Automatización de técnicas de división por importancia para la simulación de eventos raros

por

Carlos E. Budde

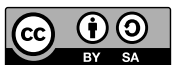
Mayo de 2017

Director: Pedro R. D'Argenio

Co-director: Holger Hermanns

Presentado ante la Facultad de Matemática, Astronomía, Física y Computación como parte de los requerimientos para la obtención del grado de Doctor en Ciencias de la Computación de la

Universidad Nacional de Córdoba



Clasificación en CCS concepts: • **Computing methodologies** → **Rare-event simulation**; *Discrete-event simulation*; Modeling and simulation; Model verification and validation.

Palabras clave: verificación formal de sistemas, modelado y simulación, simulación de eventos raros, división por importancia, RESTART, automatización de la división por importancia.

Resumen

Existen muchas técnicas para estudiar y verificar descripciones formales de sistemas probabilísticos. El *model checking probabilista* es un ejemplo sobresaliente, que abarca muchos formalismos de modelado a través de varios ángulos de estudio y grados de detalle. Sin embargo el núcleo de su algoritmia depende de la propiedad de pérdida de memoria inherente a las distribuciones exponenciales. Aún más, el espacio de estados del modelo debe caber en la memoria física del computador.

La simulación de Monte Carlo por eventos discretos ofrece una alternativa para la generalidad de procesos estocásticos descriptos como autómatas. El término *model checking estadístico* se refiere al uso de simulaciones en el entorno de model checking, con sistemas formalmente descriptos y propiedades expresadas en alguna lógica temporal (LTL, CSL, PCTL*, etc.), cuyos valores son estimados dentro de los criterios de confianza especificados por el usuario.

Esta solución alternativa puede sin embargo fallar, siendo incapaz de responder la pregunta formulada por el usuario. Esto ocurre típicamente cuando el análisis estadístico de las trazas generadas indica que los datos son insuficientes para satisfacer el criterio de confianza solicitado, es decir que se requiere la simulación de más trazas. Cuando los valores a estimar dependen de la ocurrencia de eventos raros cuya presencia en una traza es muy poco probable, la situación puede degenerar en requerimientos inviables, e.g. dos meses de simulación Monte Carlo estándar pueden ser necesarios para generar el intervalo deseado con un 90 % de confianza.

Estrategias de simulación especializadas para combatir estos problemas han sido ideadas, que disminuyen la varianza del estimador y reducen por ende los tiempos de simulación. La división por importancia es una de ellas, que requiere la definición de una función guía para dirigir la generación de trazas hacia el evento raro. Esta *función de importancia* es comúnmente ideada de forma ad hoc por un experto en el área del modelo estudiado. Una elección inadecuada puede resultar en simulaciones ineficientes y grandes tiempos de simulación.

En esta tesis se presentan técnicas automáticas para derivar la función de importancia, basadas en una descripción formal del modelo y de las propiedades a estimar. Esto abarca procesos estocásticos generales dado el uso de simulación por eventos discretos. El formalismo en el que se describen los modelos se denomina Autómatas Estocásticos con Entrada/Salida (IOSA por sus siglas en inglés, [DLM16]), y tanto propiedades transitorias como de equilibrio (estado estacionario o *steady-state*) que involucren eventos raros pueden ser estimadas. Dado que IOSA es un formalismo modular, se han desarrollado y estudiado dos técnicas complementarias: derivar la función de importancia del modelo global ya compuesto, y derivarla localmente en los módulos individuales del sistema. La segunda opción requiere la composición de las funciones generadas localmente para construir una función global, lo cual ofreció otro tema de investigación

también incluido en esta tesis.

Herramientas prototípicas pero extensibles fueron implementadas para comprobar la factibilidad y eficiencia de estas técnicas. Algunos detalles sobre su implementación, junto con los resultados de varios experimentos realizados, se presentan a lo largo de la tesis.

Agradecimientos

¿Cómo poner en palabras estos sentimientos? Imposible. Pero esta gente vale el intento, así que ahí va. Creo que cualquier persona que haya tenido la tenacidad, y la suerte, de culminar algún estudio de doctorado, no puede dejar de reconocer la contención humana que lo hizo posible.

Mi mamá y mi hermano, Lucía y Leopoldo, me han aguantado en días buenos y días malos. Mi papá Carlos ya no está, aunque siempre está. Antes pensaba que, al estar tan cerca, es difícil apreciar lo mucho que necesitamos a nuestra familia; ahora ya soy más viejo. Sepan que sin ustedes esto no existe.

También hay familia más grande: mi tía Luisa, el Pablo, el tío Ale, Ucacha y Etruria. Y familia encontrada/elegida: Lichi, Zerep, los vergas de CN1, los ñoños de FAMAF, los locos de Muay, Sergio, Pao, ... Quiero y ansío que estos vínculos se afiancen, y no me caben dudas que así será.

No me olvido de vos, dire, que bastante me ayudaste y bastante me hiciste renegar también. Ni de Raúl, con quien tanto vivimos en esta carrera de doctorandos, y quien pronto va a estar escribiendo algo como ésto. Mirando atrás y viendo lo que construimos, pucha, no es tan poco el fin y al cabo.

Hay un montón de otras personas de FAMAF a quienes les debo mucho: Nico, Pedro, Charly, Oscar, Damián, Laura, Silvia, Pablo, Félix, ... no termina la lista. Y gente de Saarbrücken: mi co-director Holger, el gran Arnd Hartmanns, Gilles, Luis, Yuliya, Hasan, y muchos más. Espero que nuestros caminos se vuelvan a cruzar pronto.

Quiero agradecer también a José Villén-Altamirano, quien me dio una mano enorme en el entendimiento de RESTART. En tres días no sólo me demostró lo hospitalarios que son en Madrid, sino que además me dio consejos y explicaciones que meses de lectura no habrían logrado esclarecer con mayor atino. Sólo desearía haber hecho esa visita antes, y así haber tenido tiempo de implementar parte de estas nuevas ideas para la tesis.

De seguro me estoy olvidando injustamente de muchas otras personas, que ya tendrán tiempo de recriminármelo y demandar una compensación en términos de cerveza o afines. Con gusto pagaré la deuda, y nos tomaremos un tiempo para ponernos al día.

A todos, así pues, ¡gracias!

Índice general

Introducción 1

Está profundamente arraigado en la naturaleza humana, asumiendo que tal cosa existe, el estudiar y modificar nuestro ambiente con el fin de minimizar amenazas e incrementar nuestras posibilidades de supervivencia y confort. En una sociedad cada día más tecnologizada con aparatología electrónica, estos esfuerzos se materializan en el desarrollo de sistemas de cómputo y de almacenamiento de información. Estos procesos y herramientas basados en el uso de computadoras pueden volverse extremadamente complejos, y dado que nuestro bienestar depende de ellos, se los somete a revisión constante, tanto humana como automatizada, para asegurar su correcto funcionamiento.

Son omnipresentes los ejemplos de este tipo de actividades: desde verificaciones y *controles mecánicos en trenes* realizadas periódicamente, o revisiones del *código fuente de programas*, hasta los protocolos altamente estructurados que se aplican en cada fase de fabricación y ensamblaje de las *aeronaves espaciales*.

A pesar de tales esfuerzos, los inextricables cimientos de la realidad imposibilitan evitar por completo la ocurrencia de accidentes. Ya sea por error humano o falla mecánica, el 22 de Febrero de 2012 "*la tragedia de Once*" se cobró con la vida de más de cincuenta personas, en el peor accidente ferroviario argentino de los últimos treinta años.

Los resultados indeseables también se observan en procesos aislados de un ambiente natural hostil. Considérese *Heartbleed*, el *bug* de seguridad en la librería criptográfica OpenSSL, usada a lo largo y a lo ancho del mundo para asegurar el valor más importante de la civilización occidental: el capital privado. El código fuente del que estamos hablando es una implementación profesional de un protocolo estandarizado, sometido a varias fases de control como ser la revisión-de-pares (*peer-review*). Aún así, el código contenía una falla que permitía infringir la privacidad del usuario a través de una sobre-lectura de registros (*buffer over-read*). Esta vulnerabilidad fue difundida de forma inmediata y masiva, al punto que la compañía Codenomicon incluso ideó un logo que hoy día se asocia mundialmente con el bug (ver Figura 1.1).

Este tipo de bugs encuentran rendijas donde ocultarse incluso en cadenas de producción protocolares, saliendo a la luz para causar caos en formas netamente dañinas. Los programas de transbordadores espaciales son famosos por la minuciosidad de sus controles de seguridad y sus procedimientos supervisados. Así y todo, el *desastre del transbordador espacial Columbia* acabó con siete vidas y destruyó investigaciones e inversiones tasadas en los millones de dólares, en un accidente que técnicos e ingenieros de la NASA no supieron prevenir.

Figura 1.1: logo del bug Heartbleed[†]

Los límites de la revisión humana son innegables. Se pueden realizar inspecciones, se pueden seguir protocolos, se puede revisar código; pero el factor subjetivo, ese inyector de fallas imposible de medir, estará presente siempre que haya seres humanos involucrados en el proceso. Es por ello que las garantías formales han ganado popularidad a lo largo del último cuarto de siglo [CW96]. Desde la rigurosidad de la lógica y la matemática, contar con técnicas que aseguran que un *modelo del sistema* satisface ciertas propiedades vitales, no sólo es beneficioso sino también cada vez más necesario en el mundo moderno. Dos de los tres incidentes mencionados ocurrieron hace menos de seis años, lo que da cuenta de la actualidad de esta afirmación.

Model checking ("chequeo de modelos" en inglés) es un ejemplo prominente. Los procedimientos de verificación del model checking se basan en una exploración exhaustiva del espacio de estados del modelo del sistema [CES86, BK08, Har15]. El usuario provee el modelo y una formalización de la propiedad que desea verificar, y los procedimientos de model checking indican si la propiedad es válida o no en el modelo (típicamente para consultas cualitativas), y en ciertos escenarios también pueden medir hasta qué punto es satisfecha la propiedad (consultas cuantitativas).

No obstante, los resultados de estas pruebas formales sólo son válidas en los modelos donde se realizaron las verificaciones. Por ende mientras más realista sea el modelo, más útil resulta el resultado. Así, desde sus comienzos discretos y deterministas con el álgebra de procesos y los sistemas de transiciones, los análisis se han vuelto más complejos para incluir comportamiento no-determinista, probabilidades discretas, tiempo continuo, e incluso comportamiento estocástico continuo.

El precio a pagar por tales avances son técnicas de verificación cada vez más complejas y un espacio de estados que no deja de crecer, cuyo almacenamiento en la memoria física de la computadora puede fácilmente volverse inviable. Enfocándonos en la dimensión del espacio de estados, varias técnicas de reducción son actualmente conocidas para trabajar sobre una abstracción de la descripción original del modelo. Ejemplos de tales técnicas son *computer slicing* [Wei84], reducción de orden parcial [Val90], reducción por confluencia [BvdP02], y muchas otras [Bry86, CFM+93, dAKN+00, DJJL02, DN04, BGC04]. Varias de estas técnicas se basan en verificar modelos reducidos, relacionados con el modelo original a través de una *relación de bisimulación* [Mil89]. Desgraciadamente y con bastante frecuencia, las hipótesis sobre las que estas técnicas se basan son un tanto restrictivas, e.g. describir el comportamiento estocástico del sistema empleando únicamente funciones de densidad de probabilidad con la propiedad de pérdida de memoria [BK08]. Otro problema conocido del que muchos procedimientos de minimización

[†] Por Leena Snidate / Codenomicon - http://heartbleed.com/heartbleed.svg

sufren, es requerir acceso a todo los estados alcanzables, lo cual luego resulta en mejores tiempos de verificación pero definitivamente no soluciona el problema de la dimensión del espacio de estados [Har15].

Hay otro camino, popularmente conocido como *model checking estadístico* [YS02, LDB10], que puede operar sin esta problemática exploración del espacio completo de estados. Dicha técnica, que difiere considerablemente del model checking *estándar* previamente mencionado, se basa en la producción aleatoria de ejecuciones del (modelo del) sistema. Cada ejecución generada es interpretada como una nueva muestra independiente del comportamiento del sistema, almacenada para aumentar una *muestra aleatoria* del mismo. Dicha muestra aleatoria es luego analizada estadísticamente para otorgarle al usuario una respuesta tentativa a su consulta.

La naturaleza de esta respuesta es muy diferente a aquella que puede producir el model checking estándar, el cual tiene certeza (o como mínimo está seguro de no tenerla) de su veredicto final [BK08]. En cambio, el model checking estadístico genera una estimación de la respuesta, en la cual el usuario puede confiar con cierta noción (cuantitativa y medible) de confianza. A causa de sus orígenes estadísticos, esta estimación es usualmente dada en términos de un intervalo, dentro del cual el verdadero valor de la propiedad consultada por el usuario podría encontrarse. Por consiguiente, cuando el usuario desee una respuesta más confiable, puede solicitar la producción de intervalos más ajustados y con un mayor grado de confianza [LDB10].

Dado que las muestras son producidas y analizadas sobre la marcha, esta técnica no necesita representar el espacio completo de estados del modelo del sistema. En consecuencia, los problemas derivados de su gran tamaño son evitados. Como contrapartida los tiempos de cómputo usualmente son mayores en el model checking estadístico, lo cual está ligado a la producción de trazas de ejecución del sistema. Puede ocurrir que un gran número de nuevas muestras provea muy poca nueva información, y por ende que la estimación progrese lentamente. Esta situación se exacerba cuando la propiedad estudiada depende de la observación de un *evento raro*, cuya ocurrencia en trazas generadas aleatoriamente es muy poco probable. Existe un campo entero de investigación, conocido como *simulación de eventos raros*[‡], cuyo objetivo específico es contrarrestar este tipo de escenarios [RT09b].

1.1. Motivaciones y metas

Hay dos técnicas que sobresalen en el campo de la simulación de eventos raros: muestreo por importancia y división por importancia. *El muestreo por importancia* (*importance sampling* en inglés, [GI89, Hei95, JS06]) juega con el comportamiento estocástico del sistema de formas reversibles. De esta forma la probabilidad de observar el evento raro en una traza de ejecución generada aleatoriamente es

[‡] Aquí "simulación" significa la generación aleatoria de trazas de ejecución del sistema; no está relacionado con la noción de "bisimulación" previamente mencionada.

incrementada, y las estimaciones progresan a velocidades más razonables. *La división por importancia* (*importance splitting* en inglés, [KH51, Bay70, VAVA91]) no modifica el modelo original sino que altera la forma de simular, clonando las simulaciones (e.g. trazas de ejecución) que prometen generar un evento raro, y truncando las que se desvían del mismo. De esta forma la mayoría del esfuerzo computacional se emplea en la producción de muestras ricas en información.

Cada técnica tiene sus ventajas y sus desventajas, complementándose entre ellas de cierta forma, como se discutirá más adelante en Sección 2.4. No obstante, la reversibilidad del cambio de medida requerido por el muestreo por importancia es difícil de llevar a cabo de forma sistemática. Incluso si no nos preocupásemos por una potencial automatización, la mayoría de las estrategias conocidas se ven forzadas al estudio de sistemas markovianos, debido a lo difícil que resulta generar un cambio de medida provechoso y reversible. Ver por ejemplo [GSH$^+$92, LT11] y [LMT09]. Estas características van en sentido contrario a las motivaciones generales de esta tesis, las cuales describiremos a continuación.

Una de las mayores atracciones del model checking estándar es lo que vulgarmente se conoce como *push-button approach*: ni bien el modelo y la propiedad a consultar han sido especificados, el usuario puede obtener las respuestas que desea de forma completamente automática. Consideramos que esto es una clara ventaja del método por sobre otras técnicas formales como la prueba asistida de teoremas. **Por consiguiente, buscaremos desarrollar procedimientos lo más cercanamente posible a tal automatización.**

Sin embargo, el model checking estándar sufre del infame *problema de explosión de estados*, que fuerza a sus usuarios a aplicar e.g. reducciones por bisimulación y otras estrategias con equivalente fin. Es de suma importancia reducir la representación del modelo, forzándolo a que quepa en la memoria física de la computadora, para poder aplicar los algoritmos de verificación. **Nosotros buscaremos evitar completamente este problema, recurriendo al análisis de modelos por simulaciones** (i.e. model checking estadístico).

Existe otra ventaja en la elección de simulación por sobre model checking estándar, relacionada con la amplitud del tipo de modelos abarcados por cada técnica. Decididamente, deseamos ser tan generales y abarcadores como nos sea posible. Los primeros algoritmos de model checking sólo podían lidiar con sistemas markovianos, lo cual es demasiado restrictivo para nuestras intenciones. La situación a mejorado con los años, derivando en "múltiples soluciones para múltiples formalismos" (ver The Modest Toolset en [Har15]). Aún así y en contraste, *dejando de lado el no-determinismo*, las técnicas de análisis por simulación pueden ser trivialmente extendidas para lidiar con todo tipo de comportamiento probabilístico, temporizado, o estocástico (continuo). Haremos de esto una nueva motivación: **nuestro producto final debería ser fácil de aplicar a tantos tipos de modelos como nos sea posible.**

Más aún, nos interesan los desafíos planteados por el análisis de sistema en regímenes de eventos raros, donde la generación de trazas de ejecución no puede llevarse a cabo con técnicas Monte Carlo estándar, so pena de que la estimación tarde demasiado en converger a un resultado útil. **En ese sentido nos ocuparemos**

de implementar técnicas de simulación eficientes, más específicamente que empleen división por importancia, la cual se adecúa formidablemente a nuestros intereses.

> Resumiendo, las motivaciones generales de esta tesis involucran el desarrollo de técnicas automáticas para análisis de (modelos de) sistemas, usando simulación y análisis estadístico de trazas de ejecución. Los sistemas modelados deben ser tan generales como sea posible, pero las propiedades estudiadas deben incluir algún evento raro, difícil de observar cuando se generen las trazas de ejecución. A su vez y más específicamente, enfocaremos nuestros estudios en el perfeccionamiento de la técnica de división por importancia, armonizándola con estas motivaciones.

El incremento de eficiencia derivado del uso de división por importancia yace en la elección de una *función de importancia* adecuada [VAVA91, VAVA02, Gar00, LLGLT09]. Es esta función la que indica cuándo una traza de ejecución se acerca al evento raro, y cuándo se está alejando de él. Por ende, obviando algunos detalles técnicos, podemos pensar que que la elección de una función de importancia eficiente es equivalente a tener una buena implementación de división por importancia.

Cuando se decide abordar la simulación de eventos raros mediante el uso de métodos de división por importancia, el usuario suele quedar a cargo de proveer de manera ad hoc su propia función de importancia, junto con el modelo del sistema y las propiedades a estudiar en él [VAVA91, CAB05, LLGLT09]. Sin embargo y en vista de las motivaciones generales expresadas, en esta tesis deseamos automatizar la construcción de tal función, sin que el usuario deba intervenir directamente en el proceso.

Además es notorio el hecho de que muchos estudios en materia bibliográfica referente a la simulación de eventos raros, más prominentemente aquellos dedicados a la técnica de muestreo por importancia, formalizan alguna medida de la eficiencia de la estrategia presentada. A manera de ejemplo mencionamos los trabajos [GSH$^+$92, GHSZ98, KN99].

Esta clase de estudios se interesan por implementar métodos con eficiencia óptima o asintótica (también conocida como eficiencia logarítmica [LMT09]). En general, contar con mecanismos de simulación que presenten tales propiedades es de suma utilidad. Esto se debe a que los mismos ofrecen garantías formales de convergencia, asegurando que la estimación convergerá rápidamente—o tan rápidamente como sea posible—de forma independiente al grado de rareza del evento estudiado.

Por desgracia, tales estudios suelen especializarse en los sistemas específicos que están siendo tratados, desarrollando hipótesis restrictivas que descartan la generalización en varias direcciones. Tal especificidad es por supuesto inevitable si se desea obtener semejante grado de eficiencia. El desarrollo de métodos con eficiencia óptima requiere pruebas formales de que la varianza del estimador es mínima en la configuración escogida. Lograr una eficiencia logarítmica requiere pruebas de que dicha varianza crece polinomialmente a medida que la rareza del evento crece exponencialmente (ver la Sección 2.4). Por consiguiente este

tipo de resultados implica amoldar el método de simulación en alto grado a la clase de sistemas considerados, lo cual se encuentra en evidente conflicto con las motivaciones generales mencionadas.

En vista de las últimas reflexiones listamos los objetivos específicos de la tesis:

- desarrollar algoritmos para construir la función de importancia empleada por las técnicas de división por importancia,
  - ▷ estos algoritmos deben tomar los mismos datos de entrada que se requerirían para analizar el modelo con simulación estándar;
- embeber esta función en un procedimiento, automatizado hasta el nivel "push-button", que implemente la división por importancia;
- construir una herramienta de software que implemente dicho procedimiento;
- dar pruebas empíricas de la eficiencia de nuestras estrategias,
  - ▷ buscamos que nuestra implementación sea más eficiente que el uso de técnicas de Monte Carlo estándar,
  - ▷ no se busca lograr métodos óptimos ni asintóticamente eficientes,
  - ▷ cuando sea posible, los resultados obtenidos deberán ser validados contra datos verificados en la bibliografía,
  - ▷ la experimentación deberá desarrollarse en modelos diversos, incluyendo sistemas no markovianos.

1.2. Trabajo relacionado

Existe cierto número de estudios en la actualidad que se encaminan en esta misma dirección. Primero y principal [JLS13] comparten varias de nuestras motivaciones generales. También proponen la derivación de una función de importancia, llamada *función de puntaje* en [JLS13, JLST15], a partir de las mismas entradas que requiere el model checking estadístico. Ellos se enfocan en la propiedad consultada por el usuario, la cual debe ser replanteada equivalentemente de forma que puedan identificarse "capas" o "niveles" en ella. Así, el valor de importancia (o puntaje) de un estado del sistema está dado por el número de capas de la propiedad que el mismo satisface.

Esta idea le otorga poca relevancia al modelo estudiado cuando se deriva la función de puntaje. Creemos sin embargo que la estructura del modelo debería ser considerada para realizar dicha derivación. Además, si la propiedad consultada por el usuario no puede ser replanteada de la forma propuesta en [JLS13], se debe recurrir al uso de heurísticas aproximativas.

En [ZM12] y [RdBSH13] el formalismo de modelado son las Redes de Petri Estocásticas (SPN por sus siglas en inglés). Ambos trabajos usan la estructura de la red para favorecer ciertas simulaciones prometedoras en un régimen de evento

raro; una comparación entre ambos trabajos puede hallarse en [ZRWCL16]. En particular, en [ZM12] se deriva una heurística que aproxima una medida de la distancia entre un *marking* arbitrario y los markings que satisfacen la propiedad estudiada. Esa información se utiliza para derivar una función de importancia, siguiendo una estrategia comparable con la que introducimos en el Capítulo 3 de esta tesis.

Sin embargo, ciertas decisiones tomadas por [ZM12] se alcanzan mediante algoritmos de Programación Lineal, aplicable sólo a una clase restringida de SPN (los *T-semiflujos libremente relacionados*, según se aclara en [ZRWCL16]). Estas decisiones involucran aspectos centrales como la selección del *factor de división* y de los valores de importancia *umbrales* para la aplicación de RESTART (un mecanismo de división por importancia en particular). Las motivaciones generales de esta tesis apuntan a un campo de aplicación de mayor amplitud. De otro modo, dejando de lado el uso de SPN, los métodos descriptos en [ZM12, Sec. IV] tienen cierta correspondencia con nuestras propuestas en la Subsección 3.2.3.

En [RdBSH13] se enfocan en el estudio de muestreo por importancia en lugar de la división por importancia, si bien afirman que la función de distancia derivada mediante el método presentado podría ser usada para implementar división por importancia. La idea es aplicar la estrategia de Booth & Hendriks como se encuentra descripta en [LDT07], midiendo la distancia entre markings arbitrarios y el evento raro. Así logran aceleraciones en la simulación de eventos raros sin expandir el espacio completo de estados.

El método desarrollado en [RdBSH13] es muy elegante, pero depende de: trabajar exclusivamente con demoras de disparo markovianas; poder parametrizar la *intensidad* de todas las transiciones mediante algún parámetro de rareza; y resolver varias instancias de Programación Lineal Entera (un problema NP-completo). En [RdBSH13] no se reportan tiempos de simulación, pero sí se lo hace en [ZRWCL16], mostrando una aplicación efectiva de la estrategia. Aún así, en ese mismo trabajo se mencionan problemas de cómputo para modelos de mayor tamaño de los que aparecen en la publicación. Además el método completo está restringido a SPN markovianas, y un objetivo específico de esta tesis es poder operar sobre sistemas no markovianos.

Barbot también se enfoca en [Bar14] en operar sobre SPN mediante muestreo por importancia. Por lo demás, muchas de sus motivaciones y objetivos coinciden con los de esta tesis. Restringiendo sus estudios al campo de los procesos markovianos, la tesis de doctorado de Barbot provee pruebas formales de reducción de varianza en varios escenarios. Además, en la última parte de [Bar14], Barbot ejemplifica empíricamente la eficiencia de sus propuestas, corriendo un benchmark de muestreo por importancia mediante una herramienta de software que implementa su técnica.

Por último, señalamos que todos estos trabajos mencionados (al igual que esta tesis) se basan en un análisis estático del modelo y/o las propiedades provistos por el usuario. En cambio en [GVOK02] se le asigna importancia a los estados, es decir se construye la función de importancia, aplicando simulación invertida secuencialmente sobre todos los estados que conforman al sistema. Esto requiere

ciertos conocimientos a priori acerca de la distribución estacionaria del modelo, y la aplicabilidad del método se muestra sobre cadenas de Markov finitas de tiempo discreto.

1.3. Contribuciones y esquema de la tesis

Además de esta introducción, de las conclusiones, y de algunos apéndices finales, esta tesis está organizada en tres capítulos extensivos.

El Capítulo 2 cubre los aspectos teóricos fundamentales requeridos para seguir la tesis. El Capítulo 2 es autosuficiente, dejando de lado algunas referencias a los apéndices. Más precisamente:

Las Secciones 2.1 y 2.2 ofrecen un vistazo de varios formalismos de modelado y lógicas temporales, usados para analizar y consultar las propiedades que exhibe un modelo de sistema.

La Sección 2.3 repasa algunas técnicas conocidas que automatizan las verificaciones y chequeos de dichos modelos. Usando las motivaciones generales de la Sección 1.1 como guía, construimos un camino a través de una variedad de estrategias y algoritmos. Al hacerlo identificamos las fortalezas y debilidades de cada técnica con respecto a nuestras intenciones de aplicación.

La Sección 2.4 introduce formalmente la simulación de eventos raros, y motiva la elección de un criterio de parada para las estimaciones, usado más adelante en las experimentaciones. Esta sección justifica la transición del ámbito general de las secciones 2.1 a 2.3, al campo más específico de las secciones 2.5 a 2.7.

Las Secciones 2.5 y 2.6 introducen primero la división por importancia en un carácter formal, luego presentan un amplio panorama de implementaciones conocidas, y finalmente se enfocan en la técnica que será usada durante las experimentaciones.

La Sección 2.7 explora los límites de la división por importancia e identifica algunas preguntas abiertas en el área. La discusión final conecta todas las nociones presentadas a lo largo del capítulo con los objetivos específicos de la tesis.

El Capítulo 3 presenta nuestra primera estrategia (monolítica), desde las ideas originales que la motivaron, hasta los resultados numéricos de la experimentación con casos de estudio tomados de la bibliografía en el área. Más en detalle:

La Sección 3.1 reflexiona sobre el rol crítico de la función de importancia a la hora de obtener una buena implementación del método de división por importancia. Se usan ejemplos para introducir e ilustrar los tópicos sensibles, que se abordan luego en las siguientes secciones.

La Sección 3.2 presenta nuestro primer algoritmo, diseñado para cumplir con el primer objetivo específico que se detalló en Sección 1.1: derivar una función de importancia a partir de otras entradas provistas por el usuario. El entorno de aplicación es descripto formalmente y se provee una prueba de terminación del algoritmo.

La Sección 3.3 desarrolla un marco completo para implementar una aplicación automatizable de división por importancia. Esto tiene lugar desde una perspectiva de modelo monolítico, inherente al algoritmo presentado en la Sección 3.2. En particular la Subsección 3.3.3 introduce el algoritmo que usaremos para seleccionar los umbrales requeridos para aplicar división por importancia. Todo esto responde a nuestro segundo objetivo específico.

Las Secciones 3.4 y 3.5 se ubican en el plano empírico, introduciendo la primer herramienta de software desarrollada durante la evolución de esta tesis. La herramienta es utilizada en Sección 3.5 para experimentar con varios casos de estudio markovianos tomados de la bibliografía. El resultado de estos experimentos sirve para validar el correcto funcionamiento de la herramienta, y para dar muestras prácticas de la eficiencia de nuestras técnicas. Así alcanzamos los dos últimos objetivos específicos, aunque el segundo sólo es cubierto parcialmente, puesto que faltaría experimentar sobre sistemas no markovianos.

La Sección 3.6 concluye el capítulo analizando las limitaciones de la estrategia en él propuesta. La más severa es la necesidad de generar el espacio de estados del modelo compuesto, inherente a la naturaleza monolítica de dicha estrategia. Si bien la mayoría de los objetivos son satisfactoriamente alcanzados en el Capítulo 3, el problema de la *explosión de estados* es sumamente restrictivo en materia de su aplicabilidad. Esto nos incentivó a buscar nuevas soluciones, originando la investigación que reportamos en el Capítulo 4.

El Capítulo 4 introduce un segundo método (composicional) para automatizar la división por importancia, que intenta resolver o al menos mitigar los problemas en los que incurre la estrategia monolítica desarrollada en el Capítulo 3. Los temas principales tratados en este capítulo se organizan de la siguiente forma:

La Sección 4.1 explora los cimientos de una estrategia composicional. Discute ciertos aspectos que deben ser tratados cuando se deriva una función de importancia de naturaleza distribuida, y formula dos desafíos concretos.

Las Secciones 4.2 y 4.3 responden a los desafíos de la Sección 4.1, satisfaciendo el primer objetivo específico de la Sección 1.1 en este nuevo escenario, viz. derivar una función de importancia composicional. Más específicamente, Sección 4.2 presenta una forma de descomponer la propiedad (global) consultada por el usuario, con el fin de construir funciones de importancia locales a cada componente del sistema. Se provee un algoritmo que encaja en el entorno general del Capítulo 3. Luego Sección 4.3 presenta varias estrategias de re-composición del conjunto resultante de funciones locales de importancia, para así obtener una función global de importancia a ser usada en las simulaciones. La Sección 4.3 también incluye una comparación entre la estrategia monolítica del capítulo anterior, y la estrategia composicional de este capítulo.

La Sección 4.4 Presenta un formalismo de modelado desarrollado recientemente, llamado IOSA por sus siglas en inglés, enteramente libre de las restricciones markovianas que restringen al formalismo usado en el Capítulo 3. IOSA es la base sobre la cual se construyen todas las aplicaciones prácticas del Capítulo 4.

La Sección 4.5 proyecta las propuestas y resultados de las secciones anteriores en un plano más empírico: la Subsección 4.5.2 describe una sintaxis concreta para el formalismo IOSA, y la Subsección 4.5.3 presenta la segunda herramienta de software desarrollada durante las investigaciones de esta tesis. Los objetivos dos y tres de la Sección 1.1 son así satisfechos.

La Sección 4.6 presenta datos empíricos que evidencian la aplicabilidad y eficiencia de la estrategia composicional desarrollada en este capítulo. Dado que IOSA puede operar con distribuciones estocásticas arbitrarias, algunos modelos estudiados no son markovianos, con lo cual alcanzamos el último objetivo específico en toda su extensión.

El Capítulo 5 reflexiona sobre los resultados generales obtenidos en este trabajo, y menciona posibles continuaciones para mejorar algunos de ellos y para extender su campo de aplicación.

Los Apéndices A a C son anexos que contribuyen a la reproducibilidad de nuestros experimentos, y que revisan superficialmente las nociones más formales detrás de algunas teorías en las que esta tesis se basa. Concretamente:

El Apéndice A incluye el código de todos los modelos de sistemas usados para producir los resultados numéricos presentados. Se usan dos lenguajes de modelado: los sistemas estudiados en el Capítulo 3 se expresan en el lenguaje de entrada de PRISM; los del Capítulo 4 se expresan en la sintaxis de modelado de IOSA.

El Apéndice B añade a la tesis algunas definiciones y resultados elementales de teoría de la medida, requeridos para comprender el Apéndice C y los aspectos más formales del Capítulo 2.

El Apéndice C presenta las nociones básicas del formalismo NLMP, que conforma la base sobre la cual se le da semántica al formalismo IOSA del Capítulo 4.

Marco teórico 2

En este capítulo se cubren superficialmente varios conceptos fundamentales para la comprensión de esta tesis. Referimos al lector interesado en una entendimiento más profundo de los temas que aquí se introducen al siguiente (excelente) material:

- *Principles of Model Checking*, por Christel Baier y Joost-Pieter Katoen, [BK08], donde diversos aspectos del modelado y la verificación de sistemas son explicados sobre sólidas bases matemáticas y computacionales;
- *Rare Event Simulation using Monte Carlo Methods*, editado por Gerardo Rubino y Bruno Tuffin, [RT09b], una monografía sobre simulación de eventos raros (RES por sus siglas en inglés) resultante del esfuerzo colaborativo de varias autoridades del área;
- *The splitting method in rare event simulation*, la tesis doctoral de Marnix Garvels, [Gar00], que ofrece un análisis de gran profundidad sobre la aplicación de división por importancia para resolver el problema de RES.

Recomendamos al lector que le dedique un tiempo a la lectura de este capítulo, incluso si ya se encuentra familiarizado con las nociones que en él se introducen. La intención trasciende la mera presentación del material teórico necesario, buscando asimismo revisar los conceptos y preguntas abiertas que motivaron esta tesis. Esto se expone de forma convergente, comenzando sobre la generalidad del modelado y la verificación formal de sistemas, y culminando con la (necesidad de la) derivación de una función de importancia para aplicar división por importancia como solución al problema de la simulación de eventos raros.

2.1. Modelado de sistemas

Existe una técnica para estudiar y comprender los sistemas que diseñamos, que tiene el atractivo beneficio de la automatización (parcial), y que puede proveer garantías sobre los resultados que genera: *el modelado y la verificación formal*. Para seguir esta estrategia la funcionalidad primaria del sistema debe ser interpretada y descripta en términos de algún lenguaje formal, lo cual constituye la fase (en general) no automatizable. Este proceso de abstracción no es para nada trivial, una de cuyas muchas dificultades yace en la elección de los componentes y comportamientos relevantes para incluir en la abstracción. No obstante los

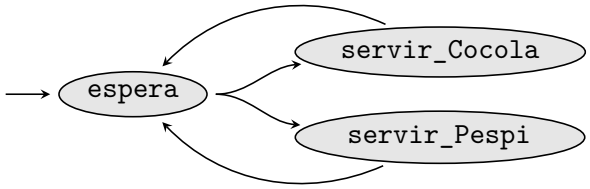


Figura 2.1: Máquina expendedora (versión pirata)

resultados valen el esfuerzo: ni bien se cuenta con un modelo formal, muchos estudios pueden llevarse a cabo tan sólo presionando un botón. Vale la pena mencionar que sí existen algunas técnicas para extraer automáticamente modelos de algunos sistemas a partir de una descripción formal de los mismos, como ser el código fuente en el caso de programas de software, siempre y cuando se disponga de tal descripción.

Se han desarrollado numerosos formalismos de modelado para expresar los diversos aspectos en que un sistema puede ser descripto y analizado, los cuales varían de acuerdo al ángulo de estudio. Muchos adoptan una estrategia basada en autómatas, donde el concepto de *estado* describe la "configuración actual de los componentes", que va evolucionando acorde a alguna dinámica descripta formalmente y por tanto libre de ambigüedades[†]. Así, desde el estado actual s, el autómata que describe al sistema puede moverse (o *transitar*) hacia un próximo estado s' siguiendo alguna *función de transición* (o relación de transición), lo cual típicamente se denota $s \to s'$.

Desde esta perspectiva basada en estados el *no determinismo* surge naturalmente al usar abstracciones, e.g. cuando el sistema es afectado por un ambiente inespecificado, o cuando varios componentes corren en paralelo y sólo es de interés el comportamiento global conjunto. A manera de ejemplo consideremos una máquina expendedora de refrescos bebibles, donde un cliente puede solicitar una *Pespi* o una *Cocola*. Para simplificar la cuestión supongamos que Gottfrid Svartholm y sus secuaces hackearon los circuitos de forma que no se requiere un pago; es posible obtener uno de estos refrescos tan sólo presionando el botón correspondiente. En un modelo que se abstrae del cliente y considera exclusivamente a la máquina expendedora, no hay manera de predecir cuál bebida será escogida a continuación. Por lo tanto desde un estado de `espera` hay una *elección no determinista* entre un próximo estado `servir_Pespi` y un próximo estado `servir_Cocola`. La Figura 2.1 es una representación gráfica del proceso.

Al igual que en este ejemplo de juguete, el no determinismo puede ser descripto como la elección de un próximo estado entre un conjunto de posibilidades, viz. las transiciones habilitadas en cada estado son provistas sin ninguna información adicional. El formalismo de modelado de sistemas de transiciones etiquetados es uno de los más conocidos en el campo de la verificación formal de sistemas, uno de cuyos principales propósitos es describir elecciones no deterministas de transiciones entre estados.

[†] Como alternativa sin la noción de estado ver e.g. el cálculo λ [Bar84] y el lenguaje Haskell.

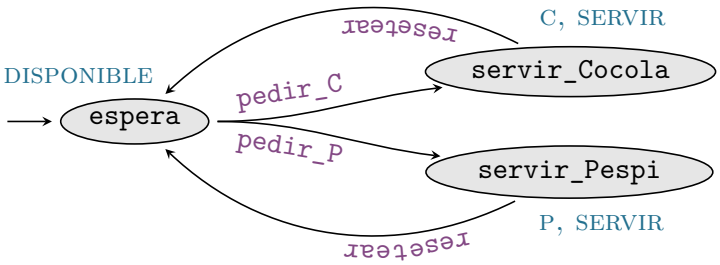


Figura 2.2: LTS de la máquina expendedora

Definición 1 (LTS)**.** Un *Sistema de Transiciones Etiquetado* (LTS por sus siglas en inglés) es una tupla $(S, s_0, A, \rightarrow, AP, Lab)$ donde:

- $S \neq \varnothing$ es un conjunto finito de *estados*;
- $s_0 \in S$ es el *estado inicial* del sistema;
- A es el conjunto de *acciones* o *etiquetas*;
- $\rightarrow \subseteq S \times A \times S$ es la *relación de transición*;
- $AP \neq \varnothing$ es un conjunto de *proposiciones atómicas*;
- $Lab\colon S \rightarrow 2^{AP}$ es una *función de identificación*.

Considerar un único estado inicial y conjuntos S y A finitos satisface los requisitos de esta tesis, si bien los Sistemas de Transiciones Etiquetados pueden definirse en términos más generales. Para una introducción más completa a esta clase de estructuras se refiere al lector a [BK08, Sec. 2.1].

En la Definición 1 las transiciones del sistema son definidas por medio de la relación $\rightarrow$. El elemento $(s, a, s') \in \rightarrow$ se denota $s \xrightarrow{a} s'$. Cuando tal transición existe decimos que s' es un *sucesor* de s, y que s es un *antecesor* de s'. Nótese que el dominio de la función de identificación Lab son los estados del sistema y que es independiente de las acciones A. Esta función relaciona un conjunto $Lab(s) \subseteq AP$ de proposiciones atómicas con el estado $s \in S$, que simboliza las propiedades satisfechas por el estado (y que por ende lo identifican).

La Figura 2.2 muestra la máquina expendedora de refrescos bebibles modelada como un LTS. Cada transición se encuentra decorada con una acción, y para $AP = \{\text{P}, \text{C}, \text{SERVIR}, \text{DISPONIBLE}\}$ cada estado está etiquetado de acuerdo a las propiedades que satisface. Por ejemplo `servir_Cocola` $\xrightarrow{\texttt{resetear}}$ `espera` indica que `espera` es un sucesor de `servir_Cocola`, y la ubicación de la proposición atómica ‘DISPONIBLE’ indica que la máquina se encuentra disponible sólo cuando el estado actual en el modelo LTS es `espera`.

El conjunto de acciones A es provisto a secas, sin ningún tipo de información adicional (e.g. sin un orden). En el ejemplo de la máquina expendedora esto significa que cuando el estado actual es `espera`, no hay información a priori que indique si el sistema evolucionará siguiendo la transición `pedir_P` o la transición `pedir_C`.

Dicha elección es no determinista. Un concepto cercano al de no determinismo es el de probabilidad. Dependiendo de la naturaleza discreta o continua del espacio de estados subyacente, los términos *elección probabilista* o *elección estocástica* son empleados respectivamente para indicar que se provee alguna cuantificación en lo concerniente a las transiciones entre estados.

Comportamientos probabilísticos y estocásticos se hallan de forma natural en incontables situaciones de la vida real, desde colas de espera en supermercados hasta la formación de nubes y la falla y reemplazo de componentes en servicios de almacenamiento en la nube. En el caso discreto, una función de masa de probabilidad cuantifica la elección del siguiente estado. Por ejemplo si los estados sucesores de s son s_1, s_2, s_3 con probabilidad $1/2$, $1/4$, $1/4$ respectivamente, al realizar $1 \ll N < \infty$ transiciones desde el estado s *esperamos* observar, aproximadamente, $N/2$, $N/4$, $N/4$ elecciones de los estados s_1, s_2, y s_3 respectivamente. Si añadimos "*auto-ciclos*" a $\{s_1, s_2, s_3\}$ (i.e. transiciones $s_1 \to s_1, s_2 \to s_2, \ldots$), estas transiciones cuantificadas pueden condensarse en una *matriz de transición*:

	s	s_1	s_2	s_3
s	0	$1/2$	$1/4$	$1/4$
s_1	0	1	0	0
s_2	0	0	1	0
s_3	0	0	0	1

.

Las cadenas de Markov ofrecen una descripción matemática ampliamente estudiada para referirse a comportamientos probabilísticos. Cuando la noción discreta de *paso* en lugar de un *tiempo continuo* es inherente a la evolución del proceso, la variante de tiempo discreto de las cadenas de Markov puede ser usada para modelar el sistema.

Definición 2 (DTMC)**.** Una *cadena de Markov de tiempo discreto* finita (DTMC por sus siglas en inglés), es una tupla $(S, s_0, \mathbf{P}, AP, Lab)$ donde:

- S, s_0, AP, y Lab son como en la Definición 1 de LTS;
- $\mathbf{P}\colon S \times S \to [0, 1]$ es una *función de probabilidad de transición* que para todo $s \in S$ satisface $\sum_{s' \in S} \mathbf{P}(s, s') = 1$.

Las transiciones del sistema se definen pues a través de $\mathbf{P}$, que formaliza la matriz de transición ilustrada anteriormente. El valor $\mathbf{P}(s, s') \in [0, 1]$ especifica para cada estado $s \in S$ la probabilidad de tomar la transición $s \to s'$. Tal transición *existe* sii $\mathbf{P}(s, s') > 0$, en cuyo caso s' es un sucesor de s y s es un antecesor de s', análogamente al caso de los LTS. La restricción impuesta sobre $\mathbf{P}$ la convierte en una distribución de probabilidad en el espacio de estados, donde cada $\mathbf{P}(s, \cdot)\colon S \to [0, 1]$ es una medida de probabilidad sobre S. Estudios concienzudos sobre DTMC se encuentran e.g. en [Nor98, BK08].

Para dar un ejemplo concreto considérese una conexión de socket punto-a-punto en la Internet, con paquetes de datos enviados desde un extremo, que pueden ser perdidos en la red o recibidos en el otro extremo. Para estudiar la

proporción de transacciones exitosas, el total de paquetes enviados y cuántos fueron recibidos (en lugar del instante de tiempo en el que se los envió/recibió) provee toda la información relevante. Como el sistema evoluciona paso a paso, donde cada paso consiste en enviar, recibir, o perder un paquete, la Definición 2 cuenta con las herramientas para modelar el comportamiento buscado.

Típicamente, para implementar un DTMC se construye una *matriz de probabilidad de transiciones* (o matriz de transición), donde se explicita la probabilidad de transitar entre cualquier par de estados. Esta información basta para sistemas donde el comportamiento futuro depende exclusivamente del estado actual y no del camino que condujo hasta él. Además se asume que el DTMC es finito y homogéneo, e.g. las probabilidades permanecen invariantes cuando se visita un estado por segunda vez.

La matriz de transición para los estados $\{s, s_1, s_2, s_3\}$ que fue ilustrada más arriba es precisamente una matriz de probabilidad de transiciones. Nótese que los modelos con N estados requieren una matriz cuadrada con N^2 números de punto flotante. *Tipos abstractos de datos* más eficientes en espacio de almacenamiento existen para aliviar esta necesidad, como por ejemplo representaciones de matrices ralas o diagramas binarios de decisión con terminales múltiples (MTBDD por sus siglas en inglés, [CFM$^+$93, BFG$^+$97]).

En comparación con el mundo discreto que tratamos hasta ahora, the stochastic scenario is more involved because heed must be paid to measurability issues. Due to the memoryless property and the ease of analysis it offers, systems where all transitions are governed by the exponential distribution have been studied thoroughly. La variante en tiempo continuo de las cadenas de Markov es la elección tradicional para modelar este tipo de sistemas.

Definición 3 (CTMC)**.** Una *cadena de Markov de tiempo continuo* finita (CTMC por sus siglas en inglés), es una tupla $(S, s_0, \mathbf{R}, AP, Lab)$ donde:

- S, s_0, AP, y Lab son como en la Definición 1 de LTS;
- $\mathbf{R}\colon S \times S \to \mathbb{R}_{\geqslant 0}$ es una *función de tasas de transición* que para todo $s \in S$ satisface $\mathbf{R}(s, s) = 0$.

En la Definición 3 y en contraposición con $\mathbf{P}$, la matriz $\mathbf{R}$ gives the exponential rates of taking transitions between states. Having $r = \mathbf{R}(s, s')$ means the probability of taking transition $s \to s'$ within t time units is $1 - e^{-rt}$. Notice that $r = 0$ yields a null probability, so as before the transition $s \to s'$ will be said to exist iff the matrix entry $\mathbf{R}(s, s') > 0$, with the corresponding definitions of predecessor an successor states. Además, como la distribución exponencial posee la propiedad de pérdida de memoria, basta con saber el estado actual para determinar los pasos futuros, al igual que ocurre con los DTMC.

La Definición 3 permite múltiples estados sucesores, i.e. para un estado $s \in S$ puede haber más de un otro estado $s' \in S$ t.q. $\mathbf{R}(s, s') > 0$. Estas situaciones son conocidas como *condiciones de carrera*. All outgoing transitions are enabled, and the first one to *fire* (according to a sampling of the corresponding exponential distributions) will be the one taking place. Un estudio más en profundida de las

cadenas de Markov de tiempo continuo se encuentra en [Nor98, Bre68].

Un ejemplo clásico de CTMC son las colas e.g. en una caja de supermercado. The state space S would be used to represent the number of clients in the queue, whereas the rate at which new clients arrive and the cashier performs the service is encoded in $\mathbf{R}$. En un modelo simple esto resultaría en una matriz de tasa de transiciones tridiagonal, con la diagonal principal nula.

Hasta aquí hemos introducido las nociones de evolución no determinista y probabilista/estocástica. There is another key concept which naturally arises in many situations, whose explicit representation may be required. De hecho este concepto ya ha sido mencionado, si bien sólo incidentalmente: el *pasaje del tiempo*.

Un DTMC codifica la noción de evolución temporal discreta, whereas a CTMC deals, also implicitly, with continuous time. But what if the exact time point of occurrence of events needs to be modelled? For instance the soda vending machine puede caer en un estado suspendido durante 2 segundos luego de que se pidió una Cocola.

En general, consideremos sistemas en un entorno temporal continuo donde stochastically sampled events occur, yet not necessarily following the exponential distribution. A usual modelling solution is to associate a unique variable to each distinct event, which can sample time according to the desired probability density function. Los autómatas estocásticos de [DK05] siguen esta idea.

Definición 4 (SA)**.** Un *Autómata Estocástico* finito (SA por sus siglas en inglés) es una tupla $(S, s_0, A, \mathcal{C}, \to, AP, Lab)$ donde:

- S, s_0, A, AP, y Lab son como en la Definición 1 de LTS;
- $\mathcal{C}$ es un conjunto finito de *relojes* tal que cada $c \in \mathcal{C}$ tiene una medida de probabilidad continua asociada $\mu_C \colon \mathbb{R} \to [0, 1]$ con soporte en $\mathbb{R}_{>0}$;
- $\to \subseteq S \times 2^{\mathcal{C}} \times A \times 2^{\mathcal{C}} \times S$ es la *relación de transición*.

Los relojes en los SA marcan el pasaje del tiempo de forma explícita: cada $c \in \mathcal{C}$ is assigned a positive value stochastically sampled from its associated probability density function μ_C, and decrease this value synchronously with the other clocks at constant speed, satisfying the differential equation $\dot{c} = -1$. When execution starts from s_0, initial values for all clocks are sampled from their respective probability measures. Puede considerarse que el tiempo se detiene para un reloj que a alcanzado el valor nulo.

A pesar de que la noción subyacente de tiempo es continua, and just like with DTMC and CTMC, the succession of events in the execution of a stochastic automaton is discrete. More specifically, their occurrence is controlled by the *expiration* of the clocks. If the system is in state s y hay una transición $s \xrightarrow{C,a,C'} s'$, la acción $a \in A$ will be performed after all clocks in C have expired, viz. reached the null value. Then the system moves to state s' sampling new values for the clocks in C' according to their probability measures. The *relojes disparadores* of the transition are those in C, and the *relojes de reseteo* are those in C'. If $C = \varnothing$ or all triggering clocks are null as state s was reached, la transición puede ejecutarse instantáneamente.

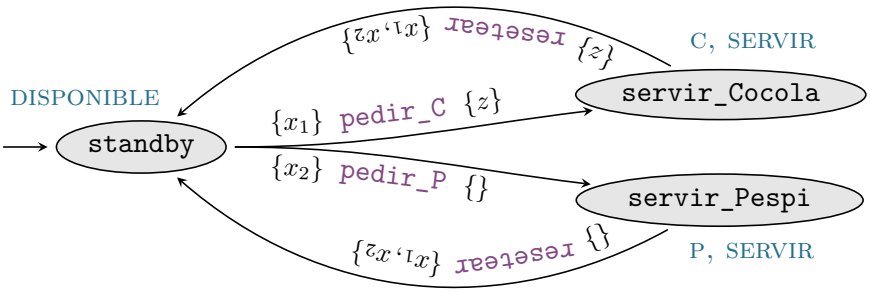


Figura 2.3: SA de la máquina expendedora

La Figura 2.3 muestra una representación posible de la máquina soda vending machine as a stochastic automaton. Los relojes x_1 y x_2 have exponential probability density functions associated, with rates λ_1 and λ_2 respectively. El reloj z instead samples its values from a continuous probability measure from the uniform distribution with support in [1.95,2.05]. Notice the state space is the same as in the LTS representation of la Figura 2.2, but the nondeterministic choice of the transitions outgoing state `espera` was replaced with a stochastic one, by the inclusion of the triggering clocks x_1 and x_2. Nótese que la máquina se suspende sirviendo por (aproximadamente) 2 unidades de tiempo cuando se pide una Cocola, lo cual está modelado en la transición que sale de `servir_Cocola` hacia `espera` empleando al reloj disparador z.

Es importante recalcar que el formalismo SA incluye nondeterministic behaviour, unlike DTMC and CTMC. That is clear from the definition of the transition relation $\rightarrow$. De cualquier estado s there can be several transitions with the same triggering clocks C, resetting clocks C', and action label a, reaching different target states s'. In later chapters a restricted version of la Definición 4 will be introduced, which rules out nondeterminism by construction. Concretamente, al imponer restricciones sobre la naturaleza de la relación $\rightarrow$, sólo puede expresarse comportamiento estocástico.

Los formalismos y ejemplos presentados fueron escogidos por su simpleza, con la intención de introducir los conceptos fundamentales de *no determinismo*, *probabilidad/estocasticidad*, y *evolución temporal*, de forma directamente aplicable a las necesidades de esta tesis. Es posible modelar sistemas mucho más generales, donde el tiempo transcurre a diferente velocidad para los distintas componentes que lo conforman, o donde comportamientos probabilísticos y no deterministas se entrelazan de forma discreta.

Existen muchos otros formalismos para satisfacer cada necesidad particular. Por ejemplo, los *procesos de decisión de Markov* (MDP por sus siglas en inglés, [Bel57]) mezclan el no determinismo de los LTS con las transiciones probabilistas de los DTMC, y los *autómatas estocásticos híbridos* (SHA por sus siglas en inglés, [BDHK06]) permiten una descripción heterogénea del transcurso del tiempo. Se recomienda la lectura de [Har15] al lector interesado en conocer de forma más amplia las opciones más estudiadas. En particular el esquema relacional presentado

en la Figura 1.2 de dicho trabajo resume sucintamente las relaciones lógicas entre varios formalismos existentes, en términos de los conceptos de comportamiento que pueden expresar.

2.2. Consultas de propiedades en los modelos

La intrincada tarea de destilar un modelo a partir de un sistema real no se realiza de puro gusto, sino para poder estudiar y comprender mejor las propiedades que el (modelo del) sistema exhibe. La intención principal suele ser asegurarse que todos los requerimientos serán satisfechos. También es posible obtener medidas de desempeño y eficiencia. Todo esto se logra *consultando* al modelo.

A grandes rasgos podemos hablar de consultas *cualitativas* y de consultas *cuantitativas*: las consultas cualitativas o funcionales tratan cuestiones absolutas—de respuesta binaria—como terminación o corrección funcional; las consultas cuantitativas se utilizan para estudiar aspectos de eficiencia como el consumo de energía o el tiempo requerido para alcanzar la terminación.

Considérese un servicio de tren ICE entre Hamburg Hbf [†] y Köln Hbf, con una única parada intermedia en Hannover Hbf. Consultas cualitativas para este ejemplo son *"¿puede saturarse el tren, dejando fuera algunos pasajeros en Hamburg Hbf?"* y *"¿siempre llega a destino el tren?"*. Consultas cuantitativas en cambio son *"¿cual es el promedio de pasajeros por trayecto?"*, *"¿qué probabilidad hay de encontrar un tren saturado en Hannover Hbf y perder el viaje?"*, y *"¿acaso más del 99 % de los viajes llegan a destino sin fallas mecánicas?"*.

Al igual que los sistemas, las consultas de propiedades también pueden ser especificadas en un lenguaje formal. Se desarrollan lógicas para describir las propiedades que se desean consultar, descriptas mediante proposiciones que respetan la correspondiente gramática. Las *lógicas temporales* son una elección usual debido a su expresividad: las gramáticas de estas lógicas producen descripciones sucintas de potenciales trazas de ejecución del sistema, i.e. de posibles sucesiones de estados permitidas por el modelo formal (de ahora en más denotadas *caminos*). Estos caminos podrían ser, en principio, infinitos.

Usando proposiciones derivadas de la gramática de una lógica temporal, conceptos relevantes para sistemas de la vida real como *alcanzabilidad* ("¿se puede llegar a esta situación?"), *seguridad* ("algo malo nunca ocurre"), y *liveness* ("algo bueno eventualmente ocurrirá"), pueden expresarse de forma clara y compacta.

Existen muchas lógicas entres las cuales escoger, donde la mejor opción depende del comportamiento que se desee describir. A continuación se listan y explican brevemente algunas variantes populares. No proveemos aquí definiciones formales para su sintaxis ni para su semántica; en su lugar, para generar una intuición de la forma en que estas lógicas hablan de ejecuciones de sistemas, se incluyen algunos ejemplos en cada caso.

La *Lógica Temporal Lineal* (LTL por sus siglas en inglés, [Pnu77]) permite razonar sobre una única línea temporal, viz. hay un único sucesor para cada

[†] Hauptbahnhof, i.e. estación principal de trenes.

estado en un camino de ejecución. LTL cuenta con los operadores de disyunción ($\vee$) y negación ($\neg$) de la lógica proposicional, y los operadores temporales modales *siguiente* ($\bigcirc$) y *hasta* (U). Los operadores proposicionales usuales se pueden derivar de $\{\vee, \neg\}$; algunos operadores temporales modales derivados de uso común son *eventualmente* ($\Diamond$) y *siempre* ($\Box$).

A las fórmulas de LTL se les da semántica sobre los caminos de ejecución del sistema. Para las Definiciones 1 a 4, las fórmulas describen los caminos a través de las proposiciones atómicas a ser visitadas. Resumiendo, $\bigcirc$ indica algo acerca del "(único) próximo estado en este camino", $\Diamond$ significa "algún estado más adelante en el camino" (viz. en el futuro). $\Box$ significa "todos los estados futuros, incluyendo el estado actual", y U dice que "algo ocurrirá en todos los estados del camino de ejecución actual, hasta que algo más finalmente ocurra". Ejemplos de fórmulas LTL son:

- $\bigcirc$ SERVIR *"lo próximo que acontecerá es la entrega de una bebida"*, así que en el LTS de la máquina expendedora de la Figura 2.2, esta fórmula es satisfecha por cualquier estado de un camino que se encuentre a una transición de distancia de los estados identificados con SERVIR, lo cual sólo se cumple para el estado `espera`.

- $\Diamond$ (SERVIR $\wedge$ C) *"eventualmente se entregará una Cocola"*, cierto cuando algún estado más adelante en el camino está identificado con ambos SERVIR y C (i.e. cuando `servir_Cocola` es alcanzable);

- $\Box \neg$ SATURADO *"el tren no está saturado y nunca lo estará"*, que en el caso del tren ICE es satisfecho por los estados de los caminos que *no* tienen delante suyo un estado identificado con SATURADO.

- $\neg$ SATURADO U KÖLN *"el tren no se saturará hasta llegar a Köln Hbf"*, donde el arribo eventual a Köln es necesario para satisfacer la propiedad.

La *Lógica de Árbol Computacional* (CTL por sus siglas en inglés, [CE82]) considera un escenario temporal ramificado, donde hay muchos potenciales sucesores para cada estado (y sobre los cuales se cuantifica). En comparación con LTL esto demanda una semántica basada en estados, dado que deben considerarse todos los caminos (en lugar de un único camino) con origen en el estado actual. Además de los operadores proposicionales y temporales de LTL, CTL cuenta con los operadores *para todos los caminos* ($\forall$) y *para algún camino* ($\exists$). El primero es satisfecho si *todos* los caminos originados en el estado actual satisfacen el resto de la fórmula (que debe comenzar con un operador temporal: $\bigcirc$, U, ...); el segundo es satisfecho si *algún* camino satisface el resto de la fórmula. Ejemplos de fórmulas CTL son:

- $\exists \Diamond$ SERVIR *"hay una ejecución del sistema originada en el estado actual, donde eventualmente se entregará una bebida"*, satisfecha en todos los estados del LTS de la máquina expendedora;

- $\forall\Diamond(\text{SERVIR} \wedge \text{C})$ *"en todas las ejecuciones del sistema a partir del estado actual, eventualmente un cliente será servido una Cocola"*, lo cual es falso en `espera` dado que hay una (extraña pero por demás válida) ejecución donde solo Pespis son escogidas:

$$\texttt{espera} \xrightarrow{\texttt{pedir\_P}} \texttt{servir\_Pespi} \xrightarrow{\texttt{resetear}} \texttt{espera} \xrightarrow{\texttt{pedir\_P}} \cdots \quad . \tag{1}$$

Vale la pena remarcar que si bien las fórmulas de LTL y CTL pueden emularse entre sí, en general la expresividad de estas lógicas es incomparable [BK08, Theo. 6.21]. For ejemplo la fórmula CTL $\forall\Diamond\forall\Box a$ no puede ser expresada en LTL, mientras que la fórmula LTL $\Diamond\Box a$ no puede ser expresada en CTL.

La *Lógica de Árbol Computacional Probabilista* (PCTL por sus siglas en inglés, [HJ94]) considera también la naturaleza probabilistic nature of the Markov chains it was designed to study. Whereas LTL and CTL talk about which states to visit and which not, PCTL allows the user to express the probability of following certain path. This logic is based on CTL with a major difference: the existential and universal quantifiers are replaced with a probabilistic operator (P), of which $\forall$ and $\exists$ could be considered special cases (even though that is not strictly true). La fórmula $\mathsf{P}_I(\phi)$, donde I es un intervalo de $[0, 1]$ con cotas racionales y ϕ es un camino, pregunta si la probabilidad de ejecutar a ϕ yace en I:

- $\mathsf{P}_{\geqslant 0,9}(\neg\,\text{SATURADO}\,\mathsf{U}\,\text{HANNOVER})$ *"la probabilidad de tener un tren ICE saturado en el primer trayecto Hamburg–Hannover es menor del 10 %"*, asumiendo que el tren efectivamente arriba a Hannover;

- $\mathsf{P}_{\leqslant 0,25}(\Diamond\,\text{SATURADO})$ *"a lo sumo un tren de cada cuatro puede saturarse"*, lo cual se aplica a cualquier punto del trayecto del tren;

- $\mathsf{P}_{>0}(\,\Box\,(\,\text{ESPERA} \vee \text{C}\,))$ *"podría ser que sólo se escojan Cocolas"*;

- $\mathsf{P}_{=1}(\,\Diamond\,(\text{SERVIR} \wedge \text{C}))$ *"eventualmente se pide una Cocola, casi guramente"*.

Curiosamente, el último ejemplo *no es equivalente* a la fórmula $\forall\Diamond(\text{SERVIR} \wedge \text{C})$ from CTL. There may exist paths with zero probability (see Apéndice B) where $\Diamond(\text{SERVIR} \wedge \text{C})$ is false, which are grasped by the $\forall$ in CTL but not by the $\mathsf{P}_{=1}$ in PCTL. En la ecuación (1) se presentó un camino de ejecución de esta naturaleza.

Algo similar ocurre entre la fórmula CTL $\exists\Box(\text{SERVIR} \wedge \text{C})$ and its probabilistic counterpart $\mathsf{P}_{>0}(\,\Box\,(\text{SERVIR} \wedge \text{C})$. En general la expresividad de CTL y PCTL son incomparables [BK08, Lemas 10.44 y 10.45].

La *Lógica Estocástica Continua* (CSL por sus siglas en inglés, [BKH99]) es otra branching-time temporal logic specifically designed for CTMC analysis. It is based on CTL and similar to PCTL, with the additions of a time bounded version of the until temporal operator ($\mathsf{U}^{\leqslant t}$), also extended to its derivatives (e.g. $\Diamond^{\leqslant t}$), and an operator to reason about the steady-state probabilities of the system (S). The bounded versions of the temporal operators limit the time horizon to consider, so $\Diamond^{\leqslant t}$ means somewhen within t time units. El operador S considera en cambio un horizonte temporal infinito, hablando de la probabilidad de caminos en un sistema *en equilibrio*. Las siguientes son fórmulas CSL:

- $\neg$ SATURADO $\mathsf{U}^{\leqslant 111}$ KÖLN *"el tren llegará a Köln dentro de 111 unidades de tiempo, y no se saturará en todo el viaje"*;
- $\mathsf{P}_{[,5,,6]}(\Diamond^{\leqslant \frac{3}{2}}$ (SERVIR $\wedge$ C)) *"dentro de una unidad y media de tiempo, la probabilidad de que un cliente pida una Cocola está entre un 50 % y un 60 %"*;
- $\mathsf{S}_{\geqslant 0,9}(\Box^{<60} \neg$ SATURADO) *"en un día laboral normal (aka en equilibrio) y con al menos un 90 % de probabilidad, el tren ICE nunca se saturará en las primeras 60 unidades de tiempo"*.

Hasta ahora hemos considerado exclusivamente fórmulas con resultados booleanos: incluso en consultas cuantitativas como $\mathsf{P}_{\leqslant 0,25}(\Diamond$ SATURADO) la respuesta es "sí" o "no" para un estado o camino de ejecución dados. Muchas veces, en vez de un requerimiento del tipo "¿es la probabilidad de que tal cosa ocurra menor/mayor que tal otro valor?", lo que se desea consultar es una medida de la *eficiencia* o el desempeño del modelo, e.g. "¿cuál es la probabilidad de que tal cosa ocurra?".

