefroid. Combining model checking and testing.

[35] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *Software Engineering, IEEE Transactions on*, (2):156–173, 1975.

[36] C. Gregorio-Rodríguez, L. Llana, and R. Martínez-Torres. Input-output conformance simulation (iocos) for model based testing. In Dirk Beyer and Michele Boreale, editors, *FMOODS/FORTE*, volume 7892 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2013.

[37] C. Gregorio-Rodríguez, L. Llana, and R. Martínez-Torres. Effectiveness for input output conformance simulation iocos. In Erika Ábrahám and Catuscia Palamidessi, editors, *FORTE*, volume 8461 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2014.

[38] C. Gregorio-Rodríguez and M. Núñez. Denotational semantics for probabilistic refusal testing. *Electronic Notes in Theoretical Computer Science*, 22:111–137, 1999.

[39] C. Gregorio-Rodríguez, L. Llana, and R. Martínez-Torres. Extending mcrl2 with ready simulation and iocos input-output conformance simulation. *ACM Symposium on Applied Computing, Software Verification and Testing (SAC)*, 2(2):1781–1797, 2015.

[40] J. F. Groote and M. R. Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.

[41] J. F. Groote and F. van Ham. Interactive visualization of large state spaces. *International Journal on Software Tools for Technology Transfer*, 8(1):77–91, 2006.

[42] J.F. Groote, J. Keiren, F. P. M. Stappers, W. Wesselink, and T. A. C. Willemse. Experiences in developing the mcrl2 toolset. pages 143–153, 2011.

[43] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[44] P. B. Hansen, E. W. Dijkstra, and C. A. R. Hoare. *The Origins of Concurrent Programming: From Semaphores to Remote Procedure Calls*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[45] A. W. Heerink. *Ins and outs in refusal testing*. Universiteit Twente, 1998.

[46] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[47] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[48] A. Hessel, K. G Larsen, B. Nielsen, P. Pettersson, and A. Skou. Time-optimal test cases for real-time systems. In *Formal Modeling and Analysis of Timed Systems*, pages 234–245. Springer, 2004.

[49] T. Higashino, A. Nakata, K. Taniguchi, and A. R Cavalli. Generating test cases for a timed i/o automaton model. In *Testing of Communicating Systems*, pages 197–214. Springer, 1999.

[50] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[51] J. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document, 1971.

[52] J.P. Katoen, T. Kemna, I. Zapreev, and D Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In O. Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 87–101. Springer Berlin Heidelberg, 2007.

[53] J. J. A. Keiren and M. Klabbers. Modelling and verifying ieee std 11073-20601 session setup using mcrl2. 2012.

[54] A. Khoumsi, T. Jéron, and H. Marchand. Test cases generation for nondeterministic real-time systems. In *Formal Approaches to Software Testing*, pages 131–146. Springer, 2004.

[55] N. Kokash, C. Krause, and E. P. de Vink. Reo + mcrl2: A framework for model-checking dataflow in service compositions. pages 187–216, 2012.

[56] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *Model Checking Software*, pages 109–126. Springer, 2004.

[57] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using UPPAAL. In Jens Grabowski and Brian Nielsen, editors, *FATES*, volume 3395 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2004.

[58] K. G Larsen and A. Skou. Bisimulation through probabilistic testing (preliminary report). In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 344–352. ACM, 1989.

[59] G. Leduc. Conformance relation, associated equivalence, and minimum canonical tester in LOTOS. In *Protocol Specification, Testing and Verification XI*, pages 249–264. North Holland, 1991.

[60] L. Llana and D. de Frutos. Relating may and must testing semantics for discrete timed process algebras. In *5th Asian Computing Science Conference, ASIAN'99, LNCS*, pages 74–86, 1742.