En general preguntas cuantitativas de esta índole, que preguntan por el valor de una medida, son omitidas en las lógicas temporales. La causa radica en que el anidamiento de respuestas con valores numéricos es difícil de capturar consistentemente. A pesar de ello, muchas herramientas ofrecen a sus usuarios la posibilidad de realizar este tipo de consulta en el nivel más externo de las fórmulas. Como ejemplos modernos mencionamos a los operadores `P=?` y `S=?` en el model checker probabilista PRISM [KNP07, Sec. 5.1].

2.3. Análisis del modelo

Una vez que el modelo del sistema y las consultas de propiedades han sido especificados formalmente, algoritmos automáticos pueden ejecutarse para verificar si el modelo satisface los requerimientos, o para descubrir la eficiencia de nuestro (modelo del) sistema. Existe más de una forma de llevar esto a cabo; a continuación se repasan algunas de las técnicas más usadas, explicando más detalladamente aquella de mayor relevancia para esta tesis.

2.3.1. Panorama y técnicas conocidas

Cuando uno piensa en verificaciones formales automáticas, las *pruebas automatizadas de teoremas* (ATP por sus siglas en inglés, también conocido como deducción automática) es quizás lo primero que viene en mente. ATP consiste en el uso de programas de computadora para demostrar que alguna formulación (la *conjetura*) es una consecuencia lógica de otro conjunto de fórmulas (los *axiomas* y las *hipótesis*). Claramente esto requiere de una formulación apropiada del problema como un conjunto de axiomas, hipótesis, y una conjetura.

ATP es un término más bien amplio asociado principalmente a los asistentes de prueba. Un *asistente de prueba* es una herramienta de software que ayuda a los

humanos a desarrollar pruebas matemáticas formales por medio de la colaboración humano-máquina. Esta técnica no es completamente automatizable y en todo caso cae fuera de los objetivos de esta tesis. El lector interesado es referido e.g. al asistente de pruebas Coq.

Más en línea con el punto de vista de modelado que ha sido introducido, el *model checking* o chequeo de sistemas suele ser una de las primeras estrategias a considerar, basado en la exploración del espacio de estados de un modelo formal del sistema estudiado. Análisis de grafos, aproximaciones numéricas, estimaciones de punto fijo, y muchos otros métodos basados en computadoras son cubiertos por este concepto tan abarcador.

En sus muchas formas el model checking puede expresar y estudiar sistemas no deterministas, probabilísticos/estocásticos, y temporizados. La idea general es correr pruebas automáticas en el modelo, donde la propiedad consultada especifica qué es lo que se busca y por ende cuál algoritmo ejecutar. Para consultas cualitativas o bien se prueba que la propiedad es satisfecha o sino (usualmente) se provee un contraejemplo extraído del modelo que la refuta. Para consultas cuantitativas que involucren procedimientos iterativos muchas implementaciones deciden, o demandan como entrada del usuario, un *épsilon de convergencia*, y finalizan el cómputo tan pronto como la diferencia entre la salida de dos iteraciones consecutivas cae por debajo de este épsilon.

Cualquiera sea su variedad, la base algorítmica del model checking requiere la representación de todos los estados del modelo. Esto desenlaza en el infame *problema de la explosión de estados*, dado que el tamaño del espacio de estados crece de manera exponencial con el número de variables del modelo (la mayoría—sino totalidad—de los formalismos conocidos están basados en variables, ver por ejemplo los lenguajes de entrada de PRISM [KNP11], UPPAAL [BDL+06], MODEST [HHHK12], STORM [DJKV16], etc.).

Existen muchas técnicas para reducir, truncar, o abstraer el espacio de estados without affecting the model checking results, or affecting them in a known and quantifiable manner. This has enabled the study of several real-life systems with very large state spaces. Yet the problem is inherent to model checking y la reducción del espacio de estados es un área activa de investigación.

Una estrategia diferente para analizar modelos, que es teóricamente oblivious of the number of system states, is *simulación de Monte Carlo* or merely *simulación*. In its standard form, model analysis by simulation comprises the generation of paths following the formal description of the system. These paths need to be finite, so either the model naturally expresses finite executions or some termination or truncation criteria has to be forced upon the generation procedure. Paths are then examined from el punto de vista de la consulta para determinar si o cómo satisfacen la propiedad.

Cuando el mecanismo de simulación está correctamente implementado, each new path provides fresh, independent information about the satisfiability of the property by the model. By means of some statistical analysis this is deemed sufficient at certain point, after *enough paths have been generated*. By then an estimate of the answer to the query is available for the user. Ver la Figura 2.4

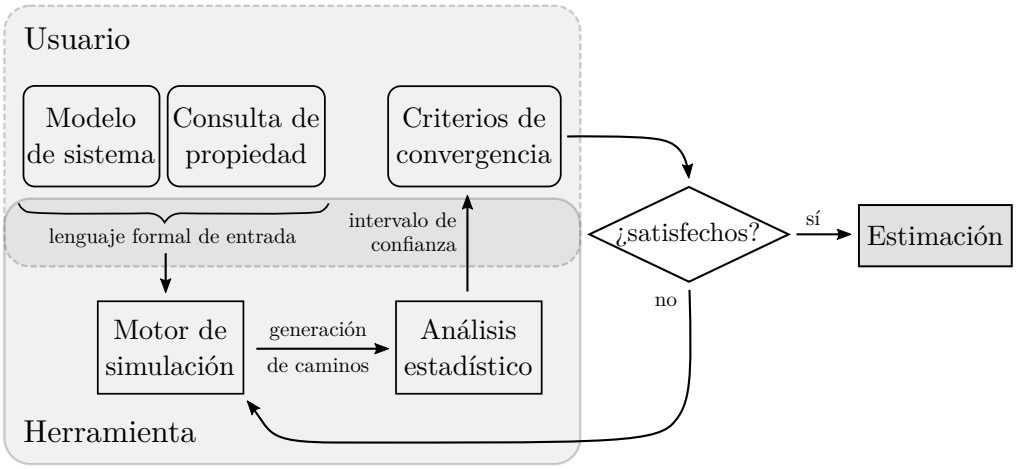


Figura 2.4: Análisis por simulación de Monte Carlo

para una representación esquemática de este procedimiento.

A todo esto, ¿qué significa *suficiente simulación*? Computation in model checking stops either when the whole state space has been analysed, a state sought has been reached, or the iterative procedure converges up to the epsilon imposed. These concepts do not apply to the approach of analysis by simulation. Instead a statistical notion of convergence derived from the law of large numbers is used to judge how far the current estimate is from the real answer. It is up to the user to choose the desired proximity, which is usually done in terms of confidence criteria, como se explicará en mayor detalle en la Subsección 2.3.2.

Esta tesis se ocupa de la autmatización de un método particular para analyse models by simulation, in a scenario where the nature of either the model or the query make it very unlikely to generate useful paths. In such scenario standard simulation techniques are rendered useless, due to the rarity of the event which needs to be observed for the statistical analysis to converge. Usualmente esto se denota *rsimulación de eventos raros* (RES por sus siglas en inglés, [RT09a]), e implica el uso de técnicas inteligentes para acelerar la convergencia.

Concluimos esta sección recordando que en la Sección 2.1 the three axis for model characterization introduced: nondeterminism, probability/stochasticity, and (explicit) time. On the one hand, the latter two can be easily encoded in simulation. Stochastic models even constitute an ideal field for applying this technique, since formal analysis and numerical approximations can be hard to develop for general and involved cases. En cambio la *simulación por eventos discretos* ofrece una solución directa y fácil de implementar.

Por el otro lado, cuando se enfrenta una elección no determinista and by the very definition of it, a simulation does not know which way to go. This poses no problem for model checking which can simply branch and follow all choices simultaneously. Path simulation however finds here a limiting factor, though some efforts are being currently made in this direction. Namely, [BFHH11] shows a way to get rid of spurious nondeterminism, which is certainly no final solution. La simulación de no determinismo *real* es tratada en [HMZ+12, BCC+14]

con algunos problemas, como requerir well-structured problems and showing bad performance in scenarios where optimal scheduling decisions are needed. The theory from [LST14] works on MDP models, encoding schedulers implicitly which saves on memory consumption. Among the most modern contributions stand [DLST15, DHLS16], que trata con Autómatas Temporales Probabilistas (que generalizan a los MDP con variables reloj) a través de *tácticas ligeras*.

Finalmente remarcamos que, cuando se encuentra embebido en el marco de la descripción y verificación formal de sistemas, el análisis por simulación muchas veces ha recibido el nombre de *model checking estadístico*. No seguimos esa tendencia en esta tesis porque las garantías formales ofrecidas por los resultados del model checking son incomparables con las nociones estadísticas de confianza y precisión inherentes a la estrategia de Monte Carlo (a pesar de un solapamiento parcial de los problemas atacados por ambas técnicas). Además hay que considerar el comportamiento no determinista, que puede ser abordado sin problemas por el model checking pero no mediante el uso de simulaciones. Por ende y de aquí en más emplearemos el término 'simulación' para referirnos a la técnica de análisis de modelos mediante la generación y el análisis estadístico de caminos de ejecución del sistema.

2.3.2. Simulación

Existen al menos dos estrategias diferentes para simular caminos de ejecución a partir de la especificación de un modelo de sistema. En la *simulación continua* se asume que la sucesión de eventos relevantes evoluciona de forma continua. Este es típicamente el caso de las ecuaciones diferenciales, que dan tasas de cambio de las variables de estado en el tiempo. Como ejemplos concretos considérense el rastreo de la trayectoria de un proyectil y la simulación de circuitos FPGA. En estos casos es usual la aplicación de técnicas de análisis numérico, e.g. integración por el método de Runge-Kutta. Otra implementación consiste en la discretización del tiempo en *intervalos* pequeños (donde la magnitud depende de cada caso particular), atendiendo en cada *paso de simulación* toda actividad que haya acontecido en el siguiente intervalo.

La *simulación por eventos discretos* (DES por sus siglas en inglés, [LK00]) ofrece una alternativa que se adecúa mejor a los sistemas que naturalmente evolucionan en instantes discretos de tiempo. No se requiere regularidad temporal, i.e. estos instantes de tiempo pueden no estar equiespaciados. Lo que realmente importa es que la evolución del sistema se de paso a paso, de manera tal que el simulador pueda construir una lista de eventos futuros que serán atendidos en orden. Evidentemente hay una fuerte correspondencia entre esta metodología y la descripción de modelos basada en autómatas que se discutió en Sección 2.1: DES es la manera usual de analizar modelos con simulación en este tipo de formalismos.

Si bien la implementación concreta puede variar, DES siempre cuenta con los siguientes cuatro ingredientes básico:

- *Estado*:
  - ▷ en cualquier instante de tiempo todo componente del sistema tiene un *estado* unívocamente definido;
  - ▷ los estados describen "la situación actual de las cosas", e.g. el número de clientes en una cola, el número de discos fallados en un cluster, un módulo de reparación ocupado/ocioso, etc;
  - ▷ el estado del conjunto de todos los componentes es el *estado del sistema*.
- *Evento*:
  - ▷ un *evento* es un cambio instantáneo en el estado del sistema;
  - ▷ no es necesario que todos los componentes del sistema se vean afectados; alcanza con que uno cambie su estado;
  - ▷ los estados *sólo* cambian durante la ocurrencia y el manejo de eventos;
  - ▷ un evento es *atómico*: aunque la implementación de bajo nivel puede cambiar el estado de unos componentes antes que otros, este orden no debe afectar el comportamiento global resultante;
  - ▷ de un conjunto de eventos posibles, usualmente el estado actual del sistema define cuales de entre ellos se encuentran habilitados.
- *Lista de prioridades*:
  - ▷ durante la simulación, los eventos futuros son programados y ordenados de acuerdo a cierta prioridad, e.g. ocurrencia en el tiempo;
  - ▷ todo evento futuro es almacenado en un tipo abstracto de dato, como una cola, para su posterior manipulación;
  - ▷ el próximo evento de mayor importancia se puede obtener eficientemente de esta lista, usualmente en tiempo constante;
- *Generación de números aleatorios*:
  - ▷ DES suele ser empleado en casos probabilísticos/estocásticos, que requieren alguna forma de aleatorizar la generación de eventos;
  - ▷ usualmente un *generador de números pseudo– o quasi–aleatorios* es la fuente de toda aleatoriedad;
  - ▷ mediante diversas técnicas se transforma la salida del generador de números aleatorios, en el rango [0,1], en comportamiento probabilístico/estocástico con la distribución deseada [PTVF07].

Asumiendo la presencia de todos estos componentes mencionados, una descripción de alto nivel para aplicar DES puede bosquejarse como sigue:

1. Establecer el estado inicial del sistema.

2. Basados en la descripción del sistema y en la configuración resultante del punto 1, generar un conjunto de eventos inicialmente habilitados y almacenarlos ordenadamente en la lista de prioridades.

3. Tomar *el* siguiente evento habilitado, de acuerdo a esta lista.

4. Modificar el estado del sistema de acuerdo al evento resultante del punto 3.

5. El nuevo estado del sistema, resultante del punto 4, puede haber habilitado nuevos eventos y deshabilitado otros. Actualizar la lista de prioridades en base a esta información, restableciendo el orden como corresponda.

6. Volver al punto 3.

Este procedimiento iterativo termina cuando se vacía la lista de prioridades donde se almacenan los próximos eventos, o cuando alguna condición definida por el usuario es alcanzada, como ser la simulación de N unidades de tiempo.

Ni bien DES finaliza, los datos relevantes que hayan sido recogidos son alimentados al mecanismo de análisis estadístico como una *nueva muestra*. La respuesta estimada para la consulta del usuario se actualiza, y se considera la terminación del procedimiento: si suficientes muestras han sido generadas para asegurar la confianza estadística solicitada, no es necesario realizar más simulaciones. En caso contrario se inicia un nuevo ciclo de DES.

También existen escenarios donde los datos para el análisis estadístico pueden ser recolectados durante las simulaciones mismas, viz. no es estrictamente necesario que DES termine. En su lugar, la información estadística puede obtenerse y analizarse *durante* la ejecución del procedimiento iterativo. Un ejemplo típico de estos casos se da cuando el usuario pregunta el número promedio de clientes en una cola.

2.3.3. Estimación

Hasta aquí la existencia de procedimientos estadísticos para procesar los datos obtenidos mediante simulaciones ha sido mencionado pero no explicado. Los nociones requeridas de estadística son introducidas en esta subsección, tanto porque es el próximo concepto lógico a cubrir, como porque nos ayudará a formular con mayor formalidad una de las motivaciones de esta tesis. Se asume al lector familiarizado con algunas nociones básicas en estadística, que de otra forma pueden ser tomadas de lecturas clásicas como [Ric06].

El Diccionario de la Real Academia Española define *estadística* como la "rama de la matemática que utiliza grandes conjuntos de datos numéricos para obtener inferencias basadas en el cálculo de probabilidades" [RA14]. Para el alcance de esta tesis todos los datos numéricos serán generados con la simulación por eventos discretos llevada a cabo en computadoras. Cada valor individual será llamado una *muestra* y denotado X o X_i. Cada muestra será el resultado de una simulación y

puede ser interpretada como la realización de una *variable aleatoria* de distribución potencialmente desconocida.

La distribución de muestreo es inherente a la naturaleza del modelo de sistema. Debería ser claro que, asumiendo un uso correcto del generador de números aleatorios [PTVF07], las muestras generadas pueden considerarse independientes e idénticamente distribuidas (IID) en el sentido estadístico.

Muestreo será el proceso de ejecutar simulaciones por eventos discretos para generar muestras IID. La secuencia de datos resultante será una *muestra aleatoria*, denotada $\{X_i\}_{i=1}^N$ para simbolizar que N muestras IID fueron generadas en el orden $X_1, X_2, \ldots, X_N$, donde N es el *tamaño de la muestra*. Desde el punto de vista matemático se puede pensar en $\{X_i\}_{i=1}^N$ como una secuencia de variables aleatorias IID.

Dada una muestra aleatoria considérese el promedio de sus valores, denotado $\overline{X}_N$ para una muestra de tamaño N:

$$\overline{X}_N \doteq \frac{1}{N}\sum_{i=1}^{N} X_i \tag{2}$$

Este valor es llamado la *media muestral* y se usa principalmente como estimador de la *media poblacional* [SS07]. Como es el resultado de transformaciones sobre variables aleatorias, la media muestral es una variable aleatoria en sí misma, con su propia distribución, media, varianza, y demás momentos muestrales. Los siguientes resultados nos ayudarán a estudiar las expresiones de estos parámetros, para obtener una intuición de la forma en que afectan al problema de RES. Se excluyen las pruebas, que pueden hallarse en incontables libros de matemática estadística, e.g. [Bil12, Ric06].

Proposición 1 (Desigualdad de Chebyshev)**.** *Sea X una variable aleatoria con media (poblacional) μ y varianza (poblacional) σ^2. Entonces para cualquier número real $\varepsilon > 0$*

$$P\left(|X - \mu| > \varepsilon\right) \leqslant \frac{\sigma^2}{\varepsilon^2}.$$

A grandes rasgos la Proposición 1 sugiere que, si el valor de σ^2 es lo suficientemente pequeño, hay una alta probabilidad de que una realización de la v.a. X se encuentre cerca de la media $E(X) = \mu$. Esto se condice con la idea de que la desviación estándar de una variable aleatoria indica el grado de dispersión de los valores que puede adoptar.

Puede probarse a partir de la ecuación (2) que la media muestral es en realidad un *estimador insesgado* para la media poblacional, viz. $E(\overline{X}_N) = \mu$ donde μ es la (usualmente desconocida) esperanza de las variables aleatorias $\{X_i\}_{i=1}^N$. La Proposición 1 dice que $\overline{X}_N$ puede hacerse arbitrariamente cercano a μ, si se conoce alguna forma para reducir la varianza de la muestra aleatoria. A su vez uno cuenta con la idea intuitiva de que mientras mayor sea esta muestra, más parecida se hará la media muestral a la media poblacional, i.e. menos se desviará este promedio del verdadero valor medio μ. Esto sugiere que N y σ^2 son

proporciones relacionadas de manera inversa, el cual es precisamente el caso si se cuenta con la independencia de las X_i:

$$Var(\overline{X}_N) = \frac{\sigma^2}{N} \tag{3}$$

La ecuación (3) nos dice pues cómo reducir la varianza en la formulación de la desigualdad de Chebyshev: substituyendo a X por $\overline{X}_N$ e incrementando el tamaño muestral N deberíamos lograr acercar la media muestral $\overline{X}_N$ a la media poblacional μ. El siguiente resultado, que es una consecuencia de la Proposición 1 y la ecuación (3), formaliza una generalización de esta afirmación.

Teorema 2 (Ley de los grandes números)**.** *Sea $X_1, X_2, \ldots, X_i, \ldots$ una secuencia de variables aleatorias independientes t.q. $E(X_i) = \mu$ y $Var(X_i) = \sigma^2$ para todo $i \geqslant 1$. Sea $\overline{X}_N$ el promedio tomando hasta la N-ésima variable aleatoria de la secuencia, viz. $\overline{X}_N \doteq \frac{1}{N}\sum_{i=1}^{N} X_i$. Entonces para cualquier real $\varepsilon > 0$*

$$P\left(|\overline{X}_N - \mu| > \varepsilon\right) \to 0 \qquad \text{as} \quad N \to \infty.$$

Nótese que el Teorema 2 no requiere que las variables aleatorias de la secuencia estén igualmente distribuidas. La condición estrictamente más débil de poseer el mismo primer y segundo momento garantiza el resultado, lo cual claramente satisfacen nuestras técnicas de simulación.

En la configuración propuesta donde el muestreo proviene de la simulación por eventos discretos, el Teorema 2 puede interpretarse así: cuando se estima el comportamiento promedio de un modelo, muchos caminos de ejecución deberán ser simulados; mientras más caminos se generen, más se parecerá la media observada al verdadero comportamiento promedio del modelo.

A su vez este parecido puede hacerse arbitrariamente cercano, siempre que uno pueda seguir generando muestras. Conceptualmente, la estimación del comportamiento promedio mejorará al aumentar el tamaño de la muestra. Esto se prueba en [CR65] asumiendo que la muestra aleatoria converge y que $\sigma^2 < \infty$.

2.3.4. Convergencia y criterios de parada

La ley de los grandes números da condiciones suficientes para dar un sentido a la noción informal de *suficiente muestreo*, donde el usuario puede elegir a priori cierta proximidad deseada a μ, y luego producir suficientes muestras hasta que la misma se cumpla. Sin embargo, este resultado sólo indica que ese punto eventualmente llegará, es decir que existe un valor de N que satisfará los deseos del usuario[†]. No da empero ningún indicio sobre cómo medir de forma efectiva la proximidad a μ que se obtiene con la muestra aleatoria actual.

Cuantificar esta proximidad es de enorme utilidad práctica. Normalmente cuando se analiza un modelo mediante simulación, el usuario solicita la estimación

[†] Estrictamente hablando esto se formula en la *ley fuerte de los grandes números*, una variante del Teorema 2.

de algún valor con cierta precisión. Simulaciones independientes son lanzadas para generar la muestra aleatoria, a partir de la cual se construye la estimación. La ley de los grandes números dice que eventualmente se habrán generado suficientes muestras como para satisfacer la precisión buscada, pero eso es insuficiente en el caso práctico. Necesitamos poder dar algún tipo de garantía respecto de la distancia entre el valor real y la estimación actual. Sólo de esta forma podremos decidir si es necesario generar más simulaciones. En otras palabras: es crucial contar con algún criterio de parada.

Por suerte hay propiedades del límite de la suma de vairables aleatorias which provide ways to measure the accuracy of the current estimate, helping to build a termination criterion. Nótese que para cualquier variable aleatoria X la *variable alteatoria estandarizada* Z definida como

$$Z \doteq \frac{X - E(X)}{\sqrt{Var(X)}}$$

satisface $E(Z) = 0, \;\; Var(Z) = 1$. Sea C_N la suma de alguna muestra aleatoria de tamaño N con media μ y varianza σ^2, i.e. $C_N \doteq \sum_{i=1}^{N} X_i$. El Teorema 2 says C_N/N converges to μ; the following result studies this convergence and gives an explicit cumulative distribution function for the standardization $Z_N \doteq (C_N - N\mu)/(\sigma\sqrt{N})$.

Teorema 3 (Teorema Central del Límite)**.** *Sea* $\{X_i\}_{i=1}^{N}$ *una muestra aleatoria con media* μ *y varianza positiva finita* σ^2*. Sea* $C_N \doteq \sum_{i=1}^{N} X_i$*, entonces*

$$\lim_{N\to\infty} P\left(\frac{C_N - N\mu}{\sigma\sqrt{N}} \leqslant z\right) = \Phi(z)$$

para $z \in \mathbb{R}$ *finito, donde* $\Phi(z)$ *es la función de distribución acumulada de una distribución normal estándar:*

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{x^2}{2}}\, dx \;.$$

El Teorema 3 habla de convergencia en la distribución, declarando que que la media muestral estandarizada Z_N converge a la función de distribución acumulada de una normal estándar. Generally speaking the *speed of convergence* depends on the real distribution of the X_i, high skewness and long tails playing against it. Several rules of thumb exists about which N is large enough to start using the approximation of the Central Limit Theorem, e.g. $N > 30$, or $C_N > 5 \;\wedge\; N * (1 - C_N/N) > 5$ for binomial proportions. Of course and in general, the larger the sample the better the approximation.

Estrictamente, el Teorema 3 should be applied if the variance of the population is known. En la mayoría de las situaciones prácticas σ^2 es desconocida y debe ser aproximada con el estimador insesgado

$$S_N^2 \doteq \frac{1}{N-1} \sum_{i=1}^{N} \left(X_i - \overline{X}_N\right)^2 \tag{4}$$

En estos casos la *distribución t de Student* con $N-1$ grados de libertad debería ser empleada en lugar de la normal, puesto que tiene una cola más larga y no depende de los valores poblacionales μ y σ^2. Formalmente:

$$\frac{\overline{X}_N - \mu}{S_N/\sqrt{N}} \sim T_{N-1} \tag{5}$$

donde $S_N \doteq \sqrt{S_N^2}$, μ es desconocida, y la distribución t de Student con $\nu \in \mathbb{R}$ grados de libertad está caracterizada por la función de densidad de probabilidad

$$f_\nu(t) = \frac{1}{\sqrt{\nu}\,\mathrm{B}(\frac{1}{2},\frac{\nu}{2})}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

donde B es la función Beta. The corresponding cumulative distribution function $T_\nu(t)$ is harder to express and thus not included. It is a known result that $T_\nu(x)$ converges to $\Phi(x)$ cuando $\nu \to \infty$, relacionando coherentemente la ecuación (5) el Teorema 3.

Desde una perspectiva práctica and given the symmetry of the cumulative distribution function of the Student's t-distribution (i.e. $T_\nu(-t) = 1 - T_\nu(t)$), this means that for sufficiently large N it is safe to assume $P(-t < Z_N < t) \approx 2\,T_{N-1}(t) - 1$. Nótese que el uso de la estandarización

$$Z_N = \frac{\overline{X}_N - \mu}{\sigma/\sqrt{N}}$$

pues $C_N = N\,\overline{X}_N$, donde S_N de la ecuación (4) debe ser usada en la ecuación anterior como aproximación para σ cuando la varianza poblacional es desconocida.

Para ver como estos resulados fit in the scenario of model analysis by simulation, suppose the user wants to find out the likelihood γ of satisfying certain property in his model. Furthermore, and here lies the core asset, he requests an upper bound de $\varepsilon > 0$ para la probabilidad de error en la estimación.

El método Monte Carlo estándar via discrete event simulation generates several, N say, independent simulations. Each simulation results in some path which will either satisfy the property or not. Thus a random sample $\{X_i\}_{i=1}^N$ is generated, where $X_i = 1$ if the i-th simulated path satisfies the property and $X_i = 0$ otherwise. This definition of the X_i means the queried likelihood is the population mean, $\gamma = \mu$. Thus a straightforward estimator $\hat{\gamma}$ for the likelihood is the sample mean $\overline{X}_N$. Denótense $\overline{X} = \overline{X}_N$ y $\sigma_{\overline{X}} = S_N/\sqrt{N}$, entonces lo que sigue devuelve una cuantificación (conservativa) del error incurrido en la aproximación:

$$\begin{aligned} P\left(|\hat{\gamma}-\gamma| \leqslant \varepsilon\right) &= P(-\varepsilon \leqslant \hat{\gamma}-\gamma \leqslant \varepsilon) \\ &= P\left(-\frac{\varepsilon}{\sigma_{\overline{X}}} \leqslant \frac{\overline{X}-\mu}{\sigma_{\overline{X}}} \leqslant \frac{\varepsilon}{\sigma_{\overline{X}}}\right) \\ &\approx 2\,T_{N-1}\left(\frac{\varepsilon}{\sigma_{\overline{X}}}\right) - 1 \end{aligned}$$

Notar que hasta ahora sólo hemos considerado *estimadores puntuales*, i.e. se le da al usuario una estimación $\hat{\gamma} \in \mathbb{R}$ del valor real γ que desea conocer, and can compute the probability of error incurred in the estimation. The approach usually followed in practice is slightly more involved and adds an interval a la información provista por el estimador puntual.

Definición 5 (Intervalo de confianza)**.** Dada una muestra aleatoria $\{X_i\}_{i=1}^N$ tomada de alguna población, un *intervalo de confianza* (IC) alrededor de algún parámetro $\theta \in \mathbb{R}$ de la población es un intervalo $[l, u] \subset \mathbb{R}$, cuyos límites l, u son variables aleatorias derivadas de la muestra, y que contiene (*cubre*) al verdadero parámetro θ con alguna probabilidad conocida.

La Definición 5 es un tanto laxa because the specific expression of the interval may vary depending on the parameter to estimate and the nature of the sample. For this thesis the main interest is to build a IC around the population mean μ. Denote by z_α the α-quantile of the standard normal distribution for $0 < \alpha < 1$, i.e. the area to the right of $z_\alpha \in \mathbb{R}$ under the curve of its density function is α. Usando la simetría de esta función junto con la aproximación provista por el Teorema Central del Límite esto se traduce en

$$P\left(-z_{\frac{\alpha}{2}} \leqslant \frac{\overline{X} - \mu}{\sigma_{\overline{X}}} \leqslant z_{\frac{\alpha}{2}}\right) \approx 1 - \alpha$$

o equivalentemente

$$P\left(\overline{X} - z_{\frac{\alpha}{2}}\sigma_{\overline{X}} \leqslant \mu \leqslant \overline{X} + z_{\frac{\alpha}{2}}\sigma_{\overline{X}}\right) \approx 1 - \alpha \tag{6}$$

La Ecuación (6) expresa que para muestras lo suficientemente grandes, la probabilidad de que μ se encuentre en el intervalo $\overline{X} \pm z_{\frac{\alpha}{2}}\sigma_{\overline{X}}$ está aproximada por $1 - \alpha$. This interval is thus called a $100(1-\alpha)$ *% confidence interval*. The value $100(1-\alpha) \in (0, 100)$ is the *confidence level* and its width $2z_{\frac{\alpha}{2}}\sigma_{\overline{X}}$ is the *precision* of the interval. The same analysis follows using Student's t-distribution instead of the normal distribution, cuando la varianza poblacional es desconocida.

Ahora sí contamos con todos los ingredientes para analizar un sistema mediante simulación. El usuario provee una descripción formal de su modelo y de la propiedad que desea consultar. También especifica el nivel de confianza y la precisión con la que se estimará el parámetro. Se simulan caminos de ejecución con la técnica de simulación de eventos discretos, que resultan en realizaciones de las v.a. $\overline{X}$ y $\sigma_{\overline{X}}$ (o el estimador que corresponda para $\hat{\gamma}$ y su varianza). Como el nivel de confianza fue prefijado, con cada nueva realización (i.e. por cada simulación finalizada) se actualiza el intervalo. Así, el valor concreto de la precisión del intervalo estimado se va actualizando, y de acuerdo al Teorema 2 este valor eventualmente disminuirá. El cómputo finaliza ni bien la precisión lograda para el intervalo se hace más chica que la que había solicitado el usuario. Alternativamente el usuario podría prefijar un nivel de confianza y un tiempo límite de cómputo, midiendo la precisión lograda en el intervalo estimado cuando las simulaciones concluyan.

2.4. Eventos raros

La receta construida en las Subsecciones 2.3.2 a 2.3.4 para estimar una respuesta que ofrecer al usuario es bastante amplia. Teóricamente sólo se encuentra limitada por la factibilidad de simular caminos de ejecución en el modelo (usualmente una tarea sencilla) y la computabilidad del estimador escogido. Sin embargo es su eficiencia práctica, más que su generalidad teórica, lo que limita su aplicación.

Denotemos con $z \in \mathbb{R}$ al cuantil de la ecuación (6), ya sea que se utilice la distribución normal estándar o la t de Student. Denotemos también con $\hat{\sigma}$ la desviación estándar medida para el estimador $\hat{\gamma}$, el cual usamos para aproximar el verdadero valor γ. La velocidad de convergencia del procedimiento simular/estimar está fuertemente relacionada con la precisión solicitada para el IC, i.e. $2z\hat{\sigma}$. Por ejemplo, cuando el estimador es la media muestral, $\hat{\gamma} = \overline{X}_N$, la precisión decrece a razón de la raíz cuadrada del tamaño muestral, ya que entonces o bien $\hat{\sigma} = \sigma/\sqrt{N}$ o sino $\hat{\sigma} = \sqrt{S_N^2/N}$. Esta *velocidad de convergencia* es moderadamente eficiente en muchas aplicaciones prácticas.

No olvidemos, sin embargo, que esta tesis pretende estudiar la ocurrencia de eventos raros, lo cual significa que $0 < \gamma \ll 1$. Para números muy pequeños e.g. $\gamma \approx 10^{-8}$ y más chicos aún, el *error absoluto* determinado por la expresión $z\hat{\sigma}$ no es lo suficientemente representativo [RT09a]. En su lugar, el *error relativo* captura de forma más flexible y significativa la precisión de nuestra estimación.

Definición 6 (Error Relativo)**.** Sea $[l, u]$ un intervalo de confianza construido alrededor de algún parámetro γ con precisión $2z\hat{\sigma}$. El *error relativo* (ER) del intervalo de confianza es su error absoluto dividido por el parámetro a estimar:

$$\mathrm{ER}_{[l,u]} \doteq \frac{z\hat{\sigma}}{\gamma}.$$

Usualmente el valor real de γ es desconocido, en cuyo caso es posible reemplazarlo por el valor estimado $\hat{\gamma}$ cuando la muestra es lo suficientemente grande. El error relativo puede ser pensado en términos de la precisión del intervalo relativa al valor estimado: solicitar una estimación con 10 % de error relativo significa e.g. que el cómputo cesará ni bien la mitad del ancho del IC construido alrededor del estimador sea menor o igual a $\hat{\gamma}/10$, i.e. un 10 % de la estimación.

La Figura 2.5 muestra gráficamente porqué es importante contar con esta flexibilidad que el error relativo otorga por sobre el error absoluto. Como uno desconoce a priori la magnitud de γ, pedir por ejemplo una precisión de 10^{-7} podría sonar razonable, pero en realidad no sabemos si lo es. Si γ es tan sólo un orden de magnitud más chico que ese valor, el IC resultante muy probablemente contendrá al 0, sugiriendo que el evento raro es potencialmente imposible. Quizás se podría haber construido un intervalo sin el 0 realizando unas pocas simulaciones más, pero si trabajamos con error absoluto tendremos que correr toda la estimación de vuelta para un error más chico. El error relativo evita este tipo de problemas desde el comienzo, asegurando que la mitad del ancho del IC será menor que el valor de $\hat{\gamma}$.

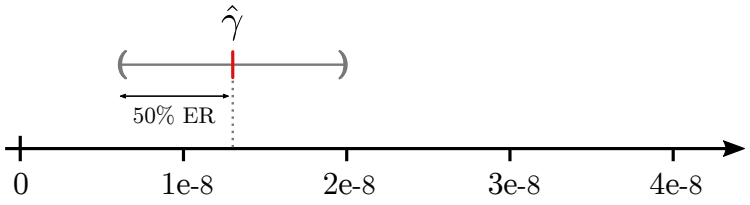


Figura 2.5: Intervalo de confianza construido con error relativo

Considérese la estimación de una proporción binomial, i.e. γ es la probabilidad de éxito en el ensayo de un experimento. Una simulación representa un ensayo, y su salida será 1 si se tiene éxito y 0 si se fracasa, lo cual encaja con los ejemplos trabajados hasta aquí. El estimador de γ será $\hat{\gamma} = \overline{X}$ y el estimador usual para la varianza en estos casos es $\hat{\sigma}^2 = \hat{\gamma}(1-\hat{\gamma})/N$ Entonces para cualquier intervalo de confianza IC

$$\mathrm{ER}_{\mathrm{IC}} \;=\; z\frac{\sqrt{\hat{\gamma}(1-\hat{\gamma})}}{\sqrt{N}\,\hat{\gamma}} \;\approx\; \frac{z}{\sqrt{N\,\hat{\gamma}}}.$$

La última expresión puede ser inmensa for very small values of γ, which is a situation naturally exacerbated by the rarity of the event. Say the user requests a 95 % confidence interval and relative error of 10 % with $\gamma \approx 10^{-8}$. Then $z \approx 1{,}96$ and hence N should be greater than $3{,}84 \times 10^{10}$ to satisfy the user needs. In this scenario where very few experiments are successful, standard-simulation times can easily become unreasonable. If each simulation takes 1 ms to complete, a computing system with a single execution thread would tomar 444 días (¡más de un año!) para satisfacer ese criterio.

Las computadoras modernas pueden aliviar this by using parallelism in its various dimensions (ILP, DLP, TLP, etc.) Together with smart implementations and to a certain extent, this can counter the presence of γ in the denominator of Definición 6. Yet there are systems characterized by an *exponential decay*, where polynomial modifications of some model parameter θ (e.g. increase by one the queue capacity) produces an exponential decrease of γ, see e.g. [Gar00, KN99]. In an exponentially decaying regime the standard approach of model analysis by simulation takes an exponential time to converge: for constants $c, k \in \mathbb{R}_{>0}$ one has $T_{\mathrm{std}}(\theta) = O(c^{k\theta})$. An *asymptotically efficient estimator* would instead converge within time polynomial in the rarity parameter: $T_{\mathrm{aeff}}(\theta) = O(\theta^{k'})$ para alguna constante $k' > 0$ [Gar00, Sec. 2.2.1].

Los tiempos inviables ejemplificados in the situation above are clearly inherent to the standard Monte Carlo approach, when it is used to analyse a model in a RES scenario. This inefficiency plus the issue of real coverage, i.e. whether or not the theoretical coverage is met by the confidence intervals built, are the main challenges presented by the rare event problem [RT09a, GRT09]. The core of the complication comes from the fact that $0 < \gamma \ll 1$, which implies very few useful paths will be generated during simulation. The two complementary techniques described next have been developed and perfected durante los últimos treinta años para contrarrestarlo.

El *muestreo por importancia* modifica la distribución de muestreo (de ahí

el nombre) de forma tal que increases the chance to visit the rare states of the model. This introduces a bias in the resulting point estimate to be corrected with a previously computed *likelihood ratio*. Evidently the change of measure requires a non-trivial understanding of the system under study. Moreover this modification needs to be tractable to allow the computation of the likelihood ratio, meaning it has to be characterized by some function selected ad hoc by the user with certain desired properties like integrability. A bad choice of measure may have a negative impact on the simulation resulting in longer computation times. In spite of these limiting factors, importance sampling has been successfully applied to several complex and incluso en sistemas de la vida real, ver e.g. [GSH+92, KN99, XLL07, dVRLR09, LT11].

La *división por importancia*, also known as multilevel splitting, works by decomposing the state space in multiple layers or *niveles*. A level should be higher as the probability of reaching the rare event from its composing states grows, so ideally the rare event would be at the top. Estimation consists in multiplying the estimates of the (not so rare) conditional probabilities of a simulation path moving one level up. The effectiveness of this technique crucially depends on an adequate grouping of states into levels, which is done by some user selected *función de importancia* (notice the resemblance with the measure change in importance sampling). This function assigns a value to each state, its *importance*, which should reflect the likelihood of observing the rare event after visiting that state. So, a state in the rare set should receive the highest importance and states importance decreases according to the probabilidad de alcanzar al evento raro desde ellos.

La mayor parte de la crítica al cambio de medida in importance sampling is also applicable to the importance function in importance splitting. This thesis conjectures assigning importance to the states, i.e. building a *good* importance function, is *easier* to be carried out by automatic procedures than choosing a *good* change of measure, providing a formalized user query and automata-based model descriptions are available. By *"easier"* it is meant that less assumptions need to be made about the nature of the system, most remarkably there is no need to rely on the memoryless property of the Markovian case. The term *"good"* is mild in the sense that it only implies an improvement over the standard Monte Carlo approach to analysis by simulation. No optimality assumptions are made; there are no conjectures about the difficulty of finding an función de importancia óptima o con eficiencia asintótica.

Basados en esa conjetura, desarrollamos algoritmos para derivar automáticamente la función de importancia a partir de la propiedad consulta por el usuario y el modelo del sistema, lo cual presentamos en los siguientes capítulos. Para entender las soluciones propuestas en todo su detalle, una descripción más profunda de la técnica de división por importancia es presentada en lo que resta del capítulo.

Es preciso hacer una última aclaración antes de concluir con esta sección. Con anterioridad se dijo que la división por importancia y el muestreo por importancia son métodos que pueden ser considerados complementarios. Por un lado esto se

debe a que las técnicas de división dependen de la división en niveles del espacio de estados, de forma tal que la probabilidad de "subir de a un nivel por vez" pueda computarse separada y eficientemente. En general esto requiere de trayectorias largas entre las condiciones iniciales del sistema y el evento raro. Si la rareza dependiese de realizar *pocas* transiciones, cada una muy improbable, entonces la división por importancia hace aguas, ya que no puede dividir la trayectoria en muchos niveles como le conviene. En estos escenarios el muestreo por importancia puede proveer una solución más natural, esto es, si es aplicable.

Por el otro lado cuando las ejecuciones a partir del estado inicial del sistema deben seguir trayectorias largas y heterogéneas para alcanzar al evento raro, puede ser muy complicado dar con un cambio de medida que seleccione la mejor transición a tomar ante cada elección. Es por ello que la división por importancia puede amoldarse mejor, en circunstancias donde muchos estados con diferentes propiedades deben ser visitados para alcanzar un estado raro, o en general cuando la naturaleza del modelo imposibilita la invención de un cambio de medida eficiente y reversible.

2.5. División por importancia

En esta sección se describen varios métodos para realizar análisis de modelos mediante simulaciones, empleando técnicas de división para mejorar la eficiencia. De aquí en adelante los términos *división por importancia*, *técnica de división*, y *división multinivel*, se utilizarán indistintamente para referirse al método descripto en esta sección para resolver RES.

2.5.1. Teoría general

Hay al menos dos ángulos desde los cuales mirar a las técnicas de división:

- Una idea original por Kahn Y Harris fue desarrollada desde un punto de vista físico en [KH51], donde la trayectoria de partículas eran salvadas y reiniciadas en ciertos puntos prometedores, para generar más observaciones del evento raro. Esta visión está naturalmente relacionada con la técnica introducida por Bayes en los setenta [Bay70, Bay72], que se diseminó veinte años más tarde cuando fue actualizada y extendida formalmente por José Villén-Altamirano y Manuel Villén-Altamirano, quienes introdujeron el nombre *Intentos de Simulación Repetitivos Tras Alcanzar Umbrales* (RESTART por sus siglas en inglés, [VAVA91, VAMMGFC94])

- Otra perspectiva más inherentemente matemática si se quiere, consiste en ver el espacio de estados del sistema como una secuencia de eventos anidados, para la noción formal de *evento* proveniente de teoría de la medida [Bre68, Def. 2.1]. Así pues dados $E_0 \supseteq E_1 \supseteq \cdots \supseteq E_n$ sean los estados en E_n los que definen al evento raro embebido en el espacio de estados completo E_0, y sea p_i la probabilidad condicional de que un camino simulado que comience en E_{i-1} alcance E_i antes de salir de E_{i-1}. Entonces

la probabilidad de visitar un estado raro es el producto $\prod_{i=1}^{n} p_i$ como se detalla en [LLGLT09, pp. 42–43].

La primera noción mencionada, que involucra el salvado y reinicio de caminos simulados, se analiza en profundidad en Sección 2.6 dado que es de particular interés para esta tesis. La segunda definición, más general y expresada en términos matemáticos, se describe en lo que resta de esta subsección, para definir formalmente la técnica de división e introducir algunas implementaciones conocidas. Todas las implementaciones comparten la idea central de el espacio de estados para resolver el problema de RES, pero difieren entre sí en la forma en que esto es explotado. Algunas nociones básicas de probabilidad son necesarias para entender las siguientes definición. Algunos aspectos teóricos fundamentales son presentados en el Apéndice B. Se refiere al lector a [Dur10, Chap. 1] o [Bre68, Chap. 2] para leer una introducción al tema.

Marco formal para la división por importancia en RES

Supongamos que la dinámica del sistema se encuentra descripta por un proceso estocástico $X \doteq \{X_t \mid t \geqslant 0\}$. Se asume la existencia de un *espacio de probabilidad* $(\Omega, \mathcal{F}, P)$ y de un *espacio medible* (S, Σ) tales que cada X_t es una variable aleatoria en Ω que toma valores en S, denotado el *espacio de estados* o conjunto de muestreo.

EL tiempo t puede ser continuo (en la recta real) o discreto (en los enteros no negativos $\mathbb{N} \doteq \{0, 1, 2, \ldots\}$). Por cuestiones de convergencia en el caso continuo, todos los caminos (viz. realizaciones de X) se asumen continuos por derecha con límites por izquierda, aka càdlàg.

A su vez y para estas definiciones, X será un proceso de Markov. Dado que la historia del proceso puede ser incorporada dentro del estado del sistema X_t, esta asunción se realiza sin pérdida de generalidad para la categoría completa de procesos estocásticos homogéneos [Gar00, Sec. 2.2].

Un *evento* será un subconjunto medible of the sampling set, i.e. an element of Σ. Let $A \subsetneq S$ be the *rare event* of interest, that is a (measurable) set of states the system can enter with positive but very small probability, e.g. a failure in a digital data storage facility leading to information loss. An event $B \subsetneq S$ is also assumed, denoting some stop or end-of-simulation condition que satisface $B \cap A = \varnothing$ y $P(B) \doteq P(X_t \in B) > 0$.

Definición 7 (Tiempo de entranda)**.** Sea $C \subseteq S$ un evento posible en el entorno formal anterior, o sea $C \in \Sigma$ y $P(C) > 0$. El *tiempo de entrada en* C es el valor casi seguramente finito

$$T_C \doteq \text{ínf}\{t \geqslant 0 \mid X_t \in C\}.$$

Hay dos formas de analizar sistemas de especial interpes para RES: transient and steady-state analysis. Both are involved with computing or estimating a very small probability value γ, viz. $0 < \gamma \ll 1$, related to the observation of the rare event A. The way of defining γ is what draws the difference between these approaches. Formalizamos estas nociones en las Definiciones 8 y 9.

Definición 8. En este entorno formal, la *probabilidad transitoria del evento raro* será el valor probabilístico

$$\gamma = P(T_A \leqslant T_B)$$

donde T_A, T_B son los tiempos de entrada en A, B respectivamente.

La Definición 8 es común in the RES literature, see e.g. [Gar00, Sec. 2.2] y [LLGLT09, Sec. 3.2.1]. It speaks of observing the rare event before reaching some stopping time T_B, here characterized by event B. This can be generalized to any (almost surely finite) time T, which might be of interest when defining simulation truncation not by an event but rather by the passage of time, e.g. *"the probability that a Cocola is chosen before T simulation time units elapse"*. Equivalently, time could be included in the state of process X, para hablar a través de eventos de un horizonte temporal finito.

Recuérdese que estamos interesados en la aplicación de importance splitting to a formal model description scenario resembling that of model checking. Thus the probability from Definición 8 should be encoded as a user query expressed in some temporal logic. Por fortuna hay una manera directa de mapear esa probabilidad a la fórmula PCTL

$$\mathsf{P}(\neg B \,\mathsf{U}\, A) \tag{7}$$

i.e. "la probabilidad de no observar la condición de parada B until the rare event A takes place". Notice such query is not pure PCTL since it asks for the numeric probability value rather than whether that value is greater or less than some bound. That is however of no concern since P can appear only at top level and not as sub-expression of neither A nor B, so the last remarks from la Sección 2.2 apply. Also events A and B need to be encoded as logic formulae. Since PCTL subsumes propositional logic then e.g. A can be simply "$\neg$ SATURADO". De aquí en más el *tanálisis transitorio* para RES será una referencia implícita a la Definición 8 ó a la ecuación (7).

Definición 9. En este entorno formal, la *probabilidad estacionaria del evento raro* será el valor probabilístico

$$\gamma = \lim_{t \to \infty} P(X_t \in A)$$

también llamado la *probabilidad a largo plazo del evento raro.*

En lugar de escribir $P(A)$, el proceso estocástico subyacente X is made explicit in la Definición 9 to highlight the dependence on the asymptotic time limit. Steady-state studies have also appeared in the RES literature, most notably in the research around the RESTART splitting technique (see e.g. [VAVA91, VAMMGFC94]). For a formalization resembling the one given above the user is referred to [Gar00, Cap. 6], donde el análisis estacionario es definito en términos de procesos regenerativos.

La Definición 9 pregunta por la proporción temporal spent visiting rare states when the system is *in equilibrium*. For Markovian processes it is easy to compute the transition rates that characterize a system in equilibrium, but for general stochastic processes that is usually too hard. El método de simulación

estándar es descartar parte de la ejecucupin inicial considered transient, and then proceed using the *batch means* technique [LK00], que favorece el descarte del comportamiento transitorio de las simulaciones.

Al igual que en el caso transitorio, some related formula from a temporal logic is desired to characterize user queries of the probability in la Definición 9. La fórmula CSL

$$\mathsf{S}(A) \tag{8}$$

habla de "la probabilidad del evento A en un sistema en equilibrio", i.e. the time proportion in the long run that states in A are visited. This fits in CSL in the same way ecuación (7) does with PCTL. From here onward the *steady-state analysis* for RES will be referring to either la Definición 9 ó la ecuación (8).

En ambos el análisis transitorio y el estacionario para RES, the value γ is positive but very small since the likelihood of reaching its characterizing set A is extremely low. Importance splitting is based on the assumption that there are identifiable *intermediate states subsets* which *must be visited* to reach the rare event, and which are *much more likely* than A to be reached by a simulation path. Formalmente, se supone la existencia de una secuencia decreciente de eventos

$$S = E_0 \supset E_1 \supset \cdots \supset E_{n-1} \supset E_n = A,$$

donde una función de proyección $f\colon S \to \mathbb{R}_{\geqslant 0}$ called the *importance function* determines events E_i. Since it defines the intermediate events which are the cornerstone of the technique, this function is a key component in multilevel splitting. Level values $L_i \in \mathbb{R}_{\geqslant 0}$ typically called *umbrales* are chosen satisfying $L_i < L_{i+1}$ for $0 \leqslant i < n$. Los eventos son definidos por medio de f y los umbrales:

$$\forall i \in \{0, 1, \ldots, n\} \,.\, E_i \doteq \{s \in S \mid f(s) \geqslant L_i\}.$$

A lo largo de la tesis y a menos que se noted otherwise it will be $L_0 \doteq 0$ and $L \doteq L_n$, resulting in $A = \{s \in S \mid f(s) \geqslant L\}$. Furthermore all simulation paths start in the initial state of the system $s_0 \in S = E_0$[†], which should ideally have minimum importance, i.e. $f(s_0) = L_0 = 0$. Nonetheless the more general condition $f(s_0) < L_1$ basta para asegurar que $s_0 \notin E_1$ como se desea.

Notar que en este entorno, every simulation path must increasingly traverse all events before reaching a state in A. Besides, considering that $P(E_0) = P(X_t \in S) = 1$ y $E_{i+1} \subset E_i$ para $0 \leqslant i < n$, la identidad

$$\begin{aligned}
\gamma &= P(E_n) \\
&= \frac{P(E_n)}{P(E_{n-1})}\frac{P(E_{n-1})}{P(E_{n-2})}\cdots\frac{P(E_2)}{P(E_1)}\frac{P(E_1)}{P(E_0)}\,P(E_0) \\
&= \frac{P(E_n \cap E_{n-1})}{P(E_{n-1})}\frac{P(E_{n-1} \cap E_{n-2})}{P(E_{n-2})}\cdots\frac{P(E_2 \cap E_1)}{P(E_1)}\frac{P(E_1 \cap E_0)}{P(E_0)}\,P(E_0) \\
&= P(E_n|E_{n-1})\,P(E_{n-1}|E_{n-2})\cdots P(E_1|E_0)\,P(E_0) \\
&= \prod_{i=0}^{n-1} p_i
\end{aligned} \tag{9}$$

[†] También podría considerarse una distribución inicial.

se sigue por definición de probabilidad condicional, donde la *probabilidad condicional de subir un nivel*, del evento E_i al evento E_{i+1}, se denota

$$p_i \doteq P(E_{i+1} \mid E_i) \qquad \text{para } \ 0 \leqslant i < n. \tag{10}$$

La eficiencia de la división multinivel depends on choosing the importance function and the thresholds s.t. $p_i \gg \gamma$ for all i. Thus a stepwise estimation of the p_i can be done more efficiently than una estimación directa de γ.

Como el tiempo no aparece explícitamente, the previous considerations can be directly mapped to steady-state analysis for a system in equilibrium (cf. [VAMMGFC94, Sec. 2.2], where the probability of the rare event is defined in a way matching la ecuación (9) para el renombrado $(E_i, p_i, \gamma) \mapsto (C_{i+1}, P_{i+1}, P)$).

Los estudios transitorios están basados en los mismos principios. Usually a filtration $\{A_i\}_{i=0}^n$ is defined for $A_i \doteq \{T_i \leqslant T_B\}$ where T_i is the entrance time into event E_i, so $P(A_n) = \gamma$ and $P(A_0) = 1$. In such setting the conditional probabilities are defined in terms of the filtration: $p_i = P(A_{i+1}|A_i)$. This line of analysis reaches an identity analogous a la ecuación (9). Ver [Gar00, Sec. 2.4] y [LLGLT09, pp. 42–45] para un estudio detallado.

La mayoría de las implementaciones prácticas de la técnica de división do not allow a fully independent estimation of the conditional probabilities, because the entrance distribution to E_i affects estimates of p_i to a great extent, conditioning also the estimates for all p_j with $j > i$. Still these probabilities can be computed somewhat separately by taking into account a statistical approximation of the entrance states to each E_i, which resemble the real entrance distributions asymptotically. A continuación describimos la estrategia multinivel general.

Haciendo abuso de la notación, el evento $Z_i \doteq E_i \setminus E_{i+1}$ será llamado de ahora en más la i-ésima *zona de importancia* o *nivel*, y los valores $\{L_i\}_{i=0}^n$ serán referidos exclusivamente como umbrales. De esta forma el i-ésimo nivel de importancia será el conjunto de estados que la función de importancia coloca entre los umbrales L_i (incluyéndolo) y L_{i+1} (excluyéndolo). El estado inicial del sistema estará situado en el 0-ésimo nivel de importancia (el *nivel de base*) y los estados raros pertenecerán n-ésimo (o último) nivel. La Figura 2.6 ofrece una representación esquemática de esta notación.

Definición 10 (Probabilidad de subir de nivel)**.** Supongamos que los eventos $\{E_i\}_{i=0}^n$ definen los niveles de importancia en un marco de división por importancia. Sea S_i la variable aleatoria que describe los estados de entrada de E_i para cualquier camino simulado. La *probabilidad de subir desde el nivel i hasta el nivel $i+1$* es la probabilidad de que un camino simulado visite E_{i+1}, condicionado en la distribución de entrada del nivel i: $P\left(E_{i+1} \mid S_i\right)$.

Se puede pensar en la probabilidad de subir de nivel como la probabilidad de que un camino simulado que empieza en el umbral inferior de la i-ésima zona alcance el umbral superior de la misma [Gar00, Sec. 2.4]. La Definición 10 ayuda a conectar las nociones matemáticas definidas hasta ahora, con las técnicas de simulación que serán descriptas algorítmicamente. En particular, el siguiente resultado será útil a la hora de probar la ausencia de sesgo del método.

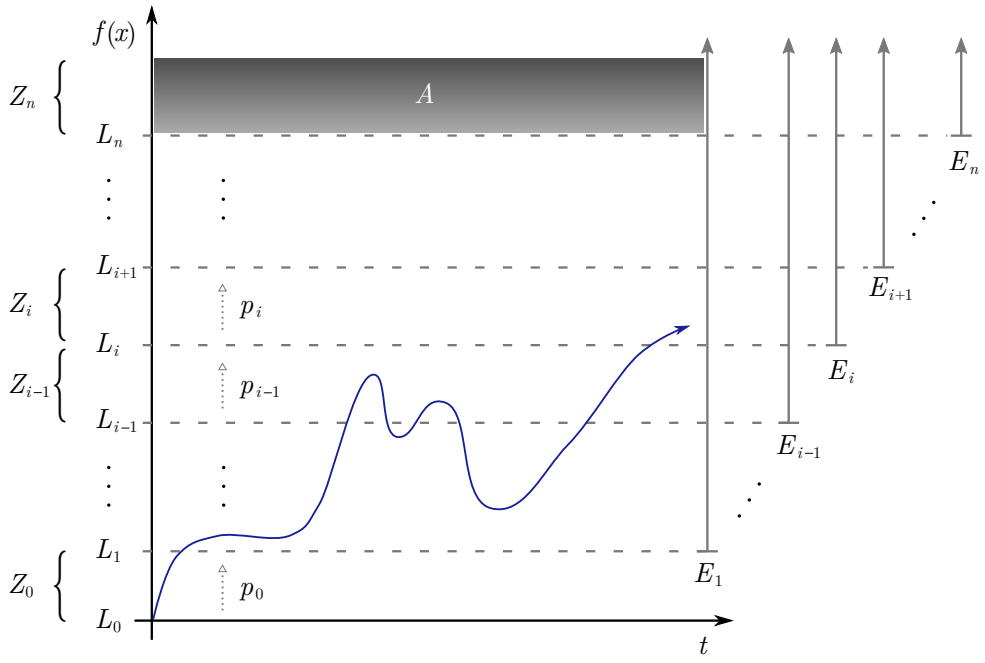


Figura 2.6: Escenario de la división multilevel

Proposición 4.

$$P\left(E_{i+1} \mid S_i\right) = p_i$$

Demostración. El marco en el que p_i fue definido para la ecuación (10) involucra caminos simulados que atraviesan E_i para alcanzar algún estado de E_{i+1}. Como el proceso X es markoviano, los estados de entrada del evento E_i determinan completamente el futuro de las simulaciones que atraviesan esta región. Esto significa que la variable aleatoria S_i de la Definición 10 condensa toda la información referida a una trayectoria que atraviesa E_i. Por ende, en el marco de las ecuaciones (9) y (10), condicionar en E_i es equivalente a condicionar en S_i. □

Método básico de división multinivel para RES

Sea T el tiempo definido por la condición de fin de simulación, ya sea éste un tiempo casi seguramente finito de entrada en T_B de un análisis transitorio, o un horizonte finito temporal (o duración de un ciclo de regeneración) en la implementación por baches de un análisis estacionario. Sea a su vez T_i el tiempo de entrada en el i-ésimo nivel de importancia, viz. la primera vez que una simulación visita un estado $s \in S$ t.q. $f(s) \geqslant L_i$.

Comience N_0 caminos de simulación independientes a partir del estado inicial del modelo del sistema, s_0. Cada uno de estos *caminos originales* avanza hasta llegar o bien a T o sino a T_1. Esto será denotado la *etapa 0* o *primera etapa* de la división multinivel.

Sea R_0 el número de simulaciones que alcanzaron el primer nivel de importancia, e.g. para los cuales $T_1 < T$. Se corrieron por ende N_0 experimentos

independientes en esta primera etapa, cada uno de los cuales tenía una probabilidad (incondicional) igual a p_0 de tener éxito. Como R_0 cuenta el número de éxitos, estamos hablando de una distribución binomial: $R_o \sim \text{Bin}(N_0, p_0)$. Se sigue que $\mathbb{E}(R_0) = N_0\, p_0$, donde $\mathbb{E}(Y)$ denota el *valor esperado* de la v.a. Y.

Nótese que $\widehat{p}_0 \doteq {}^{R_0}/_{N_0}$ es un unbiased estimator for p_0, because $N_0 \in \mathbb{N}$ and thus $\mathbb{E}(\widehat{p}_0) = {}^{1}/_{N_0}\,\mathbb{E}(R_0) = p_0$. Furthermore states $\{s_1^k\}_{k=1}^{R_0} \subseteq E_1$ realizing the *successful trajectories* are an empirical sample of the entrance distribution into E_1. Así que cada s_1^k es una observación de la variable aleatoria S_1 de la Definición 10.

A continuación, en la etapa 1, N_1 *réplicas de simulación* o *copias* are started from those R_0 states. In order to maintain a sufficiently large sampling population and assuming R_0 can be small, it is expected that $N_1 > R_0$. By the pigeonhole principle this means more than one simulation may be started from each state. The selection can be done by cloning (or *splitting*) the simulations that reached each s_1^k, or choosing randomly where to start each of the N_1 simulaciones de las R_0 opciones disponibles.

Cada nueva trayectoria es nuevamente simulated from its starting state until either T or T_2 occur, whichever happens first. Let R_1 be the number of simulations where $T_2 < T$. Stage 1 thus consists of N_1 experiments, and the r.v. R_1 counts the successful simulaciones que alcanzaron el segundo nivel de importancia.

No obstante, una distribución binomial can not be unconditionally assumed, because not all simulations are necessarily independent. Some may have started from the same state s_1^k, compartiendo la historia hasta ese punto.

Recordemos sin embargo que los estados $\{s_1^k\}_{k=1}^{R_0}$ have an asymptotic behaviour described by the distribution of S_1. Thus when conditioned on such random variable, la etapa 1 puede realmente ser considerada como un experimento Binomial.

Para ganar un poco de intuición, think that knowing the full history back to s_0, where the original N_0 *independent* simulations were bootstrapped, suffices to grant the statistical independence sought in the simulations of stage 1. Moreover, conditioning on S_1 is reasonable because the starting states of stage 1 son una muestra empírica de la variable aleatoria.

Por la Proposición 4, each of the N_1 launched simulations succeeds with probability $P(E_2 \mid S_1) = p_1$. Furthermore $\mathbb{E}(R_1 \mid S_1) = N_1\, p_1$. Consider the estimator $\widehat{p}_1 \doteq {}^{R_1}/_{N_1}$, luego claramente $\mathbb{E}(\widehat{p}_1 \mid S_1) = {}^{1}/_{N_1}\,\mathbb{E}(R_1 \mid S_1) = p_1$.

Generalizando, en la i-ésima etapa se lanzan N_i caminos de simulación desde las previas R_{i-1} simulaciones exitosas. Esto se repite hasta que se alcanza el último nivel. Entonces ya todos los estimadores $\{\widehat{p}_i\}_{i=0}^{n-1}$ habrán sido calculados, cuyo producto es un estimador del valor buscado γ:

$$\hat{\gamma} \;=\; \prod_{i=0}^{n-1} \widehat{p}_i \;. \tag{11}$$

Es interesante el hecho de que, si bien la ecuación (11) provee un estimador insesgado de γ como demostraremos a continuación, las proporciones intermedias $\widehat{p}_i = {}^{R_i}/_{N_i}$ dependen entre sí como se lo indicó (excepto por $\widehat{p}_0$). Esto es una consecuencia de la dependencia que existe entre las trayectorias generadas en la i-ésima etapa, y la distribución de entrada del i-ésimo nivel de importancia.

Esta dependencia puede ser muy fuerte si la función de importancia y los umbrales no son escogidos con cuidado, lo cual puede reducir de forma drástica la eficiencia de la técnica de división. Este tema se aborda en mayor profundidad en la Sección 2.7.

Prueba de que el estimador es insesgado

La idea de base es intercambiar producto por esperanza en la secuencia $\{\widehat{p}_i\}_{i=0}^{n-1}$ de estimadores de la ecuación (11). Nótese primero que las reflexiones con las que se probó que $\widehat{p}_1$ es insesgado pueden ser extrapoladas a cualquier estimador, siempre y cuando se conozca toda la historia de las trayectorias hasta el correspondiente nivel de importancia.

Formalmente denótese $\mathcal{F}_k$ la σ-álgebra asociada con el proceso estocástico $\{S_i\}_{i=1}^{k}$, para $1 \leqslant k < n$. Entonces

$$\mathbb{E}(\widehat{p}_i \mid \mathcal{F}_i) = p_i$$

para todo $0 < i < n$ (cf. [Gar00, equations 2.5 to 2.8]). Consecuentemente por la ley de esperanza total, para cualquier par de índices $i, j \in \{0, \ldots, n-1\}$ (s.p.d.g. sea $i < j$, cf. [Gar00, eq. 2.13])

$$\begin{aligned}
\mathbb{E}(\widehat{p}_i\widehat{p}_j) &= \mathbb{E}(\mathbb{E}(\widehat{p}_i\widehat{p}_j \mid \mathcal{F}_j)) \\
&= \mathbb{E}(\,\widehat{p}_i\,\mathbb{E}(\widehat{p}_j \mid \mathcal{F}_j)) \\
&= \mathbb{E}(\widehat{p}_i p_j) \\
&= \mathbb{E}(\widehat{p}_i)\,p_j \\
&= \mathbb{E}(\mathbb{E}(\widehat{p}_i \mid \mathcal{F}_i))\,p_j \\
&= p_i p_j\,.
\end{aligned}$$

Teorema 5 (La técnica de división es insesgada)**.** *En el método de división descripto, el valor esperado del estimador de la ecuación* (11) *es la probabilidad del evento raro de la ecuación* (9)*:*

$$\mathbb{E}(\hat{\gamma}) = \gamma.$$

Demostración. Aplicando recursivamente el argumento anterior se obtiene $\mathbb{E}(\prod_{i=0}^{n-1}\widehat{p}_i) = \prod_{i=0}^{n-1} p_i$. La igualdad buscada se desprende de esto y de las ecuaciones (9) y (11). □

2.5.2. Variantes de la técnica básica de división

Existen muchas formas de implementar la división multinivel. El método básico descripto en la Subsección 2.5.1 es sólo un ejemplo que se presta bien para propósitos introductorios y para probar la ausencia de sesgo del estimador. Un vistazo general muy bien organizado de varias implementaciones alternativas se presenta en [LLGLT09, Sec. 3.2.2], del cual se transcribe a continuación un resumen revisado.

Como elegir el número de copias N_i en cada etapa es una decisión central. Algunas estrategias típicas son:

- *División fija.* En la i-ésima etapa, cada camino de simulación exitoso que alcance al umbral superior generates the same number of offsprings $K_i \in \mathbb{N}$. The total number of simulation paths $N_{i+1} = R_i K_i$ in the next stage is thus a random variable. This is sensitive to both the $\{K_i\}_{i=1}^{n-1}$ and the $\{R_i\}_{i=0}^{n-1}$. If $R_i = 0$ the technique suffers from *starvation* since no offsprings will be produced. If $K_i \gg {}^{N_{i+1}}/_{R_i}$ then too many offsprings will be produced y la técnica sufre de *sobrecarga computacional*.

- *Esfuerzo fijo.* Un número preteterminado $N_i \in \mathbb{N}$ de copias is started during the i-th stage. If $R_{i-1} < N_i$ then these offsprings can be assigned to the available R_{i-1} starting states randomly or deterministically. This rules out the possibility of overhead, but can suffer from starvation si N_i es demasiado chico para asegurar que $R_i > 0$.

- *Éxito fijo.* El número $R_i > 0$ de simulaciones exitosas in the i-th stage is predetermined. Thus N_i is a random variable for the i-th stage, since sufficient simulations need to be launched to reach the desired R_i. This can cause computational overhead but no puede sufrir de inanición por definición.

Estas tres estrategias tienen different performance implications. Fixed splitting can be considered lightweight w.r.t. memory consumption, since it allows a *depth-first search* (DFS) implementation [LLGLT09, p. 45]. Namely, during the first stage each original path is simulated until mín(T, T_1). If T_1 takes place it yields K_1 offsprings from that path; then each of these offsprings is simulated until mín(T, T_2), and so on, antes de pasar a manejar el siguiente camino original.

Este método DFS approach can not be applied to the other two alternatives, which need to attend one importance level at a time and keep in memory all resulting estados de entrada en cada nivel.

En el método básico introducido, all simulation paths have the same weight at any importance level. In a fixed splitting scenario, consider a more general setting where trajectories can be assigned different weights. Each of the N_0 original trajectories in the bottom importance level will have weight 1. During the next stage in the first importance level, each offspring will have *peso relativo* ${}^1/_{K_1}$, dado que proviene de una trayectoria original con peso 1 que fue dividida K_1 veces.

Esto significa que los caminos exitosos in the uppermost level will have relative weight $(\prod_{i=1}^{n-1} K_i)^{-1}$. Then the estimator $\hat{\gamma}$ is the sum of relative weights of these trayectorias exitosis finales, dividido por N_0.

Esta estrategia generalizada, which takes the relative weights of the simulation paths into account, is of special use when the rare event can appear in low importance levels. Más sobre esto en la Sección 2.6.

El estimador $\hat{\gamma}$ de la ecuación (11) es *eficiente*, because its variance is smaller than the standard Monte Carlo estimator of γ [Gar00, Sec. 2.4.3]. A smaller variance means less samples are needed to converge. Nevertheless, when practical

applications are considered, the computation time of each sample has a impacto directa en el tiempo de pared de convergencia.

Por ejemplo en esfuerzo fijo y éxito fijo, paths are simulated until the upper threshold or final time T are met. In transient analysis the average computational time to reach T may increase significantly with the importance level i donde el camino comenzó [LLGLT09, p. 46].

Simétricamente, el paralelismo computacional máximo can act as bottleneck in fixed splitting. Since new paths are injected each time an upper threshold is reached, there is a risk of having an exponential explosion in the resulting número de trayectorias concurrentes.

Para mantener controlada la sobrecarga computacional derived from potentially long simulation paths, early path termination (aka *truncado de caminos*) is a typical strategy. The essence of path truncation is to select and kill ideally unpromising trajectories, before its "natural cause of death" (e.g. alcanzar T) tiene lugar, Se han estudiado diversas estrategias:

- *Truncado determinista.* Para alguna *profundidad de muerte* $\beta \in \mathbb{N}$, matar cualquier trayectoria que se submerges more than β levels from its creation level. That is, if some path originated from an entrance state into E_i, it will be truncated as soon as it visits a state in level $i - 1 - \beta$ or lower. This requires a proper weighing of paths to avoid introducir un sesgo en la estimación.

- *Truncado probabilístico.* Para la profundidad de muerte β, los caminos de i-ésimo nivel juegan a la *ruleta rusa* each time they down-cross a level deeper than $i - \beta$. More precisely, numbers $\{r_{i,j}\}_{j=\beta}^{n-1} \in \mathbb{R}_{>0}$ are chosen for paths originated in the i-th importance level. These are truncated with probability $1 - 1/r_{i,j}$ as they cross level $i - 1 - j$ downwards for $j \geqslant \beta$. On survival their weight is multiplied by $r_{i,j}$. To reduce the variance introduced by weighing, a simulated trajectory of weight w is cloned $w - 1$ times as it reaches the uppermost level; luego cada uno de estos caminos nuevos independientes tiene peso 1.

- *Truncado periódico.* Como en la versión probabilística, números $\{r_{i,j}\}_{j=\beta}^{n-1} \in \mathbb{R}_{>0}$ son escogidos para las trayectorias del i-ésimo nivel and global $\beta \in \mathbb{N}$. To reduce the variability of the Russian roulette approach numbers $D_{i,j}$ are uniformly chosen once among $\{1, \ldots, r_{i,j}\}$. Then for i-th level trajectories, every $(r_{i,j}\, D_{i,j})$-th path to go down level $i - 1 - j$ for $j \geqslant \beta$ is retained and its weight is multiplied by $r_{i,j}$; todos los otros caminos del i-ésimo nivel que hagan esto son truncados.

- *Truncado etiquetado.* Cada camino de i-ésimo nivel es *etiquetado* al nivel de importancia $(i - 1 - j)$ con alguna probabilidad que aumenta con j para $j \geqslant \beta$. Las trayectorias son truncadas sii visitan sus niveles etiquetados.

Implementaciones populares

Algunas elecciones de los criterios mencionados más arriba han sido muy estudiadas y exitosamente aplicadas a varios casos de estudio, volviéndose criterios convencionales en la comunidad de RES. A continuación describimos tres de ellos.

- *RESTART* es un método desarrollado por los hermanos Villén-Altamirano que estudiaremos en mayor detalle en la próxima sección. Sigue una implementación DFS que usa división fija cada vez que un camino simulado atraviesa un umbral hacia arriba. Cuando esto ocurre una única copia es etiquetada como la *trayectoria original del nuevo nivel*. El truncado es determinista con $\beta = 0$, i.e. toda copia que cruce hacia abajo su umbral de creación es terminada. Para evitar la inanición (i.e. quedarnos sin copias que simular) la copia original de cada nivel no es truncada al bajar de ese umbral, asemejándose en parte al truncado etiquetado. Hay un único camino original de simulación por cada ejecución RESTART, con peso 1, y el esquema de *pesos relativos* es empleado con $N_0 = 1$.

- *Esfuerzo Fijo* es un término utilizado por Marnix Garvels en su tesis doctoral para referirse a un método de división de naturaleza BFS (*breadth-first search*) bastante similar a la técnica básica descripta en la Subsección 2.5.1. También ha sido llamado *división* a secas en una comparación contra RESTART [VAVA06]. El método de Esfuerzo Fijo consiste en iteraciones que comienzan en el nivel de importancia de base, donde se truncan los caminos simulados ni bien alcanzan el tiempo de parada T (*fracasan*) o el siguiente nivel de importancia (*tienen éxito*). Los estados de entrada al siguiente nivel, señalados por las simulaciones exitosas de esta primera etapa, son salvados y usados como puntos de partida para las simulaciones de la siguiente etapa. Así se va cubriendo un nivel de importancia por etapa, hasta que finalmente se alcanza el nivel del evento raro. Al llegar a ese punto todos los estimadores $\widehat{p}_i$ para las probabilidades condicionales de la Definición 10 han sido computados. Sólo resta multiplicarlos para estimar el evento raro siguiente la ecuación (11).
  Este método usa esfuerzo fijo con elección determinista como mecanismo de generación de copias, y truncado determinista. Nótese que los caminos simulados que descienden niveles de importancia no son truncados, como sí ocurriría en RESTART. Además, no es necesario un mecanismo de pesos relativos dado que no se permite a los caminos simulados el cruce de los umbrales; en ese sentido lo único que hacen los caminos es señalar los estados de entrada del próximo nivel, a partir de los cuales se generarán las simulaciones de la nueva etapa.

- *División Multinivel Adaptativa* [CG07] y *Monte Carlo Secuencial* [CDMFG12] son métodos más difíciles de ubicar en el escenario descripto, dado que saltean la pre-selección de los umbrales $\{L_i\}_{i=1}^{n}$. En su lugar, estos métodos descubren los umbrales dinámicamente mientras simulan caminos de ejecución que van siendo guiadas hacia el evento raro. El usuario sólo

debe indicar la probabilidad p_i de subir de nivel a priori, lo que vuelve al número n de umbrales la nueva variable aleatoria. Para mayor claridad digamos que $p = p_i$ para todo $i \in \{0, \dots, n\}$, donde n es desconocido. En la i-ésima etapa, m caminos son simulados independientemente a partir de estados predefinidos hasta que alcanzan algún criterio de parada, e.g. llegar al tiempo T. Cada camino simulado visitó varios estados con sus respectivos valores de importancia. Sea v_j^i el máximo valor de importancia observado por el j-ésimo camino simulado, entonces la i-ésima etapa genera el conjunto de datos $V_i = \{v_j^i\}_{j=1}^m$. Para $k = \lceil p\, m \rceil$, sea ν_k^i el k-ésimo cuantil de V_i. Es decir, si ordenamos a V_i en orden creciente, sea ν_k^i el valor en la k-ésima posición. Entonces ν_k^i es el candidato a "umbral superior" de la i-ésima etapa, debido a cómo se lo seleccionó. Además, todos los estados visitados con importancia ν_k^i sirven como potenciales puntos de entrada para las simulaciones de la próxima etapa.
Eventualmente el evento raro es alcanzado y el número de etapas n es determinado, lo que deriva en el estimador para el evento raro $\hat{\gamma} \doteq p^n$. Estas implementaciones del método de división por importancia son ideales cuando se cuenta con un espacio de estados continuo, mientras que puede presentar problemas prácticos si se aplica a modelos discretos.

2.6. RESTART

En el material bibliográfico existente para RES, uno de los métodos de división por importancia de mayor renombre es RESTART. Ya en 1970 A. J. Bayes introdujo un mecanismo de simulación acelerado para estimar la probabilidad de que un proceso estocástico se halle en un estado perteneciente a un conjunto raro [Bay70], con muchas de las propiedades que mencionamos más arriba. Veinte años más tarde en 1991 José y Manuel Villén-Altamirano redescubrieron el método en [VAVA91], atinando el famoso acrónimo. Más adelante generalizaron su trabajo para que opere con múltiples umbrales en [VAMMGFC94], y luego para que soporte un evento raro que podía ocurrir en cualquier nivel de importancia [VA98]. Ambas la versatilidad de la técnica y su relativa facilidad de implementación la vuelven un candidato perfecto para el grado de generalidad que busca esta tesis. Un análisis más profundo de sus características se presenta por ende en esta sección.

Una explicación minuciosa de (una versión madura de) RESTART can be found in [VAVA11, Sec. 2], a transcription of which is given next using a notation more compliant with the one presented. A Markov process $X = \{X_t \mid t \geqslant 0\}$ is assumed, and thresholds $\{L_i\}_{i \geqslant 0}^n$ are defined on the real line, determining *importance regions* $S = E_0 \supset E_1 \supset \cdots \supset E_n \supseteq A$ for a rare set A. This is done via an importance function $f \colon S \to \mathbb{R}$ which maps the state space S of X into the real line, so $E_i = \{s \in S \mid f(s) \geqslant L_i\}$. Notice "region E_i" here is nothing else but "event E_i" from the formal setting previously introduced for general importance splitting. Likewise importance zones $Z_i \doteq E_i \setminus E_{i+1}$ (denoting $Z_n \doteq E_n$) create a partition of S where the higher the value of i, mayor es la importancia de los estados contenidos.

Dado un camino simulado que atraviesa S, let a *rare event* or A *event* refer to this path taking a transition whose target is a rare state $s \in A$. Define an E_i *event* in the same way for any region E_i. Let also a B_i *event* denote the path taking a transition $s \to s'$ whose originating and target states satisfy $f(s) < L_i$ and $f(s') \geqslant L_i$ respectively. That is, a B_i event tells when the simulation has "gone up" into the i-th importance region. Equivalently define a D_i *event* when the simulation "goes down" from the i-th importance region into cualquier zona Z_j con $j < i$. RESTART trabaja así.

1. Un camino simulado llamado *el camino principal* comienza del estado inicial $s_0 \in Z_0$. This path will last until a predefined end-of-simulation condition is met, say a finite time horizon T or an almost surely finite tiempo de entrada en el conjunto de parada.

2. Cada vez que un evento B_1 occurs in the main trial the system state is saved, the main trial is interrupted, and $K_1 - 1$ offsprings or *copias de nivel 1* son generadas.

   ▷ Una copia es sólo an independent simulation path which originates from the entrance state into some higher region by another trial. In this case, por el camino principal ingresando en E_1.

   ▷ Una copia de nivel 1 is truncated when it causes a D_1 event (viz. goes down to zone Z_0) or meets the condición de fin de simulación.

   ▷ El hilo de ejecución de la computadora switched from the main trial to its offsprings, following the DFS approach described for fixed splitting in Subsección 2.5.1. An equivalent mechanism will be set in motion as estas copias generan eventos B_2.

3. Luego de que las $K_1 - 1$ copias have been attended until truncation, the main trial is restored at the saved state del evento B_1.

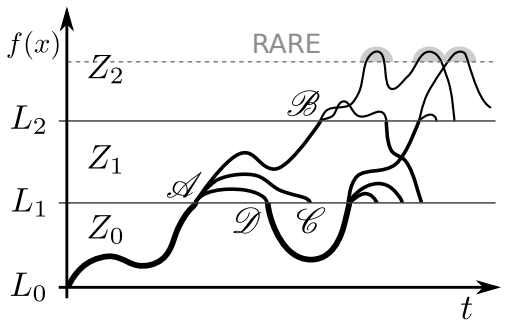
   ▷ Incluyendo al camino original, the total number of simulated paths between events B_1 and D_1 is K_1. Each of these K_1 trajectories se llama una copia $[B_1, D_1)$.

   ▷ Sólo el camino principal can continue after D_1, potentially generating new B_1 events y así evita la inanición.

4. Cada copia $[B_1, D_1)$ puede have triggered a B_2 event during its execution. As this happens an analogous process is set in motion: $K_2 - 1$ offsprings of level 2 are launched, starting in the state which caused B_2 y terminando en un evento D_2.

   ▷ La copia de nivel 1 that generated the B_2 event is tagged the original trial for this level, and is the only one que sobrevivirá al evento D_2.

   ▷ Al igal que el camino original, this original trial of level 1 can then generate more B_2 events. It will be killed however if it generates a D_1 event, since it is a retrial of nivel 1 y por ende una copia $[B_1, D_1)$.



La traza simulada evoluciona desde el estado inicial hasta que el umbral L_1 es cruzado *hacia arriba* en $\mathscr{A}$. Allí se divide en $K_1 = 3$, pues ese es un evento B_1. Las dos copias de nivel 1 evolucionan independientemente: una llega al umbral L_2 en $\mathscr{B}$ generando un evento B_2 y dividiéndose en $K_2 = 2$; la otra cruza L_1 *hacia abajo* en $\mathscr{C}$, generando un evento D_1 que la trunca. La simulación original también genera un evento D_1 en $\mathscr{D}$, pero lo sobrevive porque es la traza original del nivel 0.

Figura 2.7: Ejemplo de la ejecución de RESTART

- ▷ Contando al camino original de nivel 1, there are K_2 trials of level 2, denotadas copias $[B_2, D_2)$.

5. En general para $1 \leqslant i \leqslant n$, $K_i \in \mathbb{N}$ is the number of $[B_i, D_i)$-trials launched each time a B_i event is triggered por una copia $[B_{i-1}, D_{i-1})$.

   - ▷ K_i se llama el *i-ésimo factor de división* or *splitting value*. It is a constant chosen a priori by the user, con la restricción $K_i > 1$.

Ver la Figura 2.7 para una descripción esquemática de este procedimiento de división.

El método descripto se basa en an ideal implementation, where the rare event is entirely contained in the uppermost region and a simulation path can rise by at most one importance region at each step. In such setting any trajectory visiting a rare state has traversed all splitting stages, which stacked up on every threshold crossed. An unbiased estimator is obtained applying the relative weighing scheme with a weight equal to 1 for the main trial. Thus the relative weight of a level n retrial producing a rare event in zone Z_n is $1/K$, para el factor de división acumulado $K \doteq \prod_{i=1}^{n} K_i$.

Nótese que, si las simulaciones were monotonically increasing in importance, such a stacked splitting factor is actually the maximum number of offpsrings which could be concurrently running in the uppermost importance region. Thus K can also be introduced as the *statistical oversampling* incurred by the offsprings de nivel n que visitan al conjunto raro A.

Para dar una fórmula explícita del estimator derived from a RESTART simulation, consider first a steady-state analysis in a continuous time model. Say M retrials of level n eventually make it to the rare set. Say also the simulation time each one spends on the rare event is given by $\{t_j^*\}_{j=1}^{M}$, with some finite time horizon $T < \infty$ of a batch means run. Sea $T^* \doteq \sum_{j=1}^{M} t_j^*$, entonces un estimador

insesgado para la proporción temporal (viz. la probabilidad estacionaria) del evento raro es

$$\hat{\gamma} \doteq \frac{T^*}{K\,T}$$

para el factor de división acumulado K, in a single batch means run with T units of simulation time. Proofs for the unbiasedness of this estimator can be found in [VAVA02, VAVA11]†.

RESTART también puede usarse para análisis transitorio, obtaining an equally unbiased estimator (see e.g. [GHSZ98, GHSZ99, GK98, GVOK02]). The idea is to launch N_0 main trials instead of the single one of steady-state analysis, since each trial is expected to be short lived. Say M retrials of level n make it to the rare set A before entering the stopping set B. These have been benefited from the stacking up of the splitting mechanism, thus each has relative weight $1/K$. There is no permanence time to measure, since simulations are truncated as soon as they visit a rare state; what counts is them having reached A before B. So M can be thought of as the number of successes in a "Binomial-RESTART experiment", where each of the N_0 main trials is a Bernoulli-RESTART experiment. El estimador en este caso es:

$$\hat{\gamma} \doteq \frac{M}{K\,N_0}$$

donde K pondera el sobre-muestreo actuando como peso relativo de las M simulaciones exitosas.

Respecto de la perspectiva Binomial and compared to the binary outcome of a standard Bernoulli experiment, a Bernoulli-RESTART run has a $(K{+}1)$-ary outcome. That is because each experiment can take any value in $\{0, 1/K, 2/K, \ldots, K-1/K, 1\}$, debido a la naturaleza de división de las simulaciones.

Como ya se mencionó, all of the above applies to an ideal implementation of the technique, where it is assumed a simulation path can rise by at most one importance region at each step. Generally speaking the definition of the Markov process X and the choice of importance function may allow a simulation path to jump over some importance zone, viz. taking a transition $s \to s'$ with $s \in Z_{i-1}$ and $s' \in E_{i+1}$. In such cases it must be considered that several B_i events occurred simultaneously, and the corresponding splitting, tagging, y salvado de los estados debe ser llevado a cabo acordemente.

Por ejemplo, digamos que la transición $s \to s'$ jumps over zone Z_i, e.g. $f(s) < L_i$ and $f(s') = L_{i+1}$. Since that is a B_i event, $K_i - 1$ retrials of level i have to be launched starting in state s' and finishing when an event D_i occurs. Yet taking that transition also means each of those trials (including the original one from level $i-1$) is causing a B_{i+1} event. Since the total number of $[B_i, D_i)$-trials is K_i, then $K_i(K_{i+1} - 1)$ retrials of level $i+1$ are also started from state s', which will finish when an event D_{i+1} takes place. En total hay pues $K_i K_{i+1}$ caminos simulados: el camino original de nivel $i-1$; las $K_i - 1$ copias de nivel i que serán

† Una prueba generalizada muy interesante se encuentra en *Recent Advances in RESTART Simulation*, un seminario que los hermanos Villén-Altamirano presentaron en RESIM 2008.

trunvadas por un evento D_i; y las $K_i(K_{i+1} - 1)$ copias de nivel $i + 1$ que serán truncadas por un evento D_{i+1}.

Otra complicación potencial es tratar con un evento raro que puede ocurrir en cualquier región de importancia, no sólo en E_n [VA98]. Desde el punto de vista implementativo esto puede arreglarse con relativa facilidad. Basta con considerar el peso relativo de los caminos simulados que visitan los estados raros, que no será necesariamente $1/K$ sino más bien $W_\ell \doteq 1/\prod_{i=1}^{\ell} K_i$ para un estado raro que se ubique en la zona Z_ℓ con $\ell \leqslant n$. Esta actualización puede inyectarse en los estimadores descriptos más arriba si se emplean etiquetas en las simulaciones: copias de de nivel i serán etiquetadas con su correspondiente peso relativo W_i y la división global por K es reemplazada por la multiplicación con el correspondiente W_i, como se hace en [VA98, eq. (2)].

Nótese sin embargo que en un escenario donde hay estados raros en zonas Z_ℓ con $\ell < n$, el muestreo del evento raro no estará beneficiado por todo el poder del mecanismo de división. Por ende la eficiencia del método se deteriora, lo cual es analizado en profundidad en [VA98, VAVA11].

2.7. Aplicabilidad y eficiencia de la división multinivel

RESTART es realmente versátil, en tanto que puede ser aplicado a muchos tipos de sistemas, por ejemplo [VAMMGFC94, GK98, GHSZ99, VAVA06, VA07b, VA07a, VA09, VAVA13, VA14, BDH15]. Aún así es importante saber qué tan eficientemente puede ser aplicado en cada caso.

En [GHSZ98, GHSZ99] se dan condiciones suficientes para lograr una aplicación asintóticamente eficiente de RESTART, una de cuyas hipótesis básicas es trabajar con cadenas de Markov con cantidad numerable de estados. Este tipo de restricciones, comunes en el material bibliográfico debido a las buenas propiedades de la distribución exponencial, son demasiado fuertes para los objetivos generales de esta tesis.

Los siguientes capítulos presentan técnicas automáticas para implementar la división por importancia, que intentan cubrir el amplio espectro de los procesos estocásticos (con la única restricción de que sean homogéneos). Esta automatización debe introducir tan pocas restricciones como sea posible. No olvidemos que no hay objetivos de optimización, sino que deseamos lograr una aplicación efectiva de la técnica de división, de cualquier forma que le gane en eficiencia a la estrategia de Monte Carlo estándar de la Subsección 2.3.2.

El instrumento teórico usual para medir y comparar la eficiencia de diferentes mecanismos de estimación, es la varianza de los estimadores comparados. Esta varianza tiene un impacto directo en la precisión de los intervalos de confianza y por ende en los tiempos de convergencia. En consecuencia, para decidir si RESTART es un buen candidato para usar en las experimentaciones de división multinivel, es deseable contar con alguna caracterización matemática de su varianza.

Los autores Manuel y José Villén-Altamirano have developed several expressions for such variance, some of them reported in [VAVA02]. They all are purely theoretical in nature, lo que significa que no se pueden computar efectivamente

en aplicaciones de la vida real, al menos no para modelos estocásticos generales. A pesar de ello muestran que for optimal, quasi-optimal, and even merely good implementations of the method, there is indeed an expected gain por usar RESTART por sobre la simulación de Monte Carlo estándar [VAVA11].

Notablemente, la selección óptima y quasi-óptima of parameters proposed in [VAVA11] are impractical due to the assumptions these make on the systems [VAVA11, Sec. 5]. Notwithstanding this inconvenience, an *effective application* of RESTART can anyhow be performed. Specific guidelines for such a "good implementation" of the method are give in [VAVA11], fitting the objectives of this thesis wonderfully. Específicamente, reportan cuatro factores que afectan al desempeño:

f_O *Ineficiencia por la sobrecarga de cómputos.*

Esto tiene que ver con la implementación concreta de los computational methods, which depends also on the model. It is affected by the evaluation of the importance function every time a state is visited (e.g. at each simulation step), the comparison of the resulting importance to the threshold values, and the context switch for saving and restoring states during replication and truncation. So for instance the number of variables of the system influences RESTART negatively. There is no universal solution for this source of inefficiency: in general smaller models should be favoured, and good design patterns and programming techniques are paramount to minimize the overhead derived from state manipulation and from the evaluación de la función de importancia.

f_K *Ineficiencia por los valores de división.*[†]

Cuando se discutió la estrategia de división fija for offsprings generation in la Subsección 2.5.1, it was said that a careless selection of the splitting factors $\{K_i\}_{i=1}^n$ can lead to overhead or starvation. Optimal and quasi-optimal values for all K_i require a dense state space S. For the general case there is a procedure based on pilot runs starting from the initial state s_0, which increasingly chooses the K_i trying to fulfill the *balanced growth* equation [Gar00, eq. (2.25)]. This requires a previous fixing of the thresholds and operates with a balancing procedure: when the i-th threshold is granted a splitting value bigger than desired (e.g. due to the discreteness of the state space), it will be compensated with a smaller K_{i+1}, and vice versa. The performance incidence of un error en este mecanismo de selección debería ser moderado [VAVA11].

f_L *Ineficiencia por los umbrales escogidos.*[‡]

La selección del número y la ubicación de los umbrales is similar in impact to the choice of splitting values, since these two parameters are intimately related in a fixed splitting strategy. Elegir umbrales demasiado próximos

[†] En [VAVA11] esto aparece como el factor f_R; se lo renombró de acuerdo a la notación local.

[‡] En [VAVA11] esto aparece como el factor f_T; se lo renombró de acuerdo a la notación local.

entre sí puede ser contrarrestado reduciendo la división, pero colocarlos demasiado alejados entre sí puede incur in unavoidable starvation. Notice the extreme case of a single threshold at the boundary of A, which turns RESTART into standard Monte Carlo simulation. The authors recommend using pilot runs and statistical analysis, trying to fix the values $\{L_i\}_{i=1}^{n}$ so that the probability of level up is near $1/2$. If the nature of the model makes this impossible, e.g. the probability of a single transition $s \to s'$ is already lower than 0.5, just set the thresholds as close to each other as possible. The later choice of splitting values will try to counter estas situaciones utilizando la heurística de crecimiento balanceado.

f_V *Ineficiencia por la varianza de los eventos B_i.*

Esto habla de la varianza en la *importancia real* de un evento B_i, cuando es disparado por different entrance states into region E_i. That is, the unknown theoretical probability of observing a rare event after visiting the state that caused the B_i event. If this importance varies much with the entrance states into the regions, the performance can deteriorate greatly, since the splitting at the i-th threshold will not cause an homogeneous oversampling. Even worse, some trajectories may be favoured over others, which could yield an incorrect estimation. This in spite of the unbiasedness of the method, since the computation of the IC could prematurely converge to a wrong estimate, because no trajectories have yet been sampled from an unlikely but representative set. RESTART is quite sensitive to this factor, which is affected by the concrete modelling of the system and the choice of the importance function $f\colon S \to \mathbb{R}$. Guidelines are provided in [VAVA11] to reduce f_V, pero parecen difíciles de generalizar fuera del marco de ese trabajo.

De los cuatro factores mencionados es el último, f_V, is the most difficult to counter systematically. That is because it depends on the nature of the particular problem under study, which has mostly led to ad hoc solutions, very well suited for the situations where they are proposed, but inapplicable in a different scenario. As stated in the early sections of this chapter, formal system modelling supplies several magnificent tools to structure the description of a system. This provides a partial solution to the f_V problem, as long as the user can distill a representative e idealmente sucinta descripción del modelo.

Queda por tratar la elección de la función de importancia. Esto también ha sido resuelto, en general, para cada caso concreto: la mayoría de los artículos sobre división por importancia proponen funciones ad hoc para sus casos de estudio y evalúan su desempeño. El cómputo genérico de funciones es un campo novedoso, cubierto tan sólo por los estudios mencionadas en la Sección 1.2.

Evidentemente, el impacto en la eficiencia de una buena elección de función de importancia trascienden a RESTART. El factor f_V descripto con anterioridad es sólo un ejemplo concreto de la criticidad de esta elección. Dado que todas las técnicas de división guían los caminos de simulación para que visiten a los estados con la mayor importancia (calculada), *una mala elección de función de importancia implica una mala técnica de división*, independientemente de sus

características particulares. Esto será teórica y empíricamente ilustrado en los siguientes capítulos.

Como ya se mencionó, las motivaciones generales de esta tesis involucran automatizaciones en el análisis por simulación de modelos estocásticos generales en un régimen de evento raro. Dado el alto potencial de innovación y el impacto directo que podría tener en la industria, *la derivación automática de la función de importancia* para la aplicación de técnicas de división en RES es un área prometedora de investigación, y el principal objetivo específico de esta tesis.

División multinivel: automat. monolítica 3

Este capítulo presenta dos contribuciones principales de esta tesis: cómo derivar una función de importancia a raíz de un modelo global del sistema; y cómo usar dicha función en un método automatizado de división multinivel. La relevancia y sensibilidad de la técnica de división a la elección de función de importancia es ilustrada mediante algunos ejemplos prácticos. Luego, un marco formal para el modelado de sistemas es determinado, sobre el cual se define el algoritmo de derivación. Una implementación automática del proceso completo de análisis para RES es introducida a continuación. Finalmente, la eficiencia de esta técnica, como así también su principal limitación, son ejemplificadas mediante casos de estudio.

3.1. La importancia de la función de importancia

El mismo marco formal empleado en la Sección 2.5 se utilizará aquí: habrá un proceso estocástico homogéneo $X = \{X_t \mid t \geqslant 0\}$, con tiempo discreto o continuo t, que describe al modelo bajo estudio. Para las siguientes definiciones X debe satisfacer la propiedad de Markov. Esto puede asumirse s.p.d.g. dado que la historia del sistema puede incluirse en el estado X_t del proceso (ver [Gar00, Sec. 2.2]).

El espacio de estados se denota S y el conjunto de estados raros es $A \subset S$, el cual es muestreado por el proceso con probabilidad positiva muy cercana a cero $0 < \gamma \ll 1$. Un *evento* será un subconjunto medible de S, y A también será llamado el *evento raro*. El objetivo general en la simulación de eventos raros es estimar el valor probabilístico γ de observar al evento raro, realizando un análisis estadístico sobre un conjunto de caminos simulados a partir del estado inicial del sistema $s_0 \in S \setminus A$. Estudios transitorios y estacionarios son de interés por igual, como fuesen introducidos en las Definiciones 8 y 9.

La técnica de división se basa en una descomposición de S que agrupe a los estados con similar probabilidad de hacer que una simulación desemboque en el evento raro. La táctica computacional es la simulación de caminos de ejecución, i.e. estamos hablando de la probabilidad de que uno de estos caminos alcance al evento raro luego de haber visitado un estado. Además, como X es markoviano, basta con considerar los caminos que *comienzan* en el estado cuya visita estamos considerando. Desde este punto de vista A puede describirse como una meta que queremos alcanzar, y el *grado de importancia* de un estado $s \in S$ debería reflejar la probabilidad de alcanzar la meta cuando los caminos de ejecución comienzan

en s. La siguiente definición formaliza una cuantificación de esta propiedad.

Definición 11. Sea X un proceso de Markov homogéneo, con espacio de estados S y evento raro $A \subseteq S$. La *importancia real* de un estado $s \in S$ es la probabilidad de que un camino simulado que comienza en s visite eventualmente algún estado de A, viz. que una variable aleatoria de X observe al evento A, condicionada en el estado inicial s.

Nótese que la importancia real del estado inicial s_0 is precisely γ. Evidently these values are unknown in general but for the states in A, and one of the goals of RES is to estimate them. In importance splitting this is done for s_0 employing some predefined scaled approximation of such values, which must cover the full state space. Such (arbitrary) approximation is known in the literature under the name of *importance function* (also score function [JLS13] and rate function [GHSZ98]), and it can be any projection which maps the states en algún conjunto o cuerpo con un orden total:

Definición 12. Una *función de importancia* es una función

$$f \colon S \to \mathbb{R}.$$

Para cada $s \in S$ el valor $f(s) \in \mathbb{R}$ es la *importancia calculada* o simplemente *la importancia* de s.

La calidad de esta aproximación está íntimamente relacionada to the performance of the technique. The more it resembles the (scaled) true importance of the states, the faster the method should converge (see Sección 2.7). As a matter of fact once the system model has been formalized, deciding on an importance function is usually the first step for the application of multilevel splitting. Most techniques even have procedures to select and tune other execution parameters based on a user-provided definition of the function. Take for instance [CG07] which only needs the choice of ratio k/n besides this function, or the guidelines given in [VAVA11] to derive the thresholds and the splitting factors para una aplicación práctica de RESTART.

Por desgracia la Definición 12 le deja todo el trabajo al implementador. La heurística debería asignar los valores de importancia coherentemente con la Definición 11, pero esto no es cabalmente posible desde el punto de vista práctico, pues si lo fuese entonces ya conoceríamos el valor de γ. Históricamente, la forma tradicional de lidiar con esta elección es definir una función ad hoc que se amolde al sistema específico de interés. Así y todo, además de ser inextensible, esta estrategia necesita de una decisión humana calificada que podría empero estar errada, con severas consecuencias para la eficiencia. Para ilustrar esta sensibilidad de la técnica de división por importancia a la elección de la función de importancia, recurrimos al siguiente caso práctico.

Ejemplo 1: Sistema de colas con rupturas.

Considere el sistema markoviano de colas de la Figura 3.1 donde hay un único buffer, `buf`, al cual varias fuentes le envían paquetes de forma concurrente. Las fuentes son de tipo $i \in \{1, 2\}$ y se encuentran operativas/rotas de acuerdo a tiempos descriptos por exponenciales caracterizados por los parámetros α_i y β_i respectivamente. Cuando están operativas, las fuentes de tipo i envían paquetes a tasa λ_i que son encolados en `buf`. Estos paquetes son procesados por el servidor a tasa μ siempre y cuando se halle operativo. El servidor se rompe periódicamente con tasa ξ y se repara con tasa δ.

Este sistema fue estudiado originalmente en [KN99] para un estado inicial donde hay un único paquete en `buf`, el servidor está roto, y todas las fuentes están rotas excepto por una de tipo 2. Usando muestreo por importancia los autores estimaron la probabilidad transitoria de alcanzar una capacidad máxima paramétrica $K \in \mathbb{N}$ en el buffer, antes de que `buf` se vacíe.

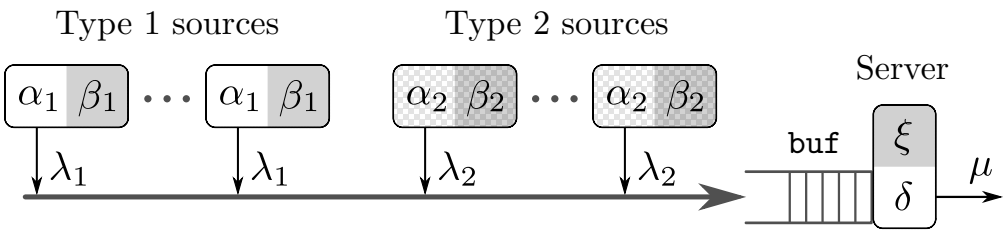


Figura 3.1: Sistema de colas con rupturas

La presencia de `buf` y de varios otros componentes, cada uno con su propio estado, makes this also an attractive case for splitting. The state of the corresponding Markov process $X = \{X_t \mid t \geqslant 0\}$ should include the server status: an inherently boolean random variable which can be encoded as an integer taking the value 1 when the server is operational and 0 otherwise. Process X should also include information regarding the status of each source, which can be equally encoded as integers, and of the number of packets in the buffer, of clearly integral nature. Given thus X, applying multilevel splitting in any of its flavours requires one to decide how important each state X_t is. Equivalently: ¿como deberíamos definir la función de importancia f?

Sea $buf \in \mathbb{N}$ el número de paquetes encolados en `buf`. Como el evento raro tiene que ver sólo con una sobrecarga del buffer, una propuesta ingenua sería usar la función $f(X_t) = buf$. Let $f_1 \colon S \to \mathbb{N}$ denote this importance function, then f_1 is oblivious del número de fuentes que le envían activamente paquetes al buffer. Esto no suena razonable dado que si no hay arribo de paquetes entonces no hay posibilidad de sobrecarga. Sea pues $f_2(X_t) \doteq buf + \sum s_1^k + \sum s_2^k$ be the importance function which adds to buf the number of active sources, with random variable s_i^k corresponding to the activation status de la k-ésima fuente de tipo i.

Los tiempos de convergencia de RESTART usando estas dos funciones (y otras más) fueron comparados en [BDH15], for 95 % Confidence Intervals with 5 % Relative Error, testing buffer capacities $K \in \{20, 40, 80, 160\}$ with the same system parameters as in [KN99]. Surprisingly f_1 behaved at least as well as f_2, outperforming it in several occasions. In some settings the difference was remarkable, e.g. for $K = 80$ and with a global splitting value of 5, f_1 convergió más de 8 veces más rápidamente que f_2.

Este comportamiento inesperado podría deberse al uso de RESTART como técnica de división, o a las particularidades de su implementación en la herramienta usada en las pruebas. También podemos esbozar una explicación con bases teóricas, argumentando que la tasa total de encolado en el buffer y la tasa de atención del servidor, son demasiado rápidas con respecto a los tiempos de ruptura y reparación de las fuentes. Por ende incluir el estado de estos componentes en el cálculo de la importancia generaría niveles irrelevantes en el espacio de estados, que causarían sobrecargas infructuosas de cómputo debido a los mecanismos de copiado y truncamiento. □

Elecciones óptimas y asintóticamente eficientes para la función de importancia requieren, evidentemente, de un conocimiento cabal y de algunas suposiciones restrictivas del tipo de modelos tratados. Sin embargo, a pesar de la naturaleza markoviana y de la simpleza general del sistema de colas que presentamos en el Ejemplo 1, elegir una función de importancia meramente *buena* resulta más difícil de lo que podríamos haber esperado. Por ejemplo no queda claro si tener en cuenta el estado de las fuentes es una estrategia del todo errada. Podría ser que su inclusión en el cómputo de la importancia es útil, pero deber ser disminuido por algún factor. Pero en ese caso ¿cuál factor dará el mejor resultado?

Para colmo de males, cualquier cambio en la definición del evento raro puede degradar el funcionamiento de funciones que otrora funcionaban de maravilla. Supongamos que el evento raro del Ejemplo 1 es definido como una sobrecarga del buffer cuando todas las fuentes de tipo 1 se encuentran activas. En ese caso la medida de importancia definitivamente tendrá que considerar el estado de las fuentes, y por ende f_2 debería comportarse mejor que f_1 (¿verdad?). Es decir que una definición particular de f debe ser ideada no sólo para cada sistema, sino también para cada análisis que queramos hacerle al sistema.

Empeorando la cuestión aún más, no sólo la estructura general del sistema sino también sus rasgos dinámicos específicos influyen en la elección. Supongamos que el evento raro no deja dudas respecto de cuales componentes del sistema deben ser tenidos en cuenta. Incluso en tales escenarios puede haber complejidades ocultas (e.g. valores específicos de ciertos parámetros), que vuelvan ineficiente a una función de otro modo muy natural. Veamos esto con un nevo caso de estudio.

Ejemplo 2: Cola tándem.

Considérese una red Jackson en tándem que consiste en dos colas conectadas como en la Figura 3.2. Los clientes llegan a la primera cola siguiendo un proceso de Poisson con parámetro λ. Luego de ser atendidos por el primer servidor a tasa μ_1 entran en la segunda cola, donde son atendidos por el segundo servidor a tasa μ_2. Finalmente tras ser atendidos por segunda vez, los clientes abandonan el sistema.

El tiempo entre eventos está exponencialmente distribuido y es independiente entre las distintas estaciones. Esto significa que el tiempo entre arribos es independiente de los tiempos de atención, y que el tiempo transcurrido entre dos servicios en el primer (resp. segundo) servidor es independiente de los tiempos de arribo y de los tiempos de servicio en el segundo (resp. primer) servidor. Luego el proceso estocástico $\{(q_1, q_2)_t \mid t \geqslant 0\}$, donde $X_t \doteq (q_1, q_2)_t$ es el número de clientes en la primer y segunda cola al tiempo t, es markoviano y homogéneo. Este modelo ha recibido considerable atención en la bibliografía de RES, ver por ejemplo [GHSZ98, Gar00, GVOK02, VA07b, LDT07, VAVA11, BDH15].

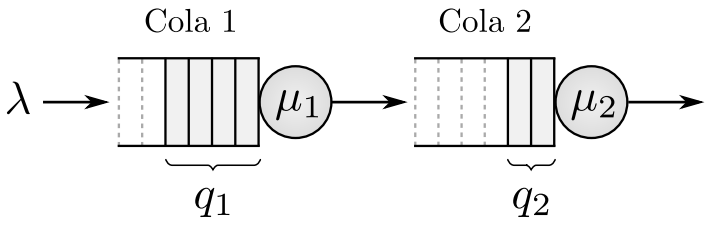


Figura 3.2: Cola tándem

Para alguna capacidad máxima $L \in \mathbb{N}$ estamos interesados en estudiar el comportamiento transitorio de las colas para el evento raro de una saturación en la segunda cola, viz. $q_2 \geqslant L$. Supongamos que el sistema comienza a correr sin clientes en la primer cola y con uno solo en la segunda, y supongamos también que nos interesa medir la probabilidad de saturación en la segunda cola antes de que el sistema se vacíe (esto es denominado un *ciclo regenerativo* en [GHSZ99, VAVA06]). ¿Cómo deberíamos definir a f?

Como el evento raro sólo involucra a la segunda cola, una alternativa ingenua sería $f((q_1, q_2)_t) = q_2$. Sería extraño empero that the value of q_1 played no role at all in the true importance of X_t: even with $L - 1$ customers in the second queue, no rare event can be immediately observed if $q_1 = 0$. Name $f_1(X_t) \doteq q_2$; most modern literature discourages choosing f_1 as importance function, en vista de consideraciones similares.

En contraste al estado $(0, L - 1)$ as presented above, let the current system state be $(L, {}^3\!/_4\, L)$. Then, providing the first server is not much slower than the second one, X_t could quickly lead to the desired overflow, a pesar del hecho

de que q_2 dista de estar llena. Esto nos lleva a proponer a $f_2(X_t) \doteq q_1 + q_2$ como importancia para el estado. However, if the first queue is the bottleneck and the rarity of the overflow is due to very fast service times at the second queue, the value of q_1 has little influence on the true importance of X_t, y f_2 no debería comportarse tan bien.

Generalizando en esta dirección surge the family of functions $f_{\alpha_1,\alpha_2}(X_t) \doteq \alpha_1\, q_1 + \alpha_2\, q_2$, where the selection of the weights $\alpha_i \in [0,1]_{\mathbb{R}}$ is behind the resulting performance of the function in each particular framework. Then $f_1 = f_{\alpha_1=0,\alpha_2=1}$ and $f_2 = f_{\alpha_1=1,\alpha_2=1}$. As it happens, the optimal choice of weights depends on the comparative order between the loads of the queues, ρ_1 and ρ_2 [VAVA06, LDT07, LLGLT09]. These loads are inversely related to the service rates following the formula $\rho_i \doteq \lambda/\mu_i$. But the essence of the question still remains: ¿cómo deberíamos esoger a α_1 y α_2 para maximizar la eficiencia de la función f_{α_1,α_2}?

Si la configuración es $\rho_1 < \rho_2$, [VA07b, Sec. 4.1] suggests using $\alpha_1 = \alpha_2 = 1$, i.e. f_2. The author derives this formula when looking for the linear combination between q_1 and q_2 which would minimize some expression of the variance of the estimator. An adaptation of this approach is also followed in [LDT07, Sec. 5.2]. Both works report good results: la varianza medida para f_1 fue bastante mayor que la de f_2.

En contraste, en un escenario donde $\rho_1 > \rho_2$ (viz. la primer cola es el cuello de botella), [VAVA06, VA07b] sugiere que se utilicen $\alpha_1 = 0{,}6$ y $\alpha_2 = 1$ como pesos óptimos para minimizar la varianza. Esto no se corresponde del todo con los tiempos reportados en la Tabla 1 de [BDH15], donde f_1 fue la función ad hoc más eficiente en un marco de experimentación consistente con el de los otros dos trabajos. Dicha tabla muestra que ambas f_2 y $f_{\alpha_1=1,\alpha_2=2}$ fueron notablemente más lentas para converger que f_1. Para descartar comportamiento extravagante se realizaron más pruebas con la función $f_3 \doteq f_{\alpha_1=3,\alpha_2=5}$, para la cual la proporción $\alpha_1/\alpha_2 = 0{,}6$ es la misma que para los pesos óptimos mencionados. La eficiencia medida para f_3 es comparable a la de f_2 en [BDH15], i.e. le tomó considerablemente más tiempo que a f_1 la construcción del IC deseado. □

Sacamos dos conclusiones principales del Ejemplo 2. Primero, que la selección de pesos $\alpha_i \in [0,1]_{\mathbb{R}}$ óptimos (o razonables) para la función de importancia f_{α_1,α_2} no es en absoluto trivial. Depende al menos del orden comparativo de las cargas de las colas ρ_i y también puede estar influido por otros parámetros, e.g. la finitud de las colas, o la relación entre el tamaño de las colas y los valores concretos de carga (más que de su orden comparativo). Esto a pesar de la clara sencillez del sistema, que sólo cuenta con dos componentes cuya dinámica de interacción es bien clara. Ocurre que variaciones en los parámetros del sistema pueden modificar el comportamiento general, deteriorando el desempeño de funciones de importancia que en otras circunstancias podrían ser muy buenas.

La segunda conclusión derivada del Ejemplo 2 es que a pesar de su precisión,

el análisis teórico puede ser engañoso si algún factor es pasado por alto. Lograr un análisis comprensivo completo puede ser difícil: para derivar los pesos óptimos para el caso $\rho_1 > \rho_2$ en [VA07b], el autor asume que la primer cola tiene capacidad ilimitada y que los servidores se comportan de cierta manera específica[†]. Además de la diferencia en las tasas exactas λ, μ_1, μ_2, y además del escaleo de α_1, α_2 en f_3 para imitar la proporción de los pesos óptimos $^{\alpha_1}/_{\alpha_2} = 0{,}6$, estas suposiciones podrían ser la razón por la cual hay discrepancias entre [VA07b] y los resultados empíricos de [BDH15]. Remarcamos a su vez que el estudio teórico busca reducir la varianza del estimador, lo que debería ser proporcional a los tiempos totales de convergencia, pero podría ser subóptimo para la implementación concreta de cierta técnica.

Queda pues claro que encontrar pesos α_i que optimicen el desempeño de la función de importancia f_{α_1,α_2} en una configuración particular de la cola tándem es una tarea no trivial. La elección dependerá de los valores específicos del sistema, cualquier suposición empírica del comportamiento general (e.g. la primer cola no puede saturarse), y la definición del evento raro. Respecto de esto último, no olvidemos que el Ejemplo 2 trata con la probabilidad de saturación en la segunda cola *dentro de un ciclo regenerativo*. ¿Qué función de importancia habría que escoger para estimar la probabilidad del evento $q_1 + q_2 \geqslant L$? ¿Y si quisiéramos estudiar el comportamiento estacionario del proceso?

En suma, cada vez que el sistema o el ángulo de estudio cambian, una nueva expresión para la función de importancia deber ser ideada a raíz de nuevos análisis. Como se ilustró en los ejemplos previos, esta tarea no sólo es ardua sino también de alcance y extensibilidad limitadas. Es por ello que, para el uso efectivo de la técnica de división en situaciones de la vida real, es crucial contar con algún algoritmo que automatice la derivación de la función de importancia, a partir de las descripciones concretas del modelo del sistema y de la propiedad consultada.

3.2. Derivando la función de importancia

La destilación de una función de importancia puede ser abordada desde varios ángulos, dependiendo de las necesidades e intenciones del estudio. En esta sección explicamos nuestra estrategia específica, estableciendo para ello la bases formales necesarias. La sección concluye con el pseudocódigo de un algoritmo para derivar la función de importancia a partir del modelo y de la especificación del evento raro.

3.2.1. Objetivo

Hay dos caminos clásicos a seguir a la hora de dar instrucciones en la selección de una función de importancia adecuada. Desde una perspectiva predominante-

[†] Tomado de [VA07b, p. 153]: "... we will make the following assumption: In a two-queue tandem Jackson network with loads $\rho_1 > \rho_2$, if the initial system state is $(q_1, 1)$ and ..., q_1 customers will be in the second queue when the first one becomes empty."

mente práctica, es posible decidirse por alguna categoría específica de sistemas. Guías rigurosas pero no necesariamente formales son provistas, para construir funciones de varianza reducida para alguna técnica de división multinivel en particular. Esta estrategia es muy popular a la hora de estudiar procesos que surgen de problemas de la vida real, e.g. redes de colas Jackson, dado que provee soluciones de flexibilidad limitada pero que son fáciles de implementar y de usar en el entorno práctico que fueron diseñadas para atacar. Ver e.g. [CAB05, VAVA13, VA14].

En su lugar, desde una perspectiva más abstracta the intention could be to build a function with appealing theoretic properties. Asymptotic efficiency, bounded normal approximation, and optimality (minimizing an expression of the variance of a given estimator) are typical goals. This is usually approached in an analytic fashion, formalizing rather strong restrictions on the nature of the systems covered, for which the presented strategy shows the desired properties. The main advantage of this approach is the quality of the resulting function, which will perform well, or as good as possible, regardless of la rareza del evento. Ver e.g. [GHSZ99, DD09, GHML11].

En esta tesis el objectivo es proveer un algoritmo automátic que, for user provided formalisations of a time-homogeneous stochastic process X and some rare event to study, will yield an importance function on the full state space of X. To this aim the modelling formalisms from la Sección 2.1 are used for specifying the system models. In turn the rare event will be described as a property query que concuerde con las de la Sección 2.2.

La *función de importancia automática* resultante is not required to be optimal or even asymptotically efficiency. The goal is to allow an effective application of multilevel splitting which outperforms the standard Monte Carlo approach from Sección 2.3. Moreover, the automatic function is expected to perform at least as well as any simple and general ad hoc choice. That is, discarding solutions formally tailored for the particularities of the system under study, the function derived by the algorithm proposed should be as efficient como cualquier alternativa que el usuario pudiese idear.

La última meta no puede ser descripta formalmente, ya que el usuario siempre podría *simplemente idear* una solución óptima para el problema considerado. Al respecto, seguimos la estrategia de verificar extensivamente cada caso analizado, tratando de formar una noción del grado hasta el cual este objetivo fue satisfecho. Hemos desarrollado una herramienta que implementa el algoritmo de derivación automática de la función de importancia, y la técnica de división RESTART para realizar simulaciones. También es posible pasarle a la herramienta una función de importancia definida por el usuario. Esto nos permitió comparar el desempeño de varias funciones alternativas en términos de igualdad. En ese entorno, varios modelos y definiciones del evento raro han sido estudiados, comparando la eficiencia de la función automáticamente derivada contra varias alternativas ad hoc.

3.2.2. Marco formal

El algoritmo de derivación que presentaremos realiza una búsqueda a lo ancho (*breadth-first search*) en el grafo de adyacencia inherente al espacio de estados del

modelo. Por consiguiente contar con una cantidad finita de estados bastará para asegurar su terminación. Además trabajaremos con un único estado inicial para simplificar ciertas suposiciones.

En este capítulo nos enfocaremos en las cadenas de Markov de tiempo discreto y continuo de las Definiciones 2 y 3, a pesar de que la propiedad de pérdida de memoria de la distribución exponencial es transparente para nuestros algoritmos de derivación. Esto fue motivado por la herramienta de software desarrollada para estudiar nuestras propuestas, y también porque los formalismos DTMC y CTMC ofrecen una base conocida que facilitará muchas explicaciones.

Sólo hay dos propiedades relevantes que necesitan ser distinguidas durante la ejecución de una simulación: si el estado actual es *raro*, i.e. si es parte del evento raro, y si indica truncamiento, i.e. si es un estado de *parada*. El conjunto de proposiciones atómicas será pues en general $AP = \{\text{RARO}, \text{PARAR}\}$. El evento raro $A \subset S$ estará compuesto por los estados $s \in S$ para los cuales $Lab(s) = \{\text{RARO}\}$, y en los análisis transitorios el conjunto de parada $B \subset S$ estará identificado por 'PARAR', i.e. los caminos de simulación serán truncados cuando alcancen un estado con dicha identificación. Como $B \cap A = \varnothing$ entonces $\forall s \in S \,.\, |Lab(s)| \leqslant 1$.

Las Definiciones 2 y 3 dan una pista sobre cómo representar en un tipo abstracto de dato la estructura de un modelo DTMC o CTMC. Ambos son bastante similares entre sí: difieren sólo en las restricciones particulares que los elementos de $\mathbf{P}$ y de $\mathbf{R}$ deben satisfacer respectivamente. En ambos casos las transiciones del sistema están descriptas por matrices (usualmente ralas) de dimensión S^2, a partir de las cuales se puede extraer el grafo de adyacencia. Esto es explotado por el algoritmo que introducimos a continuación.

3.2.3. Algoritmo de derivación

En la Sección 2.7 se explicó que la elección de función de importancia tiene serias implicaciones en la eficiencia de la división multinivel, cualquiera sea el método específico seleccionado. Los Ejemplos 1 y 2 proveen evidencia de la sensibilidad de la técnica de división ante tal elección. En general queda claro que las simulaciones que siguen alguna estrategia de división son guiadas por esta función, que selecciona las direcciones en que el esfuerzo computacional debe ser intensificado.

En la Sección 3.1 se dijo que las funciones de importancia intentan aproximar la verdadera importancia de los estados de acuerdo con la Definición 11. Dicha definición fue expresada intencionalmente en términos de trayectorias a través de los estados: recuérdese que para un camino simulado a través del modelo, la verdadera importancia del estado $s \in S$ puede describirse como la probabilidad de observar al evento raro (i.e. visitar un *estado raro* del conjunto $A \subsetneq S$) luego de visitar a s. Dicha probabilidad tiende a decrecer con la distancia, medida en número de transiciones, entre s y el evento raro. Esto claro en el caso de los modelos DTMC donde las transiciones tienen pesos probabilísticos. Mientras más pasos de simulación sean necesarios, menor será el valor producto que representa la probabilidad conjunta de tomar todas las transiciones correctas que nos llevan desde s hasta A de la forma más directa posible.

Algo muy similar ocurre con los modelos CTMC. Iniciando una trayectoria desde el estado s, en cada paso se genera una condición de carrera que puede desviar al camino simulado de la vía más directa que la llevaría hasta A[†]. Por ende mientras más corta sea la ruta, mayor chance habrá de alcanzar el evento raro sin incurrir en desviaciones.

Este análisis sugiere que para ambos formalismos, caminos más largos hasta A implican menores probabilidades de observar el evento raro. Por consiguiente en el caso promedio de las simulaciones que comienzan desde $s \in S$, la distancia hasta A (medida como el número de transiciones) y la importancia de s deberían ser magnitudes inversamente proporcionales. Si pudiéramos rastrear o al menos conjeturar las trayectorias que guían a las simulaciones desde s hasta A, alguna noción de la distancia entre ellos podría ser determinada, y usada para elegir una importancia apropiada para s.

Claramente el peso probabilístico (o la tasa) de las transiciones afecta la importancia real de s. Pospondremos empero consideraciones que incluyan estas magnitudes; ellas serán tratadas en estadíos más avanzados de la estrategia que proponemos en este capítulo. La derivación de la función de importancia será enteramente determinada por el grafo de adyacencia inherente a la matriz de transiciones del modelo.

La idea central es bastante simple: comenzando en simultáneo de todos los estados que componen a A, realizar un análisis concurrente de alcanzabilidad revertida en el sistema de transiciones del modelo, i.e. una corrida de BFS que atraviese el grafo de transiciones usando las aristas con sus direcciones invertidas. Cada iteración del algoritmo visitará una nueva capa de estados, que estarán un paso más alejados de A que los estados de la iteración previa. Las capas sucesivas de estados visitadas en cada iteración del algoritmo son decoradas con valores de importancia decreciente. El pseudocódigo se presenta en el Algoritmo 1, donde $\mathcal{M}$ es el modelo del sistema y s_0 es su estado inicial.

Así la distancia del camino más corto que lleva desde cada estado hacia A es computado con una búsqueda a lo ancho (*breadth-first search*) de complejidad $\mathcal{O}(b\,n)$, donde n es el tamaño del espacio de estados y b es el grado de ramificación del grafo de adyacencia. Nótese que aunque $b \approx n$ en el peor escenario, i.e. si se cuenta con una matriz de transiciones densa, b es normalmente muchos órdenes de magnitud más chico que el número total de estados.

Para cada estado $s \in S$, su importancia $f(s)$ es pues calculada como el inverso de su distancia al estado raro más cercano, donde la distancia de s_0 a A es la mayor considerada. En ese sentido nótese que el ciclo exterior puede finalizar *antes* de haber visitado todos los estados, ni bien hallamos a s_0. Esto es así porque en el Algoritmo 1, el estado inicial del sistema tendrá la menor importancia, concretamente $f(s_0) \doteq 0$. La descripción de RESTART implica que así debe ocurrir, dado que no hay eventos D_0 que puedan truncar a la simulación original. En general no se le puede sacar provecho a los estados que tienen menor importancia que s_0; dividir/truncar para aumentar el muestreo en regiones tan alejadas del evento raro generará sobrecarga innecesaria de cómputo,

[†] Esto puede formularse con más rigor estudiando la cadena markoviana embebida en el CTMC.

Algoritmo 1 Derivación de la función de importancia de un modelo.

Entrada: módulo $\mathcal{M}$
Entrada: conjunto de estados raros $A \neq \varnothing$

$g(A) \leftarrow 0$
`queue.push`(A) {denota a los estados de A como visitados}
repeat
  $s \leftarrow$ `queue.pop`()
  for all $s' \in \mathcal{M}$.`predecessors`(s) **do**
    if s' not visited **then**
      $g(s') \leftarrow g(s) + 1$
      `queue.push`(s') {denota a s' como visitado}
    end if
  end for
until `queue.is_empty`() **or** s_0 visited
$g(s) \leftarrow g(s_0)$ for every non visited state s
$f(s) \leftarrow g(s_0) - g(s)$ for every state s

Salida: función de importancia $f \colon S \to \mathbb{N}$

sin recompensas que justifiquen el esfuerzo.

El Algoritmo 1 hace dos suposiciones de sus entradas:

1) asume un acceso de caja negra al grafo de adyacencia (invertido) de $\mathcal{M}$ por medio de la función $\mathcal{M}$.`predecessors`$\colon S \to 2^S$;
2) espera que se le provea el conjunto de estados raros $A \subsetneq S$ como entrada explícita.

La suposición 1 puede satisfacerse fácilmente, dado que la dinámica de cadenas de Markov de tiempo discreto y continuo suele ser almacenada en formato matricial, del cual el grafo de adyacencia puede obtenerse directamente. La suposición 2 es menos directa, dado que la entrada del usuario en ese sentido es una consulta de propiedad como en las Ecuaciones (7) y (8). Algún mecanismo debe ser ideado, para transformar la fórmula lógica que expresa al evento raro en el conjunto de estados que la satisfacen. Cubrimos este punto en la Subsección 3.3.2. Con estas consideraciones en mente podemos proveer una prueba de terminación.

Proposición 6 (Terminación del algoritmo de derivación de la función de importancia)**.** *Sea $\mathcal{M}$ un modelo DTMC o CTMC finito, y $A \neq \varnothing$ el conjunto de estados raros del espacio de estados S de $\mathcal{M}$, derivado de alguna consulta transitoria o estacionaria del usuario. Entonces, comenzando de las entradas $\mathcal{M}$ y A, el Algoritmo 1 termina luego de realizar un número finito de iteraciones.*

Demostración. Como $\mathcal{M}$ es un modelo DTMC o CTMC finito, el conjunto S es finito y también lo es el grafo de adyacencia derivado de la matriz de transiciones $\mathbf{P}$ o $\mathbf{R}$. Trabajar con un grafo finito nos garantiza que el ciclo **for**–**do** terminará todas las veces que sea iniciado, puesto que hay un número finito de antecesores

para cada estado. Denótese "visitar" la acción de meter un estado en la cola `queue`. El predicado condicional dentro del ciclo interno nos asegura que cada estado será visitado a lo sumo una vez, y el ciclo externo quita un elemento de `queue` con cada iteración. Por ende contar con un S finito basta para garantizar que la primera condición de la guarda del ciclo externo **repeat–until** eventualmente será satisfecha. Como dicha condición de guarda es una disyunción, esta argumentación asegura que el ciclo externo realizará un número finito de iteraciones. □

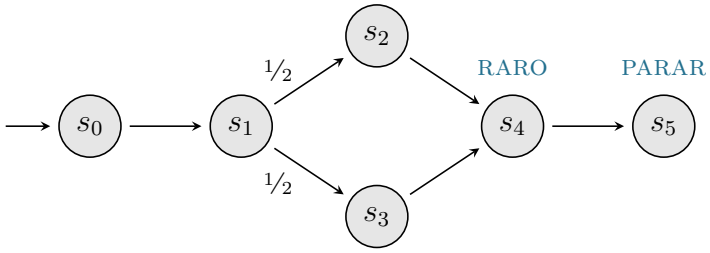
De hecho la Proposición 6 puede ser aplicado al modelo $\mathcal{M}$ de cualquier proceso estocástico finito, dado que la propiedad de pérdida de memoria de las cadenas de Markov no fue usada en la prueba. La única complicación posible está ligada a la forma en que se expresa el sistema de transiciones, pero igualmente el algoritmo terminará en un número finito de iteraciones, siempre y cuando se cuente con una función de caja negra $\mathcal{M}.\texttt{predecessors}\colon S \to 2^S$.

El Algoritmo 1 genera una función donde todo camino simulado que siga una trayectoria más corta desde el estado actual hasta el conjunto de estados raros, atravesará un secuencia monótonamente creciente de valores de importancia. Llamaremos a esto la *condición de monotonicidad* de la función de importancia. Pero hay que resaltar que la función resultante no es necesariamente *correcta*, en el sentido de que no siempre calculará la importancia real de los estados del modelo. Esto es evidente ya que los pesos o tasas de las transiciones son ignorados. Tomemos por ejemplo un DTMC donde los estados s y s' se encuentran respectivamente a dos y una transiciones de distancia de A, pero donde ambas transiciones que llevan desde s hasta A tienen probabilidad 1, mientras que la que conecta a s' con A tiene probabilidad ½. El Algoritmo 1 le dará mayor importancia a s', lo cual se contradice con la Definición 11 de importancia real.

Este problema es contrarrestado en la estrategia abarcadora que se introduce en la siguiente sección para automatizar la división por importancia. Contar con implementaciones completamente abarcadoras como ésta es conveniente desde un punto de vista práctico. Pero lo que es más importante, también son una forma de evitar errores derivados de una malinterpretación de las sutilezas de cada técnica particular. Una implementación errónea combinada con una elección pobre de función de importancia puede en ocasiones generar estimaciones incorrectas, como se muestra en el siguiente ejemplo.

Ejemplo 3: Estimación errónea de un RESTART incorrecto.