[61] L. Llana and R. Martínez-Torres. IOCO as a simulation. In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods - SEFM 2013 Collocated Workshops: BEAT2, WS-FMDS, FM-RAIL-Bok, MoKMaSD, and OpenCert, Madrid, Spain, September 23-24, 2013, Revised Selected Papers*, volume 8368 of *Lecture Notes in Computer Science*, pages 125–134. Springer, 2013.

[62] R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer, 1980.

[63] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[64] R. Milner. Elements of interaction: Turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.

[65] R. Milner. *Interactive Computation, the new paradigm*, chapter Turing, Computing and Communication. Springer, 2006.

[66] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1–2):83 – 133, 1984.

[67] B. Nielsen and A. Skou. Automated test generation from timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 343–357. Springer, 2001.

[68] D. Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer Berlin Heidelberg, 1993.

[69] M. Phalippou. Executable testers. In *Proceedings of the IFIP TC6/WG6. 1 Sixth International Workshop on Protocol Test systems VI*, pages 35–50. North-Holland Publishing Co., 1993.

[70] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.

[71] G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computation*, 5:452–487, 1976.

[72] G. D Plotkin. A structural approach to operational semantics. 1981.

[73] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[74] P. Rabanal, I. Rodríguez, and F. Rubio. Testing restorable systems: formal definition and heuristic solution based on river formation dynamics. *Formal Aspects of Computing*, 25(5):743–768, 2013.

[75] F. Ranzato. A more efficient simulation algorithm on kripke structures. In Krishnendu Chatterjee and J. Sgall, editors, *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 753–764. Springer, 2013.

[76] C. Gregorio Rodríguez. *Entendiendo las Semánticas de Procesos*. PhD thesis, Departamento de Sistemas Informáticos y Computación, 2009.

[77] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *CONCUR '92 - Concurrency Theory, Third International Conference*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.

[78] G. J. Tretmans. A formal approach to conformance testing. 1992.

[79] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[80] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, (TR-CTIT-96-26), 1996.

[81] J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR'99 Concurrency Theory*, pages 46–65. Springer, 1999.

[82] J. Tretmans. Model based testing with labelled transition systems. In R. M. Hierons, J. P. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008.

[83] J. Tretmans. Model-based testing and some steps towards test-based modelling. In Marco Bernardo and Valérie Issarny, editors, *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 297–326. Springer, 2011.

[84] J. Tretmans and E. Brinksma. Torx: Automated model-based testing. 2003.

[85] J. Tretmans and E. Brinksma. Torx: Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, editors, *First European Conference on Model-Driven Software Engineering*, pages 31–43, December 2003.

[86] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, ser. 2(42):230–265, 1936.

[87] R. J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Free University, Amsterdam, 1990. Second edition available as *CWI tract* 109, CWI, Amsterdam 1996.

[88] R. J. van Glabbeek. The linear time-branching time spectrum. In *CONCUR '90 Theories of Concurrency: Unification and Extension*, number 458 in Lecture Notes in Computer Science, pages 278–297. Springer-Verlag, 1990.

[89] R. J. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR '93 - Concurrency Theory, 5th International Conference*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[90] R. J. van Glabbeek. *Handbook of Process Algebra*, chapter The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes, pages 3–99. Elsevier, 2001.

[91] R. J. van Glabbeek and B. Ploeger. Correcting a space-efficient simulation algorithm. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 517–529. Springer, 2008.

[92] M. Veanes and N. Bjørner. Alternating simulation and ioco. In A. Petrenko, Adenilso da Silva Simão, and José C. Maldonado, editors, *ICTSS*, volume 6435 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2010.

[93] M. Veanes and N. Bjørner. Alternating simulation and ioco. *STTT*, 14(4):387–405, 2012.

[94] M. Veanes, C. Campbell, W. Schulte, and N. Tillmann. Online testing with model programs. In Michel Wermelinger and Harald Gall, editors, *ESEC/SIGSOFT FSE*, pages 273–282. ACM, 2005.

[95] `http://cadp.inria.fr/resources/vlts/`.