Considérese el DTMC con seis estados representado aquí, con la función de importancia $\{(s_0, 0)\,,\ (s_1, 1)\,,\ (s_2, 1)\,,\ (s_3, 0)\,,\ (s_4, 2)\,,\ (s_5, 0)\}$. El estado inicial es s_0 y el evento raro (si bien no es tan raro...) es $A = \{s_4\}$. Las regiones de importancia estarán dadas por el único umbral $L_1 = 1$, i.e. las zonas $Z_0 \doteq \{s_0, s_3, s_5\}$ y $Z_1 \doteq \{s_1, s_2, s_4\}$ forman la partición de los estados que será usada en la división multinivel.



Como s_5 es un estado de parada, la probabilidad transitoria de que un camino simulado desde s_0 alcance al evento raro antes de parar es trivialmente $\gamma = 1$, como lo sugeriría cualquier análisis por simulación de Monte Carlo estándar.

Recordemos que el estimador RESTART para análisis transitorio es $\hat{\gamma} = \frac{M}{K\,N_0}$, donde M es el número de caminos simulados que alcanzaron el evento raro, N_0 es el total de simulaciones RESTART iniciadas en s_0, y K es la división acumulada entre el estado inicial y el máximo valor de importancia.

Como hay un único umbral, L_1, K es justamente la división que se lleve a cabo en L_1. Es razonable asumir que $K > 1$, pues de lo contrario RESTART no diferiría de Monte Carlo estándar. Digamos por ejemplo que $K = 2$.

Nótese que las simulaciones que escojan el camino a través de s_3 truncarán todas las copias creadas en s_1, dado que que éstas se trasladarán desde la zona Z_1 hacia la zona Z_0 generando un evento D_1. El único camino de simulación que puede sobrevivir la transición $s_1 \to s_3$ es la copia original del nivel 0. En el siguiente paso de simulación, cuando esta copia tome la transición $s_3 \to s_4$, se introduce en el conjunto raro A al mismo tiempo que genera un evento B_1.

Hay dos implementaciones posibles de RESTART: atender primero el evento B_1, o considerar primero la entrada en A. En el segundo caso no se estará dividiendo al camino de simulación cuando éste se traslada desde Z_0 hacia Z_1, y estadísticamente tendríamos

$$\lim_{N_0 \to \infty} M = \frac{3}{4}\, N_0 \,,$$

porque la mitad de los caminos simulados tomarán la transición $s_1 \to s_3$, y con $K = 2$ la mitad de dichos caminos serán copias de nivel 1 que serán truncadas cuando visiten a s_3.

Como puede verse, aplicar RESTART de la forma propuesta resulta en la estimación $\hat{\gamma} \approx \frac{3/4\, N_0}{2\, N_0} = 3/8 \neq \gamma$. Nótese a su vez que la brecha entre $\hat{\gamma}$ y la verdadera probabilidad transitoria γ se exacerba cuando el valor de división en L_1 es aumentado. □

Hay dos problemas cuya conjunción llegó a RESTART a estimar incorrectamente un valor para $\mathsf{P}(\neg \text{PARAR}\ \mathsf{U}\ \text{RARO})$ en el Ejemplo 3. Primero, la función de importancia propuesta no satisface la condición de monotonicidad. Es decir, no todos los caminos que llevan a una simulación desde el estado inicial hacia el

evento raro visitan valores de importancia monótonamente crecientes, a pesar de ser los caminos más cortos. Segundo, la implementación de RESTART consideró primero la entrada en A, antes de atender la división generada por el evento B_1.

En una metodología ad hoc, incluso asumiendo una implementación sin fallas de la técnica de división escogida, todavía queda la problemática de la función de importancia. Usar una función sin la condición de monotonicidad no basta de por sí para producir la estimación incorrecta del Ejemplo 3. Así y todo, cualquier función que no satisfaga esta condición puede resultar ineficiente. Esto se pone en evidencia en el ejemplo anterior en la división-truncado-división que se genera en las simulaciones que sigan la trayectoria $s_1 \to s_3 \to s_4$. Teóricamente esto es cuantificado por RESTART en el factor f_V mencionado en la Sección 2.7.

En sistemas sencillos como el de arriba resulta trivial generar una propuesta ad hoc de función de importancia que satisfaga la condición de monotonicidad. Sin embargo, las complejidades de los modelos y las consultas de evento raro provenientes de casos de la vida real tienden a complicar las cuestiones. Por ejemplo hay sistemas donde una falla puede dispararse por distintas configuraciones de los componentes, sin que necesariamente éstos estén relacionados entre sí. En tales situaciones la técnica de división puede aprovecharse de las capas generadas en el espacio de estados de forma tal que el evento raro no se encontrará exclusivamente en las regiones de mayor importancia.

Técnicas automáticas como las del Algoritmo 1 nos garantizan que la función de importancia derivada tendrá las propiedades deseadas. Embebida en la estrategia abarcadora de la siguiente sección, esto evita problemas de estimación como los que se ilustraron en el Ejemplo 3.

3.3. Implementando técnicas automáticas de división

Contar con un algoritmo para derivar la función de importancia es un gran primer paso en el camino de las automatizaciones de técnicas de división multinivel, pero no basta de por sí. Con contadas excepciones estos métodos requieren a su vez de la elección del número de umbrales y su valor de importancia preciso. Además las tácticas que emplean división fija deben seleccionar el nivel de división a realizar en cada umbral, y las tácticas de esfuerzo fijo necesitan una elección análoga del esfuerzo a dedicar en cada nivel de importancia.

Esta sección presenta otra contribución de esta tesis: una estrategia automatizable para aplicar división por importancia al análisis de sistemas por RES. Esto incluye un marco para modelar sistemas y especificar las consultas del usuario, un algoritmo para seleccionar los umbrales, y la ejecución automática de simulaciones usando técnicas de división por importancia en respuesta a la consulta realizada.

3.3.1. Lenguaje de modelado

A pesar de su precisión matemática, las Definiciones 2 y 3 de cadenas de Markov son insuficientes para proveer un formalismo ameno para la descripción de procesos estocásticos y probabilísticos. Esto se debe a que una especificación

explícita de S no es práctica. Modelos realistas pueden fácilmente contar con miles de configuraciones diferentes que deberían ser representadas como el conjunto S de estados opacos [Har15].

Una solución tradicional consiste en añadir una capa de abstracción con *variables tipadas*. Para el alcance de esta tesis alcanza con considerar variables enteras, donde los booleanos pueden ser codificados como enteros de rango $\{0, 1\}$, sin considerar las variables que toman valores en un subconjunto denso de la recta. El conjunto de posibles valuaciones de las variables consideradas conformará el espacio de estados de la cadena markoviana. Para preservar la condición de finitud estas variables sólo podrán tomar una cantidad acotada de valores.

Definición 13 (Estados simbólicos)**.** Sea $\{v_i\}_{i=1}^m$ una secuencia de variables enteras acotadas, i.e. cada v_i puede tomar valores en algún conjunto finito no vacío $V_i \subsetneq \mathbb{Z}$. Un vector $(v_1, v_2, \ldots, v_m) \in \mathbb{Z}^m$ de valuaciones concretas de las variables será llamado un *estado simbólico*.

Definición 14. Dada una variable entera acotada v que toma valores en V, el *rango de* v es $\#v \doteq |V|$, i.e. el número de valores diferentes que v puede adoptar.

Definición 15 (Estados concretos)**.** Supóngase que el espacio de estados S de alguna cadena de Markov $\mathcal{M}$ se corresponde con el conjunto de valuaciones de las variables enteras acotadas $\{v_i\}_{i=1}^m$. Entonces cada estado $s \in S = \{s^j\}_{j=1}^M$ será referido como un *estado concreto* de $\mathcal{M}$, donde M es la cantidad de posibles valuaciones diferentes, viz. $M = \prod_{i=1}^m \#v_i$.

Nótese que un estado simbólico es un vector m-dimensional expresado en términos de variables, mientras que un estado concreto yace en el conjunto finito (adimensional) S. Las Definiciones 13 a 15 establecen una biyección implícita entre el *espacio de estados simbólicos* correspondiente a una secuencia de variables, y el *espacio de estados concreto* de la cadena de Markov $\mathcal{M}$ para la cual estas variables fueron definidas. Se dirá que $\mathcal{M}$ *está ligado a las variables* $\{v_i\}_{i=1}^m$ y viceversa.

En los Ejemplos 1 y 2 de la Sección 3.1 los eventos raro y de parada, viz. the states sets A and B, were defined in terms of some system component, namely an overflow in a queue. This naturally speaks of a particular valuation of the variable representing the number of customers in the queue. However, sets A and B must be declared in terms of the atomic propositions AP and the labelling function Lab defined on top of S. Claramente sería conveniente contar con una forma más práctica de expresarlos, en lo posible en términos de variables.

En general y a menos que se indique lo contario, we will assume that the rare and stopping sets will be declared by the user as symbolic states. The bijection established by the previous definitions ensures this defines unique A and B subsets of concrete states in S. For instance in the tandem queue example where the rare event was defined as $q_2 \geqslant L$, the concrete states labelled with 'RARE' will be those corresponding to all symbolic states where the variable q_2 tiene una valuación mayor o igual a L.

Queda por tratar el tema de las transiciones del modelo. Algún tipo abstracto de dato eficiente podría usarse para representar la matriz rala correspondiente

a las propabilidades de **P** o las tasas de **R** de las Definiciones 2 y 3. Elecciones comunes son CSR and CSC sparse representations, and MTBDD. Representation efficiency notwithstanding, it is impractical to request such input directly from the user. Even in very sparse cases the size of the matrix will most likely be demasiado grande para emplear una declaración explícita.

Muchos lenguajes abstractos han sido diseñados to specify the dynamics of a process. These typically allow to speak directly in terms of symbolic states, i.e. of certain variables valuations. *Edges* are then defined at the abstraction level of variables, each corresponding to una o más transiciones en el nivel más bajo de los estados concretos.

Definición 16 (Aristas)**.** Para la cadena de Markov $\mathcal{M}$ ligada a las variables $\{v_i\}_{i=1}^m$, sea *pre* una condición boolean en las variables. También para $k \leqslant m$ e índices i_ℓ que tomen valores disjuntos en $\{1, \ldots, m\}$, considérense las expresiones aritméticas $\{ex_{i_\ell}\}_{\ell=1}^k$ sobre estas variables. Supóngase que la expresión ex_{i_ℓ} devuelve valores válidos para la variable v_{i_ℓ}, y sea $pos \doteq \bigwedge_{\ell=1}^k (ex_{i_\ell})$. Entonces $pre \to pos$ es una *arista* en $\mathcal{M}$, donde *pre* es la *precondición* o *guarda* de la arista, y *pos* es la *postcondición* o *acción* de la arista.

Intuitivamente, la guarda de una arista tells when its actions can be applied. Notice several transitions at the level of S can be covered by a single edge, since a guard can speak of a range of valuations. Consideremos por ejemplo

$$(0 < q_1 < L \wedge \mathit{arribo}) \;\to\; (q_1 + 1)_{q_1}$$

que representa las transiciones cuyo originating state corresponds to the variables valuation $\mathit{arrival} \equiv \top$ and $q_1 \in \{1, \ldots, L-1\}$. For $L > 2$ that is strictly more than one transition at the level of S. The sub-index decorating the postcondition is used to signify that the value resulting from the expression '$q_1 + 1$', for the current value of q_1, será aplicado a la variable q_1.

La Definición 16 no puede ser usada para describir modelos DTMC ni CTMC porque carece de un componente esencial: the probabilistic weights of the transitions from **P** and the transition rates from **R**. This is covered in different ways by the modelling languages available in the literature. Here the PRISM language is adopted since: it has a clear and relatively simple syntax; it is a de facto standard in the field of probabilistic model checking; and the tool implementing the derivation algorithm was developed as a modular extension of the PRISM tool.

Una descripción extensiva de la sintaxis de PRISM se encuentra en la sección 'Manual' de su página web. Mostramos un ejemplo de juguete en el Código 3.1, que modela un interruptor de luz en un entorno discreto. Cuando está apagada la luz tiene probabilidad `p` de ser encendida. Inversamente, la luz se apaga con probabilidad `q` cuando está encendida. La semántica asociada a la sintaxis del lenguaje PRISM en términos de DTMC y CTMC se encuentra detallada en la tesis doctoral de Dave Parker, [Par02].

Código 3.1: Ejemplo de un DTMC en la sintaxis de PRISM

```
1  dtmc
2  const double p = 0.4;
3  const double q = 0.001;
4  module Luz
5      encendida: bool init false;
6      [] !encendida -> (  p): (encendida'=true)
7                       + (1-p): true;          // i.e. no hacer nada
8      []  encendida -> (  q): (encendida'=false)
9                       + (1-q): true;
10 endmodule
```

De aquí en más, referencias al *nivel concreto* de una cadena de Markov estarán hablando implícitamente de S, $\mathbf{P}$ o $\mathbf{R}$, y las transiciones entre estados concretos. A su vez, referencias al *nivel simbólico* harán alusión a las variables y las aristas de alguna descripción de alto nivel (digamos, usando el lenguaje de entrada de PRISM) de la cadena markoviana. Entonces por ejemplo el nivel simbólico del modelo `Luz` del Código 3.1 involucra e.g. la variable `encendida` y las aristas de las líneas 6 a 9, mientras que el nivel concreto se refiere al DTMC sobre el cual se le da semántica a este modelo de PRISM.

Algunas decisiones implementativas moldean la manera en que los modelos DTMC y CTMC son expresados en PRISM. Por ejemplo: cómo se interpreta el solapamiento de guardas en varias aristas; la forma de manejar la ejecución paralela cuando el modelo se compone de varios módulos; y cómo sincronizar la ejecución de dichos módulos. Todos los detalles relevantes para esta tesis son repasados fugazmente los siguientes ejemplos prácticos.

Ejemplo 4: Modelo DTMC de una cola.

Considérese un servidor que opera en un entorno de tiempo discreto. En cada *tick temporal* el sistema recibe un nuevo paquete con probabilidad `lambda`, encolándolo en el buffer `q1`. A su vez, cuando la cola no está vacía, en cada tick se procesará y desencolará un paquete con probabilidad `mu1`. Nótese que arribo y procesamiento podrían ocurrir en simultáneo durante el mismo tick, resultando en un estado sin modificaciones cuando el tick haya transcurrido. Abajo mostramos un modelo de este proceso en el lenguaje PRISM.

Código 3.2: modelo de cola discreta en PRISM

```
1  dtmc
2
3  const int c = 12;           // Capacidad de la cola
4  const double lambda = 0.1;  // Probabilidad de que entre un paquete
5  const double mu1 = 0.14;    // Probabilidad de procesar  un paquete
6
7  module ColaDiscreta
8
9      q1: [0..c] init 1;
10
11     [] (q1=0) -> (  lambda): (q1'=q1+1)
12                  + (1-lambda): true;
13     [] (0<q1 & q1<c) -> (  (lambda)*(  mu1)): true
```

```
14                        + (  (lambda)*(1-mu1)): (q1'=q1+1)
15                        + ((1-lambda)*(  mu1)): (q1'=q1-1)
16                        + ((1-lambda)*(1-mu1)): true;
17      [] (q1=c) -> (  (1-lambda)*mu1): (q1'=q1-1)
18                   + (1-(1-lambda)*mu1): true;
19  endmodule
```

Obsérvese que cada posible valuación de la variable `q1` es tratada en las tres aristas. Las tres guardas correspondientes forman una partición del espacio de estados. If some value was not covered, it would result in unspecified behaviour (a *deadlock*), since the model could not take any action upon reaching such valuation. If instead two guards overlap, the model would show nondeterministic behaviour since two potentially different actions (i.e. two different transitions) could be performed from the same system state. The PRISM tool recognizes these undesired situations y advierte al usuario de su presencia en el modelo.

A pesar de su simplicidad, este modelo puede ser analysed for rare behaviour. For instance one could study the probability of reaching the maximum queue capacity, starting from a non-empty state, before all packets in the queue get processed by the server. Este análisis transitorio puede forzarse al ámbito de los eventos raros simplemente jugando con las probabilidades `lambda` y `mu1`, y con la capacidad de la cola `c`. □

Cada arista en el modelo PRISM del DTMC especifica todas las transiciones que parten de los estados que satisfacen su guarda. Como cada transición que parte de un estado en un DTMC es probabilista, sus pesos deben sumar 1. As mentioned in el Ejemplo 4, when two guards from two edges overlap, i.e. if both can become true for some valuation of the variables, nondeterminism arises. Specifically, PRISM interprets such situation as two disjoint sets of probabilistic transitions outgoing the states which satisfy these guards. A simulation path would have to choose between those sets, with no hint regarding which of them to follow. This is nondeterministic behaviour, and falls outside the scope of the DTMC formalism. Resumiendo, un modelo PRISM DTMC *no puede tener aristas con guardas que se solapen.*

La situación es diferente en el escenario estocástico, since race conditions naturally express the existence of several unrestricted (other than having a positive rate) transitions leaving a state. Therefore, a PRISM CTMC model *can have edges whose guards overlap.* The concrete states satisfying multiple guards would simply resolve the resulting race condition, realizando la transición que se disparó primero y descartando las otras.

Como las hemos definido hasta ahora, una cadena de Markov models the sequential evolution of a probabilistic or stochastic process. In reality however most hard- and software systems are not sequential but parallel in nature [BK08, sic]. A process can be defined by the parallel execution of its components, also called *modules*. Notice the **`module`** and **`endmodule`** keywords in Códigos 3.1 y 3.2. The PRISM language allows the definition of several modules, and the process resulting from the parallel execution of all of them es llamada el *modelo global del sistema.*

Los módulos individuales pueden ser totalmente independientes during the parallel execution of the global system model, evolving autonomously, or can communicate and cooperate in some way. The first option is called *entrelazado* and for the second option there are many alternatives, like shared variables and channel systems the *mecanismo de handshake* will be used along the thesis, where modules execute certain transitions de forma *sincrónica* y entrelazando la ejecución de todas las demás transiciones.

El uso de de handshake introduce un nuevo tipo de elemento, las *etiquetas o acciones de sincronización*, usadas por los módulos para comunicarse entre ellos. Estas acciones proveen los medios para que varios componentes tomen una transición en sincronía. Están muy relacionadas con el conjunto de acciones A de las Definiciones 1 y 4 de LTS y SA, aunque aquí las usaremos para sincronizar cadenas de Markov.

En el lenguaje de PRISM cada arista en un módulo está etiquetada con una acción envuelta en corchetes. Si los corchetes están vacíos se asume la transición especial τ (tau), que no realiza sincronización alguna e implica una *arista de interleaving*. Esta es la situación de todas las aristas del modelo de PRISM en el Código 3.2. En contraposición, cuando dos o más aristas de diferentes módulos comparten una etiqueta que no es τ, deben ser ejecutadas en simultáneo (o no pueden ocurrir). Ilustramos esto con un nuevo ejemplo.

Ejemplo 5: Modelo CTMC de la cola tándem.

La cola tándem markoviana del Ejemplo 2 fue introducida usando un entorno de tiempo continuo, y por ende podría ser expresada como un CTMC. El Código 3.3 muestra un modelo PRISM posible. El sistema que representamos tiene tres módulos dependientes, `Arribos`, `Cola1`, y `Cola2`, que corren en paralelo y sincronizan usando las acciones `arribo`, `servicio1`, and `servicio2`.

Código 3.3: modelo PRISM de la cola tándem

```
ctmc

const int      c = 8;  // Capacidad de ambas colas
const int lambda = 3;  // tasa(-> q1         )
const int    mu1 = 2;  // tasa(   q1 -> q2   )
const int    mu2 = 6;  // tasa(         q2 ->)

module Arribos
    // Arribo externo de un paquete
    [arribo]  true -> lambda: true;
endmodule

module Cola1
    q1: [0..c-1] init 0;
    // Arribo de un paquete
    [arribo]  q1<c-1 -> 1: (q1'=q1+1);
    [arribo]  q1=c-1 -> 1: true;
    // Procesamiento de un paquete
    [servicio1] q1>0 -> mu1: (q1'=q1-1);
endmodule

```

```
22 module Cola2
23     q2: [0..c-1] init 1;
24     perdido: bool init false;
25     // Arribo de un paquete
26     [servicio1] q2<c-1 -> 1: (q2'=q2+1);
27     [servicio1] q2=c-1 -> 1: (perdido'=true);
28     // Procesamiento de un paquete
29     [servicio2] q2>0 -> mu2: (q2'=q2-1);
30 endmodule
```

Consideremos el arribo externo modelado en el módulo `Arribos`, línea 10. Para que ocurra un arribo, la acción `arribo` es transmitida para la sincronización con los otros módulos. La `Cola2` puede ignorar el asunto puesto que ella sólo reacciona a las acciones `servicio1` y `servicio2`. El módulo `Cola2` en cambio sí sincroniza con la acción `arribo`, por lo que el arribo externo será permitido sii alguna de las aristas en las líneas 16 y 17 está habilitada. Las guardas de estas aristas fueron escogidas para que en cualquier instante de tiempo, exactamente una se encuentre habilitada. Por consiguiente el arribo siempre puede tener lugar, lo cual es consistente con un sistema de colas realista.

Así, el arribo de un paquete es modelado vía la ejecución en síncronía de las correspondientes aristas `arribo` in both modules. An analogous mechanism is set in motion when the server in the first queue processes a packet. Then the action is `servicio1` and the modules participating in the synchronous transition are `Cola1` and `Cola2`. No synchronisation uses action `servicio2`, so it could be replaced with τ by using the empty brackets '`[]`' in line 29, without affecting the global behaviour.

Este modelo markoviano de una cola tándem presenta muchas oportunidades for the study of rare behaviour. From the transient point of view, it is interesting to know which is the probability of loosing a packet in the second queue due to an overflow (indicated by `perdido'=true`), before the server processes all packets and `q1` takes the null value. From the steady-state point of view one can be interested in the long run probability of observing such packet loss. The rarity of these events can be tuned with the system parameters expressed como enteros constantes en el Código 3.3.

Notemos por último que en la forma en que fueron presentados e ignorando la variable indicadora `perdido`, los módulos `Cola1` y `Cola2` son copias exactas módulo un renombrado de variables y acciones de sincronización. Más colas podrían ser añadidas similarmente, extendiendo el modelo a una red Jackson de n colas en tándem para $n \in \mathbb{N}$ arbitrario. □

En el lenguaje PRISM una acción también puede ser bloqueante, cuando uno de los módulos que participan de la sincronización no tiene una guarda habilitada. Removamos por ejemplo la línea 17 del Código 3.3 (en el módulo `Cola1`). When `q1=c-1` no arrival would then be allowed, since `Cola1` synchronises on label `arribo` but the single `arrival`-edge left would have a disabled guard. Por ende las transiciones en el módulo `Arribos` se hallarían bloqueadas.

Este esquema de sincronización entre módulos también permite condiciones de carrera a nivel del modelo global del sistema, but only in interleaving transitions. Cuando two guards from different modules are enabled and the edges do not synchronise, a race condition is formed and resolved. The values sampled from the exponential distributions involved will determine which one takes place, al igual que en el caso intra-modular.

Para ilustrar esta sincronización, considérese el estado inicial de la cola tándem del Ejemplo 5. Notice the single guard in the `Arribos` module is always satisfied. Also, since initially `q2` is assigned the value `1` in `Cola2`, the last edge of that module is enabled as well. Hence two transitions are enabled in the initial global state of the system model: one corresponding to an arrival in module `Arribos`, and another one corresponding to a packet processing in module `Cola2`. Esta es una condición de carreara de alcance global dado que ambas aristas no sincronizan.

Una última cuestión de peso a considerar acerca del lenguaje PRISM es la tasa resultante de la sincronización de transiciones. Para aristas en interleaving que se ejecutan sin sincronización, la tasa de las transiciones subyacentes en el nivel concreto es el número racional que precede al punto y coma de la arista. Este es e.g. el caso de `mu2` en la línea 29 del Código 3.3.

Cuando en cambio varias aristas de diferentes módulos son ejecutadas en sincronía, *el producto de sus tasas* será la tasa de ejecución de la transición gobal subyacente a nivel concreto. Como se hizo en el Ejemplo 5, una estrategia para obtener la tasa deseada consiste en poner la tasa deseada para la transición global en la arista de un único módulo representativo. Todas las otras aristas que sincronicen con ella serán otorgadas la tasa '1', de manera que el producto sea la tasa global deseada.

3.3.2. Especificación de la consulta del usuario

Recuérdese que hay dos ángulos de estudio desde los cuales estudiamos los eventos raros en esta tesis: transitorio y estacionario. Las ecuaciones (7) y (8) de la Subsección 2.5.1 ya proveen fórmulas de lógicas temporales para expresar consultas de comportamiento transitorio y estacionario respectivamente.

Sin embargo, esas expresiones asumen una representación explícita de los conjuntos de estados raros y de parada, A y B, o a lo sumo en términos de sus proposiciones atómicas características, RARO y PARAR. Como A y B pueden ser definidos a nivel simbólico, sería razonable esperar que expresiones que involucren variables, y no proposiciones atómicas ni mucho menos estados concretos, sean el medio por el cual se consulten los valores probabilísticos buscados.

El lenguaje de especificación de propiedades de la herramienta PRISM incluye varias varias lógicas temporales, incluyendo PCTL y CSL. Además, al tope de la fórmula ofrece los operadores cuantitativos `P=?` y `S=?` relacionados con estas lógicas, que respectivamente devuelven el valor numérico para comportamiento de naturaleza transitoria o estacionaria. Específicamente, estos operadores devuelven "la probabilidad" del conjunto de estados que satisface la subfórmula englobada, lo cual encaja a la perfección en nuestro marco de análisis.

Para consultar por el comportamiento transitorio, el usuario especificará

$$\texttt{P=? [!}\textit{parar}\ \texttt{U}\ \textit{raro}\ \texttt{]}$$

donde ambas *parar* y *raro* son expresiones booleanas que pueden incluir literales, constantes, y variables definidas en el modelo PRISM del sistema. La expresión *parar* identifica los estados de parada, que truncan los caminos de simulación que los visitan. La expresión *raro* identifica al evento raro de interés.

El análisis estacionario se consulta mediante la expresión

$$\texttt{S=? [}\textit{raro}\ \texttt{]}$$

para la misma definición de *raro*. Es interesante el hecho de que, si bien la lógica CSL fue diseñada para estudiar sistemas en un entorno de tiempo continuo, la expresión introducida también puede ser empleada en modelos DTMC. Devolverá la probabilidad de visitar los estados que satisfagan *raro*, una vez que el sistema se encuentre en equilibrio.

Ejemplo 6: Consultas para la cola tándem.

Volviendo al estudio de la cola tándem, supóngase que el usuario desea saber la probabilidad de perder un paquete en la `Cola2` antes de que su buffer se vacíe. Dicha pérdida implica que el servidor de la primera cola procese y envíe un paquete a una segunda cola saturada, viz. cuando `q2=c-1`. Usando la variable indicadora `perdido`, la propiedad consultada para realizar el correspondiente análisis transitorio es:

```
P=? [ q2>0 U perdido ],
```

que formalmente pregunta por "la probabilidad de no observar una segunda cola vacía, hasta que un paquete sea perdido en `Cola2`".

Recordemos que inicialmente hay un paquete almacenado en `q2`, lo cual es necesario para mantener el estado inicial del sistema fuera del conjunto de parada B. Desde el punto de vista práctico esto significa que sin la directiva '`init 1`' en la definición de `q2` (línea 23 del Código 3.3), todas las simulaciones se detendrían tan pronto como empiecen.

Si en cambio el usuario está interesado en la probabilidad a largo plazo de perder un paquete en la `Cola2`, la consulta debería ser:

```
S=? [ perdido ].
```

Como la cola tándem como está descripta en el Ejemplo 2 y modelada en el Ejemplo 5 es un sistema ergódico[‡], el valor inicial de `q2` no es importante para esta propiedad. □

[‡] La cadena de Markov subyacente es ergódica porque todos sus estados son aperiódicos y positivos recurrentes, ver e.g. [Tso92].

3.3.3. Selección de los umbrales

Las dos subsecciones previas proveen un marco para especificar el modelo y la consulta del evento raro. Todo se corresponde con el lenguaje de entrada de PRISM, y esta herramienta cuenta con un motor de simulación propio, por lo que el análisis por simulación de Monte Carlo estándar de la Subsección 2.3.2 podría ser llevado a cabo sin más.

Usar división por importancia no es tan directo, debido a la información extra que esta técnica requiere. La Subsección 3.2.3 explica cómo derivar automáticamente una función de importancia a partir de las entradas que el usuario normalmente le proveería a PRISM, pero eso no es todo lo que se necesita. La mayoría de las técnicas de división necesitan escoger los valores de los umbrales, donde se realizará el copiado de los caminos de simulación (e.g. para división fija), o donde cada etapa de la simulación comienza y termina (e.g. para esfuerzo fijo).

Podría usarse sencillamente cada valor de importancia como un umbral, e.g. splitting each time a simulation visits a state with higher importance than the previous one. Obviously this has big chances of incurring in a large computational overhead, which could easily render useless any gain derived from the use of splitting. Choosing every other importance value as a thresholds sounds more reasonable, pero nuevamente no es otra cosa que una elección a ciegas.

Reconocemos dos soluciones al problema planteado: selecting the thresholds ad hoc, tailored for the specific system under study, or using an algorithm to analyse the automatic function produced by Algoritmo 1, trying to "derive the thresholds" as well. In general and following the automatable approach of the whole thesis, it is desirable to choose the thresholds adaptively, considerando la estructura de cada modelo particular.

De entre las implementaciones populares de técnicas de división que mencionamos al final de la Subsección 2.5.1, la División Multinivel Adaptativa sobresale por su alegada "selección dinámica de umbrales". Recall this technique and its successor, Sequential Monte Carlo, perform simulations on a system where only the importance of the states is assumed known. By statistical analysis of the maximum importance reached by each simulation path, the values of the thresholds (or its analogous in that setting) are incrementally discovered, del estado nicial hasta el evento raro.

Por ende, para implementar un método de división automático tenemos las siguientes opciones:

1. usar División Multinivel Adaptativa (o Monte Carlo Secuencial) como técnica de división estándar, descartando por completo la necesidad de escoger umbrales;
2. extraer la idea algorítmica de estas estrategias para hallar los umbrales, usándola luego con cualquier otra técnica de división.

RESTART was selected as default splitting algorithm due to its nice practical properties (see Sección 2.6). Therefore we chose the second approach; the pseudocode for the selection of the importance values to use as thresholds se presenta en el Algoritmo 2. Toma los siguientes parámetros:

$\mathcal{M}$ - el modelo del sistema;
f - la función de importancia;
n - el número de simulaciones independientes lanzadas por simulación;
k - el número de simulaciones "exitosas"de entre las n lanzadas;
m - el número de eventos discretos a generar en cada simulación.

Además, el algoritmo usa las siguientes variables internas:

`sim` - un arreglo para los estados con la máxima importancia de cada simulación;
`T` - una cola que almacena los valores de importancia escogidos como umbrales.

En el Algoritmo 2 la rutina $\mathcal{M}$.`simulate_ams`$(s, n, m, f,$ `sim`$)$ lanza n simulaciones independientes que comienzan del estado s, y almacena en la variable arreglo `sim` los estados que representan los máximos valores de importancia observados en esas simulaciones (n en total). Ordenar estos estados de acuerdo a su valor de importancia deja en la k-ésima posición de `sim` al estado representante del k/n cuantil de importancia, cuyo valor de importancia es tomado como un nuevo umbral.

Algoritmo 2 Selección de los umbrales con División Multinivel Adaptativa.

Entrada: módulo $\mathcal{M}$
Entrada: función de importancia $f\colon S \to \mathbb{N}$
Entrada: setup de las simulaciones $k, n, m \in \mathbb{N}_{>0},\ k < n$

Var: `sim`$[n]$ **Tipo:** arreglo de estados
Var: `T` **Tipo:** cola de enteros {los umbrales}

$s \leftarrow \mathcal{M}$.`initial_state`()
`T`.`push`$(f(s))$
`falla` $\leftarrow$ **false**
repeat
  $\mathcal{M}$.`simulate_ams`$(s, n, m, f,$ `sim`$)$
  `sort`(`sim`$, f$)
  $s \leftarrow$ `sim`$[k]$
  if `T`.`back`$() < f(s)$ **then** {`T`.`back`() no desencola}
    `T`.`push`$(f(s))$ {nuevo umbral: k/n cuantil de importancia}
  else
    `falla` $\leftarrow$ **true**
  end if
until `T`.`back`$() =$ máx(f) **or** `falla`
for $i \leftarrow$ `T`.`back`$() + 1$ **to** máx(f) **do**
  `T`.`push`(i) {los valores de importancia no alcanzados serán umbrales}
end for

Salida: la cola `T` con los valores de los umbrales

Las entradas numéricas del algoritmo, $k, n, m \in \mathbb{N}$, deben ser escogidas ad hoc por el usuario. Heurísticas basadas en la naturaleza de la función de importancia f

y del modelo $\mathcal{M}$ son fáciles de implementar y han sido empleadas en la herramienta de software desarrollada. De acuerdo a nuestras observaciones empíricas, siempre que las cotas $m \in [10^3, 10^5]$, $n \in [10^2, 10^4]$, y $k \approx {}^{n}/_{s_m}$ sean respetadas, donde s_m es el máximo valor de división en cualquier umbral, estos valores sólo tienen un impacto moderado en la eficiencia del algoritmo.

Esta *elección adaptativa de umbrales* provee toda la información complementaria needed by an importance function, ad hoc or automatic. In particular, the function derived with Algoritmo 1 exploits the structure of the adjacency graph of each model $\mathcal{M}$, but disregards all probabilistic and stochastic information. The paths generated by $\mathcal{M}$.`simulate_ams`$(\cdots)$ in Algoritmo 2 do consider such information, in a state space already labelled with importance. The resulting thresholds, which are the most important metadata used by RESTART during simulations, por ende reflejando el comportamiento completo del modelo bajo estudio.

Hay ciertas cuestiones a considerar a la hora de usar esta estrategia en el entorno de la tesis, relacionadas con la forma en que la División Multinivel Adaptativa fue introducida por Cérou et al. en [CG07]. Las discutiremos a continuación.

Espacios continuos vs. espacios discretos

La teoría que prueba que la División Multinivel Adaptativa devuelve un estimador de varianza óptima está basada en un espacio de estados continuo y una función de importancia sobre ellos [CG07]. Esto significa que en el algoritmo original, los umbrales pueden ser seleccionados arbitrariamente cerca unos de otros, lo cual es una de las hipótesis con las que se asegura la optimalidad.

Para el alcance de esta tesis el espacio de estado es finito, and even though optimality is not a major concern, the continuity hypothesis of Adaptive Multilevel Splitting may have repercussions regarding termination of the idea behind Algoritmo 2. More precisely, simulation paths tend to "go down" as the rarity of the states increases. That is, several simulations launched by $\mathcal{M}$.`simulate_ams`$(\cdots)$ could only visit states whose importance is below la del punto de partida en el estado 's'.

If more than $n - k$ simulations go down in the way described above, the ${}^{k}/_{n}$ importance quantile from `sim` will not be above the previously defined threshold. An iteration of the **repeat**–**until** loop in Algoritmo 2 would then fail to provide a new value to store in `T`.

To remedy such situations it was decided to consider all unreached importance values as thresholds. This strategy is coherent: if that many simulations go down without visiting higher importance states, then they are dwelling in a very rare zone, where the chances of observing the "next importance value" are less than ${}^{k}/_{n}$. Thus the best that can be done is to regard such next importance value as a threshold. Notice this is exacerbated by the discreteness of the sates and the importance function.

En el Algoritmo 2 la variable booleana `falla` identifica y resuelve los casos en los que el cuantil ${}^{k}/_{n}$ de importancia no devuelve un umbral más alto. Esto nos da condiciones suficientes para probar la terminación.

Proposición 7 (Terminación del algoritmo de selección de umbrales)**.** *Sean $\mathcal{M}$ un modelo DTMC o CTMC finito, f una función de importancia con imagen en los números naturales, y sean $k, n, m \in \mathbb{N}, k < n$. Entonces, tomando esas entradas, el Algoritmo 2 termina luego de ejecutar una cantidad finita de instrucciones.*

Demostración. El ciclo **for–do** del final tiene un rango finito y un instrucción de tiempo constante en su cuerpo. Por ende terminará, y basta con probar la terminación del ciclo principal **repeat–until**. Cada una de las n simulaciones lanzadas por la rutina $\mathcal{M}$.`simulate_ams`$(\cdots)$ genera M eventos discretos y luego termina. La rutina de ordenamiento `sort`$(\cdots)$ también lleva a cabo sus tareas usando un número finito de instrucciones. Por ende sólo resta probar que la guarda del ciclo es satisfecha tras un número finito de pasos. Si una iteración no devuelve un estado cuya importancia es mayor a la del umbral anterior, `falla` se pone en `true` y se satisface la guarda del ciclo. Sino, el condicional dentro del ciclo asegura que cada llamada la rutina $\mathcal{M}$.`simulate_ams`$(\cdots)$ en cada iteración del ciclo devolverá un estado con importancia más alta que el último valor almacenado en `T`. Como S es finito, también lo es el codominio de f, y por ende el número de valores distintos que puede considerarse es finito. En consecuencia, tras una número finito de iteraciones el valor 'máx(f)' será almacenado en `T`, y la guarda del ciclo será satisfecha. □

Al igual que con la Proposición 6 para el algoritmo de derivación de la función de importancia, la Proposición 7 hace caso omiso de la propiedad de pérdida de memoria. Eso significa que el Algoritmo 2 en realidad puede ser usado con cualquier proceso estocástico homogéneo.

A pesar de su coherencia, la estrategia de terminación empleada en el Algoritmo 2 es un tanto ríspida. Hay alternativas más moderadas a la variable booleana `falla`, como usar un contador de fallas y un número predeterminado de cota de tolerancia. Cada vez que la condición `T.back`$() < f($`sim`$[k])$ falle, el ciclo sería repetido para los mismos `T` y s, pero incrementando el contador de fallas y el esfuerzo de $\mathcal{M}$.`simulate_ams`$(\cdots)$ (e.g. aumentando m o n). Si el contador alcanza la cota de tolerancia, entonces el ciclo **repeat–until** se rompe como en el Algoritmo 2. Si, en cambio, un nuevo umbral es hallado, el contador y cualquier otra variable modificada como m o n puede ser restablecida a sus valores originales.

La importancia del evento raro

Otro tema con nuestro entorno de aplicación en comparación con la teoría original de [CG07] es el valor de importancia de los estados raros. En la División Multinivel Adaptativa el evento raro está definido como todos los estados de proceso de Markov fuerte por encima de cierta barrera M. Esto coincide con el marco de la Subsección 2.5.1 donde $A = E_n$, i.e. cuando f mapea todos los estados raros a los valores de importancia más altos, que es el caso de la función automáticamente derivada. Pero algunos sistemas no encajan naturalmente con esta caracterización, como se mostrará en la Sección 3.6.

Una solución sencilla sería detenernos exclusivamente por valor de importancia en lugar de considerar la condición de rareza. Esta es precisamente la táctica que

sigue el Algoritmo 2. No hay verificaciones respecto de la definición del usuario del evento raro; sólo importa la distribución de importancia impartida por f.

La desventaja de esta técnica es que en algunos casos, máx(f) representa un situación extremadamente rara, y los eventos raros "más mundanos" están representados por estados cuya importancia es tan sólo una fracción de dicho valor. Entonces el Algoritmo 2 podría tardar mucho tiempo en converger, intentando alcanzar el valor máx(f), en lo que bien podría considerarse un malgaste de esfuerzo computacional. Dado que si hay estados raros con mucha menor importancia que máx(f), la mayor parte de las observaciones del evento raro involucrarán a estos estados, y no a aquellos que poseen el máximo valor de f. Este tema es discutido en mayor detalle en las Secciones 3.5 y 3.6.

3.3.4. Estimación y convergencia

Junto con el Algoritmo 1 para derivar automáticamente una función, y dejando de lado momentáneamente el valor de división concreto a utilizar, las Subsecciones 3.3.1 a 3.3.3 proporcionan suficientes mecanismos para implementar *simulaciones* que usen la división multinivel automáticamente.

Cada simulación genera un estimador puntual $\hat{\gamma}_i$ for the probability γ of the rare event. The next step is using the statistical theory from Subsección 2.3.3, to analyse the sample $\{\hat{\gamma}_i\}_{i=1}^N$ and produce the desired interval estimate around the mean of the data, $\hat{\gamma} \doteq {}^1/_N \sum_{i=1}^N \hat{\gamma}_i$. The intention is to provide the user with a reliable guess of γ, where reliability is quantified in terms of coeficiente de confianza y precisión del intervalo.

El intervalo de confianza de la Definición 5 y la Ecuación (6) en la Subsección 2.3.3 assumes the population mean is estimated without information about the distribution of the samples. Since the variance σ^2 is unknown and approximated with the estimator S_N^2 for a sample of size N, el Teorema Central del Límite es usado con la distribución t de Student para garantizar que

$$P\left(\mu \in \left[\, \overline{X} \pm t_{\frac{\alpha}{2}} \sqrt{\frac{S_N^2}{N}} \,\right]\right) \approx 1 - \alpha \tag{12}$$

donde t_α es el cuantil α de la distribución t de Student, $\overline{X} = \hat{\gamma}$ es el valor medio de la muestra aleatoria $\{X_i\}_{i=1}^N = \{\hat{\gamma}_i\}_{i=1}^N$, y $\mu = \gamma$ es la media poblacional desconocida.

No obstante, cuando se realiza un análisis transitorio, each path will either find a rare state or get prematurely truncated upon encountering a stopping state. Thus each simulation can be regarded as a Bernoulli trial, donde observar al evento raro es sinónimo de éxito.

Bajo esta configuración, correr N simulaciones is equivalent to performing an $\langle N, \gamma \rangle$-Binomial experiment, where the variance of each Bernoulli trial is characterised by the expression $\sigma^2 = \gamma(1-\gamma)$. Usar $S_N^2 = \hat{\gamma}(1-\hat{\gamma})$ para estimar la varianza de la población, ecuación (12) genera pues el IC

$$\hat{\gamma} \pm t_{\frac{\alpha}{2}} \sqrt{\frac{\hat{\gamma}(1-\hat{\gamma})}{N}} \ . \tag{13}$$

Para criterios de confianza provistos por el usuario, i.e. un coeficiente de confianza α y un ancho de intervalo d, esto sugiere el siguiente método para generar el IC: increasingly generate samples, viz. simulation paths, computing each time the value of $t_{\frac{\alpha}{2}}\sqrt{\hat{\gamma}(1-\hat{\gamma})/N}$; as soon as it falls below $d/2$, se satisfizo el criterio deseado y la estimación concluye.

Recuérdese sin embargo que el modelo y la consulta de propiedad están bajo un régimen de evento raro, where asking for relative error is more robust than requesting some width 'd' fixed a priori. Moreover the expression of ecuación (13) is unreliable for the extreme cases when $\gamma \approx 0$ and $\gamma \approx 1$. There are more robust estimators, like the Wilson score interval [Wil27], que cubre estos casos con mejor precisión.

Teniendo todo esto en cuenta, esta estrategia basada en la transient nature of the simulations provides a simple convergence decision mechanism. Unfortunately, its apparent suitability notwithstanding, no puede usarse confiablemente en general como criterio de parada.

El problema nace del uso de la división multinivel. When simulations are standard Monte Carlo, each sample has a success/failure outcome. Instead, when using e.g. RESTART with a stacked splitting factor K, the result can take any value in $\{0, 1/K, 2/K, \ldots, K-1/K, 1\}$. Therefore, and in general for any splitting technique, no puede asegurarse que la muestra aleatoria $\{X_i\}_{i=1}^{N}$ siga una distribución Binomial $\langle N, \gamma \rangle$ [†].

Para la estrategia de estimación que seguimos en este trabajo, the complication described above materializes in a premature stop: the convergence criterion deems the current data set sufficient, when in truth more simulations were needed to build an interval containing γ con la confianza deseada.

El problema de cobertura real del parámetros is one of the most difficult to solve in RES [GRT09]. One strategy is to use the standard IC expression of ecuación (12), based on the Central Limit Theorem which makes few assumptions on the random sample. Notice anyway that in a typical scenario where the distribution of the simulated paths is unknown, the sample size required to satisfy the confidence criteria can only be discovered a posteriori. Esto puede ser un tema delicado cuando el presupuesto de cómputo está prefijado.

Este análisis sigue líneas similares a las del uso del relative error to define the desired interval width, since that also implies simulating first and estimating later, with no foreseeable notion of termination. Reijsbergen et al. estudian complicaciones análogas en [RdBS16], cuando se usa muestreo por importancia para test de hipótesis.

Dándole prioridad a la confiabilidad sobre la velocidad de convergencia, this thesis uses the expression from ecuación (12) to build the IC in both transient and steady-state analysis. Reemplazando la nomenclatura estadística estándar con sus contrapartes en RES, la ecuación que se aplicará en las estimaciones es

$$P\left(\gamma \in \left[\hat{\gamma} \pm t_{\frac{\alpha}{2}}\sqrt{\frac{\hat{\sigma}_{\hat{\gamma}}}{N}}\right]\right) \approx 1-\alpha \tag{14}$$

[†] Se sabe de problemas similares que afectan al muestreo por importancia [RdBS16].

donde $\hat{\sigma}_{\hat{\gamma}}$ es la varianza empírica de la muestra $\{\hat{\gamma}_i\}_{i=1}^{N}$ (ver la ecuación (4)), y $t_{\frac{\alpha}{2}}$ es el cuantil $\alpha/2$ de la distribución t de Student.

Imponemos la cota inferior convencional $N \geqslant 30$, tras la cual las comparaciones para evaluar la convergencia comienzan. Cada nueva muestra (viz. cada nuevo resultado de simulación) actualiza el valor estimado $\hat{\gamma}$ y por ende también el ancho deseado del intervalo, por medio de la técnica de error relativo. Cada muestra también actualiza el ancho empírico computado para el IC siguiendo la ecuación (14). Cuando este ancho empírico se torna más angosto que la precisión derivada de $\hat{\gamma}$ como su error relativo, se asume la convergencia y se devuelve el IC resultante.

Notablemente, esta estrategia de estimación es equivalente al "test de Chow-Robbins" de [RdBS16], reportado en ese trabajo como la única alternativa que devuelve una estimación correcta independientemente de la distribución real de la muestra. El precio a pagar es la inhabilidad para predecir cuantos caminos de simulación serán necesarios para satisfacer el criterio de convergencia del usuario.

3.4. Herramientas de soporte

Desarrollamos la herramienta de software BLUEMOON, que implementa nuestro método automatizado para división por importancia como lo describimos en las sección anterior. Está escrita en C++ y Java como una extensión modular del model checker probabilista PRISM [KNP11], versión de desarrollo 4.3, que se ejecuta sobre la Máquina Virtual de Java.

La herramienta BLUEMOON es software gratuito, libre y de código abierto, disponible con licencia Pública General de GNU (GPL v3). El código fuente puede ser descargado de la página de la herramienta, localizada en la página web del Grupo de Sistemas Confiables en http://dsg.famaf.unc.edu.ar/tools.

Cabe destacar la naturaleza prototípica de BLUEMOON, which was devised to validate the theory presented in this chapter. The desire for empirical validation motivated the choice of continuous and discrete time Markov chains. Many studies already exist for this kind of systems, facilitando la tarea de reproducir resultados conocidos.

A pesar de ello, y como discutimos en las Subsecciones 3.2.3 y 3.3.3, the algorithms and techniques introduced do not make any assumption of memorylessness. They can thus be employed to study more general stochastic processes, como mostraremos en el Capítulo 4.

Siendo una extensión de la herramienta PRISM, BLUEMOON reads DTMC and CTMC models described in the PRISM input language, and property queries expressed in the PRISM property specification language (ver las Subsecciones 3.3.1 y 3.3.2). The model and queries are written down in a text file and the tool is invoked in the command line, passing as input the file and the options `--rare vent <tipo> <estrat>` y `--rareconf <conf> <prec>`. Los argumentos son obligatorios; su sintaxis y semántica es la siguiente:

`<tipo>` Especifica si el análisis es transitorio o estacionario, lo cual se indica

resp. con los valores `tr` y `ss`.

`<estrat>` Especifica la estrategia de simulación a emplear. Su valor debe ser uno de los siguientes:

- `nosplit` para usar la táctica de Monte Carlo estándar;
- `auto` para usar la táctica de división por importancia con la función de importancia automática, derivada del modelo y la consulta usando el Algoritmo 1;
- `adhoc` para usar división por importancia con una función ad hoc, a ser definida por el usuario.

`<conf>` Especifica el coeficiente de confianza deseado por el usuario y debe ser un número (racional) en el intervalo abierto $(0, 1)$.

`<prec>` Especifica la precisión del intervalo, y puede ser un número racional concreto, o un porcentaje con el formato `p%` para usar la técnica de error relativo, donde `p` es un número natural en el intervalo $[1, 100]$ interpretado sobre el ancho total del intervalo.

Por ejemplo la línea de comando

```
>_ prism-bm modelo.prism --rarevent tr auto --rareconf .9 20%
```

invoca la herramienta (identificada con el comando `prism-bm`) sobre el modelo PRISM `modelo.prism`, para ejecutar un análisis transitorio usando división por importancia con la función automáticamente derivada por BLUEMOON, solicitando un nivel de confianza del 90 % y un error relativo del 10 % (el ancho empírico del IC debe ser a lo sumo un 20 % del valor estimado, i.e. menor que $0{,}2\,\hat{\gamma}$).

También hay una opción `--rareparams` que toma como argumento una lista separada por comas con personalizaciones, como el nivel de división a usar, la técnica de construcción del intervalo de confianza, un timeout de tiempo-pared de ejecución, etc. One of its most relevant uses is to define the importance function when the ad hoc approach is selected. La función debe ser una expresión aritmética de tipo entero que use literales, constantes, y variables del modelo. Por ejemplo el comando

```
>_ prism-bm modelo.prism --rarevent ss adhoc  \
     --rareconf .95 20%                       \
     --rareparams "timeout=5,ifun=acc^2-5*q2"
```

corre un análisis estacionario en `modelo.prism`, usando división con la función de importancia ad hoc que le substrae cinco veces el valor de `q2` al cuadrado de `acc`. Both `acc` and `q2` should be defined in `modelo.prism`, either as constants or variables, and the expression must evaluate to an integer.

La función de importancia ad hoc can also be defined as a PRISM formula inside of the model. The example above is equivalent to appending "`formula importance = acc^2-5*q2`" como nueva línea al archivo `modelo.prism`, y luego

ejecutar el mismo comando pero con `timeout=5` como único elemento de la lista pasada como argumento a la opción `--rareparams`.

Cuando se selecciona la construcción automática de la función de importancia, el Algoritmo 1 is used to build an explicit function on the state space of the global system model. This means the importance value of each concrete state is stored as an integer in a vector. Interestingly, the backwards BFS of the algorithm uses a column major sparse matrix representation (CSC) of the adjacency graph, which eases the reversed traversing of the model transitions. Since simulations need to take transitions in a forward manner, the matrix is made row major (CSR) una vez que la función de importancia ha sido construida.

La técnica de división multinivel implementada en la herramienta BLUEMOON es RESTART. Whenever the `auto` or `adhoc` argument is passed to the option `--rarevent`, the thresholds are selected for the corresponding importance function using a variant of Algoritmo 2. Once the thresholds are ready, RESTART simulations are executed to generate the collection $\{\hat{\gamma}_i\}$ of estimates, de la cual el IC es construido como se lo detalló en la Subsección 3.3.4.

Un valor de división global es usado para todos los umbrales. By default it equals two, meaning each time a trial crosses a threshold upwards, two trials will continue execution, i.e. one clone is created. Este valor puede ser modificado con la personalización `split=<num>` que ofrece la opción `--rareparams`.

El valor de división global influencia la elección de los umbrales by means of the *balanced growth* approach [Gar00, eq. (2.25)]. Generally speaking, the higher the splitting value, the further the thresholds will be from each other. The aim is to have roughly the same level-up probability (Definición 10) in all importance levels, dado que esto debería aumentar la eficiencia de RESTART [VAMMGFC94].

La salida principal de la herramienta displays the resulting point estimate $\hat{\gamma}$, the precision of the IC, and the interval itself. It also shows the current stage of the execution: building the model, the importance function, the thresholds, etc. Específicamente cuando se está simulando, muestra cuantas muestras $\hat{\gamma}_i$ se han generado hasta el momento. Un extracto de las salida se ve así:

```
>_ PRISM
   =====

   Version: 4.3.dev
       ⋮
   Type:        CTMC
   Modules:     ContinuousTandemQueue
   Variables:   q1 q2 arr lost
   ---------------------------------------------
   [DEV] Rare event simulation chosen.
   [DEV] Simulation type: TRANSIENT
   [DEV] Simulation strategy: RESTART_AUTO
       ⋮
```

```
Identifying special states... done.
Building importance function... done.
Setting up RESTART simulation environment... done.
Estimating rare event probability {5}{6}{8}{52} done:
- Point estimate: 5.873E-6
- Precision: 1.175E-6
- Confidence interval: [ 5.286E-6 , 6.46E-6 ]
    ⋮
```

Cuando las estimaciones terminan exitosamente, como en el ejemplo de arriba, se imprime información temporal luego de las estimaciones numéricas. El tiempo-pared total de ejecución se discrimina en las diferentes etapas que componen toda la ejecución. Una salida de ejemplo (continuación de la anterior) es:

```
>_
    ⋮
- Confidence interval: [ 5.286E-6 , 6.46E-6 ]

Processing times information
- Total elapsed time: 29.18 s
- Setup time: 0.68 s
    > Initial setup: 0.02 s
    > Rare/Reference states identification: 0.01 s
    > Importance function building: 0 s
    > Thresholds selection: 0.65 s
- Simulation time: 28.5 s

[DEV] Skipping other model checks.
```

Sin embargo la ejecución se puede interrumpir prematuramente, truncando la estimación antes de que el criterio de confianza haya sido satisfecho. This can happen either by reaching a predefined timeout (set with the `timeout=<num>` customization of `--rareparams`), or by a user or system interrupt. If simulations had already started and there is estimation data available when interrupted, BLUEMOON muestra el estimador puntual alcanzado junto con algunos IC para niveles de confianza típicos. Una salida de ejemplo es:

```
>_
    ⋮
Estimating rare event probability {8}{12}{13}{41} wall time
limit reached.

[rarevent.RareventSimulatorEngine] Interrupted, shutting down
- Point estimate: 4.851E-6
- 90% confidence: precision = 1.285E-6
```

```
                    interval = [ 4.208E-6 , 5.493E-6 ]
- 95% confidence: precision = 1.531E-6
                    interval = [ 4.085E-6 , 5.616E-6 ]
- 99% confidence: precision = 2.012E-6
                    interval = [ 3.845E-6 , 5.857E-6 ]
```

Además de su salida principal, BLUEMOON tiene una salida técnica donde los pasos de ejecución son descriptos en mayor detalle. Information like which concrete states are rare/stopping, the seed fed to the random number generator, the importance value of the thresholds, and even an extract of the importance function, are printed in the technical output of the tool.

Estos datos son útiles para analizar la ejecución en profundidad, e.g. cuando deseamos comparar los umbrales seleccionados, o para debuggeo. Por defecto es lanzada como texto plano en un archivo nombrado en base al tipo de ejecución, entonces por ejemplo

```
>_ prism-bm modelo.prism --rarevent ss auto ...
```

imprimirá la información técnica en "`rarevent_STEADYSTATE_RESTART_AUTO.log`". Esto puede modificarse con la opción `--rareparams` mediante `techlog=<name>`.

Todas las personalizaciones ofrecidas por BLUEMOON pueden consultarse mediante la interfaz de ayuda de PRISM. Por ejemplo el comando `prism-bm--help rareparams` muestra las personalizaciones ofrecidas a través de la opción `--rareparams`. Se invita al lector interesado a visitar el tutorial en la página de la herramienta, donde algunos casos de estudio son ilustrados para el modelo de la cola tándem.

3.5. Casos de estudio

Muchos ejemplos fueron tomados de la bibliografía de RES y analizados con BLUEMOON. Las descripciones generales de los sistemas y los resultados de la experimentación son mostrados a continuación. Los modelos usados para generar estos datos se listan en el Apéndice A.

3.5.1. Entorno de experimentación

Todos los modelos estudiados son cadenas de Markov de tiempo continuo o discreto, descriptas en el lenguaje de entrada de PRISM. En consecuencia, pudimos usar el model checker para validar que los modelos implementados producen las salidas deseadas, i.e. los valores publicados en los trabajos de donde se los extrajo.

Corrimos experimentos independientes para cada caso, estimating intervals for confidence coefficients and relative errors fixed a priori. All experiments ran until the confidence criteria was met, o un límite temporal de reloj pared (*límite de tiempo pared*) fuese alcanzando. El hardware usado fue un procesador Intel

Xeon E5-2620v3 con 12 núcleos de 2.40GHz, con 128 GiB 2133MHz de memoria DDR4 RAM.

Para cada sistema variamos algún parámetro, stressing out the convergence conditions by increasing the rarity of the event hence decreasing the value of γ. For each model and parameter value, we tested three simulation strategies: RESTART using the automatic function, RESTART using ad hoc importance functions (some taken from the literature), and standard Monte Carlo. Four global splitting values were tested en las simulaciones de división multinivel.

Comprobamos la consistencia de los intervalos de confianza obtenidos, comparing them against the values produced by the PRISM tool. This sección presents charts displaying the convergence time for each strategy. Time measurements cover the full computation process, including preprocessings like the compilation of the model file and the selection of the thresholds. We repeated each experiment thrice; the values shown are the average de los tiempos pared de ejecución medidos para cada experimento.

Presentamos todos los resultados mostrando un gráfico por valor de división evaluado. The outcomes of the standard Monte Carlo simulation appear repeated in all charts to ease visual comparisons. In each case, we span along the x-axis the parameter varied to increase the rarity of the event. On the y-axis we show the average time to convergence, en segundos y usando una escala logarítmica.

Por un lado, una barra que alcance el extremo superior de un gráfico significa que hubo un timeout previo a la convergencia, lo cual denotamos como *una falla*. Por el otro lado y excepto donde se aclare lo contrario, todas las simulaciones que convergieron antes del límite de tiempo-pared produjeron intervalos que contenían al valor computado por PRISM para el mismo modelo.

3.5.2. Cola tándem

Análisis transitorio

Recordemos la red Jackson en tándem presentada en el Ejemplo 2, que consiste en dos colas conectadas. Para un entorno de tiempo continuo, replicamos el experimento de [Gar00, p. 84], usando los valores para los parámetros $(\lambda, \mu_1, \mu_2) = (3, 2, 6)$. Iniciando las simulaciones desde el estado $(q_1, q_2) = (0, 1)$, estamos pues interesados en observar una segunda cola saturada antes de que la misma se vacíe. La expresión que usamos para consultar esta propiedad es `P=? [ q2>0 U q2=c ]`, donde la variable `c` es la máxima capacidad de las colas (C).

Usamos el modelo de PRISM del Apéndice A.1. Notice we represent the queue monolithically, in contrast to the modular implementation previously shown in Código 3.3. Both models are semantically equivalent, but the monolithic version makes it easier to signal events, like an external packet arrival, sin la necesidad de usar variables globales.

Nótese a su vez que la tasa de servicio de la segunda cola, μ_2, is greater than the one at the first queue. That means the first queue is the bottleneck and hence the rarity of the saturation comes from the fast service times at the second queue. According to [VAVA06, VA07b, LLGLT09] in respect of the tandem queue, este es

el escenario más difícil de resolver.

Evaluamos capacidades máximas $C \in \{8, 10, 12, 14\}$, para las cuales los valores de γ aproximados por PRISM son respectivamente 5.62e-6, 3.14e-7, 1.86e-8, 1.14e-9. A IC 95 \| 10 criterion was imposed. This means estimations had to reach a 95 % confidence level and 10 % relative error, i.e. the empirical precision of the interval had to be smaller than 0,2 times the estimate $\hat{\gamma}$. Esto debía lograrse dentro de las 3 horas de ejecución pared.

Para las simulaciones con división por importancia, besides the automatic function computed by BLUEMOON (denoted `auto`), we tested three ad hoc importance functions: counting the number of packets in the second queue alone (`q2`), counting the packets in both queues (`q1+q2`), and a weighed variant of that second function (`q1+2*q2`). We used the global splitting values 2, 5, 10, and 15. Standard Monte Carlo simulations are denoted `nosplit` en los gráficos a lo largo de esta sección.

Los promedios del tiempo-pared de ejecución se muestran en la Figura 3.3. Recuérdese que mostramos un gráfico por valor de división, donde las salidas de las simulaciones `nosplit` se encuentran repetidas en todas las gráficas. Además, como el parámetro variado para incrementar la rareza del evento es la capacidad máxima de las colas, expandimos a lo largo del eje de las abscisas los valores de C testeados.

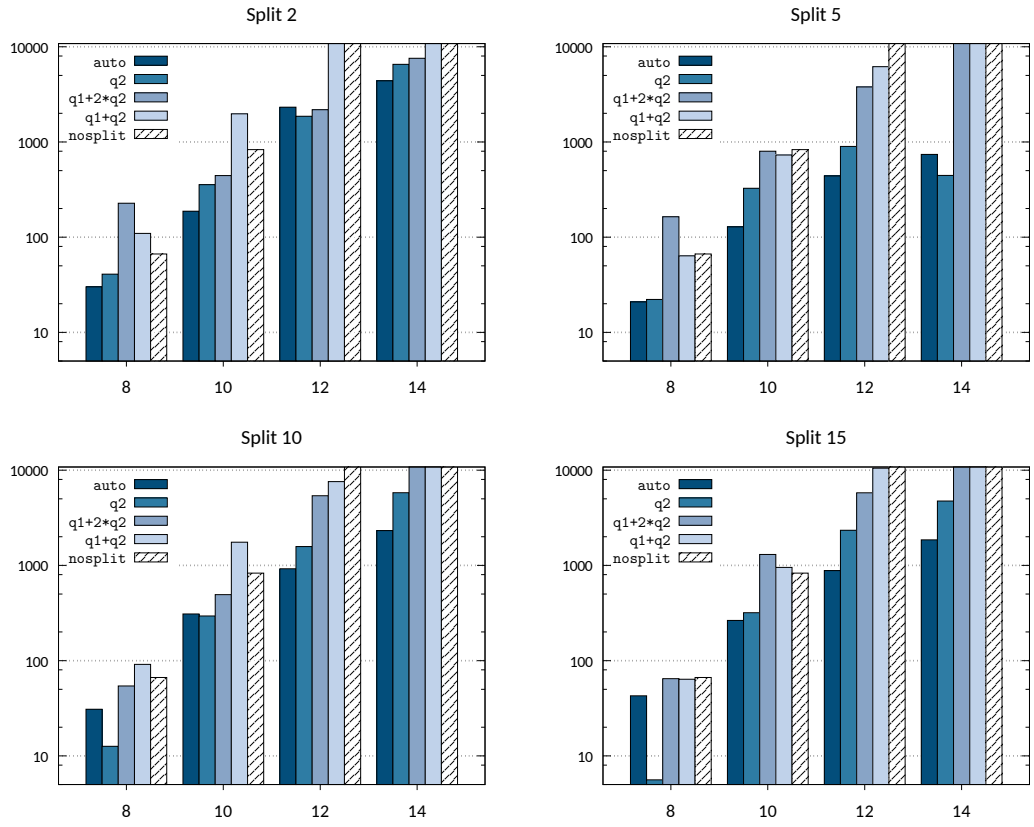


Figura 3.3: Tiempos del análisis transitorio de la cola tándem (modelo CTMC)

Para los mayores valores de C y como cabía esperar, las simulaciones de Monte Carlo estándar fallaron, i.e. no lograron alcanzar el criterio escogido para los intervalos de confianza dentro del límite de tiempo impuesto.

Sorprendentemente y con muy pocas excepciones (e.g. para $C = 8$ con división 10 y 15), the `auto` importance function outperformed all ad hoc variants in most configurations. The closest competitor was `q2`, which sometimes resembled or improved the convergence times of `auto`, most notably for the smallest queue size (cuando el evento no es tan raro).

En general, los resultados parecen indicar que the global splitting strategy implemented in BLUEMOON is quite sensitive to the value chosen. In particular, Figura 3.3 suggests 5 is the best option among the four splittings values used for experimentation. In that respect the `auto` importance function showed less variance than `q2`; compare e.g. the performance of these two functions para los distintos valores de división cuando $C \in \{8, 14\}$.

Como explicamos en la Sección 3.4, BLUEMOON employs the balanced growth approach when automatically selecting the thresholds, in an attempt to reduce the variability of RESTART due to the relationship between the thresholds location and their splitting. The apparently unpredictable behaviour observed when varying the splitting value seems to indicate that the chosen strategy is suboptimal. Las siguientes secciones seguirán discutiendo este tema.

Por último, queremos resaltar los tiempos de convergencia de las variantes ad hoc que mostraron el peor desempeño. Para algunos valores de división cuando $C \in \{8, 10\}$, ambas `q1+q2` y `q1+2*q2` tardaron más en converger incluso que las simulaciones `nosplit`. Esto evidencia sin una elección apropiada de función de importancia, umbrales, y valor de división, la sobrecarga de cómputo de las técnicas de división como RESTART puede degradar el desempeño.

Análisis estacionario

También estudiamos el comportamiento estacionario de una saturación en la segunda cola. Aquí γ representa la proporción de tiempo que la segunda cola permanece en un estado de saturación en corridas a largo plazo. La consulta correspondiente es `S=? [ q2=c ]`.

Para las capacidades máximas testeadas $C \in \{10, 15, 20, 25\}$, los valores de γ aproximados por PRISM son 3.36e-6, 1.62e-8, 7.42e-11, y 3.29e-13. Las estimaciones construyeron IC $95\,|10$ dentro de las 2 horas de tiempo-pared de ejecución. Evaluamos las mismas funciones de importancia y valores de división que en el caso transitorio. Los resultados se presentan en la Figura 3.4.

Las simulaciones de Monte Carlo estándar convergieron sólo para las capacidades más pequeñas de las colas. Esto era de esperar dado que, exceptuando a $C = 10$, las capacidades de las colas usadas en este experimento excedían las del análisis transitorio, donde las simulaciones `nosplit` habían fallado para los valores más altos de C.

Prominentemente, todos los experimentos con Monte Carlo estándar o bien fallaron, o sino tardaron más en converger que las corridas que emplearon división por importancia, cualquiera sea el valor de división seleccionado. Esto

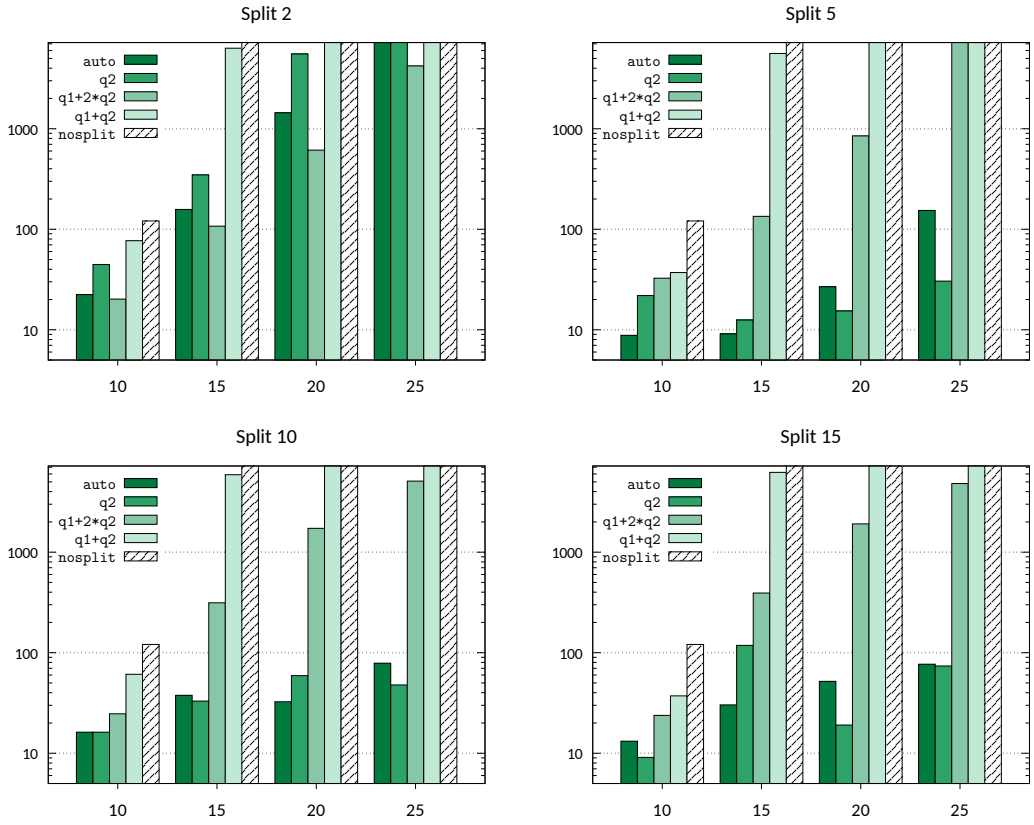


Figura 3.4: Tiempos del análisis estacionario de la cola tándem (modelo CTMC)

sugiere que RESTART puede adecuarse mejor a realizar estudios estacionaros en lugar de transitorios, al menos en este entorno donde hay dos colas conectadas secuencialmente.

Nuevamente la función de importancia `auto` importance function was either the best or the runner up in terms of performance. From the four splitting values tested, it converged the slowest for split 2. Its general behaviour did not vary much among the other splitting values, unlike its closer competitor, es decir `q2`.

Subrayamos que en algunos pocos casos, convergence times decreased as the rarity of the event increased. See e.g. `q2` with splitting 5 for queue capacities $C \in \{10, 15, 20\}$, and also `q2` with splitting 15 for queue capacities $C \in \{15, 20, 25\}$. Studying the technical output of BLUEMOON reveals the reason may be the automatic selection of thresholds. Basamos esta conjetura en lo siguiente.

Tomemos por ejemplo el desempeño de `q2` for splitting 15, where BLUEMOON selected 5–6 thresholds for $C = 15$, 8–9 thresholds for $C = 20$, and 8–11 thresholds for $C = 25$. Since the number of thresholds is almost the same for $C \in \{20, 25\}$, the convergence times are mostly influenced by the rarity of the event, viz. the size of the queue. This results in longer convergence times for $C = 25$ than for $C = 20$, como muesta la Figura 3.4 (y como se esperaba).

Para $C = 15$ empero, la convergencia tardó más que para $C \in \{20, 25\}$. A su vez, las mediciones son consistentes en los tres experimentos que corrimos para esta configuración, que usaron diferentes semillas del generador de números

pseudo-aleatorios. Esto está en conflicto con el comportamiento esperado.

La explicación más plausible parece tener sus bases en el número de umbrales: el algoritmo chose too few of them for $C = 15$, hence the full gain derived from the use of splitting could not be achieved, and the performance of the splitting simulations fue incluso peor que para $C = 25$.

Esta teoría también está respaldada por los experiments which did behave as expected, that is, where convergence times increased together with the value of C. The conjecture is that in these cases, an increment in the value of C should be reflected in an increment in the number of thresholds, particularmente con más de seis umbrales para $C = 15$.

Tomemos por ejemplo el caso de la función de importancia `auto` for splitting 2. The number of thresholds automatically selected for $C = 10, 15, 20, 25$ was respectively 3, 8, 13, and 18, and these numbers were consistent in all (three) experiments run for each configuration. Something analogous is observed for splitting 10 with the importance function `q1+2*q2`, where the number of thresholds automatically selected were 2-3, 7, 9-10, and 13, respectively for the values of $C = 10, 15, 20, 25$. This is thus more evidence in favour of the conjecture that the unexpected higher times for smaller values of C podrían estar causados por una mala selección de umbrales.

Este tipo de anomalías sugieren una implementación ineficiente de RESTART, derivada de un mecanismo subóptimo de selección de umbrales. La situación es similar (y se asume relacionada) a la variabilidad en desempeño debido al valor de división escogido, que ya fue observado en el anterior estudio transitorio. Discutimos posibles soluciones a este problema en la Subsección 3.5.3.

3.5.3. Cola tándem de tiempo discreto

También estudiamos la cola tandem en un entorno de tiempo discreto. Recuérdese que en la cola individual presentada en el Ejemplo 4 de la Subsección 3.3.1 había *ticks de tiempo* que marcaban puntos discretos de evolución temporal. Aquí seguimos la misma estrategia, en un escenario donde múltiples eventos pueden tener lugar en el mismo tick (e.g. un arribo externo de paquete y un servicio de paquete en la segunda cola).

Queremos estudiar un sistema donde la primera cola is the bottleneck. The rarity of the saturated state in the second queue, `q2=c`, is due to its fast service times con respecto a la primera cola.

Como el modelo es discreto, the rates from the continuous scenario need to be replaced with probabilities, defining the odds of an event taking place at each time tick. Let thus `parr` denote the probability per time tick of an external packet arrival, and `ps1` and `ps2` the probability per time tick of a packet service in the first and second queues respectively. We used the PRISM model from Apéndice A.2. Notice that just like in the continuous case, implementamos la cola como un único módulo en lugar de hacerlo composicionalmente[†].

[†] Para un DTMC en PRISM esto implica declarar *todos* los eventos que pueden ocurrir en un tick; es por ello que el modelo discreto casi triplica en longitud a su contraparte continua.

Hicimos un análisis estacionario para las probabilidades `parr` $= 0.1$, `ps1` $= 0.14$, and `ps2` $= 0.19$ and maximum capacities $C \in \{10, 15, 20, 25\}$. The corresponding long run saturation values (γ) approximated by PRISM are 4.94e-7, 1.28e-8, 3.22e-10, and 7.96e-12. Estimations had to build a IC $90\backslash 10$ dentro de las cuatro horas de ejecución pared.

Las funciones de importancia son similares a las que evaluamos en el caso de tiempo continuo. Concretamente, además de `auto` se emplearon las funciones ad hoc `q2`, `q1+q2`, y `q1+5*q2`. Comparamos los valores globales de división 2, 5, 10, y 15. La Figura 3.5 muestra los promedios de los tiempos-pared medidos.

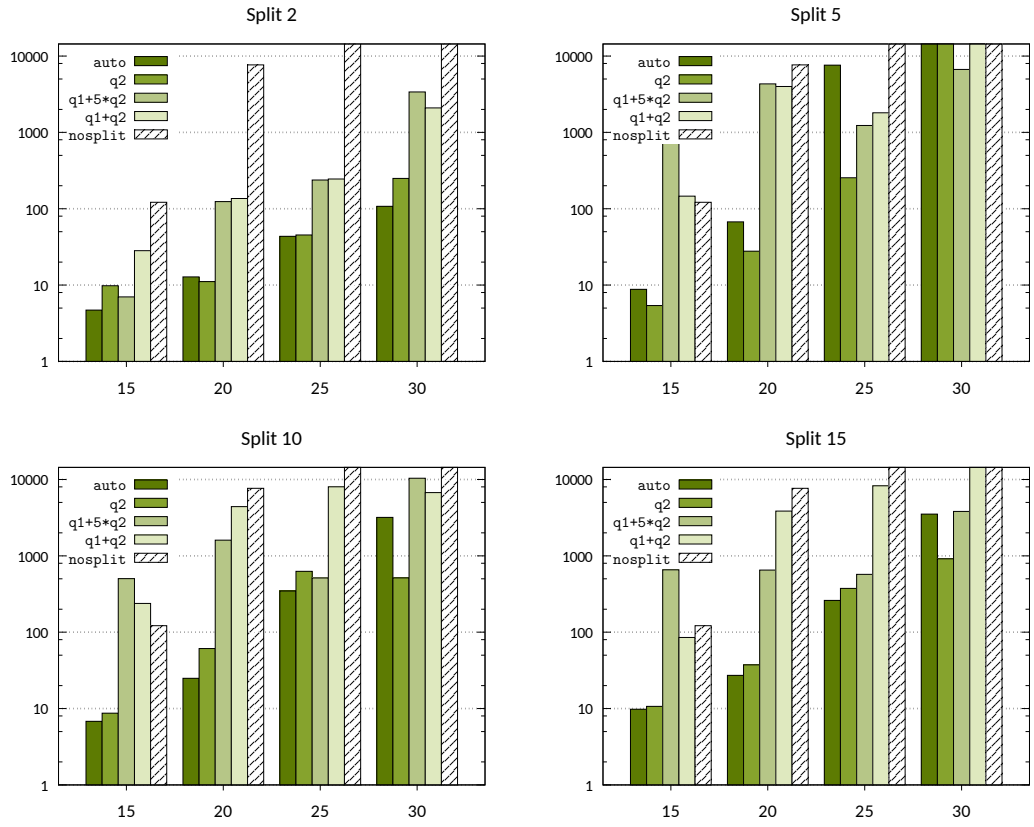


Figura 3.5: Tiempos del análisis estacionario de la cola tándem (modelo DTMC)

A diferencia del caso con tiempo continuo, los valores de división más pequeños produjeron los menores tiempos de convergencia. Es notable el hecho de que la función de importancia `auto` fue la más veloz en converger en esos casos, para $C \in \{15, 25, 30\}$. A su vez, dejando de lado el valor de división 5, podemos observar un excelente desempeño general de `auto`. La función que resultó en segundo lugar es `q2`, al igual que ocurrió con los experimentos en el modelo CTMC de la cola tándem

Los tiempos de ejecución pared resultantes for splitting value 5 deserve special attention. For the values $C \in \{15, 20, 25\}$, `auto` took considerably (and inconsistently) longer than `q2`. We studied each individual experiment in further depth, y sacamos las siguientes conclusiones.

En el caso de $C = 15$, dos experimentos de `auto` tardaron menos de 5 segundos y uno tadó 16 sengundos, whereas all experiments of `q2` took around 5 seconds. Since the number of thresholds did not vary much we attribute this to a bad seed of the random number generator. The influence of such incident is exacerbated by the small computation times, resulting in a high relative variance. The same experiments were repeated with different seeds of the random number generator, and the convergence times observed were very similar between `auto` and `q2`, lo que respalda esta explicación.

Para $C \in \{20, 25\}$ la situación es bastante difernte. For these cases we observed that *fewer thresholds* tend to yield *faster convergence times*, contrary to the overall observations in the continuous time case. We also witnessed this behaviour in the outcomes of experimentation with a splitting value of 10, pero no así en los experimentos con división 2.

Una explicación posible es que la teoría used for the selection of the thresholds, which is based on a *global* (unique) splitting value, is inadequate for purely discrete systems. Already for the CTMC model of the tandem queue, there is evidence that the implementation in BLUEMOON of the thresholds selection mechanism is not optimal. In a DTMC model not only the state space but also the transitions of the system are discretised in time. In this setting, tiny variations in the splitting or the thresholds may have a snowball effect in RESTART, causing starvation or overhead in the upper importance levels, degradando así la eficiencia del método.

Una solución para este problema podría ser to increase the effort spent in selecting and probing the thresholds. In that respect, Algoritmo 2 implements the technique Adaptive Multilevel Splitting, for which an improved version is known. This newer version is named Sequential Monte Carlo [CDMFG12], and it enhances the statistical properties of the algorithm, reducing the variance of the outcome. An alternative way to tackle with the issue would be to reduce the snowball effect derived from having a single global splitting value, eligiendo en su lugar la división a realizar en cada umbral de forma individual.

Un último comentario: los tiempos promedio de ejecución de `q1+5*q2` para división 5 y $C = 15$ se hallan ocultos por la leyenda del gráfico. En esa configuración la función tardó 6535,5 s en converger. Esto es casi lo mismo que le tomó para el mismo valor de división pero con $C = 30$, donde convergió en 6674,7 s.

3.5.4. Cola abierta/cerrada

Considérese una red que consiste en dos colas *paralelas* atendidas por un mismo servidor. Una *cola abierta*, q_o, recibe paquetes a tasa λ desde una fuente externa. Una *cola cerrada*, q_c, recibe (envía) paquetes de (a) un buffer interno del sistema. El mismo servidor procesa los paquetes de ambas colas, dándole prioridad a la cola cerrada. Es decir, los paquetes de q_o son atendidos a tasa $\mu_{1,1}$, a menos que q_c no esté vacía, en cuyo caso será ésta atendida primero por el servidor a tasa $\mu_{1,2}$. A su vez, los paquetes que se hallen en *circulación interna* son procesados en el buffer interno del sistema a tasa μ_2, tras lo cual son enviados de vuelta a q_c. Cuando hay un único paquete en circulación interna, la red es en

realidad una cola $M/M/1$ donde el servidor está sujeto a fallas y reparaciones. Una representación esquemática de este sistema se ofrece en la Figura 3.6.

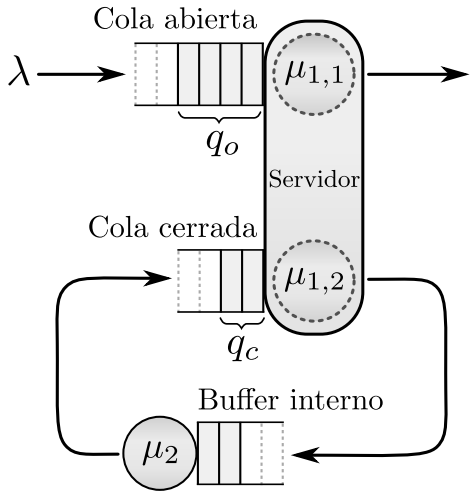


Figura 3.6: Cola abierta/cerrada

Este proceso fue estudiado en [GHSZ99, Sec. 4.1] en un entorno de tiempo continuo. Comenzando desde un estado vacío $(q_o, q_c) = (0, 0)$, se realizó un análisis transitorio para estimar la probabilidad de que q_o alcance una capacidad máxima B, antes de que ambas colas se vacíen nuevamente. La configuración estudiada en ese trabajo tiene un único paquete en circulación interna, usa las tasas $\lambda = 1{,}0$, $\mu_{1,1} = 4$, $\mu_{1,2} = 2$, $\mu_2 \in \{0{,}5, 1{,}0\}$, y las capacidades $B \in \{20, 40\}$.

Usamos el modelo PRISM del Apéndice A.3. The implementation is modular, although the variables representing queue occupancy have global scope. With that model, we used the property `P=? [ !reset U lost ]` para consultar la probabilidad transitoria buscada.

Analizamos el comportamiento transitorio para este sistema in the setting of [GHSZ99], for maximum capacities $B \in \{20, 30, 40, 50\}$ of q_o. The corresponding probabilities of the rare event approximated by PRISM for rate $\mu_2 = 1{,}0$ are 5.96e-7, 5.82e-10, 5.68e-13, and 5.55e-16. Instead for $\mu_2 = 0{,}5$ they are respectively 3.91e-8, 8.89e-12, 2.02e-16, and 4.60e-19. Estimations were set to build a IC $95\,|\,10$ dentro de las 2.5 horas de ejecución pared.

Las simulaciones RESTART incluyeron a la función de importancia `auto` and three ad hoc variants: counting solely the packets in the open queue (`oq`), adding information of whether the packet in internal circulation is currently in q_c (`cq+oq`), and a weighed version of that last variant (`cq+5*oq`). Corrimos experimentos para los valores globales de división 2, 5, 10, y 15.

Los tiempos de ejecución resultantes para un internal server with rate $\mu_2 = 1{,}0$ are presented in Figura 3.7. Standard Monte Carlo simulation failed for all but the smallest queue size, $B = 20$. This comes as no surprise given the values of γ approximated by PRISM for $B \in \{30, 40, 50\}$. Still, these results provide further evidence that (a reasonable implementation of) importance splitting is a good choice when analysing the properties of a model en un régimen de evento raro.

Sorprendentemente, casi todas las simulaciones que usaron división tardaron menos de un minuto para converger, yielding confidence intervals which contained the values of γ approximated by PRISM. En cambio, la única simulación `nosplit` que logró converger tardó casi 40 minutos, y esto sólo en el entorno menos demandante.

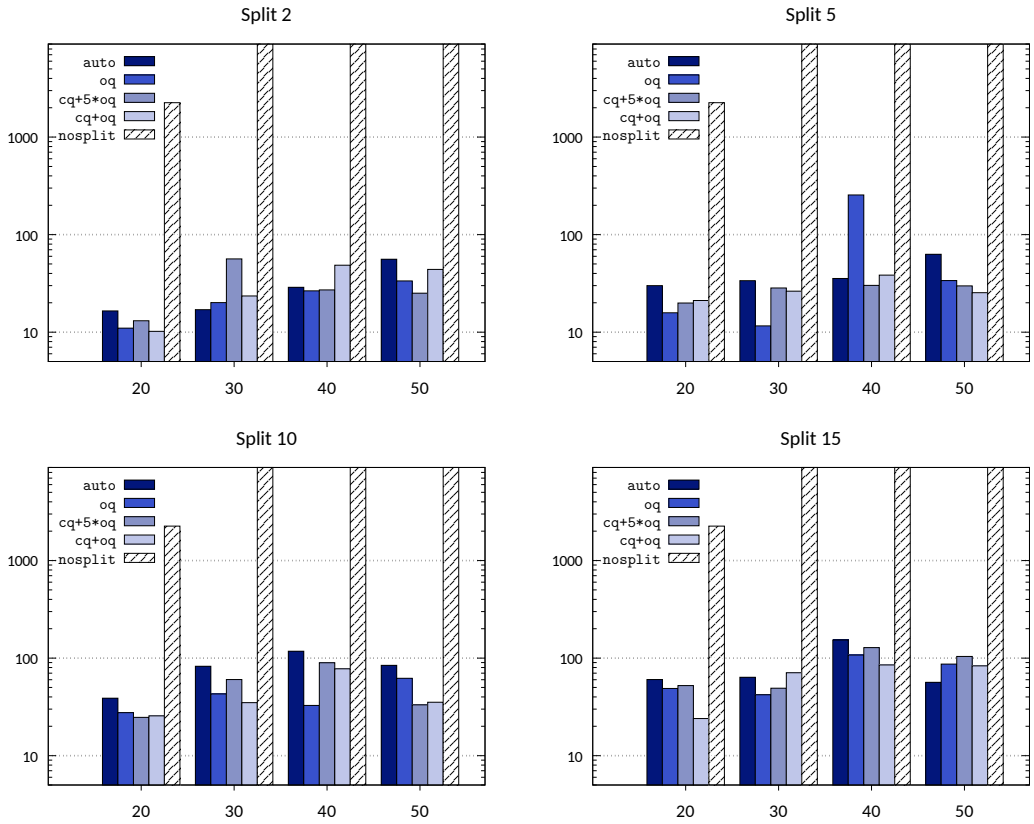


Figura 3.7: Tiempos del análisis transitorio de la cola abierta/cerrada ($\mu_2 = 1{,}0$)

Estudiando el desempeño de las diferentes funciones de importancia, this is the first situation where `auto` is not among the favourites. It behaved quite well for $B \in \{30, 40\}$ with splitting 2, which was the splitting yielding the best performance in most configurations. In general however it was slower to converge than the ad hoc variants, though together with `cq+oq` it showed smaller sensitivity al valor de división que las otras dos funciones.

Esta es también la primer situación donde la more complex ad hoc importance functions rivalled the plain count of packets in the target queue, i.e. `q2`. The best average convergence times where: 10.2 seconds for $B = 20$, lograda por `cq+oq` with splitting 2; 11.6 seconds for $B = 30$, lograda por `oq` with splitting 5; 26.5 seconds for $B = 40$, lograda por `oq` with splitting 2; and 25.1 seconds for $B = 50$, lograda por `cq+5*oq` con división 2.

Los tiempos promedio de convergencia incongruentemente largos de `oq` for $B = 40$ with splitting 5 deserves special attention. When we looked at the individual experiments the cause became evident: the third experiment took 12

minutes, whereas the other two converged in 33 and 12 seconds. Una vez más, la anomalía puede explicarse al mirar los umbrales. El valor extremo usó 18 umbrales, y los otros dos experimentos usaron 13 y 14 umbrales respectivamente.

Este caso (`oq` para el valor de división 5 y $B = 40$) es un buen ejemplo de la dicotomía de inanición/sobrecarga, que afecta a RESTART cuando los umbrales y la división no son seleccionados apropiadamente. Con 13 umbrales la estimaciones convergieron en 33 segundos. Para la misma configuración pero con 14 umbrales las cosas mejoraron, requiriendo sólo 12 segundos para alcanzar los criterios de confianza. Esto sugiere que 13 umbrales genera muy poca división, causando que algunas simulaciones no lleguen hasta el evento raro. Pero luego con 18 umbrales se producen demasiadas divisiones y truncamientos, sobrecargando el número de simulaciones que deben ser manejadas en simultáneo. Estudiando la salida técnica de cada bache, pudimos ver que la varianza de las salidas es demasiado alta, y por ende la convergencia estadística tardó más.

La Figura 3.8 muestra los tiempos de ejecución para la tasa alternativa $\mu_2 = 0{,}5$. Omitimos los resultados de la experimentación con $B = 50$ porque ninguno de ellos convergió dentro del límite de 2 horas y media. Recordemos que PRISM computó el valor de probabilidad 4.6e-19 de observar el evento transitorio. Como esto se encuentra tres órdenes de magnitud por debajo del caso de $B = 50$ cuando $\mu_2 = 1{,}0$, estas fallas no son de sorprender. Ni tampoco resulta inesperado el hecho de que todas las simulaciones de Monte Carlo hayan fallado.

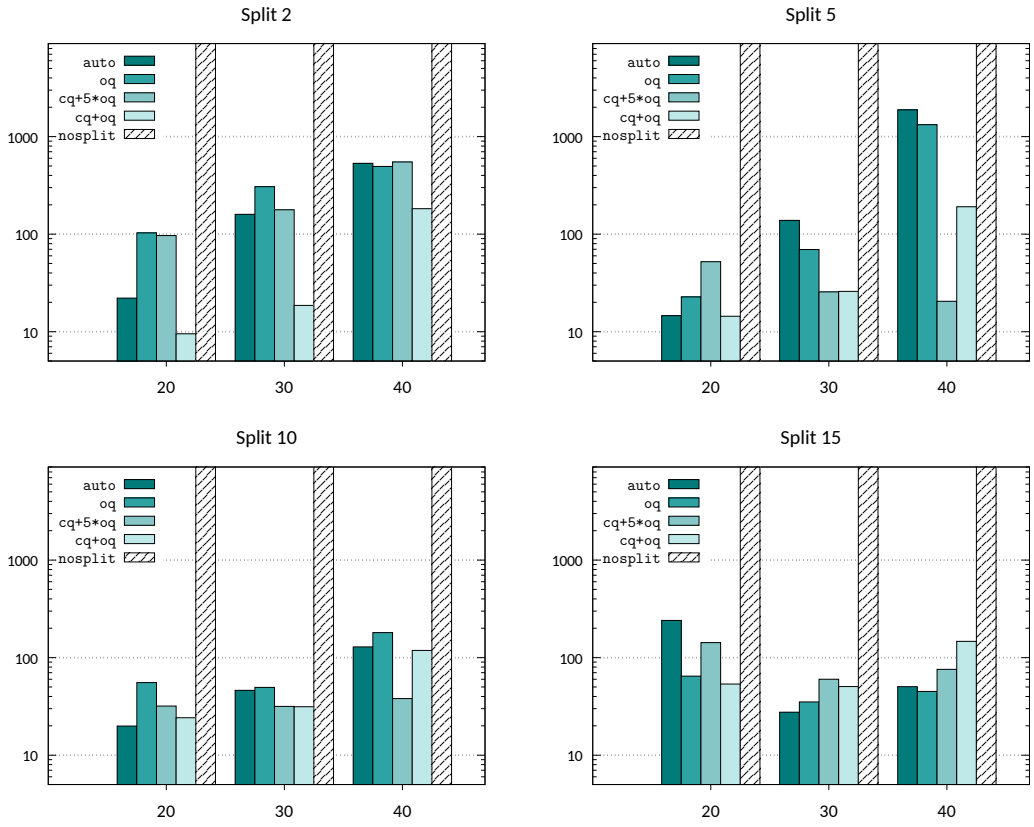


Figura 3.8: Tiempos del análisis transitorio de la cola abierta/cerrada ($\mu_2 = 0{,}5$)

Para interpretar los resultados que mostramos en la Figura 3.8, es crucial comprender qué se modifica al cambiar el valor de μ_2. Recordemos que μ_2 es la tasa con la cual un paquete en circulación interna es enviado a la cola cerrada (q_c) desde el buffer interno. Además recordemos que el servidor que atiende a la cola abierta (q_o) no desencolará paquetes de q_o mientras q_c tenga algo en su interior. Esto era análogo a tener un servidor que se rompe atendiendo a la cola q_o. Por consiguiente un menor valor de μ_2 implica *mayores períodos* del servidor *desencolando paquetes de* q_o, lo cual resulta en menores probabilidades de observar el evento raro de una cola abierta saturada.

A medida que μ_2 disminuye, having a packet in q_c (viz. a broken server) becomes less common, and the state of q_c *grows in importance*. That is likely the reason why, for most splitting values, the importance functions `cq+oq` and `cq+5*oq` converged faster than `oq`. Contrasting against the previous setting where $\mu_2 = 1{,}0$, the performance of the two functions which consider q_c mejoró con respecto `oq`, aquí donde $\mu_2 = 0{,}5$.

Evitar incurrir en este tipo de análisis is precisely one of the goals behind the automatic derivation of the importance function. Notice that the `auto` importance function converged faster than `oq` in many situations, and even better than the best candidates (`cq+oq` and `cq+5*oq`) in a few cases, e.g. $B \in \{30, 40\}$ with splitting 15 and $B = 20$ with splittings 5 and 10. That is why, even though `auto` was not the best performing importance function, de todas formas consideramos que estos resultados son satisfactorios.

La alta variabilidad entre los distintos valores de división probados merece las mismas consideraciones que en los casos de estudio previos. Finalmente, observamos que la convergencia más lenta de `auto` para la división 15 con $B = 20$ en comparación con $B \in \{30, 40\}$, fue causada por uno de los tres experimentos. Dos de ellos convergieron en menos de 2 minutos, mientras que el tercer experimento tardó 10 minutos. Este problema, ya observado para la representación DTMC de la cola tándem, puede mitigarse incrementando el número de experimentos repetidos por cada configuración. Por desgracia, los recursos computacionales disponibles no lo permiten.

3.5.5. Sistema de colas con rupturas

Recuérdese el sistema que presentamos en el Ejemplo 1, donde varias fuentes (divididas en dos tipos) envían paquetes a un único buffer, y todas ellas pueden dejar de estar operativas y luego ser reparadas. El único servidor que atiende al buffer también puede romperse y repararse, por lo que este sistema puede ser visto como una generalización de la cola abierta/cerrada de la subsección previa.

Kroese et al. estudiaron estos procesos [KN99] using importance sampling, in a continuous time setting with five sources of type 1 and another five of type 2. The rates used in [KN99, Sec. 4.4] are: $(\alpha_1, \beta_1, \lambda_1) = (3, 2, 3)$ for sources of type 1, describing the speeds of repair, failure, and packet production respectively; $(\alpha_2, \beta_2, \lambda_2) = (1, 4, 6)$ for sources of type 2; and rates $(\delta, \xi, \mu) = (3, 4, 100)$ for the server, donde μ representa el procesamiento de un paquete en lugar de su producción.

Usamos el modelo PRISM del Apéndice A.4. La implementación es monolítica, taking advantage of the Markovian properties to reduce the state space[†]. Starting with a single packet enqueued in the buffer, a broken server, and a single operational source of type 2, the rare event is a buffer saturation for some maximum capacity K before it empties. The corresponding transient property is `P=? [ buf>0 U buf=k ]`, donde `k` es K, la máxima cpacidad del buffer por alcanzar.

Realizamos un análisis transitorio para las capacidades máximas $K \in \{40, 80, 120, 160, 200\}$. The corresponding values of γ obtained with PRISM are 4.59e-4, 3.72e-7, 3.02e-10, 2.45e-13, and 1.98e-16. We used a IC $95\backslash 10$ criterion together with a wall time execution limit of 3 hours. Like in the previous case studies, the splitting values tested en las simulaciones de división por importancia fueron 2, 5, 10, y 15.

Además de la función `auto` construida por BLUEMOON, we tested three ad hoc importance functions. The simplest one, `buf`, just counts the number of packets in the buffer. That seems too naïve, since the up/down state of the sources is crucial to generate the desired saturation. Therefore, another variant also counts the number of sources which are up, using weights related to their production rate to discriminate between the two source types. Such function is denoted `buf+nSu`, which stands for "buffer occupancy plus number of sources up". The specific expression we used to compute the importance is `buf + src1*lambda1 + src2*lambda2`. We also implemented a third strategy, counting the number of sources which are "down". This last function is denoted `buf+nSd`, y la expresión que usamos para medir la importancia es `buf + NSrc1 + NSrc2 - src1 - src2`.

La elección de estas funciones de importancia ad hoc merece una breve reseña antes de que presentemos los tiempos promedio de convergencia que generaron. Including the state of the sources when computing the importance of a system state sounds sensible: the more sources producing packets, the faster a full occupancy should be observed. That suggests that the more operational sources it has, the more important a state should be deemed, which points at `buf+nSu` as a good importance function alternative. Why then try `buf+nSd`, which uses an opposite heuristic? The answer is a stark "why not? we may just try". This queueing system is the most complex introduced so far. The mixed open/closed queue from the previous subsección is simpler, yet it showed a behaviour not so easy to foresee, when changing the value of a single parameter. Perhaps some hidden subtleties in the interrelationship among components makes the heuristic behind `buf+nSu` derivar en una mala implementación de RESTART.

La Figura 3.9 presenta los resultados de la experimentación. Standard Monte Carlo simulations converged in time only for the smallest buffer size, $K = 40$. However, in doing so they outperformed all importance splitting simulations. This is not so surprising since $\gamma > 4.5\text{e-}4$ for that buffer size, comprising the *less rare event* studied so far. Moreover, in half of the cases only one threshold was selected by BLUEMOON, and in most of the other half of the cases, se eligieron

[†] En lugar de las fuentes individuales usamos contadores `src1` y `src2` de rango 6, puesto que N fuentes activas de tipo i significan una tasa de entrada al buffer igual a $N\lambda_i$. Así el espacio de estados crece aritméticamente con el número de fuentes.

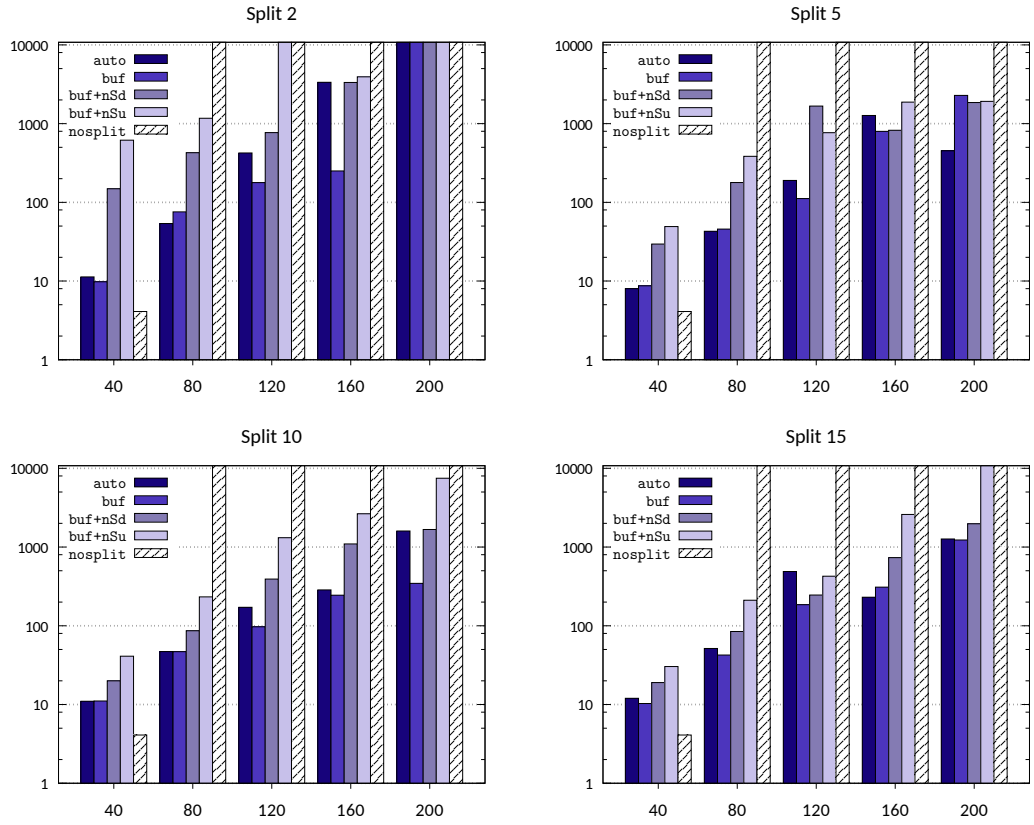


Figura 3.9: Tiempos del análisis transitorio de la cola con rupturas

dos umbrales. Comparemos ésto con los 13–18 umbrales que discutíamos en la Subsección 3.5.4, y los 3–18 umbrales discutidos en la Subsección 3.5.2. Recall the gain derived from using RESTART is exploited when the splitting is performed at the thresholds. When the event is not so rare, Algoritmo 2 selects few importance values as thresholds, and the overhead of splitting (e.g. computing the importance of a state at each simulation step) belittles its gain. We believe that is the most likely reason por la que las simulaciones `nosplit` superaron a las que usaban RESTART para $K = 40$.

Estudiando el desempeño de las diferentes funciones de importancia, puede verse que `auto` y `buf` fueron las más rápidas en converger en casi todas las configuraciones. Esto desafía las especulaciones previas acerca de la importancia real de un estado del sistema, y cómo podría ser afectado por el estado operativo/roto de las fuentes.

Es sin embargo poco claro el si los estados de las sources should be left out of importance computation altogether. Maybe their influence must be merely scaled down by some *unknown factor*. In other words, perhaps the value of `buf` debería ser escalado en las expresiones de `buf+nSu` y `buf+nSd`.

Basamos esta hipótesis en las tasas de producción de falla/reparación of the sources, which are almost two orders of magnitude lower than the service rate $\mu = 100$. This means packets are quickly dequeued from the buffer, luego el hecho

de ver a muchos de ellos encolados es poco probable, y en particular puede ser mucho más importante que tener una fuente activa más o menos.

Asimismo, el número de fuentes operativas is *scaled up* by the production rate λ_i in the expression of `buf+nSu`. For instance, 20 packets in `buf` and 3 sources up (of type 2) is deemed as important as 14 packets in `buf` and 4 sources up. This could be yielding a computed importance transverse to the real importance, con el incremento en la varianza que esto implica.

Para ilustrar esta última acotación y su incluencia in the convergence times, consider a state where `buf=k-1` and one source of type 2 is broken. Suppose the importance function `buf+nSu` is used. Suppose also that the current system importance is i, and that the importance values $i+1$ and $i+6$ were selected as thresholds. Then for splitting value $k \in \mathbb{N}_{>1}$, a simulation which repairs the source and then enqueues a packet, produces k^2 more rare events than one that just enqueues a packet. These kind of scenarios augment the variability between the outcomes of the RESTART runs, aumentando los tiempos de cómputo para alcanzar la convergencia.

Por último en esta línea de análisis, we draw attention to `buf+nSd`, which performed better than `buf+nSu` in all but one of the configurations tested. This in spite of the seemingly unreasonable heuristic behind `buf+nSd`, emphasising once again the difficulties of choosing a good function. Anyhow and in correspondence with the goals of the thesis, all conjecturing and reviewing can be dropped when an algorithm para derivar una función de importancia está a nuestra disposición.

En particular, subrayamos la buena calidad de la función de importancia `auto` a través de un ranking de funciones para las distintas capacidades del buffer. De los tiempos promedio para convergencia presentados en la Figura 3.9, las funciones de importancia que se comportaron mejor para for $K = 40, 80, \ldots, 200$ fueron respectivamente `auto` con división 5 (8.0 s); `buf` con división 15 (42.6 s); `buf` con división 10 (97.1 s); `auto` con división 15 (230.5 s); y `buf` con división 10 (346.4 s). Además, en los tres casos donde `buf` fue la ganadora, `auto` salió en claro segundo lugar, y aveces por muy poco. Por ejemplo le tomó 42,9 s converger con división 5 para $K = 80$, i.e. 30 ms (0,7 %) más que a `buf`.

3.6. Limitaciones de la estrategia monolítica

Este capítulo presentó una estrategia automatizable para realizar análisis de modelos mediante simulación, utilizando técnicas de división multinivel para mejorar la velocidad de convergencia de los mecanismos de estimación. Las únicas entradas requeridas son el modelo del sistema, la consulta de propiedad especificando qué análisis transitorio o estacionario realizar, y un criterio de confianza o un presupuesto de ejecución a satisfacer (o ambos). Es decir, requiere las mismas entradas que las estrategias estándar de análisis por simulación de Monte Carlo.

La sección previa da empirical validation of the relatively good efficiency that can be obtained using this approach. In all the systems and for a vast majority of their configurations, la estrategia automática descripta en la Sección 3.3 resultó

en una implementación de la división multinivel que sobrepasó con creces a la táctica de Monte Carlo estándar. As desired, this difference in performance was exacerbated by the rarity of the event: the smallest the probability to estimate, the better RESTART behaved con respecto a las simulaciones que no usaron la técnica de división.

Además, la función de importancia automáticamente derivada por BLUEMOON usando el Algoritmo 1 rivalled the best ad hoc alternatives tested, most of them suggested as sensible or optimal choices by the authors of the articles de los que se extrajeron estos sistemas.

Por desgracia, este éxito no es general.BLUEMOON was designed on top of a model checker which, for the concerns of this thesis, studies Markov chains only. This is however no theoretical bound, since both algorithms and also the automatic technique as a whole is oblivious of the memoryless property. Rather, the restriction limiting the applicability of the monolithic approach stems from a more practical concern: it is the same state space explosion problem que hace temblar los cimientos del model checking.

This is not directly related to the use of importance splitting. RESTART simulations require, at most, the execution history that led to each state. The issue comes from the technique used to automatically build the importance function. Algoritmo 1 needs an explicit representation of the full state space of the model, to store the importance values computed. Even worse, it also needs to traverse the transition matrix, whose likely sparsity helps little.

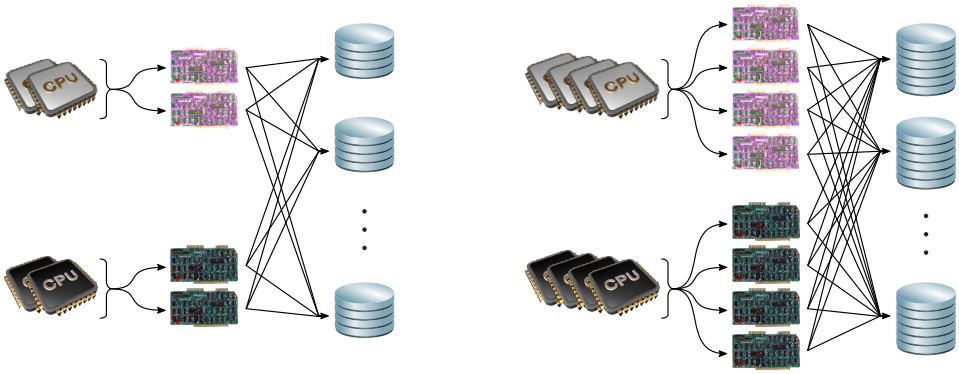
Es fácil apreciar la gravedad de estos requerimientos bajo la luz de alguna aplicación proveniente de la vida real, que podría necesitar la definición de cientos de módulos. Esto vuelve inviable cualquier representación explícita del espacio de estados, y más aún de la matriz de transiciones. Sin embargo no es necesario recurrir al mundo real para darse contra los límites de la estrategia monolítica. El modelo simplificado de una base de datos que presentamos en el siguiente ejemplo es prueba de ello.

Ejemplo 7: Sistema de base de datos con redundancia.

Considérese una instalación de base de datos que consiste en discos organizados en clusters, controladores de discos, y procesadores. Para redundancia R el sistema está formado por dos tipos de *procesadores* (con R copias de cada tipo), dos tipos de *controladores* de disco (con R copias de cada tipo), y seis clusters de *discos* (con $R+2$ discos cada uno). La Figura 3.10 muestra una representación esquemática para dos valores de redundancia: Figura 3.10 (a) muestra un sistema donde $R=2$ y Figura 3.10 (b) uno donde $R=4$.

El tiempo de vida de las unidades está exponencialmente distribuido según tasas de fallo μ_D, μ_C, y μ_P para los discos, controladores, y procesadores respectivamente. Cuando un procesador de un tipo falla, causa una falla en un procesador del otro tipo con cierta probabilidad. Además, una unidad puede fallar con igual probabilidad en uno de dos modos, cada uno con su propia tasa de reparación. El sistema se considera operativo mientras menos

de R procesadores de cada tipo, R controladores de cada tipo, y R discos en cada cluster, hayan fallado.



(a) Base de datos con $R = 2$ (b) Base de datos con $R = 4$

Figura 3.10: Instalación de base de datos con redundancia

Este sistema fue estudiado originalmente en [GSH+92] using importance sampling, and then in [VA98, VA07a, VAVA11] with RESTART using importance functions defined ad hoc. The interest lies in studying the steady-state unreliability of the system, i.e. where γ reflects the proportion of time the database is not operational. We focus on the setting used in the RESTART articles, where $(\mu_D, \mu_C, \mu_P) = (1/6000, 1/2000, 1/2000)$, the inter-processor failure probability is $1/100$, and failures of type 1 and 2 yield repair rates of 1 and 0.5 components por unidad de tiempo respectivamente.

En [VA98] el autor estudia redundancias mayores than two, namely $R \in \{2, 3, 4\}$, and observes how multilevel splitting performs better for the higher values of R. As discussed in [BDH15] and later in [BDM17], such observations are reasonable since the underlying adjacency graph is highly connected, and R steps can make a system with no failed units to become non-operational. Thus small values of R provide a meager setting for the splitting strategy, which relies on an efficient layering of the state space, stacking up between the initial state and the rare set. In short: the higher the redundancy, the better RESTART (y la división multinivel en general) debería comportarse.

Consideremos ahora el modelo para este sistema presentado en el Apéndice A.5. Even though the description is modular, the variables forming the state space are global. Also, the same Markovian trick used in the queueing system from Apéndice A.4 is employed, grouping the state of potentially individual modules en lugar de implementarlos por separado.

A pesar de estos trucos, the resulting model has four variables of range $R + 1$ and six of range $R + 3$, plus the boolean variable `f` in module `Repairman` to distinguish between the two failure types. So the total number of states is $2\,(R+1)^4\,(R+3)^6$. For redundancy two, the lowest one considered, this adds up to 2531250 states in a system con una matriz de transición densa, para la

cual PRISM reporta 57825000 transiciones.

Esto generará problemas casi inmediatamente relacionados con la disponibilidad de memoria física, donde el punto exacto de inflexión depende de la memoria total disponible y del tipo de datos usado internamente. Con la versión de desarrollo 4.3 de PRISM y 8 GiB de RAM a disposición, el model checker lanza una excepción std::bad_alloc para $R = 4$, que habría contado con $2 \cdot 5^4 \cdot 7^6 = 147061250$ estados y 3774852200 transiciones según se reportó en la salida de la herramienta. □

El Ejemplo 7 ilustra una limitación práctica de la estrategia monolítica presentada en este capítulo. El ejemplo también sugiere otro problema, más sutil y menos crítico en el sentido de que no impide la aplicación general del método, pero aún así con un claro impacto negativo en su eficiencia.

Este problema, de una mayor sutileza, is rooted in the high connectivity of the adjacency graph inherent to the database system. The efficiency of multilevel splitting increases as more levels are placed between the initial system state and the set of rare states. When few transitions can take a simulation path from any state to any other state, una división rica en niveles no es factible.

Para redundancia R, la base de datos puede moverse from the initial (fully operational) state to a rare state by taking R transitions. Therefore, the `auto` importance function computed by BLUEMOON yields only $R \in \{2, 3\}$ importance levels in the configurations tested, meaning a maximum of 1–2 thresholds. That seems hardly enough to get the full gain from splitting: recall the discussions for the queue with breakdowns from Subsección 3.5.5, where standard Monte Carlo simulations converged faster than RESTART simulations for the less rare setting. Something similar happens when running BLUEMOON em el modelo del Apéndice A.5 para los valores de redundancia $R \in \{2, 3\}$.

La rareza del evento estudiado en el Ejemplo 7 is based on the very low probability of choosing a few transitions. These scenarios are detrimental for importance splitting, as mentioned at the end of Sección 2.4, y usualmente son manejados mejor por el muestreo por importancia (si es factible).

En ese sentido podría argumentarse that the database system is inherently *flat*, and that R importance levels is the best we can do. Nevertheless that is not entirely true: compare a system with just a single disk failed, against a system where there is one failed component of each type (one disk per cluster, one CPU of each type, one controller of each type). It is much more likely to observe a rare event in the second case, since any further failure of any other component will cause a system failure. Still, the `auto` importance function computed by BLUEMOON deems both cases equally important, puesto que ambos están (¡realmente!) a una transición de distancia del conjunto de estados raros.

El Algoritmo 1 no puede distinguir entre estos casos because it operates on the fully composed model, where the *modular structure* is amalgamated into a unified, highly connected system. A solution would require us to exploit such modular structure, understanding the contribution of each component to the global rare event, y usarlo para construir la función de importancia.

Por último y para conlcuir con este capítulo, volvamos a la preocupación principal que aqueja a la estrategia monolítica. The inability to avoid *the state explosion problem* comes from the assumptions of Algoritmo 1, which builds the automatic function on top of a single module. If a modular formalism like the PRISM input language is used, Algoritmo 1 requires the composition of all the modules of the system, which generates a model whose state space crece exponencialmente con el número de módulos involucrados.

Una estrategia distribuida pareciera ser la mejor solución, where we would apply (some variant of) Algoritmo 1 locally on each module. The difficulty lies in achieving this without loosing track of the global behaviour, since the north of the importance funcion is the *global* rare event. Therefore, when computing a function *local* to each system component, we must consider how does such global rare event reflect on each module. Otherwise we would be unable de elegir la importancia para sun conjunto de estados locales.

Hemos identificado pues dos desafíos: distribuir la estrategia monolítica de este capítulo, honrando el evento raro global mientras se construyen funciones de importancia localmente en cada componente del sistema; y tener en cuenta la estructura modular del sistema, favoreciendo así la división en capas del espacio de estados para incrementar las ganancias del uso de división por importancia.

Esos dos desafíos son el tema principal del siguiente capítulo, donde proponemos estrategias para atacar los problemas, y métodos para automatizar su implementación.

División multinivel: automat. composicional 4

Este capítulo introduce estrategias que adaptan las técnicas automáticas del Capítulo 3 para articularlas con una descripción composicional (o modular) del sistema. Esta es otra contribución principal de la tesis, que involucra descomponer al evento raro global para construir funciones de importancia localmente en cada módulo, y luego combinar esta información para calcular la *importancia global* que las técnicas de división requieren.

Las discusiones comienzan alrededor de la descomposición de la propiedad que describe al evento raro. La meta es proyectar un *evento raro local* dentro de cada módulo, con el cual derivar una *función de importancia local*. Luego, estudiamos diferentes formas de calcular la importancia del modelo gobal del sistema, usando las funciones locales de importancia como material de construcción. Un formalismo recientemente introducido para modelar procesos estocásticos generales (homogéneos y libres de no determinismo) es considerado. Además, una nueva herramienta de software es presentada, la cual implementa las estrategias y algoritmos introducidos, usando el formalismo mencionado como núcleo teórico. Finalmente ejemplificamos la eficiencia de la estrategia mediante casos de estudio.

4.1. El camino hacia la modularidad

La necesidad de representar explícitamente el espacio de estados del modelo compuesto es la limitación más severa de la estrategia monolítica del Capítulo 3. Este requisito compromete la factibilidad de la misma, como se expuso con la instalación de base de datos introducida en el Ejemplo 7.

Para evitar este problema, consideremos la solución ingenua de aplicar la técnica de derivación de función de importancia separadamente a cada componente del sistema. Esto sólo requeriría la representación explícita del espacio de estados de cada *módulo*, pero no de su composición. En particular para el sistema de la base de datos, cada componente puede estar o bien roto o sino operativo. Si consideramos pues un modelo con redundancia $R \geqslant 2$, donde cada componente está implementado como un módulo independiente, la solución simplista que proponemos sólo necesitaría $6(R+2)+2R+2R = 10R+12$ representaciones independientes de espacios binarios de estados. Difícilmente habrá problemas de memoria incluso para e.g. $R = 10$.

Sin embargo las complicaciones ocultas detrás de esta estrategia no pueden ser resueltas de forma tan directa. La "técnica de derivación de función de importancia" a la cual hacer referencia la solución ingenua es precisamente el Algoritmo 1, que

toma dos entradas: el módulo $\mathcal{M}$ y el conjunto de estados raros A. Para aplicar el algoritmo sobre cada módulo individual $\mathcal{M}_i$ que forma parte de un sistema compuesto, debemos identificar antes que nada a los *estados raros locales*, A_i.

Identificar los conjuntos $\{A_i\}$ en todos los módulos puede ser más difícil de lo que suena, al menos en el caso general. Considérese por ejemplo la cola tándem cuando el usuario solicita un análisis estacionario del evento raro

$$q_1 > \frac{C}{2} \quad \Rightarrow \quad q_2 = C$$

donde '$\Rightarrow$' representa la implicación lógica habitual. Estudiando el módulo de la segunda cola, donde la variable local q_2 está definida, no queda claro si los estados concretos asociados a $q_2 < C$ pueden ser considerados raros. Ellos satisfacen la propiedad del evento raro, pero sólo si $q_1 \leqslant {}^{C}/_{2}$. ¿Cuál es pues el conjunto de estados raros locales en el módulo de la segunda cola?

Esto trae a colación el primer desafío en el camino hacia la modularidad:

(a) proyectar el evento raro global sobre el espacio de estados de cada módulo, para guiar la construcción de funciones locales de importancia

Continuando con nuestra argumentación, supongamos que el desafío (a) ha sido resuelto satisfactoriamente. Recordemos ahora que las técnicas de división multinivel trabajan con la importancia de los estados del modelo compuesto, i.e. con la importancia de los *estados globales*. En ese sentido nuestra estrategia ingenua es incompleta, porque devuelve un conjunto de funciones que sólo pueden interpretarse separadamente en los *estados locales* de los módulos. Para cada configuración del sistema global esto resulta en un conjunto de valores de importancia. Dicho conjunto *debe ser combinado* de alguna forma para calcular la importancia global que requieren las técnicas de división.

Persistiendo en nuestra ingenuidad, supongamos que queremos definir esta importancia global como la suma de todas las importancias locales. Consideremos ahora un sistema donde hay contribuciones heterogéneas por parte de los distintos módulos al evento raro global. Ya hemos estudiado uno de estos casos: la cola con rupturas introducida en el Ejemplo 1 con la que se experimentó en la Subsección 3.5.5. Nuestros análisis sugieren que el número total de paquetes encolados en el buffer —digamos, la importancia del módulo que representa al buffer— es mucho más relevante en términos de importancia (global) real que el estado operativo/roto de las fuentes —digamos, la importancia de los módulos que representan a las fuentes—.

Por consiguiente, la solución ingenua de sumar ciegamente la importancia de todos los módulos por igual no es satisfactoria en el caso general. Cuando se combina el conjunto de valores de importancia locales en una importancia global, el grado de contribución de cada módulo al evento raro global es claramente relevante. Todo lo cual genera el segundo desafío el camino hacia la modularidad:

(b) construir una función local de importancia por sobre la información almacenada localmente en cada módulo, considerando el rol de cada módulo en el evento raro.

Los desafíos (a) y (b) representan las principales dificultades teóricas entre la estrategia general del Capítulo 3 y una versión distribuida de la misma. El desafío (a) es analizado en mayor profundidad en la Sección 4.2, donde proponemos una solución algorítmica simple y robusta. El desafío (b) es el tema de estudio de la Sección 4.3.

4.2. Funciones locales de importancia

Como hemos visto, encontrar una manera de descomponer modularmente el método general del Capítulo 3 no puede hacerse en línea recta. El primer desafío que uno encuentra en esta dirección es la identificación del evento raro en el espacio de estados local de cada módulo. En esta sección proponemos una estrategia automatizable para llevar a cabo esta tarea.

4.2.1. Proyectando al evento raro sobre los módulos

Los formalismos modernos de modelado, como el lenguaje de entrada del model checker PRISM, usualmente están basados en una sintaxis de descripción composicional. Esto significa que le permiten al usuario expresar el modelo de su sistema como la composición paralela de un conjunto de *módulos de sistema* de menor tamaño que el modelo compuesto. Cada módulo tiene su comportamiento propio y generalmente puede definir su propio conjunto de *variables locales*, cuyo alcance no trasciende al módulo donde fueron definidas.

A pesar de la localidad de su alcance, the expression of the user query for the (global) rare event can include variable names from any module. Consequently, the property conveyed by such expression has semantics on the global system model, and not on the modules taken individually. In other words, the rare event property is interpreted as a set of estados *globales*, posiblemente describiendo configuraciones simultáneas específicas en varios módulos.

El primer paso hacia una estrategia modular es identificar, in every module taken individually, the set of local states A_i corresponding to the global rare event. The non-trivial question is how to interpret the *global property*, provista por el usuario como una consulta del evento raro, localmente en cada módulo.

Para ilustrar la dificultad fundamental consideremos la red de colas tándem del Ejemplo 2, descripta como un modelo composicional en el Código 3.3. Compararemos las siguientes definiciones del evento raro, para analizar el comportamiento estacionario del modelo:

(a) $q_2 = C$,
(b) $q_1 = C \ \wedge \ q_2 = C$,
(c) $q_1 \geqslant {}^{C}/_{2} \ \Rightarrow \ q_2 = C$.

Las tres definiciones hablan de una saturación en la segunda cola (`Cola2`), pero el rol de la primera cola (`Cola1`) es más difícil de comprender.

La variable q_1 no aparece en la definición (a), so the module of the first queue could be ignored when deriving the local importance functions, since it does not

change the validity of the formula $q_2 = C$. In other words, for definición (a) la función local de importancia de `Cola1` podría ser nula, e.g. darle importancia 0 a todos los estados.

La Definición (b) sí incluye a la variable q_1. More precisely, given the logical expression is a *conjunction* of two terms, the occurrence of q_1 in one of those terms implies it is a key component of the global rare event. This indicates that the local importance function of the module of the first queue will not be null as it happened with definición (a). Moreover, the local rare states in `Cola1` deben ser los que satisfagan $q_1 = C$.

Finalmente, la definición (c) deserves a deeper analysis. That formula is equivalent to $\neg\,(q_1 \geqslant {}^{C}\!/_{2}) \vee q_2 = C$. Therefore, the local rare states in `Queue1` are those which *do not satisfy* $q_1 \geqslant {}^{C}\!/_{2}$. This is at odds with situation (b), where the term containing q_1 was used as it occurs in the definition. Here instead the local rare states están identificados por la *negación* del término donde ocurre q_1.

En general, la dificultad radica en saber whether to take positively or negatively the occurrence of a variable in the rare event. The whole expression can have several levels with nested negations, entangling matters. Hence when analysing a single module, it is generally unclear how to interpret the subformulas where its variables appear. For instance, in the previous example with the module of the first queue, the occurrence of q_1 in a comparison must be taken positicamente para la definición (b) del evento raro, y negativamente para la definición (c).

Revisando el análisis aplicado en el caso (c), notemos que la conclusión de usar $\neg\,(q_1 \geqslant {}^{C}\!/_{2})$ fue alcanzada por medio de la equivalencia

$$q_1 \geqslant \frac{C}{2} \;\Rightarrow\; q_2 = C \quad\equiv\quad \neg\left(q_1 \geqslant \frac{C}{2}\right) \vee\; q_2 = C\,.$$

Específicamente, el lado *más simple* de la equivalencia, viz. donde sólo se utilizan los operadores lógicos de disyunción y negación, establece con mayor claridad cómo debe interpretarse la expresión lógica a la hora de identificar los estados locales.

De modo similiar, considérese la expresión de evento raro $\neg\,(q_1 \neq C \vee q_2 < C)$. Podría ser más apropiado emplear en su lugar la expresión equivalente $q_1 = C \wedge q_2 \geqslant C$ cuando estamos interpretando cuales estados deberían identificarse como raros.

La idea a grandes rasfos es transformar the rare event formula into an equivalent expression in some normal form, where all nesting has been solved and there are no logical operators of implication nor equivalence. En ese sentido la *forma norma disyuntiva* (DNF por sus siglas en inglés) es un buen candidato. A DNF formula is a disjunction of *clauses*, each of which is a conjunction of *literals*. A literal is an atomic proposition or the negation of one, i.e. a boolean variable or the comparison of numeric expressions involving numbers and variables.

Usar fórmulas en DNF tiene varias ventajas:

- es un estándar de representación de fórmulas, con las implicaciones habituales que esto conlleva, e.g. se puede asumir al lector familiarizado;

- toda fórmula proposicional puede ser expresada equivalentemente en DNF, por lo que no hay restricciones sobre qué puede analizarse;
- el evento raro puede ser identificado mediante la satisfacción de *cualquier* cláusula en la expresión;
- no hay negaciones anidadas y por ende no hay ambigüedad a la hora de interpretar los literales, dado que cada uno declara con claridad cómo (positivamente o negativamente) contribuye al evento raro.

De los tres ejemplos expuestos, los primeros dos ya se encuentran en DNF: la definición (a) is a single literal, and (b) is a single clause composed of two literals. Instead, definición (c) is not in DNF; an equivalent DNF formula is $\neg (q_1 \geqslant {}^{C}/_{2}) \vee q_2 = C$, which is composed of two clauses, cada una con un único literal.

La principal ventaja de tratar con formulae expressed in DNF is that the literals have all the information regarding how to interpret the occurrence of a variable. Therefore a simple projection of the formula onto the namespace of each module should provide a correct and unambiguous identificación de los estados raros locales.

Los literales de una fórmula en DNF can be regarded as its building blocks. In that respect they are considered indivisible during a projection. This means e.g. projecting $q_2 > C$ onto the scope of `Cola1` yields an empty expression, for even though the constant C (e.g. the identifier `c`) has global scope and is thus known by `Cola1`, the variable q_2 (e.g. the identifier `q2`) existe sólo dentro de `Cola2`.

Continuando con el ejemplo previo de la cola tándem, consider a projection of definición (a), which is already in DNF, onto the namespace of the module of the first queue. Since `q2` is outside the scope of `Cola1`, the projection will yield an empty expression as previously described. This suggests there is little or no information regarding the rare event in such module, which could thus be safely omitted from importance computation. Contrarily, the same projection onto the namespace of `Cola2` yields the expression $q_2 = C$ (e.g. `q2=c`). This will correctly identify the concrete states corresponding to una segunda cola saturada como los estados raros locales.

En el caso de la definición (b), que también está em DNF, una proyección en el espacio de nombres del módulo de la primera cola genera $q_1 = C$. Esto significa que los estados raros locales son los que se corresponden con una primera cola saturada, tal como queríamos. Es decir que en este caso la `Cola1` sí es relevante para el cómputo de la importancia. La situación con la `Cola2` es igual que en (a).

Por último, consideremos la fórmula en DNF $q_1 < {}^{C}/_{2} \vee q_2 = C$, equivalente a la definición (c) del evento raro. Una proyección en el espacio de nombres de la `Cola1` devuelve $q_1 < {}^{C}/_{2}$, identificando como estados raros locales aquellos que corresponden a una primera cola ocupada hasta menos de la mitad de su capacidad. Como se discutió con anterioridad, este es el resultado deseado, y pudo ser alcanzado sin mayores procesamientos gracias a que la expresión empleada en la proyección estaba en DNF. La situación con la `Cola2` es la misma que para las otras dos definiciones del evento raro.

4.2.2. Algoritmos y cuestiones técnicas

La subsección previa establece que el usuario debe realizar su consulta mediante una expresión lógica en forma normal disyuntiva. Pero hasta aquí la proyección de la fórmula en el entorno local de cada módulo ha sido introducida informalmente. Es preciso contar con un algoritmo que automatice este procedimiento.

La proyección de un literal que contenga un nombre de variable fuera de entorno genera una "expresión vacía". Sin embargo los elementos neutros de la disyunción y la conjunción son diferentes, y el algoritmo de proyección debe distinguir estos casos. Por un lado, un literal proyectado que se anula en el interior de una cláusula debe ser reemplazad con $\top$, i.e. el *verdadero* lógico. Por otro lado, una cláusula completa que anulada cuando se la proyecta debe ser reemplazada con $\bot$, i.e. el *falso* lógico.

A su vez, si se anula la expresión completa del evento raro cuando se la proyecta sobre un módulo, e.g. porque todos los literales que en ella ocurren contienen nombres de variables fuera de rango, la expresión resultante debe ser $\top$. Esto significa que todos los estados locales de un módulo irrelevante serán considerados como raros locales. La razón detrás de este proceder está relacionada con la posterior composición de las funciones locales de importancia; será justificada en Sección 4.3.

El Algoritmo 3 presenta un procedimiento para realizar la proyección descripta. Nótese que hay un número finito de cláusulas por cada fórmula DNF φ, cada una con un número finito de literales. Esto significa que ambos ciclos del algoritmo realizarán un número finito de iteraciones. La rutina `free_vars`(σ) busca el nombre de las variables libres que ocurren en el literal σ. Como cada literal tiene longitud finita la rutina terminará en tiempo finito. Además, las rutinas `global_vars`() y $\mathcal{M}$.`vars`() devuelven los nombres de las variables dentro del entorno global y del entorno de cada módulo $\mathcal{M}$ respectivamente. Como hay un número finito de variables en cada módulo del sistema, estas rutinas terminarán en tiempo constante finito, viz. son de orden $\mathcal{O}(1)$. En vista de lo expuesto, no se requiere una prueba formal para garantizar la terminación del Algoritmo 3.

Éste algoritmo parte la expresión global del evento raro, produciendo fórmulas más chicas que pueden ser interpretadas independientemente en el entorno de cada módulo del sistema. Nótese sin embargo que la política de proyección *no puede tratar casos diagonales*, i.e. situaciones donde el evento raro involucra operaciones con variables provenientes de distintos módulos en comparación directa.

Esto es inevitable en lo concerniente a la comparación de operadores aritméticos, pero puede ser esquivado en otras situaciones. Por ejemplo, si el usuario solicita un análisis transitorio de la cola tándem para el evento raro

$$\text{mín}(q_1, q_2) \geqslant \frac{C}{2}$$

la relación entre los operadores mín$(\cdots)$ y $\geqslant$ nos permite reescribir al evento raro como $q_1 \geqslant {}^{C}\!/_{2} \wedge q_2 \geqslant {}^{C}\!/_{2}$, que puede ser respectivamente proyectado como $q_1 \geqslant {}^{C}\!/_{2}$ y $q_2 \geqslant {}^{C}\!/_{2}$ sobre los módulos `Cola1` y `Cola2`. Claramente este es el comportamiento buscado.

Algoritmo 3 Proyección de una expresión DNF sobre un módulo

Entrada: módulo $\mathcal{M}$
Entrada: expresión del evento raro (global) en DNF φ

$\widetilde{\varphi} \leftarrow$ **false**
for all cláusula $\psi \in \varphi$ **do**
  $\widetilde{\psi} \leftarrow$ **true**
  for all literal $\sigma \in \psi$ **do**
    if `free_vars`$(\sigma) \subseteq \mathcal{M}$.`vars`$() \bigcup$ `global_vars`$()$ **then**
      $\widetilde{\psi} \leftarrow \widetilde{\psi} \wedge \sigma$
    end if
  end for
  if $\widetilde{\psi} \not\equiv$ **true then**
    $\widetilde{\varphi} \leftarrow \widetilde{\varphi} \vee (\widetilde{\psi})$
  end if
end for
if $\widetilde{\varphi} \equiv$ **false then**
  $\widetilde{\varphi} \leftarrow$ **true**
end if

Salida: expresión del evento raro local $\widetilde{\varphi}$

Supongamos en cambio ahora que el evento raro global es

$$q_1 > 0 \wedge \frac{q_2}{q_1} \geqslant 2 \quad \Rightarrow \quad q_2 \geqslant C \,,$$

el cual pregunta por la probabilidad de saturación en la segunda cola, cuando tiene al menos el doble de paquetes que la primer cola. Una fórmula DNF equivalente es $q_1 \leqslant 0 \vee q_2 < 2q_1 \vee q_2 \geqslant C$ „ donde q_1 y q_2 son *comparadas directamente* en la segunda cláusula. The projection of the literal $q_2 < 2q_1$ will yield an empty expression for both modules of the tandem queue, because q_1 is out of scope for `Queue2` and q_2 is out of scope for `Queue1`. Thus, the identification of the local rare states will be given by $q_1 \leqslant 0$ in `Queue1`, and by $q_2 \geqslant C$ in `Queue2`, which is at odds with the original user query. In particular, such projection does not reflect the dependence between the saturation in the second queue y la ocupación en la primera cola.

Así y todo, no creemos que este sea un problema grave due to a pragmatic reason and a theoretic reason. On the practical side, throughout our research we have encountered few studies involving these kind of properties. Most articles seem to deal with quite simple definitions of the rare event involving a single system variable. On the theoretical side, when the property query compares variables directly, a model of the system can (and most likely would) define these variables within the same module. After all, when a direct comparison is relevant the semantics behind the variables must be related, y dividirlos en módulos separados podría ser un signo de que se está modelando al sistema de forma errónea.

A pesar de esas razones hay ejemplos de casos diagonales en la bibliografía de RES. Por ejemplo [VAVA06] study the transient behaviour of the tandem queue for the rare event $q_1 + q_2 \geqslant C$. The only workaround in such cases is to define the variables affected within the same module, because Algoritmo 3 will otherwise yield empty projections as discussed. For the tandem queue this results in a monolithic representation like the one from Apéndice A.1. We highlight however that in complex systems with several components it is unnecessary to merge everything into a single monolithic model. Componer los módulos cuyas variables se hallan en comparación directa debería ser suficiente.

Ni bien se ha concluido con las proyecciones, puede procederse a construir las funciones locales de importancia de cada módulo. When the projection is empty for a component we assume it does not play a primary role in the rare event, and render a null local importance function. Otherwise we use the formula resulting from the projection, which is in DNF by construction. The projected formula $\widetilde{\varphi}_i$ is used to identify the local rare states A_i in module $\mathcal{M}_i$. Then $\mathcal{M}_i$ and A_i are provided as input to Algoritmo 1, que devuelve la función de importancia local del i-ésimo módulo.

El Algoritmo 4 presenta el procedimiento completo. Las rutinas `project`$(\cdots)$ y `derive_importance_function`$(\cdots)$ son respectivamente los Algoritmos 3 y 1. El método `identify_states`$(\cdot)$ de cada módulo $\mathcal{M}_i$ devuelve el conjunto de estados concretos locales identificados por su argumento. Como la identificación tiene lugar dentro del alcance local del módulo, la fórmula $\widetilde{\varphi}_i$ alimentada debe contener sólo variables globales o locales al módulo $\mathcal{M}_i$. La rutina `project`$(\cdots)$ se asegura de que así ocurra.

Algoritmo 4 Cómputo de funciones locales de importancia

Entrada: conjunto de módulos $\{\mathcal{M}_i\}_{i=1}^m$
Entrada: fórmula del evento raro global φ

```
assert_DNF(φ)
for all módulo M_i do
  φ̃_i ← project(M_i, φ)
  if φ̃_i ≡ true then
    f_i ← null
  else
    A_i ← M_i.identify_states(φ̃_i)
    f_i ← derive_importance_function(M_i, A_i)
  end if
end for
```

Salida: conjunto de funciones locales de importancia $\{f_i\}_{i=1}^m$

El Algoritmo 4 claramente termina dado que el número de módulos es finito, y las rutinas `project`$(\cdots)$ y `derive_importance_function`$(\cdots)$ son algoritmos cuyas pruebas de terminación ya fueron dadas.

Es importante subrayar que cuando la proyección de la fórmula DNF es vacía, la naturaleza de la función **null** asignada a f_i dependerá del operador de composición. En pocas palabras, **null** debe devolver el elemento neutro de un operador aritmético, que e.g. para '+' significa que f_i será $\lambda\, x\, .\, 0$, mientras que para $*$ será $\lambda\, x\, .\, 1$. Este tema es tratado en mayor detalle en la siguiente sección.

Como consecuencia, usando los Algoritmos 3 y 4 podemos calcular y almacenar la importancia de los estados del sistema en un régimen modular, donde los requerimientos de memoria física crecen *linealmente* con el número de módulos que componen al sistema. Esto en contraste con la construcción monolítica de la función de importancia del Capítulo 3, cuya representación como un arreglo en la memoria del computador crece de forma *exponencial* con el número de módulos. El resultado es una noción de funciones locales de importancia, el conjunto $\{f_i\}_{i=1}^{m}$ producido por el Algoritmo 4, que aún deben ser combinadas entre sí para construir una función gobal de importancia.

4.3. Composición de las funciones locales de importancia

En la Sección 2.6, la descripción de RESTART hace caso omiso de la forma en la que la función de importancia es calculada o almacenada. Durante las simulaciones, RESTART sólo necesita saber la importancia del estado (global) actual luego de que ocurre una transición. La misma transparencia es requerida por todas las técnicas de división multinivel conocidas e incluso por el Algoritmo 2 para seleccionar los umbrales. Por ende, la estrategia de la sección anterior debe complementarse con algún procedimiento para decidir la importancia de cada estado del modelo compuesto, tomando como entrada las funciones locales de importancia.

4.3.1. Estrategias básicas de composición

La opción más sencilla para componer las funciones locales de importancia es dejar que el usuario decida la cuestión, quien especificaría alguna manera ad hoc de combinar (las funciones locales de importancia de) los módulos. Decimos que el usuario provee una *función de composición* ad hoc. Formalmente, la entrada requerida es una expresión algebraica que contenga (literales numéricos e) identificadores que se correspondan con las funciones $\{f_i\}_{i=1}^{m}$.

Digamos e.g. que estamos experimentando con la modular description of the tandem queue from Código 3.3, and let `Q1` and `Q2` stand for the the local importance functions of modules `Cola1` and `Cola2` respectively. Then the user could specify composition functions such as `Q1+Q2`, `2*Q1+5*Q2`, and `(1+Q1)*(1+Q2)`. La elección probablemente dependa de la naturaleza del evento raro estudiado.

Esto es comparable con una especificación ad hoc de la función de importancia. Rather than asking the user to operate with variables, the arithmetic expression provided is of higher level, dealing with whole modules. The local importance function built for a module is trustworthy, in the sense that it effectively translates the behaviour of the module into importance information (ver el Capítulo 3).

Por consiguiente usar estas funciones locales de importancia para construir la expresión, en lugar de las variables local es de los módulos, es una mejora por sobre una definición ad hoc explícita de la función de importancia.

No hay ningún problema fundamental con tal estrategia, pero la meta general de esta tesis es el desarrollo de métodos automáticos para aligerar la carga impuesta al usuario, así que una solución algorítmica sería preferible.

La solución automática más simple para combinar the local importance functions is choosing an associative binary arithmetic operator, *a composition operand*, and apply that to all functions. Natural candidates are summation, product, max, and min. Notice each operand has its own neutral element. Therefore choosing + will make Algoritmo 4 use $\lambda\, x\, .\, 0$ as the **null** function; escoger máx usará a $\lambda\, x\, .\, -\infty$, y así sucesivamente.

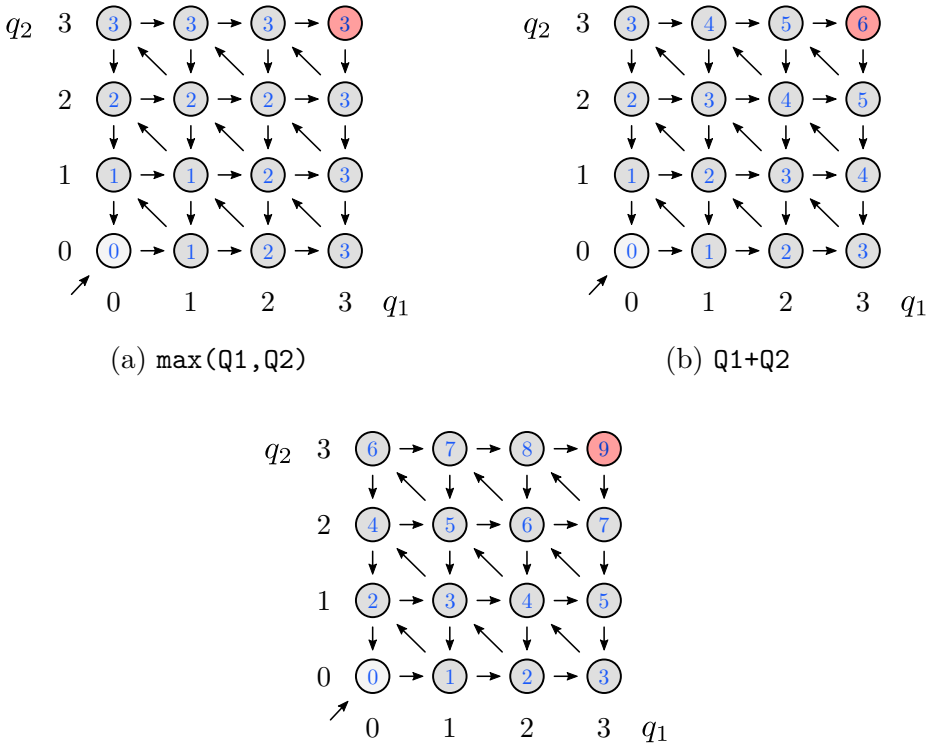
El desempeño de la división multinivel variará con la elección de operador, influenced by the nature of the model and of the rare event. That is evident since the local importance functions (i.e. los argumentos de este operando de composición) dependen de la expresión del evento raro.

Por ejemplo supongamos que la cola tándem es analizada para la definición del evento raro $q_2 = C$. Entonces usar la suma o el máximo como operando de composición produce el mismo resultado global de importancia, dado que `Q1` será una función local nula y por ende `Q1+Q2` = `max(Q1,Q2)` = `Q2`. En contraste si la definición del evento raro es $q_1 = C \wedge q2 = C$, elegir el máximo o la suma generará distintas funciones globales de importancia, como se muestra en el siguiente ejemplo.

Ejemplo 8: Composición de funciones para la cola tándem.

La Figura 4.1 muestra algunas funciones de importancia en el espacio concreto de estados de una cola tándem (de tiempo continuo) con capacidad máxima $C = 3$, donde el evento raro es $q_1 = C \wedge q2 = C$. En estas representaciones, el número de paquetes en la primer cola aumenta al movernos de izquierda a derecha, y el número de paquetes en la segunda cola aumenta al movernos de abajo hacia arriba. Las flechas que conectan los estados indican las transiciones del sistema: un arribo externo de un paquete a la primer cola es una flecha horizontal; un pasaje de paquete de la primer a la segunda cola es una flecha diagonal; un paquete que deja al sistema es una flecha vertical.

En los esquemas se presentan diferentes funciones de importancia. Los números dentro de los estados concretos (i.e. dentro de los nodos de la matriz) son los valores de importancia que cada función le asigna a los estados. Las Figuras 4.1 (a) y 4.1 (b) son funciones modulares construidas siguiendo el método de la Sección 4.2. La Figura 4.1 (a) usa el operando de composición máximo, mientras que Figura 4.1 (b) usa la suma. La Figura 4.1 (c) muestra la función monolítica que se construiría con la estrategia monolítica del capítulo anterior.



(a) `max(Q1,Q2)` (b) `Q1+Q2`

(c) `monolithic`

Figura 4.1: Funciones de importancia para la cola tándem con evento raro $q_1 = C \wedge q_2 = C$

La simetría de los valores de importancia en las Figuras 4.1 (a) y 4.1 (b) es un resultado de la naturaleza composicional detrás de la asignación de los valores globales de importancia. Mientras que la función monolítica de la Figura 4.1 (c) puede considerar cada estado global individualmente, las funciones resultantes del método composicional sólo pueden distinguir entre los estados separados por los módulos. Como consecuencia puede haber comportamientos globales que la estrategia monolítica puede distinguir pero que son invisibles a los ojos de la estrategia composicional. Esto está relacionado con los casos diagonales mencionados en la Subsección 4.2.2 y es discutido en más detalle en la Subsección 4.3.2. □

En subsecciones previas introdujimos la *condición de monotonicidad* de una función de importancia, para referirnos al caso en que una simulación que sigue un camino más corto desde el estado actual hacia el conjunto raro, sólo visitará una secuencia monótona creciente de valores de importancia. Para la situación representada en la Figura 4.1, un camino más corto es cualquiera que sólo usa flechas horizontales y diagonales para alcanzar el único estado raro en la esquina superior derecha de la matriz. Nótese que la función monolítica de la Figura 4.1 (c) satisface esta condición, como esperábamos.

La condición de monotonicidad es una propiedad deseable para una función de

importancia. Nos asegura que los caminos simulados que se mueven en la dirección correcta no serán truncados, lo cual claramente es una ventaja. Sin embargo lo mismo puede asegurarse empleando una definición más débil, pidiendo que se visiten valores de importancia "no decrecientes" en lugar de "crecientes". Bajo esta nueva definición, la estrategia de composición `Q1+Q2` de la Figura 4.1 (b) en el Ejemplo 8 también satisface la condición de monotonicidad. Esto no se cumple empero para `max(Q1,Q2)` en la Figura 4.1 (a), obsérvese e.g. la transición diagonal $(q_1, q_2) = (2, 0) \rightarrow (1, 1)$.

De esto podríamos adivinar que, para el evento raro $q_1 = C \wedge q_2 = C$, la adición es un mejor operando de composición que el máximo. Esto es también lo que nos sugiere la (mucho menos rigurosa) regla del pulgar de comparar el máximo valor de importancia asignado en cada caso. La suma de `Q1` con `Q2` produce la importancia 6 para el estado raro del sistema, mientras que `max(Q1,Q2)` sólo alcanza el valor 3. Recordemos que para un modelo y evento raro fijos, mientras mayor sea el rango de importancia, más opciones tendrá el Algoritmo 2 a la hora de elegir los umbrales. Desde esta perspectiva `Q1+Q2` también parece más prometedora que `max(Q1,Q2)`, dado que en general más umbrales implican más división y por ende mejor eficiencia de simulación, al menos cuando algoritmos adaptativos como la División Multinivel Adaptativa o el Monte Carlo Secuencial son usados para seleccionar los umbrales inteligentemente.

4.3.2. Funciones de importancia monolíticas vs. composicionales

En la estrategia del Capítulo 3, la función de importancia se construye sobre el espacio de estados del modelo compuesto. El Algoritmo 1 realiza esta tarea y, como resultado, el *comportamiento estático global* es explotado en el proceso. Por comportamiento estático nos referimos a las transiciones del grafo de adyacencia a nivel concreto, donde hay noción de los predecesores de un estado pero donde se ignora la naturaleza probabilística/estocástica de las aristas[†].

En este capítulo el comportamiento estático de cada módulo individual es usado para construir las funciones locales de importancia. Pueden sin embargo existir interacciones entre los componentes del sistema, que aparecen explícitamente a nivel global pero que no pueden ser capturadas por el Algoritmo 1 a nivel local. Si se eligen estrategias inadecuadas para componer las funciones locales de importancia, esto puede deteriorar la calidad de la función resultante si se diese alguna de las siguientes situaciones:

(a) las variables usadas en la expresión del evento raro residen en unos pocos módulos pequeños, que expresan poco y nada del comportamiento total del proceso;

(b) la sincronización de los módulos *es* gran parte de la semántica del proceso, y el estado conjunto de todos los componentes del sistema afecta mayormente el comportamiento de cada componente individual.

[†] Distinguimos esto del *comportamiento dinámico* del sistema que afecta e.g. a las simulaciones durante una corrida de RESTART.

El siguiente ejemplo ilustra los peligros de un uso inapropiado de la estrategia composicional cuando el sistema exhibe este tipo de problemas.

Ejemplo 9: Construyendo castillos de cartas.

Holgazaneando un domingo por la tarde nos hallamos hurgando en el placar del abuelo, Dios lo tenga en su gloria. Dentro de un portafolios polvoriento encontramos un viejo mazo de cartas españolas, y comenzamos a construir castillos de cartas. Construimos un castillo por palo usando todas las cartas del palo. Los construimos bien cerca entre sí, de forma tal que un sólo error tumbe a todos los castillos (¡uy!).

El Código 4.1 muestra un modelo PRISM del juego en un entorno de tiempo discreto. Como es un DTMC todas las aristas tienen pesos probabilísticos. Estos son implícitamente 1 excepto para la arista de las líneas 9 y 10, donde hay igual probabilidad de colocar correctamente una carta y de tirar todo al demonio.

El módulo `Castillo` representa cuántas cartas han sido correctamente colocadas en el castillo actual. El módulo `Palo` lleva cuenta del palo en uso; los mazos españoles contienen *Copa* (), *Oro* (), *Basto* (), y *Espada* ().

Código 4.1: Juego de castillos de cartas

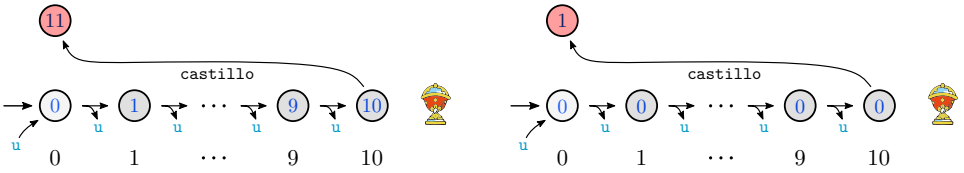
```
1  dtmc
2
3  const int NUM_PALOS = 4;  // Copa, Oro, Basto, Espada
4  const int CARTAS_PALO = 10;
5
6  module Castillo
7      estado: [0..2];  // 0:quieto ; 1:poner_carta ; 2:tumbar_todo
8      cartas: [0..CARTAS_PALO];
9      []   estado=0 & cartas<CARTAS_PALO -> 0.5: (estado'=1)
10                                          + 0.5: (estado'=2);
11     []   estado=1 & cartas<CARTAS_PALO -> (cartas'=cartas+1)
12                                           & (estado'=0);
13     [uy] estado=2 & cartas<CARTAS_PALO -> (cartas'=0) & (estado'=0);
14     [castillo]      cartas=CARTAS_PALO -> (cartas'=0);
15 endmodule
16
17 module Palo
18     palo: [1..NUM_PALOS];
19     [uy]       true -> (palo'=1);
20     [castillo] palo < NUM_PALOS -> (palo'=palo+1);
21 endmodule
```

Comenzando fácil queremos construir sólo un castillo, digamos usando copas. ¿Qué tan probable es que lo logremos en el primer intento, sin tumbar nada? Esta pregunta es de naturaleza transitoria, y el modelo del Código 4.1 nos permite consultarla de forma sucinta:

```
P=? [ estado<2 U palo=2 ].
```

Comparemos las funciones de importancia que los métodos monolítico y composicional (con la suma) generarían. Para ello la Figura 4.2 muestra una

representación espartana del espacio de estados concretos del sistema. En el esquema sólo se representa al palo de Copa, y la variable `estado` del módulo `Castillo` es abstraída. Dicha variable sólo existe para distinguir entre la colocación correcta de una carta y el tumbado de todos los castillos. En los esquemas de la Figura 4.2 eso está representado con una flecha bifurcada, ↯, donde la punta hacia abajo representa el tumbado (¡uy!), y la punta recta representa una carta correctamente colocada.



(a) Monolítico (b) Composicional (op: +)

Figura 4.2: Juego de castillos de cartas (versión para principiantes)

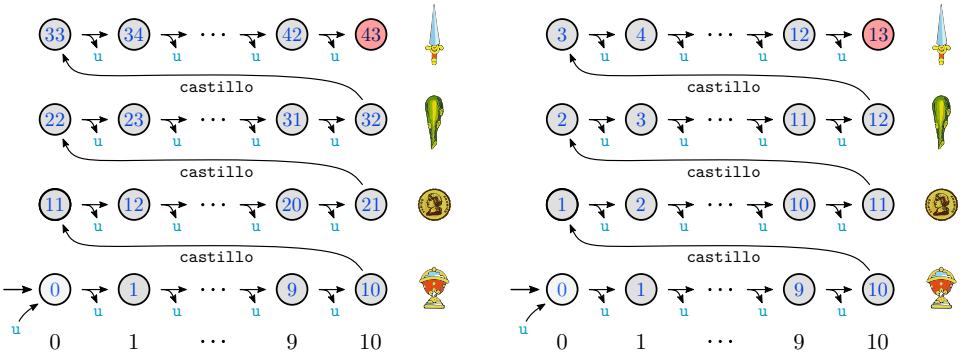
La chatez de la función de la Figura 4.2 (b), que sólo tiene los valores de importancia $\{0, 1\}$, se debe al problema (a) mencionado anteriormente. Como la expresión del evento raro en la consulta de la propiedad es '`palo=2`', la función local de importancia del módulo `Castillo` es anulada, porque ninguna de sus variables aparece en la expresión. En oposición, el método monolítico *sólo* puede observar el valor de la variable `palo` *en interacción* con la variable `cartas`, capturando así el comportamiento completo del modelo compuesto.

La estrategia composicional devuelve una función de importancia pobre porque la expresión del evento raro deja fuera a un módulo relevante, i.e. debido al problema (a). En algunos casos hay un parche sencillo para forzar la ocurrencia de variable relevantes pero no esenciales en la expresión del evento raro. Por ejemplo la consulta `P=? [ estado<2 U palo=1 & cartas=CARTAS_DEL_PALO ]` enriquece a la estrategia composicional en la situación anterior, generando la misma función de importancia que produciría la estrategia monolítica.

Sin embargo el problema (b) es más difícil de contrarrestar, pues construir funciones rigurosamente locales a los módulos implica perder cierta noción de la naturaleza de las interacciones entre módulos. Para ilustrar los peligros de este problema considérese la versión completa del juego, donde se busca construir un castillo por cada uno de los cuatro palos. La consulta es:

```
P=? [ estado<2 U palo=NUM_PALOS & cartas=CARTAS_DEL_PALO ].
```

Así ya esquivamos al problema (a) y la aplicación de los métodos monolítico y composicional produce las funciones representadas en la Figura 4.3. Al igual que antes la suma es usada para componer las funciones locales de importancia de los módulos `Castillo` y `Palo` en la Figura 4.3 (b).



(a) Monolítico (b) Composicional (op: +)

Figura 4.3: Juego de castillos de cartas (versión para expertos)

Por un lado la Figura 4.3 (a) nos muestra una continuación natural del resultado monolítico previamente presentado en la Figura 4.2 (a), aquí extendido para considerar los cuatro palos del mazo español.

Por otro lado la función de la Figura 4.3 (b) muestra una anomalía que juega en detrimento de la división por importancia: cada vez que un castillo es completado usando todas las cartas del palo correspondiente, la importancia disminuye casi hasta el valor inicial. En simulaciones de división multinivel, esto podría truncar muchas copias generadas que de hecho estaban avanzando en la dirección correcta. □

Al menos dos factores contribuyen a la anomalía representada en la Figura 4.3 (b) para el método composicional. Uno de ellos es el problema (b), i.e. el sistema del Código 4.1 usa la comunicación entre módulos como un elemento clave para evolucionar progresivamente hacia el conjunto de estados raros.

Por sí solo esto no es un problema para la estrategia. The second factor that led to the poor result of Figura 4.3 (b) is using summation as composition operand. This clearly failed to apprehend the nature de la evolución hacia el evento raro.

Combinado con una función de composición adecuada, the compositional approach can be forced to mimic the monolithic function. Take for instance `CARTAS_PALO*PALO+CASTILLO`, where `PALO` y `CASTILLO` representan las funciones locales de los homónimos en el Código 4.1.

Pero volver al uso de una función de composición requests an ad hoc intervention by the user, which defeats the purpose of the thesis. An automatable procedure is desirable, with enough plasticity to behave well when the system modules exhibit non-trivial interaction schemes. Nuestra propuesta en ese sentido es el objeto de la siguiente subsección.

El Ejemplo 9 puede dar la impresión de que, si es posible, using the monolithic approach from Capítulo 3 will always yield an importance function superior to the ones that can be built with the compositional approach. Esto definitivamente no es así.

Primero, el modelo de los castillos de cartas presentado en el Código 4.1 es un tanto artificial. It was devised with the intention of dislocating an otherwise purely sequential evolution from the initial state towards the rare event. The goal was to put in evidence that a naïve application of the compositional techniques de las secciones anteriores puede generar resultados malos.

Segundo, recordemos las limitaciones del método monolítico que presentamos en la Sección 3.6. There were two issues with the database system from Ejemplo 7: a monolithic function would occupy more physical memory in the machine than available; and the model displays a flat structure when all its modules are composed. The compositional approach is primarily designed to tackle with the first issue, but is also very useful to attack the second one.

El problema específico fue que una vez compuesto, el espacio de estados de la base de datos distingue sólo R niveles de importancia cuando se usa la redundancia $R \in \mathbb{N}_{>1}$. Eso deja muy poco espacio para dividir al espacio de estados en capas, como requiere la división multinivel. Si en cambio la estructura del sistema fuese explotada previamente a su composición, como nos permite hacerlo el método composicional, más niveles de importancia podrían ser fabricados para aumentar la división que la técnica requiere. Avanzamos más sobre esta idea en la siguiente subsección.

4.3.3. Componiendo las funciones con anillos y semianillos

El objetivo principal es concebir una *estrategia automática de composición* para las funciones locales de importancia, que exhiba un buen comportamiento independientemente de las interacciones entre los módulos, generando una función de importancia global razonable.

Recordemos la aplicación del método composicional a la cola tándem en la Subsección 4.3.1. Dependiendo del evento raro estudiado, usar los operandos de composición `max` o `+` podía producir funciones globales de importancia diferente. En particular `max(Q1,Q2) = Q1+Q2` para el evento raro $q_2 = C$, mientras que el Ejemplo 8 nos mostró que la sima es el mejor candidato cuando el evento raro es $q_1 = C \wedge q_2 = C$. Todo esto sugiere que el desempeño de una estrategia de composición está directamente afectado por la expresión que define al evento raro.

Considérese ahora una cola tándem triple, i.e. los paquetes procesados por el servidor de la segunda cola son encolados en un tercer buffer, y abandonan el sistema sólo tras haber sido procesados por el servidor de esta tercera cola. El Código 4.2 muestra un modelo de este sistema.

Código 4.2: Modelo PRISM de la cola tándem triple

```
1 ctmc
2
3 const int      c = 7;  // Capacidad de las colas
4 const int lambda = 1;  // Tasa de arribos
5 const int    mu1 = 2;  // Tasa del servidor 1
6 const int    mu2 = 4;  // Tasa del servidor 2
7 const int    mu3 = 6;  // Tasa del servidor 3
8
9 module Arribos
```

```
10      [arribo]  true -> lambda: true;
11  endmodule
12
13  module Cola1
14      q1: [0..c];
15      [arribo]    q1<c ->   1: (q1'=q1+1);  // Recibir
16      [arribo]    q1=c ->   1: true;
17      [servicio1] q1>0 -> mu1: (q1'=q1-1);  // Procesar
18  endmodule
19
20  module Cola2
21      q2: [0..c];
22      [servicio1] q2<c ->   1: (q2'=q2+1);  // Recibir
23      [servicio1] q2=c ->   1: true;
24      [servicio2] q2>0 -> mu2: (q2'=q2-1);  // Procesar
25  endmodule
26
27  module Cola3
28      q3: [0..c];
29      [servicio2] q3<c ->   1: (q3'=q3+1);  // Recibir
30      [servicio2] q3=c ->   1: true;
31      [servicio3] q3>0 -> mu3: (q3'=q3-1);  // Procesar
32  endmodule
```

Si el usuario solicita un análisis estacionario en el modelo del Código 4.2 para el evento raro

$$(q_1 = C \ \wedge \ q_2 = C) \ \vee \ (q_1 = C \ \wedge \ q_3 = C)\,. \tag{15}$$

La fórmula ya está en DNF; la proyección sobre los módulos de las colas identificará como estados raros locales a aquellos que satisfagan las fórmulas $q_i = C$ con $i \in \{1, 2, 3\}$.

Nótese que los estados de todos los módulos son relevantes. Thus one could think that summation is a natural candidate for composition strategy, viz. employing `Q1+Q2+Q3` as global importance function, donde `Qi` representa la función local de importancia del módulo `Colai`.

Nótese sin embargo que la ecuación (15) es equivalente a $q_1 = C \wedge (q_2 = C \vee q_3 = C)$, hinting at a higher relevance of the number of packets in the first queue w.r.t. the number in the second (or third) queue taken on its own. That is overlooked by the global importance function `Q1+Q2+Q3`, que considera el número de paquetes en cada cola igualmente relevante.

Esto sugiere que hay una fuerte correlación between the (precise) rare event expression used, and the performance of the composition strategy. Thus, the possibility of deriving the later using the former is quite appealing. The objective is to generate a composition strategy tan fuertemente ligada a la expresión del evento raro como sea posible.

El requisito de trabajar con fórmulas lógicas expressed in DNF already gives some structure that could be exploited. Recall the building blocks of DNF formulae are the literals, and the current focus is on literals consisting of variables local to the scope of a single module (ver la Subsección 4.2.2). One could hence map the literals in the rare event formula, a funciones locales de importancia locales al módulo donde estos literales residen.

Esto indica que la ecuación (15) produce la secuencia de funciones locales de importancia `(Q1,Q2,Q1,Q3)`. Those functions are to be composed *somehow* following the DNF structure of la ecuación (15), to build the arithmetic expression which will serve as global importance function. That sounds reasonable but leaves an open question: ¿cómo deberían interpretarse los operadores de disyunción y conjunción de la expresión en DNF?

Notemos que las reflexiones previas acerca de la mayor relevancia de `Cola1` para la ecuación (15) usan la propiedad distributiva de $\wedge$ respecto de $\vee$. En particular, el par $(\vee, \wedge)$ es una estructura algebraica conocida como *anillo*, donde $\vee$ y $\wedge$ son respectivamente la *adición* y la *multiplicación* del anillo. La distributividad de la multiplicación sobre la adición, e.g. de $\wedge$ sobre $\vee$, es un axioma de las estructuras algebraicas de anillo y semianillo.

Por ende proponemos escoger pares de operadores algebraicos $(\oplus, \odot)$ que tengan estructura de anillo o semianillo, y mapear las ocurrencias de $(\vee, \wedge)$ en la expresión DNF del evento raro a $(\oplus, \odot)$ en la estrategia de composición. En particular, máximo y suma (i.e. $(\text{máx}, +)$), y suma y producto (i.e. $(+, *)$), tienen respectivamente estructura de semianillo y de anillo.

Entonces por ejemplo el evento raro de la ecuación (15) can be turned into the composition strategy `max( Q1+Q2 , Q1+Q3 )`, which by the distributive property of $+$ with máx results in the global importance function `Q1+max(Q2,Q3)`. Remember `Qi` does not symbolize a variable of a module, sino que deber ser interpretada como la función de importancia local a un módulo.

Para contrastar el funcionamiento of this strategy against the simplistic approach of employing a binary associative operand, consider a triple tandem queue where el estado global actual es $(q_1, q_2, q_3) = (C-1, C-1, 0)$.

Usar la suma, i.e. the composition operand $+$, results in the global importance function `Q1+Q2+Q3`. This function misses out the structure that appears in the expression of the rare event, and for instance yields the same importance as a new incoming packet enters the system and moves from the first to the last queue. Yet that is a mistake: it is true that a new packet entering the system modifies the (former) state $(C-1, C-1, 0)$ increasing the importance, because a rare event can be generated by having $(q_1 = C \wedge q_2 = C)$; pero la importancia *disminuye* si un paquete alcanza la tercer cola, dado que le falta mucho a dicha cola para llenarse.

En cambio, usando el semianillo $(\text{máx}, +)$ obtenemos la función de importancia `Q1+max(Q2,Q3)` como dijimos. Entonces la importancia permanece intacta, digamos igual a $2C-1$, mientras este nuevo paquete se encuentre en la primer o segunda cola. Sin embargo, la importancia disminuiría como queríamos a $2(C-1)$, digamos, cuando el paquete se traslada hacia la tercer cola.

4.3.4. Posprocesamiento de las funciones

Hay más que hacer luego de haber escogido la estrategia de composición. Recuérdese que al seleccionar un operando de composición, el Algoritmo 4 seleccionará el elemento neutro apropiado, que será usado como la función **null**

cuando la proyección de la expresión DNF sobre el módulo esté vacía. Sin embargo, cuando uno elige una estrategia de composición de anillo o semianillo hay dos operadores en juego. Entonces por ejemplo si usamos $(\text{máx}, +)$, la función **null** debe comportarse como $\lambda\, x\,.\, 0$ cuando (el identificador que representa a) la función local de importancia correspondiente aparece como argumento de $+$, y debe comportarse como $\lambda\, x\,.\, -\infty$ cuando aparece como argumento de máx.

Una forma de lograr el comportamiento correcto is to apply a post-processing to the importance values computed by the derivation algorithms. The $(\text{máx}, +)$ semiring is in no real need of such tricks since our importance functions are non-negative. Thus 0 is the null element for both $+$ and máx. The $(+, *)$ ring however could render to zero a whole product in an expression, simply because one of the local importance functions involved is currently at its minimum, viz. 0 according to el Algoritmo 1. Nos referiremos a esto como el *problema de nulificación*.

El arreglo más sencillo es el *desplazamiento de valores*: take all the values of any local importance function and add 1 to them. Alas, there are no more zeroes and the nullification problem vanishes. All this without the importance function having really changed. Another possibility a little more far fetched is to apply *potenciación*. Take any base $b \in [1, \infty)$ and change the importance value i to the value b^i for every state. Since $b^0 = 1$ this achieves the goal just as well; however the importance function has changed, it has expanded and has now an increased range de valores de importancia (globales) diferentes.

Para ver cómo funciona consideremos la cola tándem triple para el evento raro $(q_1 = C \wedge q_2 = C) \vee (q_1 = C \wedge q_3 = C)$ como lo definimos en la ecuación (15). Digamos que el anillo $(+, *)$ is used to compose the local importance functions $\{\texttt{Q1}, \texttt{Q2}, \texttt{Q3}\}$, yielding the global function `Q1*(Q2+Q3)` as described in Subsección 4.3.3. Since `Q1` is multiplied by the result of the sum of the other two functions, a post-processing is needed to avoid the nullification problem. For capacity $C = 2$ and assuming $\texttt{Qi} \in \{0, 1, 2\}$, the shift post-processing yields *eleven* different global importance values, namely $\{2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18\}$. Using instead the post-processing `exp` with e.g. the base $b = 2{,}0$ yields the global importance values $\{2, 3, 4, 5, 6, 8, 10, 12, 16, 20, 24, 32\}$, which are *twelve* in total. If the capacity of the queues is increased to $C = 3$, then shifting yields 19 different global importance values whereas exponentiation yields 22; para $C = 4$ estos son 27 vs. 35.

Lo que ocurre es que muchos valores globales de importancia appear repeated for different combinations of values of the functions $\{\texttt{Q1}, \texttt{Q2}, \texttt{Q3}\}$. This is due to the interactions between product and addition in the expression `Q1*(Q2+Q3)`, which for very close values of `Q1`, `Q2`, and `Q3`, cause many repetitions of the arithmetic result. Setting the possible values of these local functions further apart, e.g. by means of the exponentiation post-processing, resulta en muchas menos repeticiones y por ende en un conjunto más rico de valores globales de importancia de entre los cuales escoger a los umbrales.

La Subsección 4.3.3 indica cuan beneficioso puede ser explotar la expresión del evento raro para derivar una función de importancia *natural* y eficiente. En esta subsección mostramos que una elección cuidadosa del posprocesamiento para

el anillo/semianillo escogido, puede mejorar el rango de importancia exhibido por la función global de importancia. La ventaja es evidente: mientras más valores de importancia nos otorgue la función, más opciones habrá de entre las cuales escoger umbrales, y más prometedora será la aplicación de la división multinivel.

La ganancia obtenida usando estas técnicas será ejemplificada prácticamente cuando volvamos a revisar el sistema de base de datos con redundancia del Ejemplo 7. Primero sin embargo extenderemos la expresividad de nuestros formalismos de modelado.

4.4. Autómatas estocásticos con entrada/salida

Cuando introdujimos los Algoritmos 1 a 4 resaltamos que la propiedad markoviana nunca fue parte de las hipótesis, y que la teoría presentada es aplicable a procesos estocásticos homogéneos generales. No obstante, la única sintaxis de modelado introducida hasta ahora es el lenguaje de entrada de PRISM, que para el alcance de esta tesis sólo puede representar sistemas markovianos. Ahora presentamos un lenguaje moderno de modelado que puede representar procesos en un entorno temporal continuo donde pueden emplearse distribuciones arbitrarias (i.e. no únicamente la exponencial).

Los Autómatas Estocásticos (SA por sus siglas en inglés) fueron introducidos en la Sección 2.1. Permiten muestrear eventos estocásticos de distribuciones arbitrarias, i.e. variables aleatorias continuas arbitrarias pueden ser representadas en un SA. Estas variables aleatorias son denominadas *relojes*, y toman valores positivos resultantes del muestreo de sus distribuciones (continuas) asociadas. A medida que avanza el tiempo del sistema, el *tiempo restante* de los relojes decrece en igual proporción (*en sincronía*), viz. el valor de todos los relojes decrece con la misma tasa. Cuando el valor de un reloj llega a cero "el reloj expira", habilitando la ocurrencia de eventos.

Sin embargo, la relación de transición '$\rightarrow$' de la Definición 4 de SA permite comportamiento no determinista, which is problematic from the point of view of simulations. This issue is acknowledged by [DLM16], who derive a subset of SA closed under parallel composition called *Input/Output Stochastic Automata*. The result are fully probabilistic systems, viz. where nondeterminism has been ruled out en el modelo compuesto resultante.

Para lograrlo [DLM16] restrinjen the framework of SA and work with a partition of the actions set A, splitting them into *input actions* ($A^{\mathcal{I}}$) and *output actions* ($A^{\mathcal{O}}$). Inputs synchronise with outputs, which respectively behave in a reactive and generative manner [VSS95]. This roughly means that the act of performing the transition (*generando* comportamiento) is indicated by an output, whereas inputs listen and synchronise themselves with outputs (*reaccionando* a este comportamiento).

Así, las acciones de salida tienen un rol activo and are locally controlled. As a consequence their occurrence time is controlled by a random variable. Instead, input actions have a passive role and are externally controlled. Therefore their occurrence time can only depend on their interaction with outputs. A continuación

definimos estos sistemas formalmente.

Definición 17 (IOSA, [DLM16])**.** Un *Autómata Estocástico con Entrada/Salida* (IOSA por sus siglas en inglés) es una tupla $(S, A, \mathcal{C}, \rightarrow, s_0, \mathcal{C}_0)$ donde:

- S es un conjunto numerable de estados,
- A es un conjunto numerable de etiquetas particionado en conjuntos disjuntos de etiquetas de entrada $A^{\mathcal{I}}$ y etiquetas de salida $A^{\mathcal{O}}$,
- $\mathcal{C}$ es un conjunto finito de relojes t.q. cada $x \in \mathcal{C}$ tiene una medida continua de probabilidad asociada $\mu_x \colon \mathbb{R} \to [0, 1]$ con soporte en $\mathbb{R}_{>0}$,
- $\rightarrow \subseteq S \times 2^{\mathcal{C}} \times A \times 2^{\mathcal{C}} \times S$ es una función de transición,
- $s_0 \in S$ es el estado inicial, y
- $\mathcal{C}_0 \subseteq \mathcal{C}$ son los relojes inicializados en el estado inicial.

Además, un IOSA satisface las siguientes restricciones:

(a) si $s \xrightarrow{C,a,C'} s'$ y $a \in A^{\mathcal{I}}$, entonces $C = \varnothing$,

(b) si $s \xrightarrow{C,a,C'} s'$ y $a \in A^{\mathcal{O}}$, entonces C es un conjunto unitario,

(c) si $s \xrightarrow{\{x\},a_1,C_1} s_1$ y $s \xrightarrow{\{x\},a_2,C_2} s_2$, entonces $a_1 = a_2$, $C_1 = C_2$ y $s_1 = s_2$,

(d) si $s \xrightarrow{\{x\},a,C} s'$ entonces, para cada transición $t \xrightarrow{C_1,b,C_2} s$, o bien $x \in C_2$, o sino $x \notin C_1$ y existe alguna transición $t \xrightarrow{\{x\},c,C_3} t'$,

(e) si $s_0 \xrightarrow{\{x\},a,C} s$ entonces $x \in C_0$,

(f) para cada $a \in A^{\mathcal{I}}$ y estado s, existe una transición $s \xrightarrow{\varnothing,a,C} s'$,

(g) para cada $a \in A^{\mathcal{I}}$, si $s \xrightarrow{\varnothing,a,C_1} s_1$ y $s \xrightarrow{\varnothing,a,C_2} s_2$, entonces $C_1 = C_2$ y $s_1 = s_2$.

La ocurrencia de una acción es controlada por la expiración de los relojes. Así, cuando $s \xrightarrow{\{x\},a,C} s'$ y el sistema se encuentra en el estado s, la acción de salida a se ejecutará tan pronto como el reloj x expire. En este punto el sistema evoluciona hacia el estado s', sorteando nuevos valores para cada reloj $y \in C$ que son muestreados de las correspondientes distribuciones μ_y. Para las transiciones de entrada $s \xrightarrow{\varnothing,a,C} s'$ el comportamiento es similar; la diferencia radica en el tiempo de ocurrencia de la transición, que será definido cuando la acción interactúe con una salida.

La restricción (a) establece that inputs are reactive and hence their occurrence is controlled by the environment. La restricción (b) states that outputs are generative (or locally controlled) so they have an associated set of clocks which determine their occurrence time[†].

La restricción (c) impide that a single clock enables two different transitions, which is crucial to avoid nondeterministic behaviour, since otherwise two output actions could become enabled simultaneously. Además nótese que el usar un

[†] El conjunto se pide unitario para lograr una definición limpia.

reloj después de que éste ha expirado habilitaría inmediatamente la transición de salida respectiva. That also leads to situations where two or more transitions are simultaneously enabled, e.g. if the system arrives at a state where two different (expired) clocks enable two different output transitions. Las restricciones (d) y (e) ensure that expired clocks are not used. Particularly (d) states that an enabling clock x at state s must either: be set on arrival to state s ($x \in C_2$); or is enabling in the immediately preceding state t and it has not been used right before reaching s ($x \notin C_1$).

Como los valores de los relojes son muestreados from continuous random variables, the probability that the value sampled for two different clocks coincides is null. This, together with restricciones (c) a (e), guarantees that *casi nunca* two different output transitions are enabled at the same time point. Finally, las restricciones (f) y (g) are usual restrictions on Input/Output-like automata: (f) ensures that outputs are not blocked in a composition; (g) garantiza que la composición paralela preserva el determinismo.

A los autómatas estocásticos con entrada/salida se les da semántica en términos de NLMP [DSW12,Wol12], que son una generalización de los sistemas de transición probabilistas con dominio continuo. Más precisamente, los NLMP extienden a los LMP [DEP02] con no determinismo *interno*. A continuación definimos formalmente a los NLMP, dado que serán utilizados para mostrar que los IOSA no exhiben comportamiento no determinista. Para una comprensión más profunda de la Definición 18 y una descripción más extensa de estos sistemas y sus propiedades, encomendamos al lector la lectura del Apéndice C.

Definición 18 (NLMP)**.** Un *proceso de Markov etiquetado no determinista* (NLMP por sus siglas en inglés) es una tupla $(\boldsymbol{S}, \Sigma, \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ donde:

- $\boldsymbol{S}$ es un conjunto arbitrario de estados,
- Σ es una σ-álgebra sobre $\boldsymbol{S}$,
- para cada etiqueta $a \in \mathcal{L}$ la función $\mathcal{T}_a \colon \boldsymbol{S} \to \Delta(\Sigma)$ es measurable desde Σ hacia la σ-álgebra de impacto $H(\Delta(\Sigma))$.

La semántica de un IOSA se define formalmente a través de un NLMP utilizando dos tipos de transiciones: un tipo codifica los pasos discretos, y contiene toda la información probabilística introducida por el muestreo de los relojes; el otro tipo registra el paso del tiempo, disminuyendo el valor de todos los relojes en sincronía. Para simplificar ciertas cuestiones técnicas, la Definición 19 [DLM16] asume un orden en el conjunto de relojes $\mathcal{C}$, que también se aplica a los vectores de $\mathbb{R}^N$ que representan sus valuaciones.

Definición 19. Dado un IOSA $\mathcal{I} = (S, A, \mathcal{C}, \to, s_0, \mathcal{C}_0)$ con $\mathcal{C} = \{x_i\}_{i=1}^N$, su semántica está definida por el NLMP $\mathcal{P}(\mathcal{I}) = (\boldsymbol{S}, \mathscr{B}(\boldsymbol{S}), \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ donde:

- $\boldsymbol{S} = (S \uplus \{\mathsf{init}\}) \times \mathbb{R}^N$ and $\mathcal{L} = \{\mathsf{init}\} \uplus A \uplus \mathbb{R}_{>0}$, con $\mathsf{init} \notin S \cup A \cup \mathbb{R}_{>0}$,
- $\mathcal{T}_{\mathsf{init}}(\mathsf{init}, \vec{v}) = \{\delta_{s_0} \times \prod_{i=1}^N \mu_{x_i}\}$,

- $\mathcal{T}_a(s,\vec{v}) = \{\mu_{\vec{v},C',s'} \mid s \xrightarrow{C,a,C'} s', \bigwedge_{x_i \in C} \vec{v}(i) \leq 0\}$ para toda $a \in A$, donde $\mu_{\vec{v},C',s'} = \delta_{s'} \times \prod_{i=1}^{N} \overline{\mu}_{x_i}$, con $\overline{\mu}_{x_i} = \mu_{x_i}$ si $x_i \in C'$ y $\overline{\mu}_{x_i} = \delta_{\vec{v}(i)}$ en caso contrario, y
- $\mathcal{T}_d(s,\vec{v}) = \{\delta^{-d}_{(s,\vec{v})} \mid 0 < d \leq \text{mín}(V)\}$, donde $\delta^{-d}_{(s,\vec{v})}$ es la distribución de Dirac $\delta_s \times \prod_{i=1}^{N} \delta_{\vec{v}(i)-d}$, y para todo $d \in \mathbb{R}_{\geqslant 0}$ definimos el conjunto de reales positivos $V = \left\{\vec{v}(i) \mid \exists a \in A^{\mathcal{O}}, C' \subseteq \mathcal{C}, s' \in S : s \xrightarrow{\{x_i\},a,C'} s'\right\}$, t.q. $\text{mín}(\varnothing) = +\infty$.

En [DLM16] se prueba que la estructura $\mathcal{P}(\mathcal{I})$ la Definición 19 efectivamente satisface la Definición 18 de NLMP. Nótese que el espacio de estados $\boldsymbol{S}$ de $\mathcal{P}(\mathcal{I})$ es el producto del espacio de estados del IOSA con todas las posibles valuaciones de los relojes. Un estado inicial diferenciado init se añade para codificar la inicialización aleatoria de todos los relojes. A su vez, $\boldsymbol{S}$ tiene la estructura de σ-álgebra de Borel habitual, $\mathscr{B}(\boldsymbol{S})$.

Los pasos discretos se codifican mediante $\mathcal{T}_a$ para $a \in A$. En el estado $(s,\vec{v})$ la transición $s \xrightarrow{C,a,C'} s'$ puede ocurrir si $\bigwedge_{x_i \in C} \vec{v}(i) \leqslant 0$, i.e. ni bien todos los relojes actuales haya espirado (trivialmente cierto para las acciones de entrada). El próximo estado alcanzado en el NLMP tendrá a s' como estado del IOSA, los relojes fuera de C' preservan su valor, y el valor de los relojes en C' es muestreado nuevamente de sus respectivas distribuciones.

Los pasos temporales se codifican mediante $\mathcal{T}_d(s,\vec{v})$ para $d \in \mathbb{R}_{\geqslant 0}$. Esta transición puede ocurrir sii no hay transiciones de salida habilitadas en el estado actual dentro de las próximas d unidades de tiempo. Si en efecto esto se cumple, el sistema permanece en el mismo estado IOSA s, y los valores de todos los relojes se decrementan en d, viz. transcurren d unidades de tiempo en s.

El formalismo de modelado IOSA fue diseñado para permitir la composición paralela de varios componentes de sistema. En este esquema de entrada/salida, las salidas son autónomas y sólo pueden sincronizar con entradas homónimas (no se permite la sincronización entre acciones de salida de diferentes componentes). Hay varios aspectos técnicos que atender, como la colisión de nombres de relojes. La siguiente definiciones tomada de [DLM16] formaliza la noción de composición paralela de IOSA.

Definición 20. Sean $\mathcal{I}_1 = (S_1, A_1, \mathcal{C}_1, \rightarrow_1, s_0^1, \mathcal{C}_0^1)$ e $\mathcal{I}_2 = (S_2, A_2, \mathcal{C}_2, \rightarrow_2, s_0^2, \mathcal{C}_0^2)$ dos IOSA arbitrarios. Éstos se dicen *compatibles* si no comparten acciones de salida ni relojes, es decir, si $A_1^{\mathcal{O}} \cap A_2^{\mathcal{O}} = \varnothing$ y $\mathcal{C}_1 \cap \mathcal{C}_2 = \varnothing$.

Definición 21 (Composición de IOSA)**.** Dados dos IOSA compatibles $\mathcal{I}_1$ y $\mathcal{I}_2$, su *composición paralela* $\mathcal{I}_1 \parallel \mathcal{I}_2$ es una tupla $(S, A, \mathcal{C}, \rightarrow, (s_0^1, s_0^2), \mathcal{C}_0)$ donde:

- $S = S_1 \times S_2$,
- $A = A^{\mathcal{O}} \uplus A^{\mathcal{I}}$ t.q. $A^{\mathcal{O}} = A_1^{\mathcal{O}} \uplus A_2^{\mathcal{O}}$ y $A^{\mathcal{I}} = (A_1^{\mathcal{I}} \uplus A_2^{\mathcal{I}}) \setminus A^{\mathcal{O}}$,
- $\mathcal{C} = \mathcal{C}_1 \uplus \mathcal{C}_2$,
- $\mathcal{C}_0 = \mathcal{C}_0^1 \uplus \mathcal{C}_0^2$,

- $\to$ es la menor relación definida por las siguientes reglas:

$$\frac{s_1 \xrightarrow{C,a,C'}_1 s_1'}{(s_1,s_2) \xrightarrow{C,a,C'} (s_1',s_2)}\ a \in A_1 \backslash A_2 \qquad \frac{s_2 \xrightarrow{C,a,C'}_2 s_2'}{(s_1,s_2) \xrightarrow{C,a,C'} (s_1,s_2')}\ a \in A_2 \backslash A_1$$

$$\frac{s_1 \xrightarrow{C_1,a,C_1'}_1 s_1' \quad s_2 \xrightarrow{C_2,a,C_2'}_2 s_2'}{(s_1,s_2) \xrightarrow{C_1 \cup C_2, a, C_1' \cup C_2'} (s_1',s_2')}$$

La Definición 21 provee reglas estructurales para construir la composición paralela de dos IOSA compatibles, pero no nos dice si la tupla resultante es ella misma un autómata estocástico con entrada/salida. Para ello [DLM16] demuestra que las restricciones de la Definición 17 también son satisfechas por $\mathcal{I}_1 \parallel \mathcal{I}_2$.

Teorema 8 (IOSA es cerrado para la composición paralela, [DLM16])**.** *Sean $\mathcal{I}_1$ y $\mathcal{I}_2$ dos IOSA compatibles. Entonces $\mathcal{I}_1 \parallel \mathcal{I}_2$ también es un IOSA.*

Un *IOSA cerrado* es un autómata estocástico con entrada/salida resultante de la composición paralela de dos o más IOSA, donde toda sincronización entre acciones ha sido resuelta. La Definición 21 nos garantiza que un IOSA cerrado no tendrá acciones de entrada, i.e. $A^{\mathcal{I}} = \varnothing$.

[DLM16] muestra a su vez que un IOSA cerrado es determinista, lo cual los vuelve analizables desde el punto de vista de la simulación por eventos discretos. Un IOSA es determinista si (casi seguramente) a lo sumo una transición discreta puede habilitarse en cada instante de tiempo. Equivalentemente, [DLM16] llama determinista a un IOSA que *casi nunca* alcanza un estado donde dos transiciones discretas distintas están habilitadas. La formalización de este concepto, que proveemos a continuación, requiere que recurramos a la semántica NLMP del autómata.

Definición 22 (IOSA deterministas)**.** Un IOSA $\mathcal{I}$ es *determinista* cuando en su semántica $\mathcal{P}(\mathcal{I}) = (\boldsymbol{S}, \mathscr{B}(\boldsymbol{S}), \{\mathcal{T}_a \mid a \in \mathcal{L}\})$, un estado $(s, \vec{v}) \in \boldsymbol{S}$ t.q. el conjunto $\bigcup_{a \in A \cup \{\mathsf{init}\}} \mathcal{T}_a(s, \vec{v})$ contiene más de una medida de probabilidad, casi nunca es alcanzado desde cualquier estado inicial $(\mathsf{init}, \vec{w}) \in \boldsymbol{S}$.

Por *casi nunca* [DLM16] quiere decir que la medida del conjunto de caminos de ejecución que llevan hacia un estado $(s, \vec{v}) \in \boldsymbol{S}$ donde $1 < |\bigcup_{a \in A \cup \{\mathsf{init}\}} \mathcal{T}_a(s, \vec{v})|$ es nula. Asimismo, la Definición 22 requiere que el NLMP $\mathcal{P}(\mathcal{I})$ satisfaga las nociones de *aditividad temporal*, *determinismo temporal*, y *progreso máximo* [Yi90]. Se demuestra que esto es así en [DLM16]. En particular, progreso máximo significa que cuando una transición de salida está habilitada, el tiempo no puede avanzar en el estado, sino que se debe ejecutar dicha salida.

Para aplicar las técnicas de división de simulaciones de esta tesis al formalismo de modelado IOSA, será necesario trabajar exclusivamente con sistemas que satisfagan la Definición 22. Por fortuna, [DLM16] demuestra que ni bien la sincronización de las acciones ha sido completamente resuelta, el "IOSA compuesto" resultante efectivamente es determinista.

Teorema 9 ([DLM16])**.** *Todo IOSA cerrado es determinista.*

Este capítulo está dedicado al desarrollo de técnicas que exploten la naturaleza composicional de un modelo de sistema. El Teorema 9 nos permite escoger a los autómatas estocásticos con entrada/salida como el formalismo de modelado que usaremos para verificar la eficiencia de los métodos introducidos.

4.5. Automatizaciones y herramientas de soporte

La teoría y estrategias generales que sirven de base a nuestro método composicional ya fueron cubiertas en las secciones 4.1 a 4.4. Esta sección estudia algunos aspectos prácticos, con el fin de desarrollar herramientas de software que implementen estos métodos.

4.5.1. Selección de los umbrales

En los casos de estudio presentados a lo largo de la Sección 3.5, el mecanismo de selección de umbrales fue el objeto de muchas críticas. Esto es un tanto decepcionante puesto que el algoritmo implementado en la herramienta BLUEMOON, es decir la División Multinivel Adaptativa, tiene la ventaja de poder *moldear dinámicamente* la selección al sistema particular estudiado.

Nuestra implementación de la División Multinivel Adaptativa fue presentada en el Algoritmo 2. It runs pilot simulations on the system model $\mathcal{M}$, whose statistical evaluation w.r.t. the importance values observed yielded the threshold importance values used later by RESTART. From the theoretical viewpoint, this complemented nicely the static analysis of the system and of the rare event usado por el Algoritmo 1 para derivar la función de importancia.

A pesar de su adaptabilidad, el Algoritmo 2 proved to be quite sensitive to the global splitting value selected, and even to the particular simulation run, characterized by the seed fed to the Random Number Generator. In occasions re-running the experiment produced better results, viz. faster convergence, relacionada a una elección diferente de los umbrales (ver e.g. la Subsección 3.5.3).

Esto sugiere que las propiedades estadísticas del algoritmo are not optimal. In that respect recall that the subroutine $\mathcal{M}.\texttt{simulate\_ams}(s, n, m, f, \texttt{sim})$ is called once per iteration of the main loop, in order to select a threshold with higher importance than the one previously selected. That routine launches n simulations from state s with predetermined lifetime $m \in \mathbb{R}_{>0}$. The outcomes of these simulations, en términos de los valores de importancia medidos por la función f, determinan el siguiente umbral.

Nótese que hacer que todas las n simulaciones start from the same state s introduces a potentially high correlation in the outcomes of the runs. This is recognised by [CDMFG12], who have developed a much more sound algoritmo (desde el punto de vista estadístico) llamado *Monte Carlo Secuencial*.

La principal diferencia entre Monte Carlo Secuencial y su antecesor División Multinivel Adaptativa yace en la selección de los estados de inicio en cada

iteración del ciclo principal. Con la idea de reducir la correlación entre las corridas resultantes, Sequential Monte Carlo chooses these states independently from among all the states in the system. The only condition they must comply to is having an importance value assigned por la función f mayor que el último umbral seleccionado.

En [CDMFG12] los estados son partículas generated from a Markov kernel. Thus, drawing n independent and identically distributed new particles to start simulations from, is only a matter of resampling from the kernel. In contrast, from our simulation perspective on IOSA models, it makes more sense to choose only among *reachable states*. Besides, since our scenario is discrete, we can afford to choose states to which function f le otorga *exactamente* el valor de importancia elegido para el umbral anterior.

Presentamos en el Algoritmo 5 el pseudocódigo simplificado de nuestra implementación de Monte Carlo Secuencial. We highlight that both the input and the output are the same than for our implementation of Adaptive Multilevel Splitting, viz. el Algoritmo 2. The only substantial difference between both algoritmos is that subroutine `choose_dist`$(\cdots)$ in Algoritmo 5 is used to select the starting states at each iteration of the main loop, and that `simulate_smc`$(\cdots)$ starts from n potentially different states, em lugar de un único estado como lo hace `simulate_ams`$(\cdots)$.

En particular, $\mathcal{M}$.`choose_dist`$(\cdots)$ toma el arreglo de estados `sim` $\in S^{n+k}$ as one of its inputs. The general idea is to use the first n positions of `sim` to find the new thresholds, storing there the states resulting from the pilot runs launched with that purpose. Instead, the last k positions of `sim` will hold the states which have importance equal to the last thresholds found. Thus at each step, n pilot runs will be launched. These will start from states randomly sampled from the last k positions of `sim`, i.e. from a random sample of the last threshold found. The resulting states of those n simulations, i.e. those which achieved maximum importance, will be stored in the first n positions of `sim`, para revisar si se ha hallado un nuevo umbral.

Aún más en detalle, la subrutina $\mathcal{M}$.`choose_dist`$(\texttt{sim}, n, k, t, f)$ realiza n independent simulations, which run until a state to which $f\colon S \to \mathbb{N}$ assigns importance $t \in \mathbb{N}$ is found. The starting states for these simulations *are randomly chosen* from the states at position $n, n+1, \ldots, n+k-1$ of `sim`. Upon reaching a state with importance equal to t, each of the n simulations stops and saves such state in the corresponding i-th position of `sim`, for $i \in \{0, 1, \ldots, n-1\}$. Finally, k states from among those n states are randomly selected and copied into positions $n, n+1, \ldots, n+k-1$ of `sim`, que serán usados como estados iniciales en la próxima invocación de `choose_dist`$(\cdots)$.
Tres aclaraciones adicionales ayudarán a comprender mejor al Algoritmo 5:

1. `sort`$(\texttt{sim}, f, i, n)$ ordena los estados del arreglo `sim` que están en las posiciones $i, i+1, \ldots, i+n-1$, en orden ascendente de acuerdo a los valores que la función $f\colon S \to \mathbb{N}$ les asigna;

2. $\mathcal{M}$.`simulate_smc`$(\texttt{sim}, n, m, f)$ opera como `simulate_ams`$(\cdots)$ del Algoritmo 2, con la diferencia de que lanza n simulaciones desde los estados

Algoritmo 5 Selección de los umbrales con Monte Carlo Secuencial.

Entrada: módulo $\mathcal{M}$
Entrada: función de importancia $f\colon S \to \mathbb{N}$
Entrada: setup de las simulaciones $k, n, m \in \mathbb{N}_{>0},\ k < n$

```
Var: sim[n + k]   Tipo: arreglo de estados
Var: T            Tipo: cola de enteros                    {los umbrales}
sim[0, 1, ..., n + k − 1] ← M.initial_state()
T.push(f(sim[0]))
repeat
  M.simulate_smc(sim, n, m, f)
  sort(sim, f, 0, n)
  if T.back() < f(sim[k]) then
    T.push(f(sim[k]))                          {nuevo umbral hallado}
    M.choose_dist(sim, n, k, T.back(), f)
  else
    break loop                  {no se halló un umbral más alto}
  end if
until T.back() = máx(f)
choose_remaining(T, f)
```

Salida: la cola `T` con los valores de los umbrales

en las posiciones $0, 1, \ldots, n-1$ del arreglo de estados `sim`, y luego deja en dichas posiciones los estados resultantes de dichas simulaciones;

3. cuando una iteración no logra encontrar un nuevo umbral superior, el ciclo principal se rompe (con la instrucción **break loop**) y la responsabilidad recae sobre `choose_remaining`$(\texttt{T}, f)$, quien al observar que `T.back`$() <$ máx(f) procede a seleccionar umbrales entre los valores `T.back`$()$ y máx(f) de acuerdo a alguna heurística de terminación garantizada.

El hecho de que el Algoritmo 5 termina luego de ejecutar un número finito de pasos sigue las mismas líneas de la Proposición 7, que prueba la terminación del Algoritmo 2. Por cuestiones de completitud incluimos un bosquejo de la prueba junto con su formulación formal.

Proposición 10 (Terminación del Algoritmo 5)**.** *Sea $\mathcal{M}$ un modelo IOSA finito, i.e. $\mathcal{M} = (S, A, \mathcal{C}, \to, s_0, \mathcal{C}_0)$ t.q. S y A son ambos finitos. Sea a su vez f una función de importancia con imagen en $\mathbb{N}$, y sean $k, n, m \in \mathbb{N}, k < n$. Entonces, tomando esas entradas, el Algoritmo 5 termina luego de ejecutar una cantidad finita de instrucciones.*

Demostración (bosquejo). Como S es finito y el ciclo principal selecciona un nuevo umbral *más alto* en cada iteración, el ciclo puede iterar una cantidad máxima finita de veces. El hecho de que `simulate_smc`$(\cdots)$ ejecuta un número

finito de pasos puede probarse de forma análoga a como se lo hizo para la rutina `simulate_ams`$(\cdots)$ en la Proposición 7. A su vez, al imponerle una cota máxima al número de pasos que puede realizar cada simulación lanzada por `choose_dist`$(\cdots)$, nos aseguramos que esa rutina también terminará luego de un número finito de pasos. Por último, `choose_remaining`$(\cdots)$ termina por hipótesis. □

4.5.2. Sintaxis de modelado IOSA

La Proposición 10 de la subsección anterior habla de caminos simulados sobre un IOSA finito, el formalismo de modelado que presentamos en la Sección 4.4. No obstante, hasta aquí sólo hemos presentado la teoría desarrollada en [DLM16]. Para elegir a IOSA como el lenguaje en el que expresaremos los modelos de nuestros sistemas, necesitamos una sintaxis concreta cuya gramática produzca autómatas acorde a la Definición 17.

Para ello hemos desarrollado la siguiente sintaxis[†], que presentamos informalmente al igual que hicimos con el lenguaje de entrada de PRISM en la Subsección 3.3.1. Las reglas de derivación de esta *sintaxis de modelado IOSA* son de hecho bastante parecidas a las de PRISM, con el añadido destacable de variables de tipo *reloj*, cuyos valores son muestreados de distribuciones estocásticas. A continuación proveemos una lista exhaustiva de las diferencias entre el lenguaje de entrada de PRISM como se lo usó en esta tesis, y la sintaxis de modelado IOSA que usaremos en las próximas experimentaciones de la secciones:

- en el entorno global sólo pueden definirse constantes, propiedades, y módulos;
- las constantes deben ser o bien de tipo booleano, o de tipo entero, o sino de tipo de punto flotante;
- las propiedades pueden ser especificadas en un archivo independiente, o sino pueden incluirse en al archivo con el modelo, dentro de un entorno **`properties...endproperties`**;
- las consultas de propiedades deben especificarse una por línea y son
  - ▷ *transitorias*, siguiendo el formato `P( !parar U raro )`, o sino
  - ▷ *estacionarias*, siguiendo el formato `S( raro )`,

  donde `parar` y `raro` son expresiones de valor booleano que representan respectivamente las condiciones de parada (viz. truncado de simulaciones) y de evento raro;
- las variables sólo pueden aparecer dentro del cuerpo de un módulo, viz. encerradas en un entorno **`module...endmodule`**;

[†] Mi amigazo y colega Raúl E. Monti y mi director Pedro R. D'Argenio fueron los principales responsables de esta tarea; el crédito (y la culpa) debería atribuírsele a ellos.

- las variables deben ser o bien de tipo booleano, o de tipo entero (acotado), o sino de tipo reloj;
- cada variable de tipo reloj debe estar mapeada a una función de probabilidad continua, y sólo se le pueden asignar valores aleatoriamente escogidos, que resulten del muestreo de tal función;
- las etiquetas no vacías en las aristas de un módulo deben aparecer decoradas o bien con el símbolo `?` para indicar que es una acción de entrada (y por ende una *arista de entrada*), o sino con el símbolo `!` para indicar que es una acción de salida (resp. una *arista de salida*);
- una etiqueta vacía simboliza un arista de salida no sincronizante;
- una guarda booleana vacía en una arista es interpretada como `true`;
- un símbolo `->` seguido inmediatamente por un punto y coma se interpreta como `NOP`, i.e. la ausencia de acciones;
- además de una guarda booleana, las aristas de salida deben declarar el nombre de un reloj entre el caracter `@` y el símbolo `->`, que relaciona a dicha variable de tipo reloj con las transiciones de salida concretas representadas por la arista.

Para darle al lector una idea concisa del aspecto de esta sintaxis, mostramos a continuación el extracto de un modelo descripto con ella. El sistema representado es la cola tándem modularisada que introdujimos originalmente en la Código 3.3. Como todas las variables de tipo reloj del modelo de donde extrajimos el Código 4.3 están mapeadas a la distribución exponencial, el modelo IOSA resultante es equivalente a un CTMC.

Código 4.3: Modelo IOSA para la cola tándem (extracto)

```
1  const int c = 8;  // Capacidad de ambas colas
     ⋮
14 module Arribos
15     clk0: clock;  // Arribos externos ~ Exponencial(lambda)
16     [P0!] @ clk0 -> (clk0'= exponential(lambda));
17 endmodule
18
19 module Cola1
20     q1: [0..c];
21     clk1: clock;  // Procesamiento en la Cola1 ~ Exponencial(mu1)
22     // Arribo de paquete
23     [P0?] q1 == 0          -> (q1'= q1+1) & (clk1'= exponential(mu1));
24     [P0?] q1  > 0 & q1 < c -> (q1'= q1+1);
25     [P0?] q1 == c          -> ;
26     // Procesamiento de paquete
27     [P1!] q1 == 1 @ clk1 -> (q1'= q1-1);
28     [P1!] q1  > 1 @ clk1 -> (q1'= q1-1) & (clk1'= exponential(mu1));
29 endmodule
```

```
    ⋮
43 properties
44     P( q2 > 0 U q2 == c)  // transitoria
45     S( q2 == c )          // estacionaria
46 endproperties
```

Nótese que la única arista del módulo `Arribos` en la línea 16 del Código 4.3, es una arista de salida dado que su acción `P0` está decorada con '`!`'. Su guarda booleana (entre los caracteres '`]`' and '`@`') está vacía y por ende equivale a `true`. A su vez, esta arista de salida está asociada al reloj `clk0`, el cual está mapeado a una función de densidad exponencial con tasa `lambda`.

La acción de salida `P0` del módulo `Arribos` synchronises with the input action `P0` from module `Cola1`, i.e. with any of the edges in lines 23–25. The boolean guards of those edges form a partition of the range of the integral variable `q1`. Therefore, the output edge of module `Arribos` is always enabled *from a logical point of view*, and will synchronise with exactly one of the input edges in lines 23–25. *From a temporal point of view* and by definition of IOSA, la arista de salida es habilitada cuando el reloj `clk0` expira.

Este mecanismo de sincronización es similar al del modelo PRISM para la cola tándem del Código 3.3. Esto es natural, puesto que la sintaxis de modelado IOSA se inspiró originalmente en el lenguaje de entrada de PRISM.

Nótese que los efectos de una arista, i.e. the consequences of taking the edge which are described after the symbol `->`, appear enclosed in parentheses and concatenated with the character `&`. For instance the input edge in line 23 has two effects: incrementing by one the value of the integral variable `q1`, and assigning a fresh random value to the clock variable `clk1`, muestreado de un función de densidad de probabilidad con distribución exponencial de tasa `mu1`.

Llamamos la atención a la arista de entrada en la línea 25 del Código 4.3. This edge has an empty effect, viz. a semicolon appears immediately following the symbol `->`. This is interpreted as a `NOP` or `SKIP`, that is, the absence of an effect. Semantically that edge represents a packet trying to enter a fully occupied first queue, el cual es inmediatamente descartado.

En el Código 4.3 las consultas de propiedades están en el mismo archivo del modelo, y por ende aparecen encerradas en un entorno **`properties`**. . . **`endproperties`**. la propiedad de la línea 44 es de tipo transitorio, y pregunta por la probabilidad de observar una segunda cola saturada antes de que dicha cola se vacíe. La propiedad de la línea 45 es de tipo estacionario, y pregunta por la proporción de tiempo que la segunda cola permanece saturada, viz. la probabilidad a largo plazo de observar una saturación en dicha cola.

4.5.3. Generador de Improbabilidad Finita

Desarrollamos la herramienta de software FIG, que implementa la estrategia composicional para división multinivel que describimos en este capítulo. Está escrita exclusivamente en C++ y es software autónomo. El nombre completo de la herramienta es *Generador de Improbabilidad Finita*, en honor a la obra

maestra de Douglas Adam. La herramienta FIG está disponible libremente en http://dsg.famaf.unc.edu.ar/tools bajo los términos de la licencia Pública General de GNU (GPL v3).

Los archivos con el modelo y las consultas tomados como entrada deben ser especificados siguiendo la sintaxis de modelado IOSA descripta en la Subsección 4.5.2. De aquí en más asumiremos que las consultas de propiedades se encuentran especificadas dentro del archivo donde se describe al modelo.

Destacamos el hecho de que FIG también puede tomar (ciertos tipos de) modelos descriptos en el *formato de especificación de modelos jani* versión 1.0 (JANI, [BDH+17]). Por un lado el formalismo IOSA abarca al de CTMC, así que la herramienta puede tomar cadenas de Markov de tiempo continuo descriptas en JANI. Por otro lado los Autómatas Estocásticos Temporizados (STA por sus siglas en inglés) generalizan a IOSA, y FIG puede operar sobre cierta clase de modelos STA descriptos en JANI. Específicamente, modelos STA carentes de no determinismo que concuerden con la Definición 17 pueden ser aceptados por la herramienta.

La invocación básica de FIG requiere tres opciones obligatorias: `<modelo>`, `<terminación>`, y `<estrategia>`. Su sintaxis y semántica puede ser descripta brevemente como sigue:

`<modelo>` El archivo con la descripción del modelo IOSA (o JANI) y las consultas de propiedades.

`<terminación>` El criterio (o criterios) de parada. Puede ser

- `--stop-conf` para simular hasta que el coeficiente de confianza y la precisión relativa especificadas sean alcanzados. Esos dos parámetros deben ser números en el intervalo abierto $(0, 1)$;
- `--stop-time` para simular por el tiempo (pared) especificado, descripto en el formato `<dígito>+[<s/m/h/d>]`.

`<estrategia>` La estrategia usada para simular. Debe ser una de:

- `--flat` para realizar simulaciones de Monte Carlo estándar;
- `--amono` para realizar simulaciones RESTART usando una función de importancia monolítica, construida usando el método del Capítulo 3;
- `--acomp` para realizar simulaciones RESTART usando una función de importancia composicional, construida usando el método de este capítulo con un operador (o estrategia) de composición provisto como argumento obligatorio;
- `--adhoc` para realizar simulaciones RESTART usando una función de importancia ad hoc especificada por el usuario como argumento obligatorio.

El orden de las opciones es arbitrario. Por ejemplo la línea

```
>_ fig modelo.sa --flat --stop-conf .9 .4
```

invoca a la herramienta para realizar simulaciones de Monte Carlo estándar en el archivo IOSA `modelo.sa`. Para cada consulta de propiedad especificado dentro de dicho archivo, las simulaciones correrán hasta que se haya construido un intervalo de confianza alrededor del valor estimado para la propiedad, con un nivel de confianza del 90 % y una precisión relativa del 40 % (i.e. un error relativo del 20 %).

Ambos `<modelo>` y `<estrategia>` son opciones simples, mientras que `<terminación>` es una opción múltiple. Esto significa que pueden especificarse varios criterios de parada, y se lanzarán corridas independientes para satisfacer cada uno de ellos. Por ejemplo

```
>_ fig modelo.sa --amono --stop-time 5m --stop-conf .9 .4
```

lanza, para cada consulta de propiedad especificada dentro de `modelo.sa`: primero una estimación que durará 5 minutos (pared) de ejecución; y luego una estimación que correrá hasta haber construido un intervalo alrededor de la estimación con un nivel de confianza del 90 % y una precisión relativa del 40 %.

Como la opción `--amono` was specified in the command above, all those simulations will use multilevel splitting. Like BLUEMOON, FIG implements RESTART to perform splitting simulations. The `--amono` option makes FIG build an (*automatic*) monolithic importance function for each property, subsecuentemente usado en las simulaciones RESTART.

A diferencia de BLUEMOON, la herramienta FIG does not require to be told which kind of simulation (e.g. transient vs. steady-state) to run for each property query. This is deduced from the logical expression: for transient properties several independent simulations are launched, whose average yields the desired estimate; para propiedades transitorias se utiliza el método de *batch means*.

Como se dijo, para escoger la estrategia `--adhoc` el usuario debe proveer como parámetro la función de importancia que desea usar. Esta función debe tener imagen en los números naturales, y cualquier constante o variable del modelo puede aparecer en la expresión aritmética que la define. Por ejemplo, si `tandem_queue.sa` contiene el modelo IOSA de la cola tándem del cual extrajimos el extracto presentado en el Código 4.3, entonces

```
>_ fig tandem_queue.sa --adhoc "q1+5*q2" --stop-conf .9 .4
```

ejecutará simulaciones RESTART para estimar el valor de ambas propiedades (recall these were `P( q2>0 U q2=c )` and `S( q2=c )`), using the ad hoc importance function which adds five times the number of packets in the second queue to el número de paquetes en la primera cola, i.e. `q1+5*q2`.

La situación es bastante diferente para la opción `--acomp`, aunque ella también toma un parámetro obligatorio. Se puede escoger un operador de composición, por ejemplo

```
>_ fig tandem_queue.sa --acomp "+" --stop-conf .9 .4
```

invoca a la herramienta para realizar estimaciones como se describió antes, sólo que usando una función composicional (*automática*) para las simulaciones RESTART. Addition is specified as composition operand, meaning the global importance function used will be `A+Q1+Q2`, where '`A`', '`Q1`' and '`Q2`' stand for the local importance functions built for modules `Arrivals`, `Queue1` and `Queue2` respectively. Recall these local functions are built anew for each property studied, dado que la definición del evento raro podría diferir entre consultas.

En lugar de un operando de composición, the `--acomp` option can also take an ad hoc composition strategy, defined by the user in the way described in Subsección 4.3.1. It is worthy to mention that unlike the `--adhoc` option, the arithmetic expression passed as parameter to `--acomp` must contain names of modules of the system, que son interpretadas como las funciones locales de importancia de los módulos. Esto significa que el comando

```
>_ fig tandem_queue.sa --acomp "Arrivals+Queue1+Queue2" \
     --stop-conf .9 .4
```

imita a la invocación anterior, que había utilizado a la suma como operador de composición.

Pero lo más importante es que esta utilidad permite una implementación sencilla y directa de la estrategia de anillo/semianillo de la Subsección 4.3.3. Digamos por ejemplo que la cola tándem triple que habíamos modelado con el lenguaje de entrada PRISM en el Código 4.2, se encuentra descripta con la sintaxis de modelado IOSA en el archivo `3tandem_queue.sa`. Suponiendo que los nombres de los módulos no fueron modificados, el comando

```
>_ fig 3tandem_queue.sa  \
     --acomp "Cola1+max(Cola2,Cola3)" --stop-time 2h
```

usa el semianillo $(\text{máx}, +)$ como se describió en la Subsección 4.3.3 para componer las funciones locales de importancia construidas para los módulos de las colas.

Hasta aquí hemos discutido la especificación de la función de importancia, but multilevel splitting simulations (i.e. whenever the `--adhoc`, `--amono`, or `--acomp` strategy option is specified) also require selecting the thresholds prior to running RESTART. To that aim and by default FIG runs Algoritmo 5, using as input the importance function built for the current property under study. This can be changed with the `--thresholds` option to use Adaptive Multilevel Splitting, a fixed (e.g. non-adaptive) strategy, or pure Sequential Monte Carlo. By default the tool utilises a "hybrid thresholds selection strategy" (viz. Algoritmo 5): upon a failure of Sequential Monte Carlo the subroutine `choose_remaining`$(\cdots)$ is invoked, que implementa una estrategia no adaptativa de terminación garantizada.

FIG puede tomar varias opciones adicionales para personalizar los mecanismos de estimación. The full list together with their invocation syntax and some practical examples is obtained calling the tool with the single argument `--help`.

We next describe the most relevant among these options, two of which were used to run the experiments reported in Sección 4.6. Also and from here onward, estimations whose termination is specified using the `--stop-conf` option will be referred to as *estimaciones acotadas por confianza* or merely *estimaciones por confianza*. Analogously *estimaciones acotadas por tiempo* or simply *estimaciones por tiempo* will refer to estimations cuya terminación fue especificada por medio de la opción `--stop-time`.

Independientemente del criterio de parada escogido, a maximum wall-clock execution time can be imposed by means of the option `--timeout`, which takes the same mandatory parameter than the `--stop-time` option. The `--timeout` option is very useful for confidence-bound estimations, when one has no idea whatsoever how long the estimation could take. Instead for time-bound estimations, if the `--timeout` option is given entonces las simulaciones durarán por el lapso más corto de los dos.

Remarcamos que el formato de la salida de las estimaciones por tiempo (o de aquellas que hicieron timeout) difiere del formato de las estimaciones por confianza. Si una estimación se detuvo porque alcanzó el nivel de confianza y la precisión relativa deseadas para el intervalo, la salida final se verá aproximadamente así:

```
~~~~~~~~~
 · FIG ·
~~~~~~~~~
This is the Finite Improbability Generator.
Version: 1.1
Build:   Release
    ⋮
RNG algorithm used: pcg32
Estimating P( (q2>0) U (q2==8) ),
 using simulation engine  "restart"
 with importance function "concrete_coupled"
 built using strategy     "auto"
 with post-processing     "(null)"
 and thresholds technique "hyb"
 [ 2 thresholds | splitting 5 ]
   Confidence level: 80%
   Precision: 40%
   RNG seed: 1944391357620130122 (randomized)
    · Computed estimate: 5.34e-06 (7344384 samples)
    · Computed precision: 1.67e-06
    · Precision: 2.13e-06
    · Confidence interval: [ 4.27e-06, 6.40e-06 ]
    · Estimation time: 29.04 s
```

Resaltamos que hay una *precisión computada* (“`Computed precision`”) y una *precisión a secas* (“`Precision`”). La primera es el ancho empírico del intervalo obtenido al usar las técnicas del Subsección 3.3.4. La segunda es la precisión relativa (*teórica*) solicitada, que en este caso es igual a $5.34\text{e-}06 \times 0{,}4 = 2.13\text{e-}06$.

Sin en cambio las estimaciones finalizan por razones temporales, no habrá ninguna precisión teórica que reportar. En tales casos se muestra una serie de intervalos de confianza, construidos con los datos generados para niveles de confianza típicos, e.g.

```
>_  ⋮
[ 2 thresholds | splitting 5 ]
  Confidence level: 80%
  Precision: 40%
  Timeout: 00:00:10
  RNG seed: 17463016430344695793 (randomized)
  · Computed estimate: 6.54e-06 (2508288 samples)
  · 80% confidence
      - precision: 3.00e-06
      - interval: [ 5.04e-06, 8.04e-06]
  · 90% confidence
      - precision: 3.86e-06
      - interval: [ 4.61e-06, 8.47e-06]
  · 95% confidence
      - precision: 4.59e-06
      - interval: [ 4.24e-06, 8.84e-06]
  · 99% confidence
      - precision: 6.04e-06
      - interval: [ 3.52e-06, 9.56e-06]
  · Estimation time: 10.00 s
```

Es importante remarcar que según el Algoritmo 4, las funciones construidas automáticamente usarán el cero como mínimo valor de importancia. This means that the local initial state of a module is assigned the value 0 by its local importance function, which can be problematic for the compositional approach if the product is used either como operando de composición o en una estrategia de composición.

Tomemos por ejemplo el comando

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6
```

que especifica al producto como operador de composición para usar con el método composicional. Say the rare event is a a saturation in both queues. Then e.g. the global importance function will yield the value 0, as long as the first queue is in its initial local state and regardless of the occupancy in the second queue. This is clearly at odds with the desired behaviour. Nos hemos referido a este hecho anteriormente en la Subsección 4.3.4 como el problema de nulificación.

Principalemte a raíz de este inconveniente FIG offers a `--post-process` option to modify the importance values once they have been computed. The local importance values can thus be increased or exponentiated como se describe a continuación, resolviendo el problema que causa el hecho de que cero sea el

elemento absorvente de $*$.

Retomando la situación anterior, el comando

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6  \
     --post-process shift 1
```

incrementa en uno los valores locales de importancia de todos los módulos del sistema. Eso significa que el menor valor devuelto por una función local de importancia será 1 en lugar de 0. Por consiguiente el problema de nulificación queda resuelto, dado que la primera cola tendrá importancia 1 cuando esté en su estado inicial.

En más detalle, la opción `--post-process` toma dos parámetros:

`<tipo>` el tipo de posprocesamiento que se aplicará, el cual puede ser o bien `shift` para aumentar/disminuir la importancia de cada estado por algún valor, o `exp` para aplicar potenciación usando como exponente la importancia del estado;

`<arg>` el argumento numérico a usar, que para `shift` puede ser cualquier constante entera (el valor por el que se aumentará/disminuirá cada valor), y para `exp` debe ser un número de punto flotante mayor a 1,0 (la base a usar en la potenciación).

Para ilustrar el uso de la potenciación consideremos el comando

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6  \
     --post-process exp 2.0
```

Esto significa que en cada módulo, el valor de importancia i de cada estado local será reemplazado por el valor 2^i. Nuevamente esto alcanza para solucionar el problema de nulificación, puesto que $2^0 = 1$ y por ende el menor valor de importancia en cualquier cola no será cero.

Esta subsección concluye discutiendo la interacción de la herramienta con el formato de especificación de modelos jani desarrollado por [BDH+17]. FIG puede operate as bidirectional translator between IOSA and JANI, for which the options `--to-jani` and `--from-jani` are offered. When these are specified the tool assumes the translation role and runs no estimation. Nonetheless FIG can be fed an (IOSA-compatible) JANI model as system model description, and upon a successful implicit translation the estimations will be carried out as usual. Entonces por ejemplo sea `tandem_queue.jani` un archivo con el modelo JANI correspondiente a la cola tándem modularisada que venimos tratando. El comando

```
>_ fig tandem_queue.jani --amono --stop-conf .95 .6
```

invoca a la herramienta para realizar las simulaciones RESTART usuales empleando la función de importancia monolítica, y las estimaciones concluirán una vez que se haya logrado construir un intervalo con un nivel de confianza del 95 % y un error relativo del 60 %.

En lo referido a la traducción, como ya se dijo sólo CTMC y ciertos tipos de Autómatas Estocásticos Temporizados deterministas (STA) coinciden con el formalismo IOSA. Cuando se exporta hacia JANI con la opción `--to-jani`, un STA compatible con IOSA es generado. Cuando se importa desde JANI con la opción `--from-jani` es necesario realizar muchos chequeos. Los más básicos incluyen la ausencia de variables globales y el uso de una estrategia de sincronización *tipo broadcast* como la de IOSA. Cualquier modelo CTMC que satisfaga estas condiciones debería ser aceptado. Los STA son más complicados ya que permiten varias manipulaciones de los relojes que los Autómatas Estocásticos en general desconocen (e.g. asignarle un valor preciso a un reloj), y además permiten varios relojes habilitantes por arista. Por ende la restricciones de la Definición 17 deben ser cotejadas con cuidado cuando se importa un modelo JANI de STA:

1. primero se chequean a nivel sintáctico las restricciones (a) y (b),
2. luego se construye un modelo IOSA tentativo,
3. finalmente se evalúan las restricciones (c) a (g)

Si estas pruebas son superadas con éxito, i.e. si el modelo resultante descripto en la sintaxis IOSA satisface la Definición 17, un archivo de modelo (IOSA) es producido como salida. En caso contrario se muestra un mensaje de error y la traducción aborta.

4.6. Casos de estudio

Varios sistemas fueron tomados de la bibliografía de RES y analizados con FIG. La descripción general de estos sistemas y el resultado de los experimentos se muestran en esta sección. Los autómatas estocásticos con entrada/salida usados a este propósito se encuentran listados en el Apéndice A.

4.6.1. Entorno de experimentación

Todos los modelos aquí estudiados se encuentran descriptos en la sintaxis de modelado IOSA. Algunos casos de estudio markovianos analizados en la Sección 3.5 son reproducidos en esta sección. Estas pruebas sirvieron para validar el correcto funcionamiento de la herramienta FIG. Sistemas no markovianos, que usan relojes asociados e.g. a la distribución log-normal, también son estudiados en la presente sección.

Siguiendo el formato general de la Sección 3.5 we launched independent experiments for each case, perfecting an interval around a point estimate until some convergence or time criterion was reached. Some experiments ran until meeting a confidence criterion, or were truncated upon exceeding an execution timeout; in these cases the measure of interest is the speed of convergence. Notice this was the setting for all experiments in Sección 3.5. Other experiments ran for a predefined execution time bound; in these cases the measure of interest is the precision achieved, donde el objetivo es construir el intervalo más angosto posible.

Se emplearon dos sistemas computacionales. El servidor *JupiterAce* posee un procesador Intel Xeon E5-2620v3 con 12 núcleos de 2.40GHz, with 128 GiB 2133MHz of available DDR4 RAM. Los nodos del cluster *Mendieta* count instead with 8-cores 2.70GHz Intel Xeon E5-2620 processors, each with access to a DDR3 RAM memory of 32 GiB and 1333MHz. For each case study especificamos si los cálculos fueron realizados en JupiterAce o en Mendieta.

Algunos sistemas markovianos fueron importandos del capítulo previo and analysed in the same way, that is, convergence was tested for decreasing values of the rare event probability γ. Notice this includes the database system with redundancy, which could not be evaluated thoroughly en el Capítulo 3 debido a las limitaciones de la estrategia monolítica.

En esta sección también introducimos dos sistemas no markovianos, whose analysis is carried out somewhat differently. The triple tandem queue we present in la Subsección 4.6.3 was tested for different configurations of its parameters, all of which yield roughly the same value of γ. Furthermore there are two variants of the oil pipeline system we study in Subsección 4.6.6: in one of them the system components fail according to exponentially distributed clocks; the other variant uses clocks which sample time de una distribución Rayleigh (i.e. una Weibull con parámetro de forma 2).

Cuando fue posible, evaluamos cuatro estrategias for each model and configuration: standard Monte Carlo, RESTART using ad hoc importance functions, RESTART using the monolithic importance function from Capítulo 3, and RESTART using the compositional approach from this capítulo. Also, when the system model from the literature could be imitated exactly, we checked the consistency of the confidence intervals obtained by comparing them to the published values. Otherwise we verified that all simulation strategies converged to similar values, e.g. corroborando si todos los intervalos producidos compartían una región común.

Presentamos gráficos y tablas que muestran o bien los *timepos de convergencia* o sino la *precisión de los intervalos* obtenidos, depending on whether the execution bound was *criterio de confianza* or *presupuesto de tiempo* respectively. Time measurements consider wall-clock time, including preprocessings like the compilation of the model files y la selección de los umbrales.

En algunos casos específicos los resultados evidencian a somewhat high sensitivity to the choice of seed fed to the Random Number Generator routine (RNG). This is exacerbated by the relatively low replication we could perform: three to four independent experiments were run for each configuration of each case study, which presents a risk from the statistic viewpoint. Therefore a per case replication was performed whenever needed to verify consistency. This, on top of previous studies like the ones from [BDM17], dan base a la solidez de las medidas que presentamos a lo largo de esta sección.

Se notará que en general, the simulation times obtained are longer than those presented before in la Sección 3.5, and also that the confidence convergence criteria is laxer. In that respect we highlight that a simulation step in PRISM involves accessing a matrix stored in memory, mientras que en FIG todos los

relojes de todos los módulos deben ser actualizados para realizar la misma tarea. Esta multiplicación de las instrucciones de punto flotante por paso es el precio que hay que pagar para poder manipular distribuciones arbitrarias.

Por sobre todo esto, FIG ha sido desarrollado (hasta ahora) con propósitos de corrección y no de eficiencia. Por ejemplo, la actualización de los intervalos es una tarea trivialmente paralelizable, a pesar de lo cual FIG lo hace de manera secuencial. Ésta y muchas otras optimizaciones podrían acelerar los tiempos de ejecución de forma significativa, pero caen fuera del alcance y de los objetivos de esta tesis. En ese sentido la herramienta aún puede ser considerada como prototípica.

En cualquier caso, todos los tiempos de convergencia presentados a lo largo de esta tesis, y en particular los de esta sección, buscan comparar la eficiencia entre las estrategias evaluadas en cada caso de estudio. Obtener tiempos de ejecución de menor magnitud con nuestras herramientas de software puede ser el objeto de futuras investigaciones.

4.6.2. Cola tándem

Repetimos el experimento previamente presentado en la Subsección 3.5.2, utilizando el modelo IOSA del Apéndice A.6 para correr simulaciones en Jupiter Ace. Recuérdese que este sistema consiste en una red Jackson en tándem con dos colas conectadas secuencialmente. Las tasas de arribo, primer servicio y segundo servicio son respectivamente $(\lambda, \mu_1, \mu_2) = (3, 2, 6)$. Evaluamos propiedades de tipo transitorio y de tipo estacionario.

Nótese que esta cola tándem es markoviana. Por ende, los resultados obtenidos con FIG para el modelo IOSA del Apéndice A.6 deberían coincidir con los producidos por PRISM para un modelo equivalente escrito en el lenguaje de entrada del model checker. Un modelo con estas características se presenta en el Apéndice A.7, con el cual se corroboró esta afirmación.

Análisis transitorio

La propiedad de interés es `P ( q2>0 U q2==c )`, i.e. la probabilidad de observar una segunda cola saturada antes de que ésta se vacíe, lo cual estimamos comenzando las simulaciones desde el estado $(q_1, q_2) = (0, 1)$. Evaluamos capacidades máximas de las colas $C \in \{8, 10, 12, 14\}$, para los cuales los valores de γ producidos por PRISM[†] son respectivamente 5.62e-6, 3.14e-7, 1.86e-8, and 1.14e-9. Las estimaciones fueron realizadas para un criterio IC $90\,|\,40$, es decir, FIG tuvo que construir un intervalo con un nivel de confianza del 90 % y un error relativo del 20 % para cada configuración. El timeout de ejecución fue 2.5 horas, dentro del cual FIG convergió para cada configuración generando intervalos que contenían a los valores reportados por PRISM.

Tres funciones de importancia diferentes were tested in the importance splitting simulations. La función denotada `amono` fue automáticamente construida por FIG

[†] Si el modelo del Apéndice A.7 está en `tandem.prism`, entonces el comando que utilizamos es: `prism tandem.prism -const c=8:2:14 -pf 'P=?[ q2>0 U q2=c ]'`.

usando el método monolítico del capítulo anterior. En cambio, `acomp` es la función construida following the compositional strategy, which in this case employed summation as composition operand. The third importance function tested with RESTART was the best ad hoc candidate from our previous studies, viz. counting the number of packets in the second queue, which we denote `q2`. Como antes, las simulaciones de Monte Carlo estándar son denotadas `nosplit`.

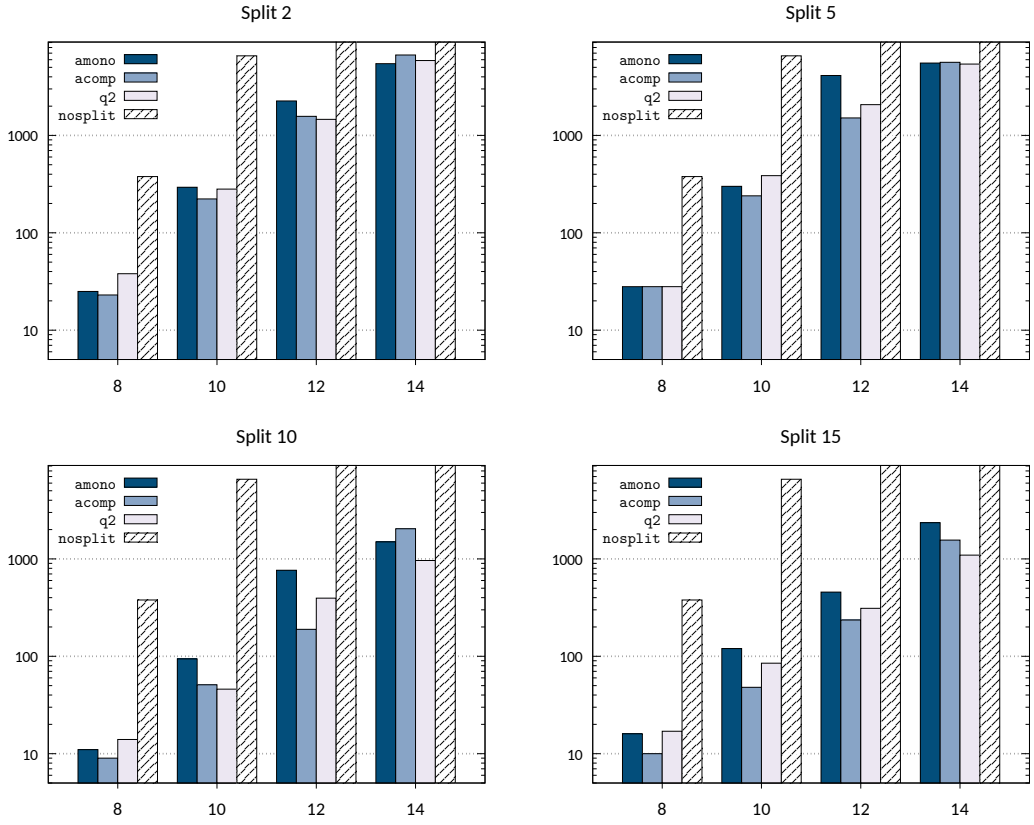


Figura 4.4: Tiempos para el análisis transitorio de la cola tándem

El promedio de los tiempos pared medidos en tres experimentos se muestran en la Figura 4.4. Recall we display one chart per splitting value, with the outcomes of the `nosplit` simulations repeated in all four charts. The maximum queue capacity C, tuned to variate the rarity of the event, se encuentra en el eje de las abscisas.

En correspondencia con nuestro estudio previo de este sistema, standard Monte Carlo simulations could converge within the time limit only for the two smallest values of C, and they were always the slowest. Contrarily, RESTART simulations converged in all settings, with no clear winner among the three functions. In several configurations the times of function `q2` resemble that of `acomp`. Notice that the rare event property involves solely a variable from the second queue. Hence the local importance function of the first queue is null, lo cual hace que `acomp` sea lógicamente muy similar a la función ad hoc.

La tendencia global de las simulaciones RESTART favorece los valores de división más grandes. La salida técnica de FIG revela que para cada valor de C,

el número de umbrales escogidos no aumenta necesariamente a medida que el valor global de división disminuye de 15 *a* 2. Esto es decididamente problemático, pues el hecho de contar con menos copias al cruzar un umbral hacia arriba debería ser contrarrestado mediante la selección de más umbrales. Creemos que este problema, de aquí en adelante denominado el *fiasco de umbrales-y-división*, está relacionado con las hipótesis de continuidad de la teoría que respalda al Algoritmo 5, y también puede estar influido por ciertos detalles implementativos de dicho algoritmo en FIG. Ahondaremos más en esta cuestión a lo largo de la presente sección.

A pesar de estos problemas, the plots show that the monolithic function was the most insensitive to the choice of global splitting value, whereas `acomp` and `q2` convergieron más velozmente para los valores de división 10 y 15.

Los gráficos también revelan un mal desempeño de `amono` w.r.t. `acomp` and `q2`, most notably for the splittings 10 and 15 when $C = 10$, and also for splittings 5, 10, and 15 when $C = 12$. This is a most unwelcome surprise, since the monolithic approach outperformed all ad hoc functions in the results previously presented in Subsección 3.5.2. As discussed in that subsección, a superior performance of the monolithic approach is expected in these kind of systems: the queues are interconnected, hence all of them might need to be considered cuando se deriva la función de importancia.

Corrimos nuevos experimentos para esas configuraciones, para ver si estos resultados estaban relacionados con el fiasco de umbrales-y-división, y para descartar la influencia de las semillas aleatorizadas que fueron alimentadas al RNG. Estos experimentos usaron otro mecanismo de selección de umbrales que ofrece FIG (i.e. no el Algoritmo 5), el cual en particular no es adaptativo. Ajustamos este mecanismo para asegurar que en este caso, una menor división global genere más umbrales escogidos. Los resultados, que detallamos en la Tabla 4.1, son destacables: la función monolítica superó a las otras dos en todas las corridas por factores que fueron desde 2x hasta 10x.

	$C = 10$		$C = 12$		
División:	15	10	15	10	5
Núm. Umbr.:	3	5	4	6	8
`amono`	71 s	41 s	164 s	224 s	68 s
`acomp`	161 s	93 s	566 s	407 s	665 s
`q2`	137 s	86 s	657 s	395 s	594 s

Tabla 4.1: Tiempos de convergencia para umbrales escogidos ad hoc

Los resultados de la Tabla 4.1 sugieren que la función monolítica es de hecho la que mejor se comporta en este sistema, y que los lentos tiempos de convergencia que se muestran en la Figura 4.4 son mayormente el resultado del fiasco de umbrales-y-división.

En cualquier caso esta experimentación logra mostrar que la estrategia com-

posicional introducida en este capítulo es plenamente operativa, y que incluso puede igualar otros métodos ad hoc muy eficientes; para lo cual no recurre a la expansión del espacio de estados del modelo compuesto, ni requiere que el usuario especifique una función de importancia explícitamente.

Análisis estacionario

En lo concerniente a simulaciones a largo plazo nos interesa la propiedad `S ( q2==c )`, i.e. la proporción de tiempo que la segunda cola pasa en un estado de saturación. Evaluamos capacidades máximas de las colas $C \in \{10, 13, 16, 18, 21\}$, para las cuales los valores de γ aproximados con PRISM[‡] son respectivamente 7.25e-6, 2.86e-7, 1.12e-8, 1.28e-9, and 4.94e-11. Las estimaciones con FIG debían converger dentro de las 6 horas de tiempo pared de ejecución logrando un IC $90\,|\,40$. Nuevamente corroboramos que estas estimaciones convergiesen a los valores producidos por PRISM. Empleamos las mismas funciones de importancia que en el caso transitorio.

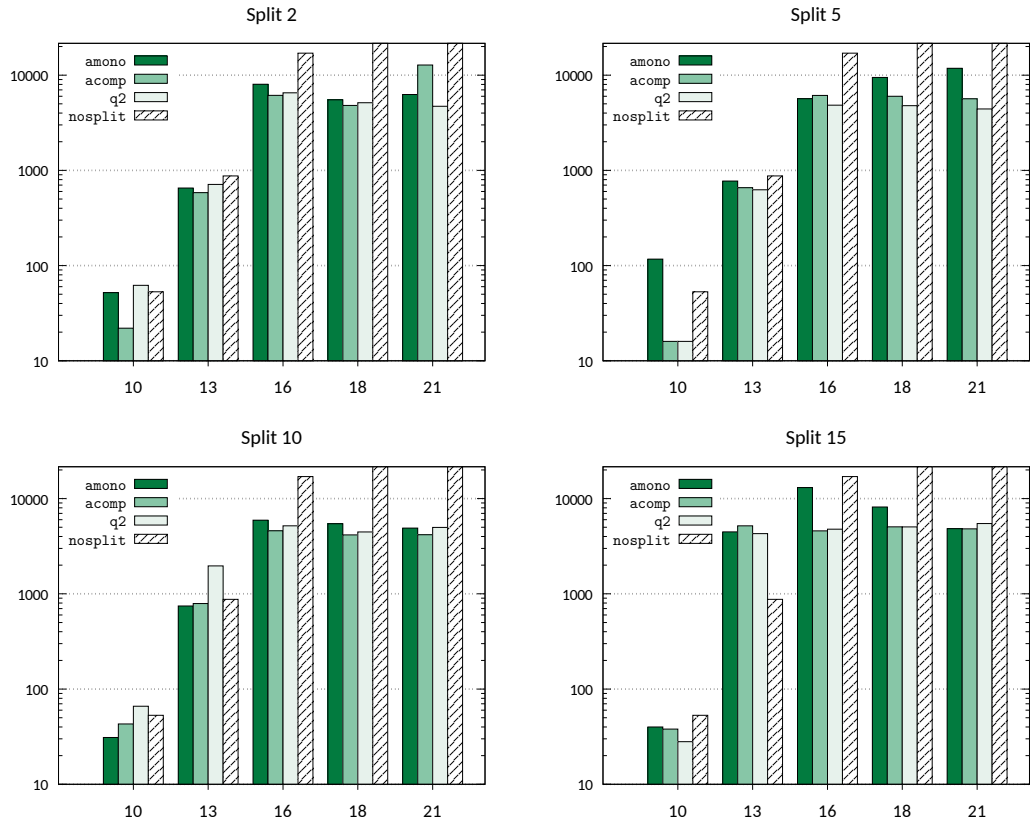


Figura 4.5: Tiempos para el análisis estacionario de la cola tándem

Los resultados obtenidos de un promedio entre tres experimentos se presentan en la Figura 4.5, siguiendo el mismo formato que en el caso transitorio. Las capacidades probadas para las colas (así como el límite de tiempo) difieren de

[‡] `prism tandem.prism -const c=10:3:21 -pf 'S=?[ q2=c ]' -jor`.

los que se usaron en la Subsección 3.5.2, pero aún así el comportamiento de las simulaciones de Monte Carlo estándar es bastante similar, convergiendo rasonablemente rápido sólo cuando $C < 15$. En contraste, ninguna de las simulaciones RESTART simulations falló en lograr los criterios de confianza dentro de los límites de tiempo.

Dejando de lado el caso de $C = 10$, which anyway is the least rare and thus the least interesting, convergence times are rather uniform for all importance functions and for splittings 2, 5, and 10. We would have expected a better performance of the monolithic approach w.r.t. `acomp` and `q2`, but as before we believe that the splittings-thresholds fiasco is creating a distortion. In particular this caused the anomaly of `amono` for splitting 15 and $C = 16$. En la Subsección 4.6.4 detallamos el problema específico detrás de este comportamiento.

Por desgracia, este problema no puede ser solucionado sistemáticamente sin llevar a cabo modificaciones profundas en los mecanismos de selección de umbrales de FIG. Tocamos este tema nuevamente en las conclusiones finales de la tesis.

A pesar de esto, the compositional approach performed very well, once again suggesting that our technique can automate the derivation of a high quality importance function, sin expandir el espacio de estados del modelo compuesto.

Como nota final notamos que la forma de los (importance splitting) plots resembles a logarithmic grow in the convergence times. Since the y-axis is in log-scale this would suggest that RESTART is showing logarithmic efficiency, with convergence times growing sub-exponentially as the rarity of the event grows exponentially. This was not the case in the previous transient study, where convergence times appear to grow exponentially, inversamente al decaimiento exponencial de γ.

En ese sentido hacemos notar que RESTART fue principalmente diseñado para estudios estacionarios (ver [VAVA91, VA98, VAVA02, VA14]). En el caso transitorio donde las simulaciones deben ser regeneradas constantemente, el costo incurrido en los mecanismos globales de división de RESTART quizás desate consecuencias de ineficiencia. En cambio, estrategias como la División Multinivel Adaptativa podrían generar mejores resultados, donde los caminos de simulación son generados y truncados en una aproximación paso a paso que se va aproximando al conjunto de estados raros. En la Sección 5.1 repasamos brevemente el tema de la implementación de otros mecanismos de simulación en FIG.

4.6.3. Cola tándem triple

Considérese una red en tándem no markoviana que opera bajo los mismos principios de la cola tándem de la subsección anterior, pero que consiste en *tres* colas con tiempos de servicio distribuidos según una función *Erlang*[§]. El parámetro de forma α es el mismo en todos los servidores, pero los parámetros de escala $\{\mu_i\}_{i=1}^3$ difieren entre las colas. Los arribos al sistema son exponenciales con tasa λ.

[§] Si bien esto podría emularse con la distribución exponencial, mantendremos un enfoque no markoviano.

El comportamiento a largo plazo de esta cola tándem triple no markoviana fue estudiado en [VA09] comenzando desde un sistema vacío. The shape parameter is $\alpha \in \{2, 3\}$ in all queues and the load at the third queue is kept at $1/3$. This means that the scale parameter μ_3 in the third queue takes the values $1/6$ and $1/9$ when α is 2 and 3 respectively. The scale parameters μ_1 and μ_2 of the first and second servers, as well as the thresholds capacity C at the third queue, are chosen to keep the steady-state probability en el mismo orden de magnitud (aproximadamente) para todos los casos de estudio.

Usamos el modelo IOSA presentado en el Apéndice A.8 para correr simulaciones en JupiterAce. The property of interest is the steady-state probability of a saturation in the third queue, i.e. `S ( q3==c )`. Following the same approach from [VA09] we choose a value of γ in the order of $5 \cdot 10^{-9}$. Por ende los valores de $(\alpha, \mu_1, \mu_2, C)$ para los seis casos de estudio **I**–**VI** son

$$\begin{aligned} \textbf{I}&: (2, 1/3, 1/4, 10) & \textbf{IV}&: (3, 1/9, 1/6, 9) \\ \textbf{II}&: (3, 2/3, 1/6, 7) & \textbf{V}&: (2, 1/10, 1/8, 14) \\ \textbf{III}&: (2, 1/6, 1/4, 11) & \textbf{VI}&: (3, 1/15, 1/12, 12). \end{aligned}$$

Las estimaciones debían lograr un IC $90\,|\,40$ dentro de las 4 horas de ejecución. Four importance functions were tested in the importance splitting simulations: the monolithic (`amono`) and compositional (`acomp`) functions which FIG can build automatically, using summation as composition operand for `acomp`; an ad hoc function which just counted occupation in the third queue (`q3`); and the ad hoc approach from [VA09] (denoted `jva`), que también considera la ocupación en las otras colas con coeficientes de peso específicos a cada caso, los cuales siempre están en el rango [0.2, 0.9].

Presentamos los resultados en la Figura 4.6. Éste experimento también fue realizado tres veces; los valores en los gráficos muestran el promedio de los tiempos de convergencia medidos. Los distintos casos de estudio **I-VI** se ubican en el eje de las abscisas de cada gráfico.

Como se esperaba, las simulaciones de Monte Carlo estándar failed to converge within the 4 hours limit imposed in almost all cases. On the other hand RESTART simulations converged in time in most settings, produciendo un intervalo estimado con las propiedades deseadas. yielding an interval estimate with the desired properties.

El caso de estudio **II** es un tanto curioso porque fue el único in which `nosplit` simulations converged, estimating the interval [6.13e-8, 9.20e-8]. And not only did they converge, but also they did so *faster than many RESTART simulations* (with quite few exceptions, perhaps most notably `amono`). Esto no es fortuito y puede explicarse de la siguiente forma:

- el caso de estudio **II** es el *menos raro* de todos; compáreselo e.g. con el caso **III** donde los estimadores puntuales convergen a valores cercanos a 1.69e-8, o con el caso **V** donde el valor es aproximadamente 3.40e-9, i.e. un orden de magnitud menos.

- la configuración del caso **II** tiene la cola con menor capacidad (e.g. el caso

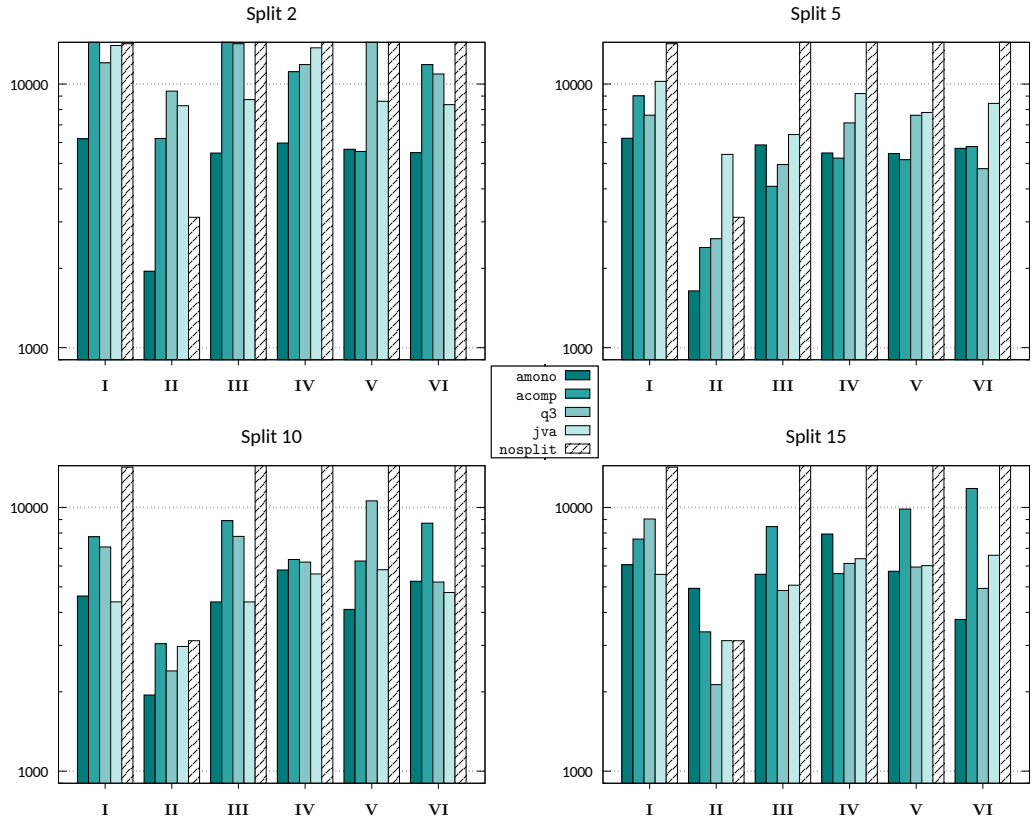


Figura 4.6: Tiempos para el análisis estacionario de la cola tándem triple

V usa un valor de C que duplica al de **II**), y por ende la división por importancia puede dar menos beneficios.

A pesar de estos argumentos, siempre hubo al menos una simulación RESTART que se comportó mejor que Monte Carlo estándar. See for instance `amono` for splittings 2, 5, and 10, and also `q3` for splittings 5, 10, and 15. In particular for splitting 5 all RESTART simulations converged faster than `nosplit`.

Unfortunately, just like it happened with the tandem queue system, there is much variability in the results for the different global splitting values tested. This is most notable for splitting 2, where RESTART simulations converged the slowest, some of them even producing timeouts in all three experiments ran (see cases **I**, **III**, and **V**). Most importance functions converged the fastest for the splitting value 10.

A pesar de estos problemas con el valor global de división, lo cual está íntimamente relacionado con el mecanismo de selección de umbrales como ya se explicó, esta experimentación con la cola tándem triple produjo tres contribuciones de considerable importancia:

1. a pesar de que el sistema puede codificarse para encajar en un entorno markoviano, nuestro modelo emplea la distribución Erlang, permitiendo una representación más compacta y también verificando nuestras afirmaciones acerca de la generalidad de nuestras técnicas y algoritmos;

2. una vez más el método composicional se desempeñó bastante bien, mostrando que la expansión del espacio de estados del modelo compuesto puede ser innecesaria para automatizar la construcción de una función de importancia razonable;

3. `amono` siempre terminó antes del timeout, y en casi todos los escenarios fue o bien la más rápida o sino la segunda, lo cual coincide con nuestras expectativas como lo detallamos en la Subsección 4.6.2.

4.6.4. Sistema de colas con rupturas

Repetimos el experimento presentado en la Subsección 3.5.5 y estudiado originalmente en [KN99], que consiste en una red de colas donde varias fuentes (de tipo 1 y 2) envían paquetes a una única cola atendida por un servidor. Todas las fuentes, y también el servidor, pueden romperse y ser reparadas más adelante. El sistema es markoviano con tasas de reparación de componentes, ruptura de componentes, y producción/procesamiento de paquetes iguales a $(\alpha_1, \beta_1, \lambda_1) = (3, 2, 3)$, $(\alpha_2, \beta_2, \lambda_2) = (1, 4, 6)$, y $(\delta, \xi, \mu) = (3, 4, 100)$ respectivamente para las fuentes de tipo 1, fuentes de tipo 2, y para el servidor.

Usamos el modelo IOSA del Apéndice A.9, para estudiar el comportamiento transitorio del sistema corriendo experimentos en JupiterAce. More precisely we are interested in the probability of observing a saturated buffer before this becomes empty, starting with a single buffered packet and all system components broken but for one source of type 2. La consulta de propiedad correspondiente es `P ( !reset U buf==K )`.

Estudiamos este sistema para las capacidades del buffer $K \in \{40, 60, 80, 100\}$, with corresponding values of γ equal to 4.59e-4, 1.25e-5, 3.72e-7, 9.59e-9. These values were approximated by PRISM in the equivalent CTMC (ver el Apéndice A.4), and the convergence of FIG to such values was checked for all settings. In particular estimations had to achieve a IC $90\backslash 40$ within 3 hours of wall time execution. Three importance functions were tested in the splitting simulations, namely the monolithic (`amono`) and compositional with summation (`acomp`) functions built by FIG, and the best ad hoc variant resulting from our studies in Subsección 3.5.5, i.e. contar el número de paquetes en el (`buf`).

El promedio de los tiempos de convergencia obtained from four experiments run with the FIG tool are presented in la Figura 4.7. The behaviour of the standard Monte Carlo simulations resembles the previous experiments with BLUEMOON, where it had converged only for $K < 80$. This time however, for $K = 40$, it took `nosplit` simulations about three minutes to build an interval with the desired properties. With the CTMC model of the queue with breakdowns running on BLUEMOON it had taken less than 10 seconds. Clearly and as expected, updating several clocks in several modules (e.g. like in IOSA) se comporta peor que el acceso a una matriz (e.g. como en la representación de transiciones de un CTMC).

Asimismo esto produjo una ventaja para las simulaciones con división, since the sheer brute force of `nosplit` is then less advantageous than selecting (proper) resplitting points like RESTART does. Como consecuencia, para $K = 40$, todas

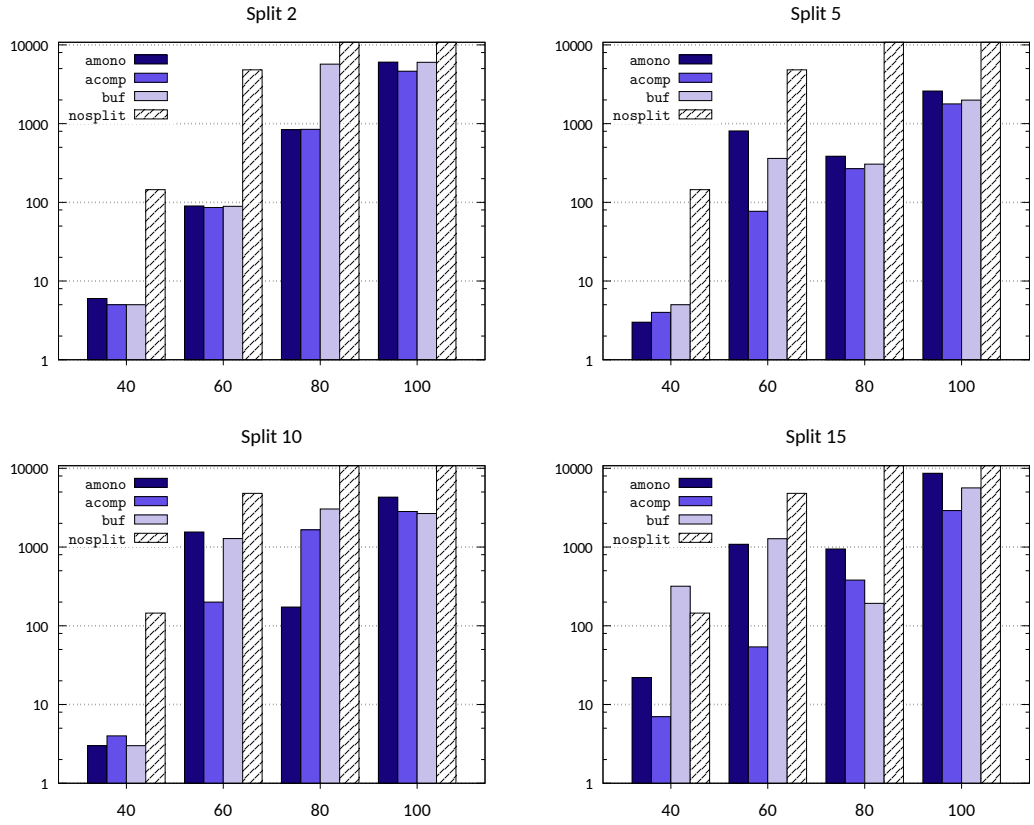


Figura 4.7: Tiempos para el análisis transitorio de la cola con rupturas

configuraciones de división salvo una (viz. `buf` para el valor de división 15) convergieron más rápidamente que Monte Carlo estándar.

Una vez más la alta variabilidad de los resultados para los diferentes valores globales de división dificulta una comparación clara entre las tres funciones de importancia. Overall however none of them clearly outperformed the rest. We use this, as we have done before, to highlight that the compositional approach can yield reasonable results incluso en las configuraciones donde una función monolítica tiene cierta ventaja teórica.

Una peculiaridad un tanto chocante de la Figura 4.7 are the incongruously long convergence times of: `buf` for $K = 40$ and splitting 15 (one configuration); and both `buf` and `amono` for $K = 60$ and splittings 5, 10, and 15 (six configurations). In particular for $K = 60$, cinco de las seis configuraciones *tardaron más en converger* que para $K = 80$.

La salida técnica de FIG revela que from the thresholds selected in those cases, half or more of them were actually not chosen by the adaptive component of Algoritmo 5. At some point in those experiments, an iteration of Sequential Monte Carlo had failed to find a higher threshold, and the algorithm fell back to the deterministic selection provided by `choose_remaining(···)`. This was observed in 1–3 out of the 4 experiments run for those configurations. Precisamente esos experimentos fueron los que produjeron los tiempos de convergencia incongruentes.

No cabe duda de que este comportamiento es problemático. Se hacen esfuerzos en la rutina `choose_remaining(···)` por imitar an intelligent selection of thresholds, taking into account the splitting value used, the post-processing (if any), and the importance range left to cover after (our implementation of) Sequential Monte Carlo has failed. However, `choose_remaining(···)` implements a deterministic selection, que ignora el comportamiento estocástico del modelo, el cual sólo puede ser tenido en cuenta por un algoritmo adaptativo.

Observar esta *falla temprana* de la componente adaptativa del Algoritmo 5, lo cual identificamos como la principal causa detrás del fiasco de umbrales-y-división, nos hace pensar en la necesidad de buscar una estrategia diferente. En la Sección 5.1 exponemos algunas reflexiones con las que diagramamos una solución prometedora.

4.6.5. Sistema de base de datos con redundancia

Recordemos el modelo de la instalación de base de datos que introdujimos en el Ejemplo 7 para mostrar las limitaciones de la estrategia monolítica. Este sistema se encuentra caracterizado por un valor de redundancia $R \in \mathbb{N}$ $(R > 1)$ y sus componentes incluyen procesadores (de dos tipos), controladores de discos (de dos tipos), y clusters de discos (seis de ellos). Denotando como *unidad* a cualquier tipo de procesador, tipo de controlador, o cluster de disco (i.e. hay diez unidades en total), el sistema estará operativo siempre y cuando menos de R componentes hayan fallado en cualquier unidad.

Esta base de datos markoviana fue estudiada originalemnte en [GSH+92] and then using RESTART in e.g. [VA98]. The failure rates of processors, controllers, and disks, are respectively μ_P, μ_C, and μ_D. Furthermore all these components can fail with equal probability in one of two types: failures of type 1 involve a repair rate equal to 1,0, and those of type 2 have a repair rate of 0,5. Rare event analyses focus on system unreliability, e.g. γ refleja la proporción de tiempo que la base de datos no se halla operativa.

Corrimos experimentos en Mendieta sobre modelos como el que presentamos en el Apéndice A.10, el cual describe (una versión resumida de) un sistema para redundancia $R = 2$. La propiedad al final del apéndice consulta por la probabilidad de tener cualesquiera dos (R) componentes falladas en la misma unidad:

```
S ( (d11f & d12f) | (d11f & d13f) | ··· | (p21f & p22f) ) .
```

Como el formalismo IOSA asocia una única función de densidad de probabilidad a cada reloj, la dependencia de fallas entre procesadores debe ser descartada (cf. [VA98]). Moreover, our IOSA models have repair clocks individual to each component, in contrast to the single (sequential) repairman scheme from [VA98, VA07a].

Estudiamos sistemas con tasas de falla $(\mu_P, \mu_C, \mu_D) = (1/50, 1/50, 1/150)$ and redundancy values $R \in \{2, 3, 4, 5\}$. Due to the long times FIG took to converge for the larger models, this experiment follows a different scheme de los que vimos hasta ahora. En lugar de solicitar que las estimaciones alcancen criterios

de confianza predefinidos, impusimos presupuestos temporales de ejecución, y medimos la precisión of the intervals built by each strategy at timeout. The goal is to estimate the narrowest possible interval in the available time, where the time budgets for the redundancy values $R = 2, 3, 4, 5$ son 10 segundos, 2 minutos, 20 minutos, y 6 houras respectivamente.

Resaltamos que nuestros modelos son incomparables con los que fuesen estudiados en [GSH+92, VA98, VA07a], due to the different hypotheses we use in order to model the database with IOSA. Besides, even though the database is Markovian, PRISM can not be utilised to approximate the results in an equivalent CTMC, owing to the physical memory issues reported in Sección 3.6. As a workaround, to verify correctness in our estimations we compared the confidence intervals yielded by all runs, corroborating that they share a common region. Los valores medios (y desviación estándar) obtenidos con este procedimiento a partir de los estimadores (puntuales centrales) para cada redundancia son:

$$\begin{array}{rccccc} R: & 2 & 3 & 4 & 5 \\ \text{avg}\{\hat{\gamma}_i\}: & \text{6.86e-3} & \text{5.14e-5} & \text{3.81e-7} & \text{8.06e-9} \\ \text{stdev}\{\hat{\gamma}_i\}: & \text{1.46e-4} & \text{2.34e-5} & \text{4.37e-7} & \text{1.49e-8}\,. \end{array}$$

Las simulaciones RESTART fueron evaluadas para cinco funciones de importancia. No se puedo emplear al método monolítico debido a los problemas de memoria que ya discutimos. The compositional approach with summation as composition operand is denoted `ac1`. Since each component can be either failed or operational, its local importance function will adopt the values 1 and 0 respectively. Hence `ac1` counts the number of failed components in the whole system. Notice that in spite of its simplicity, esta estrategia puede construir una estructura de importancia mucho más rica de lo que ningún intento monolítico podría.

La función composicional denotada `ac2` makes a distinction based on the component type, under the hypothesis that their failure rates may be different and thus they should not be mixed up (cf. `ac1`). Using the exponentiation post-processing, the local importance functions of all disks are multiplied together ($F_D \doteq \prod_{i,j=1,1}^{6,R+2} D_{i,j}$), and the same is done with all controllers ($F_C \doteq \prod_{k,\ell=1,1}^{2,R} C_{k,\ell}$) and all processors ($F_P \doteq \prod_{k,\ell=1,1}^{2,R} P_{k,\ell}$). La función de importancia global es la suma de estos valores: $F_D + F_C + F_P$.

La función `ac3` hace una distinción aún más fina, separating the product of the disks per cluster ($F_D^i \doteq \prod_{j=1}^{R+2} D_{i,j}$ for clusters $1 \leqslant i \leqslant 6$), and of the controllers and of the processors per type ($F_C^k \doteq \prod_{\ell=1}^{R} C_{k,\ell}$ and $F_P^k \doteq \prod_{\ell=1}^{R} P_{k,\ell}$ for types $1 \leqslant k \leqslant 2$). Nuevamente, la función de importancia global es la suma de estos valores: $\sum_{i=1}^{6} F_D^i + \sum_{k=1}^{2} F_C^k + \sum_{k=1}^{2} F_P^k$.

La función `ac4` usa el anillo $(+, *)$ in what can be regarded as a further refinement in the same direction than `ac2` and `ac3`: `ac2` would be the most coarse grained, mashing all components of the same type together; `ac3` contemplates the divisions of the system in independent units; y `ac4` distingue toda configuración posible que genera una falla del sistema.

Finalmente, la función `ah` tiene dos facetas. Por un lado puede ser vista como la variante composicional que implementa el semianillo $(\text{máx}, +)$. Por el otro lado

esta función coincide exactamente con la propuesta ad hoc de [VA07a], denotada $\Phi(t) \doteq cl - oc(t)$ en ese trabajo.

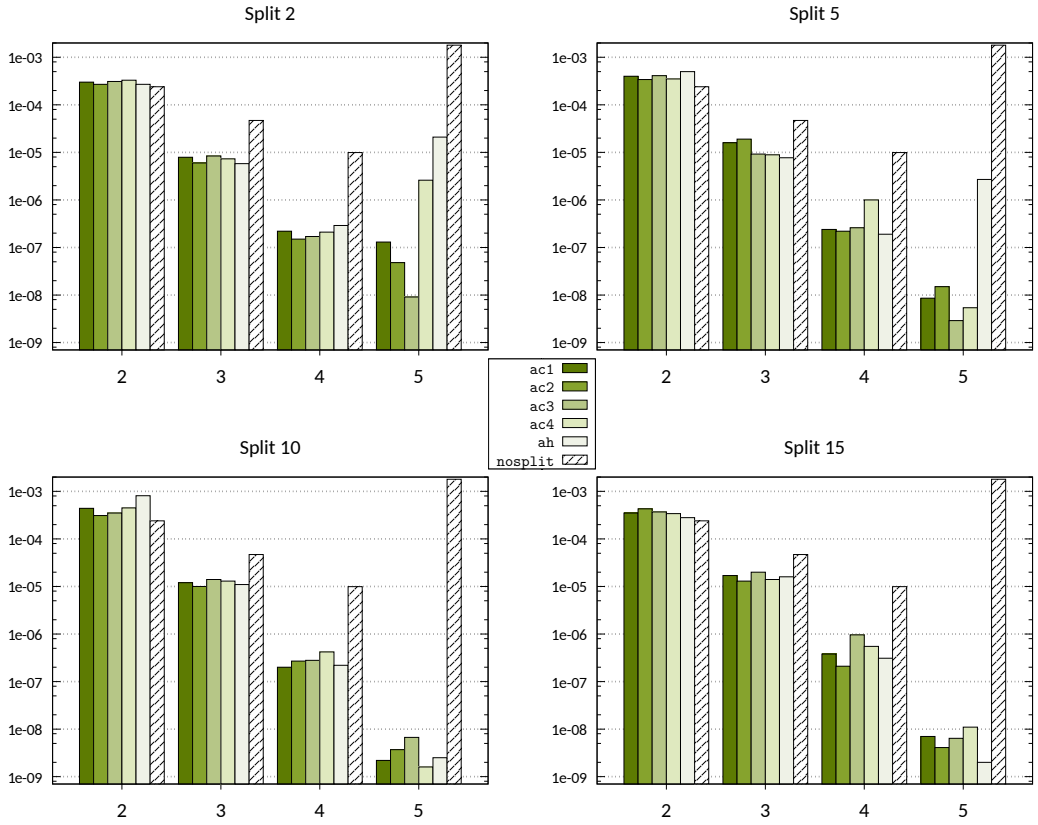


Figura 4.8: Precisión de los intervalos del análisis estacionario de la base de datos

Para un nivel de confianza del 90 %, presentamos en la Figura 4.8 la precisión de los intervalos obtenidos para los presupuestos temporales indicados. Estos valores son el promedio del resultado de cuatro experimentos independientes corridos en Mendieta.

Las simulaciones de Monte Carlo estándar estimaron los mejores intervalos para el menor valor de redundancia, $R = 2$, lo cual no es sorpredente y coincide (parcialmente) con los resultados reportados en[BDM17]. Notice however that for $R = 3$ and although to a moderate extent, la Figura 4.8 muestra un mejor comportamiento de todas las simulaciones RESTART con respecto a `nosplit`, a diferencia de [BDM17].

Aún cuando estos resultados podrían ser dejados de lado as yet another example where the event is not rare enough for a really effective application of multilevel splitting (which might well be true!), puede aprenderse algo si estudiamos más a fondo la cuestión.

Para dejar de estar operativa la base de datos requiere que R components in any unit to fail, where the number of components per unit is heterogeneous. The unit with most components would be the likeliest to produce the system failure, which is the case of the disks clusters. No obstante, el tiempo de vida

entre componentes difiere bastante, y en promedio es tres veces más corto en controladores y procesadors que en los discos. This means that it is actually very likely to observe a non operational database debido a R controladores o procesadores fallados en la misma unidad (i.e. del mismo tipo).

Es por ello que *necesitamos* aumentar el valor de redundancia in order to obtain some gain from the use of multilevel splitting. Even in the rich layering offered by `ac1` to `ac4`, most cases of a non operational database will be caused by R failed processors or controllers of the same type. Which does not imply, however, that the compositional approach is at a loss. It merely justifies why `nosplit` performed better than the splitting simulations for $R = 2$. Higher redundancies mean a more layered structure of even the likeliest, flattest failures, lo cual habla a favor del uso de la división multinivel.

Llamamos la atención ahora a los valores de redundancia $R \in \{4, 5\}$, where standard Monte Carlo ceases to be a reasonable choice and one has to resort to other strategies, e.g. importance splitting. Moreover, since the monolithic approach was infeasible already for $R = 4$ in our studies from Sección 3.6, the compositional approach introduced in este capítulo ofrece la única alternativa automática para aplicar la división multinivel.

Notablemente, ninguna de las cinco estrategias de composición clearly outperformed the rest in all configurations. This would suggest that, in this scenario where *flat failures* caused by R processors or controllers have great likelihood, the extra importance layering granted by functions like `ac4` is not a defining factor. The charts also hint at a better performance of RESTART for the higher splitting values, although convergence was slightly faster en casi todos los casos para el valor de división 10 que para 15.

Unas pocas configuracions generaron valores en discordia con el resto; the most striking cases from Figura 4.8 are observed when $R = 5$ for splitting 2 (`ac4` and `ah`) and splitting 5 (`ah`). Furthermore, for that redundancy and in one out of the four independent experiments run, `ac4` and `ac1` failed to converge to an estimate for splittings 10 and 15 respectively. These peculiarities are, to our surprise, *not related* to the splittings-thresholds fiasco. The technical output of FIG revealed an aberration in the outcome of the individual simulations, which at some point started to sample the rare event at extremely high (or low) rates, contrarily to the immediately preceding behaviour. Suspecting a relation between these aberrations and the pseudo-random number generation algorithm, we repeated such runs using a different algorithm fed with different seeds. As expected, the aberrations were not observed again, y las salidas encajaron en el marco normal correspondiente a cada caso.

Subrayamos por último que a pesar de haber observado un desempeño bastante similar entre todas la funciones de importancia, ambas `ac4` y `ah` (la segunda mirada como el semianillo $(\text{máx}, +)$) no fueron diseñadas específicamente para la base de datos, sino que pueden ser derivadas de la expresión DNF del evento raro como describimos en la Subsección 4.3.3. Esta es una muy buena razón para preferir a estas funciones por sobre las otras tres.

4.6.6. Tuberías de petróleo

Considérese un sistema k-consecutivos-de-entre-n: F (C(k, n: F) por sus siglas en inglés). El mismo consiste en una secuencia de n componentes *ordenados secuencialmente*, de manera tal que el sistema completo falla si k o más componentes *consecutivos* fallan. Para tener una imagen mental más realista y terrenal pensemos en un oleoducto donde hay n bombas de presión equidistantes. Cada bomba puede transportar petróleo, a lo sumo, por una distancia igual a la que separa a k de estas bombas. Entonces cuando $k > 1$ el sistema tiene cierta resistencia a las fallas, y puede permanecer operativo siempre y cuando haya menos de k bombas falladas consecutivas, independientemente por lo demás del número total de bombas falladas.

Los sistemas C(k, n: F) have been studied as early as 1980 [Kon80]. Several generalisations exist to the original setting; we are interested in the non-Markovian and repairable systems analysed in e.g. [XLL07, VA10]. Those works assume the existence of a repairman which can take one failed component at a time and leave it "as good as new", after a log-normal distributed repair time has elapsed [XLL07]. In particular [VA10] consider also the existence of non-Markovian failure times (namely, sampled from the Rayleigh—or Weibull—distribution) and measure the steady-state unavailability of the system. Notice the probability density function used in [VA10] para la distribución Rayleigh es $f_\beta(t) \doteq \beta t e^{-t^2 \frac{\beta}{2}}$, cuya media es $\sqrt{\frac{\pi}{2\beta}}$.

Para correr experimentos en Mendieta, usamos modelos IOSA como el que mostramos en el Apéndice A.11, que representa un oleoducto del tipo C(3, 20: F), i.e. donde hay un total de $n = 20$ bombas de presión, y $k = 3$ bombas rotas consecutivas causan una falla general del sistema. En ese ejemplo la indisponibilidad estacionaria del sistema puede consultarse con la propiedad:

```
S ( ( broken1>0 & broken2>0 & broken3>0 ) |
    ( broken2>0 & broken3>0 & broken4>0 ) |
                        ⋮
    ( broken18>0 & broken19>0 & broken20>0 ) ) .
```

Por desgracia la dependencia de Markov de $(k-1)$-pasos en las fuentes (ver [LZ00] y también [XLL07, VA10]) no puede modelarse en IOSA, puesto que requeriría la asociación de más de una distribución a los relojes involucrados[†]. We also highlight that in order to model the repairman, an extension of the basic IOSA theory and model syntax presented in Sección 4.4 y subsección 4.5.2 was employed, which allows certain use of instantaneous (or untimed) actions[§]. That, plus the possibility to define and operate with arrays of variables, es actualmente soportado por la versión 1.1 de la herramienta FIG.

[†] Hay varias extensiones al formalismo en consideración; ésta es una de ellas.

[§] Este es trabajo en curso por parte de Monti et al. en el Grupo de Sistemas Confiables.

Aún así, las políticas de reparación reportadas en [VA10] no pueden modelarse con FIG, since the tool is designed to fit a general basis of models, and such repair policy is quite singular to this system. This issue, plus the absence of the $(k-1)$-step Markov dependence hypothesis, make our implementation of the models incomparable to those studied in [XLL07, VA10]. Therefore we resort to the same strategy followed with the database, and for each system configuration studied we report the mean value de todas las probabilidades estimadas (y su desviación estándar).

Específicamente, estudiamos modelos con $n \in \{20, 40, 60\}$ componentes ordenados secuencialmente, where $k \in \{3, 4, 5\}$ sequential failures result in a non operational system. As in [VA10] we analysed both exponential and Rayleigh failure times, for rate and scale parameters $\lambda = 0{,}001$ and $\beta = 0{,}00000157$ respectively. This yields the same mean lifetime in the components. Repair time is sampled from a log-normal distribution with parameters $\mu = 1{,}21$ and $\sigma = 0{,}8$. En la Tabla 4.2 mostramos la indisponibilidad estacionaria del sistema estimada para estas configuraciones.

		Exponencial			Rayleigh		
	n:	20	40	60	20	40	60
$k = 3$	media$\{\hat{\gamma}_i\}$:	1.53e-5	4.65e-5	1.10e-4	2.02e-5	6.54e-5	1.63e-4
	desv$\{\hat{\gamma}_i\}$:	1.76e-6	4.68e-6	1.62e-5	2.49e-6	7.27e-6	2.21e-5
$k = 4$	media$\{\hat{\gamma}_i\}$:	2.92e-7	1.49e-6	4.33e-6	5.44e-7	2.65e-6	8.16e-6
	desv$\{\hat{\gamma}_i\}$:	1.39e-7	2.51e-7	6.64e-7	1.57e-7	4.04e-7	1.06e-6
$k = 5$	media$\{\hat{\gamma}_i\}$:	2.62e-9	5.93e-8	3.06e-7	7.49e-9	1.26e-7	6.97e-7
	desv$\{\hat{\gamma}_i\}$:	2.03e-9	2.54e-8	1.39e-7	6.11e-9	4.49e-8	2.77e-7

Tabla 4.2: Estimaciones de indisponibilidad para el oleoducto

Realizamos dos experimentos independientes, cada uno cubriendo las dieciocho configuraciones, corriendo a FIG en Mendieta y solicitando un IC $90\,|\,40$. We imposed differentiated wall time execution limits for the different values of k, since this parameter has the highest influence in the rarity of the event (see Tabla 4.2) and thus in the convergence times. For $k = 3, 4, 5$ we requested estimations to converge within 1.5, 3, and 6 hours respectively.

La situación con los modelos del oleoducto is similar to the database, in the sense that the high amount of components (which fail independently from each other) render any monolithic approach utterly infeasible. Therefore the automatic importance functions evaluadas con RESTART son composicionales.

La estrategia ingenua de componer las funciones locales con la suma as composition operand is denoted `ac1`. Similarly, `ac2` uses product as composition operand and an exponentiation post-processing. The $(\text{máx}, +)$ and $(+, *)$ semiring and ring composition strategies are employed por las funciones denotadas `ac3` y

`ac4` respectivamente. Por último, `ah` usa la interfaz ad hoc de FIG para implementar el semianillo $(\text{máx}, +)$, usando las variables de los módulos (i.e. in an ad hoc fashion) rather than the local importance functions which the tool could compute if requested. Esta metodología se sigue en [VA10], donde es denotada $\Phi(t) \doteq cl - oc(t)$.

Este caso de estudio incluye dieciocho configuraciones, each of them tested with standard Monte Carlo and with RESTART using five different importance functions. In addition, each multilevel splitting run was tested for the usual four different global splitting values. Los tiempos promedio de convergencia en segundos se presentan en la Tablas 4.4 y 4.5.

En ocasiones sólo uno de los dos experimentos corridos para cada setting produced a result; such single-simulation results appear enclosed in parentheses in Tablas 4.4 y 4.5. Since no simulation converged within the six hours time bound for $K = 5$, in that cases we show the interval precision achieved at timeout for a 90 % confidence interval. También y al igual que antes, dividimos las tablas por valor de división.

Puesto que hay tantas configuraciones diferentes (dieciocho en total), and also so many different settings tested for each configuration (a standard Monte Carlo run plus twenty RESTART runs if we tell apart by splitting and importance function), some simplifications are due to interpret these results. La Tabla 4.3 muestra un filtrado bastante grueso, donde contamos para cada configuración el total de simulaciones RESTART que dieron mejores resultados que las corridas de Monte Carlo estándar.

	Exponencial			Rayleigh		
n :	20	40	60	20	40	60
$k = 3$	9	8	0	3	3	0
$k = 4$	19	16	4	20	16	2
$k = 5$	7	15	6	10	10	4

Tabla 4.3: RESTART vs. Monte Carlo

Resaltamos que en la Tabla 4.3, cualquier valor de división para el cual sólo un experimento (de los dos corridos) convergió, was considered to have lost against standard Monte Carlo. This might be regarded as biased against multilevel splitting, but the only properly unbiased way of comparing all simulation settings would be to perform several more repetitions of the full experiment. Por desgracia, los prolongados tiempos de ejecución de un experimento completo[†] descartan por completo esta posibilidad.

A pesar de todo esto, en los sistemas con tiempos de falla Rayleigh observamos que usar la técnica de división multinivel comienza a dar frutos cuando la

[†] Lleva 6 días evaluar las 18 configuraciones con todos los entornos de simulación.

		Split 2					Split 5				
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah
	3	58	51	99	62	804	75	52	53	50	54
20	4	2780	2109	4398	2339	7257	6292	3535	4134	2683	3026
	5	(7.0e-10)	-	2.4e-9	(3.9e-9)	(1.7e-9)	5.6e-9	4.4e-9	(1.6e-9)	2.2e-9	2.9e-9
	3	105	1386	1388	153	132	111	1833	123	107	200
40	4	3439	3480	3590	10800	3902	4047	6344	5120	10800	3368
	5	5.8e-8	5.6e-8	4.5e-8	5.6e-8	3.8e-8	7.4e-8	5.1e-8	4.9e-8	2.1e-8	4.7e-8
	3	148	156	240	1953	256	274	383	404	202	438
60	4	10800	10800	4879	4023	4932	10800	10800	10246	4104	10800
	5	2.1e-7	1.8e-7	2.8e-7	1.7e-7	2.4e-7	3.8e-7	5.1e-7	5.3e-7	3.6e-7	3.2e-7

		Split 10					Split 15					
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah	nosplit
	3	77	53	77	44	81	88	131	74	73	80	63
20	4	10800	4019	5095	3347	4922	5084	8445	6205	4336	3452	10800
	5	3.7e-9	2.1e-9	5.8e-9	2.6e-9	3.3e-9	(5.1e-10)	3.4e-9	7.0e-9	(1.7e-9)	5.1e-9	3.7e-9
	3	89	1456	221	66	137	92	1811	147	121	155	142
40	4	3868	3568	4072	10800	5973	4449	4332	4182	10800	5152	10800
	5	5.9e-8	4.4e-8	5.3e-8	5.3e-8	5.9e-8	3.9e-8	7.1e-8	1.3e-7	8.4e-8	9.5e-8	6.2e-8
	3	263	240	649	225	765	286	245	214	240	180	137
60	4	10800	10800	2799	5430	3502	10800	10800	5277	6135	5838	4332
	5	2.5e-7	1.5e-7	1.2e-7	2.3e-7	3.6e-7	2.8e-7	1.7e-7	2.4e-7	5.0e-7	2.9e-7	2.3e-7

Tabla 4.4: Resultados para el oleoducto con fallas exponenciales

		Split 2					Split 5				
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah
20	3	50	60	57	63	644	34	51	55	48	52
	4	2884	2959	6625	2342	3775	4513	1673	2634	1598	1785
	5	(1.5e-9)	5.0e-9	-	(4.0e-9)	(2.4e-9)	4.0e-9	(5.3e-9)	5.6e-9	(4.0e-9)	6.9e-9
40	3	71	1504	139	123	193	119	1579	120	141	101
	4	5140	3881	3326	10800	2000	2539	3720	2973	10800	5467
	5	9.8e-8	1.4e-7	1.1e-7	9.3e-8	7.6e-8	8.6e-8	1.2e-7	1.4e-7	9.4e-8	8.2e-8
60	3	256	222	185	2296	163	316	283	319	345	368
	4	10800	10800	3218	3801	4528	10800	10800	7697	6488	5634
	5	6.2e-7	3.2e-7	4.4e-7	2.9e-7	4.5e-7	4.6e-7	3.0e-7	4.5e-7	1.4e-6	4.8e-7

		Split 10					Split 15					
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah	nosplit
20	3	51	53	61	53	61	76	83	74	46	76	49
	4	2322	2172	1687	1645	3420	2584	3418	2740	3241	3008	10800
	5	-	1.3e-8	1.2e-8	(4.2e-9)	7.4e-9	(2.6e-9)	2.1e-8	1.7e-8	(4.5e-9)	7.8e-9	(2.3e-9)
40	3	92	1588	119	83	161	141	1893	123	106	111	101
	4	3143	3384	3558	10800	2613	3674	4510	3911	10800	4424	10800
	5	9.6e-8	1.0e-7	6.9e-8	9.1e-8	7.6e-8	1.5e-7	1.3e-7	1.4e-7	6.3e-8	9.4e-8	9.5e-8
60	3	433	239	450	254	609	387	528	1192	430	964	100
	4	10800	10800	10800	6726	9515	10800	10800	10800	7837	10800	3769
	5	9.7e-7	5.2e-7	3.2e-7	5.5e-7	1.8e-6	1.2e-6	7.7e-7	5.7e-7	8.7e-7	1.0e-6	3.7e-7

Tabla 4.5: Resultados para el oleoducto con fallas Rayleigh

probabilidad del evento raro es inferior a (aprox.) 5.0e-6, i.e. para $n \in \{20, 40\}$ cuando $k = 4$, y para todo n cuando $k = 5$ (el caso $n = 60\,;\, k = 4$ es considerado más adelante). Algo muy similar ocurrió con los sistemas cuyos tiempos de falla siguen la distribución exponencial, pero para magnitudes inferiores, viz. alrededor de 2.0e-6.

También vale la pena recalcar que la tendencia general de la Tabla 4.3 indica que los mayores valores de n son perjudiciales para la división multinivel con respecto a Monte Carlo estándar. Esto podría deberse al hecho de que, en nuestros modelos, mayores valores de n implican eventos de menor rareza.

Si bien esto podría explicar efectivamente una variante *suave* de dicho comportamiento global respecto de n, pareciera haber alguna especie de *barrera más conspicua* entre los valores 40 y 60 de n que entre los valores 20 y 40. En ese sentido notemos que los resultados de la Tablas 4.4 y 4.5 indican que varias corridas RESTART se toparon con la cota de tiempo máxima (*hicieron timeout*), fallando así en superar a las corridas `nosplit` contra las que competían. Mientras mayor es el valor de n, con mayor frecuencia se observa este fenómeno.

Si consideramos el fiasco de umbrales-y-división, then we find a plausible explanation for this behaviour, where systems with $n = 60$ are hard to analyse for FIG using RESTART. Namely, a higher n implies longer simulation steps since more clocks need to be updated per step. If the thresholds for multilevel splitting are selected poorly, the wasted time increases with the splitting. This should be exacerbated by having higher splitting values, although not necessarily in a linear way, dado que la calidad de los umbrales seleccionados juega un papel primordial.

En este espíritu, para la configuración $n = 60$ y $k = 4$, for splitting values 2, 5, 10, and 15 respectively, la Tablas 4.4 y 4.5 show that out of the five RESTART settings: respectively 2, 2, 3, and 4 settings timed-out for Rayleigh distributed failure times; y respectivamente 2, 3, 2, y 2 configuraciones hicieron timeout para los tiempos de falla exponenciales.

En cualquier caso observamos en la Tabla 4.3 that the only configurations where, for any splitting, none or very few RESTART runs outperformed the `nosplit` runs, are those where the event is less rare. This covers mostly the configurations where $k = 3$. Higher values of k allow a more fruitful layering of the state space and hence a more efficient application of multilevel splitting. Esto coincide con el análisis y los resultados presentados en [VA10], donde se utilizó a RESTART con la función de importancia `ah`.

Las Tablas 4.4 y 4.5 muestran que las únicas corridas donde, para algún entorno de ejecución, uno o ambos experimentos no produjeron un resultado final, son precisamente aquellas donde el evento es más raro, viz. $n = 20$ y $k = 5$ para ambas distribuciones de falla. This does not only difficult a proper analysis, but also interferes with our attempts to corroborate the previous conjectures respecto del comportamiento de la división multinivel para valores más altos de k.

Por ello, para permitir un análisis más robusto y detallado we replicated the experiments for the configuration $n = 20$ and $k = 5$ of the oil pipeline, for both exponential and Rayleigh failure times in the nodes. We decided to let the simulations run for 3 h of wall time, usando la precisión de los intervalos

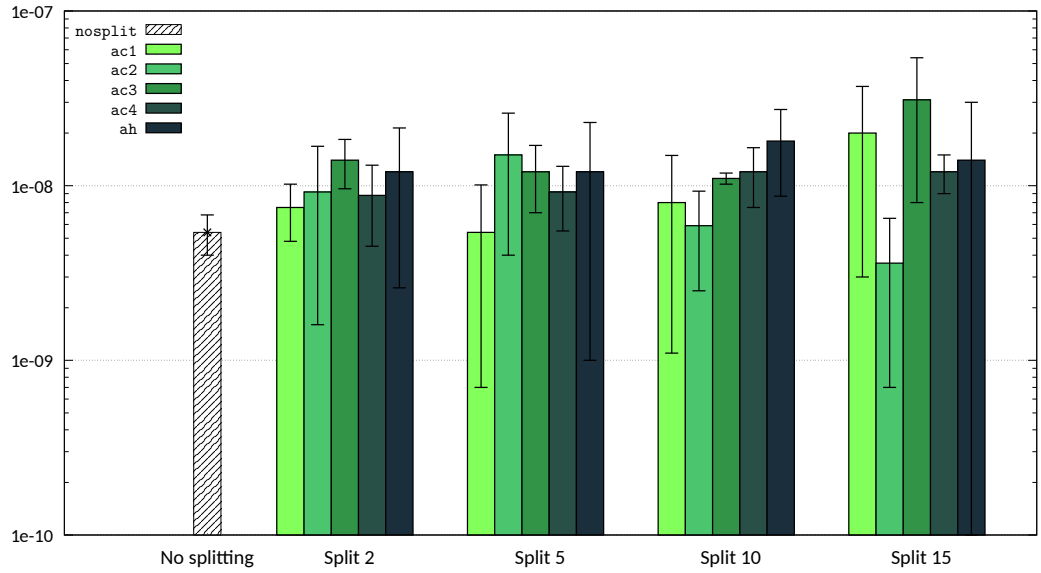


Figura 4.9: Oleoducto con fallas exponenciales; precisión para timeout de 3 h

como medida de desempeño. Tres experimentos independientes fueron corridos en Mendieta; presentamos los resultados en las Figuras 4.9 y 4.10. These values are the average of the precision of the intervals obtained from the three experiments run; la desviación estándar se muestra como marcas por encima de las barras.

Para nuestra sorpresa, incluso para estas dos configuraciones where the event is relatively rare, several splitting simulations were defeated by standard Monte Carlo. There are a few situations where particular RESTART settings clearly outperformed `nosplit`: e.g. in Figura 4.9 there is `ac2` for splitting 15; in Figura 4.10 there are `ac2` and `ac4` for splitting 2, `ac3` for splitting 5, and `ac4` and `ah` for splitting 15. However we would have expected un peor desempeño de `nosplit` con respecto a las variantes que usaron división.

Comparando estos experimentos para los diferentes failure time distributions, we observe that the simulations which use splitting behaved worse (on average) for the exponentially distributed failures. Notice also how the standard deviation of most RESTART settings in las Figuras 4.9 y 4.10 is higher than that of standard Monte Carlo, and notice that such behaviour is more pronounced in the exponential (rather than the Rayleigh) variant. Esta última observación sugiere que hay mayor sensibilidad a la semilla del RNG por parte de RESTART que de Monte Carlo estándar, lo cual creemos que está íntimamente relacionado con el fiasco de umbrales-y-división. Esto también indica que muchas de las simulaciones con las técnicas de división en realidad se habrían comportado mejor que `nosplit`, a pesar de que el comportamiento promedio favorece a `nosplit`, contrario a nuestras expectativas iniciales.

En un intento final para comprender mejor el comportamiento de este modelo de oleoducto, repetimos los experimentos para in límite de tiempo más alto, concretamente para un timeout de 5 h. The hypothesis is that a longer execution could stabilise the behaviour of the simulations in the long run. Esto debería

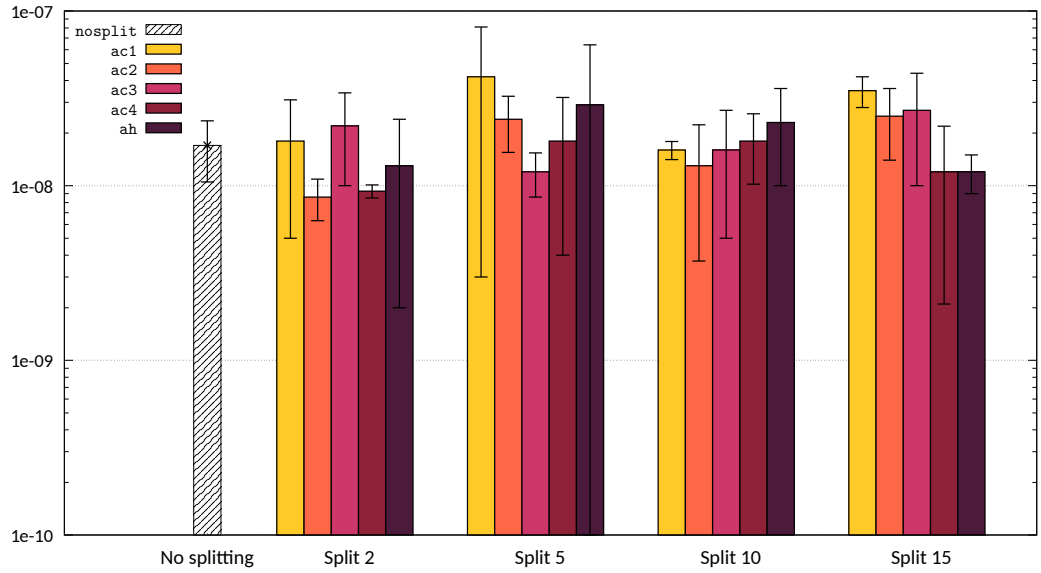


Figura 4.10: Oleoducto con fallas Rayleigh; precisión para timeout de 3 h

favorecer a las simulaciones RESTART por sobre Monte Carlo estándar, dado que una división apropiada debería aumentar el muestreo en áreas ricas en eventos raros, contrariamente a la táctica de una sola simulación de `nosplit`.

Los resultados de estos últimos experimentos, también corridos en Mendieta, are presented in Figuras 4.11 y 4.12. Four instances were launched for each configuration and execution setting. All four succeeded in each case for the experiments with exponentially distributed failures in the nodes. However, in the Rayleigh cases and due to unavoidable issues with the hardware, only two or three runs for each case finished without external interruptions. The outcomes of the interrupted runs were discarded, and thus the samples used to compute the averages shown in Figura 4.12 consist in less than four values. That is why we consider the results from those experiments de peor calidad que los presentados en la Figura 4.11.

Por un lado nuestras conjeturas acerca de la better performance of RESTART were fulfilled in the exponential variant of the model, where most settings using splitting behaved (on average) better than standard Monte Carlo. Remarcamos resultados como los de `ac2` para una división global igual a 2, `ac1` para una división de 5, y `ac4` para una división de 5 y 10, donde el ancho promedio de los intervalos se encuentra por debajo de la precisión lograda por las simulaciones `nosplit`, incluso cuando sumamos la desviación estándar de las mediciones.

Por el otro lado los resultados para el oleoducto con fallas tipo Rayleigh son un tanto desconcertantes, because they show a tendency contrary to that of the exponential case, which we could predict successfully. That is, Figura 4.12 shows that simulations using splitting behaved worse than standard Monte Carlo in general, lo cual también se halla en conflicto con los resultados previamente presentados en la Figura 4.10 para el timeout de 3 h.

No obstante, podemos zanjar la cuestión al considerar lo siguiente:

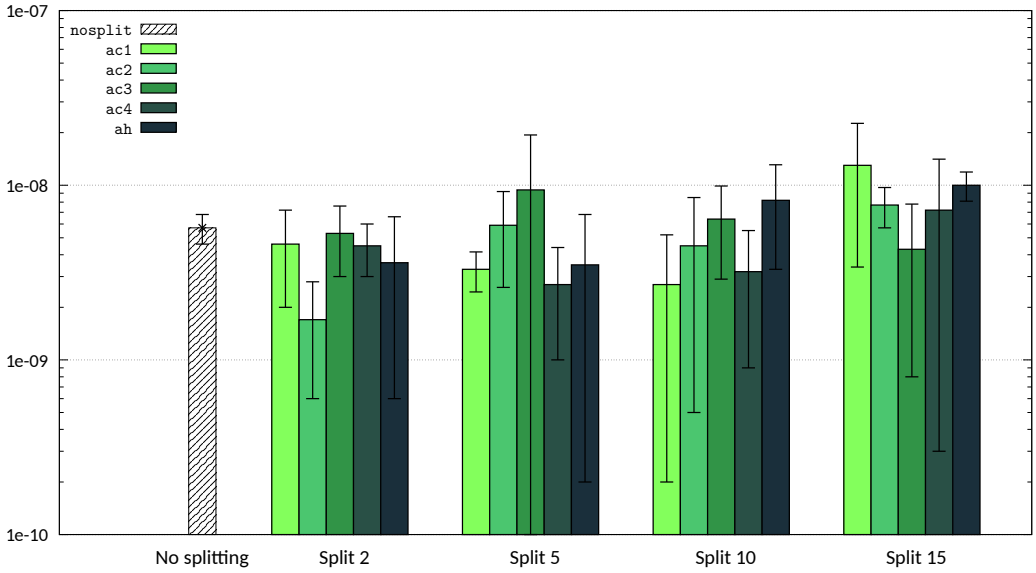


Figura 4.11: Oleoducto con fallas exponenciales; precisión para timeout de 5 h

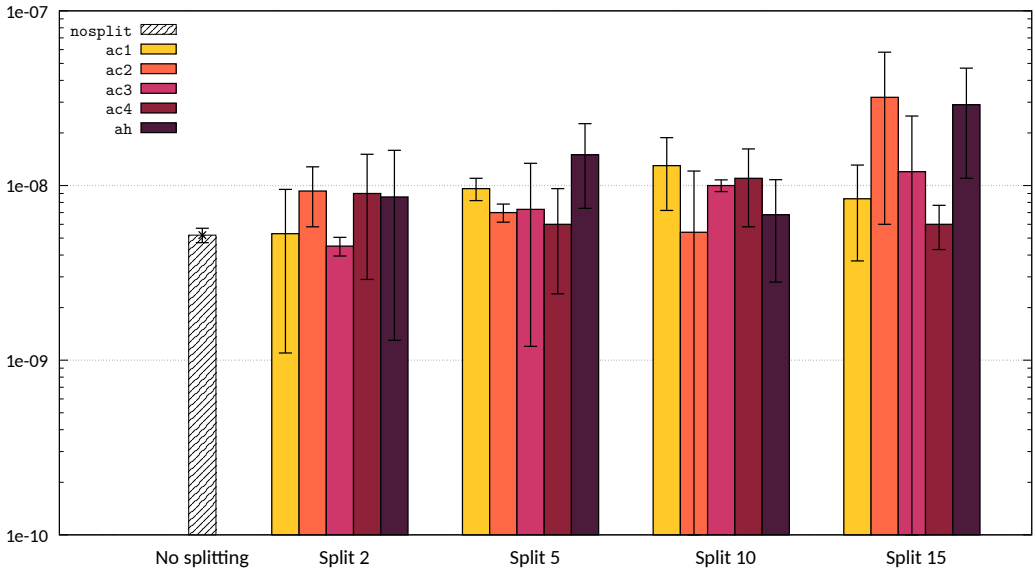


Figura 4.12: Oleoducto con fallas Rayleigh; precisión para timeout de 5 h

- la ejecución de los experimentos fue problemática desde el punto de vista técnico, produciendo muestras más pequeñas de las cuales se calcularon los valor medios presentado;
- puesto que hay evidencia de una alta sensibilidad a la semilla específica alimentada al RNG, la Figura 4.12 debe ser interpretada con reservas al tener en cuenta el ítem anterior;
- de hecho y en un sentido más abarcador, deberíamos usar muestras con más de—digamos— corridas experimentales independientes para reducir la desviación estándar a valores razonablemente pequeños;
- por sobre todo esto tenemos al fiasco de umbrales-y-división, que se evidencia en la alta variabilidad observada para los distintos valores globales de división, que dificulta la comparación de los entornos particulares de ejecuciones RESTART contra la estrategia de Monte Carlo estándar.

Todo esto se encuentra a favor de repetir toda la experimentación, corriendo muchos más experimentos independientes por configuración del sistema y por entorno de ejecución, e incluso quizás usando valores de timeout más prolongados. Sin embargo primero deberíamos estudiar mayores valores de k, since $k = 5$ may be simply not enough to observe a clear advantage of RESTART w.r.t. standard Monte Carlo. In particular [VA10] study the oil pipeline system for $k \in \{4, 6\}$, and report higher gains for the higher value of k. This will be the subject of future research.

Para concluir con esta subsección, ofrecemos algunas comparaciones entre las diferentes funciones de importancia empleadas en las corridas RESTART. Neither Tablas 4.3 a 4.5 nor Figuras 4.9 a 4.12 suggest a clearly outstanding composition strategy outperforming the rest. Indeed, the fastest converging function varied much not only with the global splitting chosen, pero también con la configuración particular del sistema estudiada.

Aún así, el rico conjunto de resultados presentado a lo largo de la subsección allows us to distill some useful information. Tablas 4.4 y 4.5 show that `ac4` was either the best performing function or the runner up in most configurations where $n = 60$. Notice that in some settings it even outperformed standard Monte Carlo, although the higher values of n favour the `nosplit` strategy as discussed. We believe this is related to the large importance range offered by this function, higher than `ac1` and `ac3` for instance. Besides, since it is derived from the specific property under study, `ac4` may fit the evolution of a simulation towards the rare event de manera más natural que e.g. `ac1` y `ac2`.

A su vez, `ac4` fue de las más resistentes al cambio in the splitting value, as it can be observed in Figuras 4.9 a 4.12. In contrast we remark that `ac2`, which yielded quite good results for e.g. splitting 15 in Figura 4.9 and splitting 2 in Figura 4.11, showed a high variability related to the splitting. The simple summation implemented by `ac1` behaved better than expected, pero siempre fue superada por las funciones que implementaban estrategias de composición de anillo/semianillo en la mayoría de las configuraciones (ver las Tablas 4.3 a 4.5).

Lo anterior indica un muy buen comportamiento promedio de `ac4`, which in addition was never the worst candidate in the most demanding configurations tested, i.e. the ones presented in Figuras 4.9 a 4.12. We believe a more thorough study of the oil pipeline system, specifically testing higher values of k and other splitting values, would maintain this assertion and favour majorly `ac4`, cuyas buenas propiedades pueden bien ser el resultado de la manera en que es derivada.

Está claro que aún hay mucho por aprender de este sistema. Es un caso de estudio interesante per se, debido a sus muchas aplicaciones en la industria, pero además ofrece gran potencial para aplicar división por importancia. En este primer intento de análisis hemos visto que regímenes más raros, donde el sistema es más tolerante a fallas debido a los altos valores del parámetro k, son beneficiosos para nuestras técnicas. Verificar hasta qué punto esto es así, con una versión más eficiente de nuestra herramienta (que en particular haya resuelto el problema de selección de umbrales), es un desafío que planeamos enfrentar en el futuro cercano.

5 Notas finales

En esta tesis hemos desarrollado técnicas para analizar automáticamente mediante simulación a sistemas en regímenes de evento raro. Empleamos división por importancia para guiar la simulación de caminos de ejecución hacia el evento raro. Contribuimos con algoritmos para derivar la función de importancia de la cual depende fuertemente la división multinivel. De esta forma, las entradas que el usuario debe proveer para emplear técnicas de división por importancia, no difieren de las entradas requeridas por los análisis que emplean técnicas de simulación de Monte Carlo estándar.

Hemos dividido nuestra estrategia en dos instancias. El Capítulo 3 presenta un primer *método monolítico* para construir y almacenar la función de importancia. La (muy) buena calidad de la función resultante es verificada empíricamente en varios casos de estudio, pero la necesidad de expandir el espacio de estados del modelo compuesto es una severa desventaja. El Capítulo 4 presenta un segundo *método composicional* que se deshace de dicho requisito, pagando por ello con la pérdida de conocimiento de parte de la semántica global del sistema. Sin embargo, si uno elige una estrategia de composición adecuada, guiada por la expresión de la propiedad del evento raro, es posible contrarrestar esta pérdida en ciertos casos. Lo que es más, dicha estrategia de composición ofrece mayor flexibilidad a la hora de construir la función de importancia (global), con lo cual es posible superar incluso las posibilidades del método monolítico.

La división por importancia es una técnica compleja. Requiere articular varias decisiones, e.g. los umbrales y la división para RESTART, para así obtener ganancias substanciales en comparación con el uso de Monte Carlo estándar. Además de desarrollar una base algorítmica, en esta tesis diseñamos mecanismos para embeber nuestros algoritmos en una aplicación automatizada de la división por importancia. Como deseábamos, el resultado se parece al *push-button approach* del model checking estándar, donde se obtienen respuestas tan sólo presionando un botón, una vez que el modelo y las consultas han sido formalizados.

Asimismo, hemos desarrollado herramientas de software (BLUEMOON y FIG) que implementan nuestra teoría. Esto permitió validar nuestras afirmaciones, corriendo experimentos en casos de estudio tomados de la bibliografía existente en el tema de la simulación de eventos raros. Comparamos el desempeño del análisis por simulación Monte Carlo estándar, contra nuestras estrategias automatizadas de división por importancia, mostrando la ganancia obtenida con nuestras propuestas en regímenes de evento raro. También comparamos el desempeño de nuestras funciones de importancia construidas automáticamente contra funciones escogidas

de manera ad hoc para cada modelo estudiado. Los resultados experimentales evidencian que nuestras propuestas son considerablemente versátiles, en añadidura a su naturaleza automática.

5.1. Trabajo futuro

Primero y principal, a raíz de los resultados y discusiones presentados en las Secciones 3.5 y 4.6, queda claro que nuestra estrategia para seleccionar los umbrales de importancia para RESTART dista de ser óptima. En este punto sospechamos que la hipótesis de un espacio de estados continuo de ambos la División Multinivel Adaptativa y Monte Carlo Secuencial, pesa demasiado en el desempeño de estos procedimientos. Adaptarlos al entorno discreto de las cadenas de Markov o IOSA produjo resultados insatisfactorios.

Esto se encuentra íntimamente relacionado con la elección de un valor global de división. Quizás estas tácticas generen resultados óptimos en entornos continuos donde los umbrales puede escogerse tan próximos entre sí como se lo desee. Empero, en nuestros experimentos, el hecho de contar con un único valor de división para todos los umbrales a veces generó falta y a veces generó sobrecarga en el número de simulaciones manejadas, a pesar de nuestros esfuerzos para contrarrestar este comportamiento a través de la elección de los umbrales.

Ulteriores discusiones con José Villén-Altamirano y Pedro R. D'Argenio nos han llevado a pensar que la táctica de escoger un valor global de división debe ser descartada, si se quiere lograr una elección (quasi) óptima de umbrales. En lugar de ello es preciso seleccionar ambos el umbral y la cantidad de división a realizar cuando se lo alcanza, en un procedimiento adaptativo

Un diagrama del algoritmo para análisis transitorio sería así[†]:

0. Inicialmente considerar a todos los valores de importancia de la función actual como umbrales potenciales;

1. Lanzar n simulaciones RESTART piloto con división global 2;

2. Si fuese necesario forzar su terminación temprana, pues se aconseja usar menos del 10 % del presupuesto de cómputo en estas *decisiones preliminares*;

3. Aproximar las probabilidades $\{P_{i|0}\}$ de [VAVA06] con el cociente entre, el número de simulaciones que alcanzaron el i-ésimo valor de importancia, y $2^i\, n$;

4. Aproximar las probabilidades $P_{i+1|1} = P_{i+1|0}P_{1|0}$ con las aproximaciones del ítem anterior;

5. Usando esos valores, calcular los coeficientes de división acumulada $\{r_i\}$ de [VAVA06] usando la ecuación $r_i = \left(P_{i|0}P_{i+1|1}\right)^{-1/2} \in \mathbb{Q}$ de dicho trabajo;

[†] José Villén-Altamirano fue quien propuso la idea original para análisis estacionario, posteriormente revisada y actualizada en discusiones con el gran Arnd Hartmanns.

6. Calcular iterativamente los valores (enteros) de división $\{R_i\}$ para cada umbral potencial mediante la fórmula

$$R_i = \text{round}\left(\frac{r_i}{\prod_{j=1}^{i-1} R_i}\right);$$

7. Si $R_i \leqslant 1$ entonces el i-ésimo valor de importancia no es un umbral;

8. En caso contrario sí lo es, y las simulaciones que lo alcancen deben dividirse en $R_i \in \mathbb{N}$ simulaciones.

Existen muchas otras mejoras potenciales para los resultados de esta tesis, por ejemplo en los algoritmos mediante los que se deriva la función de importancia. A pesar de la eficiencia que demostraron en nuestros experimentos, vale la pena probar otras variantes, quizás en una versión extendida de IOSA, o quizás en otros formalismos de modelado.

Una modificación inmediata es considerar los pesos probabilísticos de las transiciones en una adaptación del Algoritmo 1. Nótese que esto no siempre es factible, e.g. las cadenas de Markov suelen representarse con tipos abstractos de datos que lo permitirían, pero la componente estocástica de IOSA (como se lo presentó aquí) está codificada en un formato más difícil de aprovechar, *escondida en los relojes*.

Otra extensión para nuestra estrategia composicional en particular, es desarrollar nuevas técnicas de componer las funciones locales de importancia. Hemos descubierto que una expresión en DNF del evento raro es ambas natural y útil a la hora de derivar una estrategia de composición. No obstante, otras formas más complejos pero también estructuradas para expresar la propiedad pueden considerarse, sin que la capacidad de destilar una estrategia de composición a partir de ellas se vea afectada. En esta dirección estamos considerando actualmente la teoría de Árboles Dinámicos de Fallas con reparaciones (ver [RS15] y sus referencias internas).

En una apreciación más alejada, esta tesis construye la función de importancia basándose en la estructura del modelo. La propiedad del evento raro es un componente esencial, más aún para las metodologías composicionales, pero la distancia entre valores que nutre a la función de importancia está impresa en el grafo de adyacencia del modelo del sistema. Un enfoque diferente consiste en revertir esta estrategia, construyendo la función a partir de la expresión de la propiedad y modificándola a medida que se circula por el modelo, como Sedwards et al. hacen en [JLS13, JLST15]. En esta dirección consideramos que los *counting fluents* de [RDDA15] podrían proveer un marco más rico para derivar la función.

Por último pero no menos importante, una de las principales motivaciones de esta tesis es el desarrollo de herramientas de software que ofrezcan implementaciones de nuestras propuestas listas para ser descargadas y usadas. En ese sentido BLUEMOON fue desarrollado principalmente como un prototipo para corroborar

la validez de nuestras estrategias, mientras que el diseño y la implementación de FIG fue planeado a mayor escala.

Sería muy interesante ver un desarrollo creciente de la herramienta FIG. Por un lado hay muchas optimizaciones en eficiencia al alcance de la mano, como una paralelización trivial de los mecanismos de actualización de los intervalos, que podrían mejorar el desempeño de las estimaciones a muy bajo costo implementativo. Por otro lado hay cuestiones más profundas, derivadas del núcleo algorítmico de la herramienta, que también afectan al desempeño general. El mecanismo de selección de umbrales y el truncado determinista de caminos de ejecución inútiles son un par de ejemplos. Estudiar el efecto de un cambio en dichos algoritmos suena prometedor, principalmente en lo que concierne a la elección de los umbrales.

A su vez, FIG actualmente implementa un único algoritmo de división por importancia. Si bien este algoritmo (RESTART) fue escogido por sus buenas cualidades generales, no hay ningún vínculo singular que ligue a la herramienta con RESTART. Recuérdese que la función de importancia puede ser empleada como una caja negra por la mayoría de las técnicas de división multinivel. Esto, sumado al diseño modular de FIG, deberían hacer que el añadido de nuevos *motores de simulación* (como Esfuerzo Fijo, ver la Subsección 2.5.2) sea una tarea relativamente sencilla.

Apéndice: Modelos de sistemas

A

Este apéndice presenta el código fuente de todos los modelos utilizados para producir los resultados incluidos en esta tesis. Empleamos dos formalismos de modelado: todos los sistemas estudiados en el Capítulo 3 están modelados en el lenguaje de entrada de PRISM; aquellos estudiados en el Capítulo 4 están descriptos usando la sintaxis de modelado IOSA.

A.1. Cola tándem

Modelo PRISM de una cola tándem en un entorno de tiempo continuo, usado para producir los resultados presentados en la Subsección 3.5.2.

```
ctmc

const int c;                // Queues capacity
const double lambda = 3;    // rate(-> q1          )
const double    mu1 = 2;    // rate(    q1 -> q2   )
const double    mu2 = 6;    // rate(          q2 ->)
// Values taken from Marnix Garvels' Ph.D. Thesis:
// The splitting method in rare event simulation, p. 85.

module ContinuousTandemQueue

    q1: [0..c-1] init 0;
    q2: [0..c-1] init 1;
    arr:  [0..2] init 0;  // Arrival: (0:none) (1:lost) (2:successful)
    lost: [0..1] init 0;  // Package loss in q2: (0:none) (1:lost)

    // Package arrival at first queue
    [] q1<c-1 -> lambda: (arr'=2) & (lost'=0) & (q1'=q1+1);
    [] q1=c-1 -> lambda: (arr'=1) & (lost'=0);

    // Passing from first to second queue
    [] q1>0 & q2<c-1 -> mu1: (arr'=0) & (lost'=0) & (q1'=q1-1)
                                                  & (q2'=q2+1);
    [] q1>0 & q2=c-1 -> mu1: (arr'=0) & (lost'=1) & (q1'=q1-1);

    // Package departure from second queue
    [] q2>0 -> mu2: (arr'=0) & (lost'=0) & (q2'=q2-1);

endmodule

label "goal" = lost=1;
label "stop" = q2=0;
label "running" = q2!=0;
label "reference" = true;
```

A.2. Cola tándem de tiempo discreto

Modelo PRISM de una cola tándem en un entorno de tiempo discreto, usado para producir los resultados presentados en la Subsección 3.5.3.

```
dtmc

const int c;               // Queues capacity
const double parr = 0.1;   // Prob(-> q1           )
const double ps1 = 0.14;   // Prob(    q1 -> q2    )
const double ps2 = 0.19;   // Prob(           q2 ->)

module DiscreteTandemQueue

    q1: [0..c] init 0;
    q2: [0..c] init 0;
    arr1:  [0..2] init 0;  // Arrival: (0:none) (1:lost) (2:successful)
    lost2: [0..1] init 0;  // Package loss in q2: (0:none) (1:lost)

    [] (q1=0) & (q2=0)
        -> (parr): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
         + (1-parr): (arr1'=0) & (lost2'=0);

    [] (0<q1 & q1<c) & (q2=0)
        -> (parr*ps1): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
         + (parr*(1-ps1)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
         + (ps1*(1-parr)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
                                                   & (lost2'=0)
         + ((1-parr)*(1-ps1)): (arr1'=0) & (lost2'=0);

    [] (q1=c) & (q2=0)
        -> (parr*ps1): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
         + (parr*(1-ps1)): (arr1'=1) & (lost2'=0)
         + ((1-parr)*ps1): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
                                                   & (lost2'=0)
         + ((1-parr)*(1-ps1)): (arr1'=0) & (lost2'=0);

    [] (q1=0) & (0<q2)
        -> (parr*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2) & (lost2'=0)
         + (parr*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
         + ((1-parr)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*(1-ps2)): (arr1'=0) & (lost2'=0);

    [] (0<q1 & q1<c) & (0<q2 & q2<c)
        -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
         + (parr*ps1*(1-ps2)): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
         + (parr*(1-ps1)*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2)
                                                       & (lost2'=0)
         + (parr*(1-ps1)*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
         + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
                                                           & (lost2'=0)
         + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);

    [] (q1=c) & (0<q2 & q2<c)
        -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
         + (parr*ps1*(1-ps2)): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
         + (parr*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=1) & (lost2'=0)
         + (parr*(1-ps1)*(1-ps2)): (arr1'=1) & (lost2'=0)
```

```
          + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
          + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
                                                            & (lost2'=0)
          + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
          + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);

    [] (0<q1 & q1<c) & (q2=c)
        -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
         + (parr*ps1*(1-ps2)): (arr1'=2) & (lost2'=1)
         + (parr*(1-ps1)*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2)
                                                       & (lost2'=0)
         + (parr*(1-ps1)*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
         + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (arr1'=0) & (lost2'=1)
         + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);

    [] (q1=c) & (q2=c)
        -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
         + (parr*ps1*(1-ps2)): (arr1'=2) & (lost2'=1)
         + (parr*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=1) & (lost2'=0)
         + (parr*(1-ps1)*(1-ps2)): (arr1'=1) & (lost2'=0)
         + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (arr1'=0) & (lost2'=1)
         + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
         + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);
endmodule

label "goal" = lost2=1;
label "reference" = true;  // arr1!=0;
```

A.3. Cola abierta/cerrada

Modelo PRISM de una cola abierta/cerrada, usado para producir los resultados presentados en la Subsección 3.5.4.

```
ctmc

const int b;            // Open queue (oq) capacity
const int N2 = 1;       // Closed system (cq+cqq) fixed size
const double l = 1;     // oq arrival rate
const double m11 = 4;   // oq Server1 rate
const double m12 = 2;   // cq Server1 rate
const double m2;        // cq Server2 rate
// Values taken from Glasserman, Heidelberger, Shahabuddin, and Zajic:
// Multilevel Splitting For Estimating Rare Event Probabilities,
// Operations Research, Vol. 47, No. 4, July-August 1999, pp. 585-600

// System queues
global oq: [0..b]  init 0;  // Open   queue
global cq: [0..N2] init 0;  // Closed queue

module Arrival
    lost: bool init false;
    [] oq<b-1 -> l: (oq'=oq+1);
    [] oq=b-1 -> l: (lost'=true);
endmodule

```

```
module Server1
    reset: bool init false;
    [] oq>1 & cq=0 -> m11: (oq'=oq-1);
    [] oq=1 & cq=0 -> m11: (reset'=true);
    [] cq>1        -> m12: (cq'=cq-1);
    [] cq=1 & oq>0 -> m12: (cq'=cq-1);
    [] cq=1 & oq=0 -> m12: (reset'=true);
endmodule

module Server2
    [] cq<N2 -> m2: (cq'=cq+1);
endmodule

label "goal" = lost;
label "stop" = reset;
label "running" = !reset;
```

A.4. Sistema de colas con rupturas

Modelo PRISM de una cola con rupturas, usado para producir los resultados presentados en la Subsección 3.5.5.

```
ctmc

// The following values were extracted from Kroese & Nicola:
// Efficient estimation of overflow probabilities in queues
// with breakdowns, Performance Evaluation, 36-37, 1999, pp. 471-484.
// This model corresponds to the system described in the section 4.4
// (p. 481) of said article.

// Buffer capacity
const int k;

// Server
const double mu = 100;
const double xi = 3;
const double delta = 4;

// Sources of Type 1
const int NSrc1 = 5;
const double lambda1 = 3;
const double alpha1  = 3;
const double beta1   = 2;

// Sources of Type 2
const int NSrc2 = 5;
const double lambda2 = 6;
const double alpha2  = 1;
const double beta2   = 4;

module QueueWithBreakdowns

    // Initializations
    lost:  bool init false;
    reset: bool init false;
    buf: [0..K-1] init 1;     // Buffer, initially with one customer
    server: bool init false;  // Server, initially down
    src1: [0..NSrc1] init 0;  // Sources of Type 1, initially none active
```

```
37      src2: [0..NSrc2] init 1;  // Sources of Type 2, initially  one active
38
39      // Sources failure and recovery
40      [] src1>0     -> (src1 * beta1)          : (src1'=src1-1);
41      [] src1<NSrc1 -> ((NSrc1-src1) * alpha1) : (src1'=src1+1);
42      [] src2>0     -> (src2 * beta2)          : (src2'=src2-1);
43      [] src2<NSrc2 -> ((NSrc2-src2) * alpha2) : (src2'=src2+1);
44
45      // Server failure and recovery
46      []  server -> xi: (server'=false);
47      [] !server -> delta: (server'=true);
48
49      // Buffer in
50      [] src1>0 & buf<K-1 -> (src1 * lambda1) : (buf'=buf+1);
51      [] src1>0 & buf=K-1 -> (src1 * lambda1) : (lost'=true);
52      [] src2>0 & buf<K-1 -> (src2 * lambda2) : (buf'=buf+1);
53      [] src2>0 & buf=K-1 -> (src2 * lambda2) : (lost'=true);
54
55      // Buffer out
56      [] server & buf>1 -> mu : (buf'=buf-1);
57      [] server & buf=1 -> mu : (reset'=true);
58  endmodule
59
60  label "goal" = lost;
61  label "stop" = reset;
62  label "running" = !reset;
```

A.5. Sistema de base de datos con redundancia

Modelo PRISM de una instalación de base de datos con redundancia, usado en el Ejemplo 7.

```
1   ctmc
2
3   // The following values were extracted from José Villén-Altamirano,
4   // Importance functions for RESTART simulation of highly-dependable
5   // systems, Simulation, Vol. 83, Issue 12, December 2007, pp. 821-828.
6
7   // Redundancy level, viz. how many breaks produce a system failure
8   const int RED;
9
10  // Processors
11  global P1: [0..RED] init RED;
12  global P2: [0..RED] init RED;
13  const double PF = 2000;   // Processors' mean time to failure (in hours)
14  const double IPF = 0.01;  // Processors' inter-type failure rate
15
16  // Controllers
17  global C1: [0..RED] init RED;
18  global C2: [0..RED] init RED;
19  const double CF = 2000;  // Controllers' mean time to failure (in hours)
20
21  // Disk clusters
22  global D1: [0..RED+2] init RED+2;
23  global D2: [0..RED+2] init RED+2;
24  global D3: [0..RED+2] init RED+2;
25  global D4: [0..RED+2] init RED+2;
26  global D5: [0..RED+2] init RED+2;
```

```
27 global D6: [0..RED+2] init RED+2;
28 const double DF = 6000;  // Disks' mean time to failure (in hours)
29
30 // Repair rates for failures of type 1 and 2 resp.
31 const double R1 = 1.0;
32 const double R2 = 0.5;
33
34 module Processors
35     [] P1 > 0 -> (P1/PF)*(1-IPF): (P1'=P1-1)
36                + (P1/PF)*(  IPF): (P1'=P1-1)&(P2'=P2-1);
37     [] P2 > 0 -> (P2/PF)*(1-IPF): (P2'=P2-1)
38                + (P2/PF)*(  IPF): (P2'=P2-1)&(P1'=P1-1);
39 endmodule
40
41 module Controllers
42     [] C1>0 -> C1/CF: (C1'=C1-1);
43     [] C2>0 -> C2/CF: (C2'=C2-1);
44 endmodule
45
46 module DiskClusters
47     [] D1>0 -> D1/DF: (D1'=D1-1);
48     [] D2>0 -> D2/DF: (D2'=D2-1);
49     [] D3>0 -> D3/DF: (D3'=D3-1);
50     [] D4>0 -> D4/DF: (D4'=D4-1);
51     [] D5>0 -> D5/DF: (D5'=D5-1);
52     [] D6>0 -> D6/DF: (D6'=D6-1);
53 endmodule
54
55 // Number of failed components in the system
56 formula NFails = (2*RED-P1-P2)
57                + (2*RED-C1-C2)
58                + (6*(RED+2)-D1-D2-D3-D4-D5-D6);
59
60 // Operational Components in the minimal cutset
61 formula minOC = min(P1, P2,
62                     C1, C2,
63                     D1-2, D2-2, D3-2, D4-2, D5-2, D6-2);
64
65 module Repairman
66     f: bool init false;
67     // Type 1 failures on processors ...
68     [] !f & P1<RED -> 0.5 * R1 * (RED-P1)/NFails: (P1'=P1+1)
69                     + 0.5 * R1 * (RED-P1)/NFails: (P1'=P1+1) & (f'=!f);
70     [] !f & P2<RED -> 0.5 * R1 * (RED-P2)/NFails: (P2'=P2+1)
71                     + 0.5 * R1 * (RED-P2)/NFails: (P2'=P2+1) & (f'=!f);
72     // ... on controllers ...
73     [] !f & C1<RED -> 0.5 * R1 * (RED-C1)/NFails: (C1'=C1+1)
74                     + 0.5 * R1 * (RED-C1)/NFails: (C1'=C1+1) & (f'=!f);
75     [] !f & C2<RED -> 0.5 * R1 * (RED-C2)/NFails: (C2'=C2+1)
76                     + 0.5 * R1 * (RED-C2)/NFails: (C2'=C2+1) & (f'=!f);
77     // ... and on disks.
78     [] !f & D1<RED+2 -> 0.5 * R1 * (RED+2-D1)/NFails: (D1'=D1+1)
79                       + 0.5 * R1 * (RED+2-D1)/NFails: (D1'=D1+1) & (f'=!f);
80     [] !f & D2<RED+2 -> 0.5 * R1 * (RED+2-D2)/NFails: (D2'=D2+1)
81                       + 0.5 * R1 * (RED+2-D2)/NFails: (D2'=D2+1) & (f'=!f);
82     [] !f & D3<RED+2 -> 0.5 * R1 * (RED+2-D3)/NFails: (D3'=D3+1)
83                       + 0.5 * R1 * (RED+2-D3)/NFails: (D3'=D3+1) & (f'=!f);
84     [] !f & D4<RED+2 -> 0.5 * R1 * (RED+2-D4)/NFails: (D4'=D4+1)
85                       + 0.5 * R1 * (RED+2-D4)/NFails: (D4'=D4+1) & (f'=!f);
86     [] !f & D5<RED+2 -> 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1)
87                       + 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1) & (f'=!f);
```

```
88      [] !f & D6<RED+2 -> 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1)
89                        + 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1) & (f'=!f);
90      // Type 2 failures on processors ...
91      [] f & P1<RED -> 0.5 * R2 * (RED-P1)/NFails: (P1'=P1+1)
92                     + 0.5 * R2 * (RED-P1)/NFails: (P1'=P1+1) & (f'=!f);
93      [] f & P2<RED -> 0.5 * R2 * (RED-P2)/NFails: (P2'=P2+1)
94                     + 0.5 * R2 * (RED-P2)/NFails: (P2'=P2+1) & (f'=!f);
95      // ... on controllers ...
96      [] f & C1<RED -> 0.5 * R2 * (RED-C1)/NFails: (C1'=C1+1)
97                     + 0.5 * R2 * (RED-C1)/NFails: (C1'=C1+1) & (f'=!f);
98      [] f & C2<RED -> 0.5 * R2 * (RED-C2)/NFails: (C2'=C2+1)
99                     + 0.5 * R2 * (RED-C2)/NFails: (C2'=C2+1) & (f'=!f);
100     // ... and on disks.
101     [] f & D1<RED+2 -> 0.5 * R2 * (RED+2-D1)/NFails: (D1'=D1+1)
102                      + 0.5 * R2 * (RED+2-D1)/NFails: (D1'=D1+1) & (f'=!f);
103     [] f & D2<RED+2 -> 0.5 * R2 * (RED+2-D2)/NFails: (D2'=D2+1)
104                      + 0.5 * R2 * (RED+2-D2)/NFails: (D2'=D2+1) & (f'=!f);
105     [] f & D3<RED+2 -> 0.5 * R2 * (RED+2-D3)/NFails: (D3'=D3+1)
106                      + 0.5 * R2 * (RED+2-D3)/NFails: (D3'=D3+1) & (f'=!f);
107     [] f & D4<RED+2 -> 0.5 * R2 * (RED+2-D4)/NFails: (D4'=D4+1)
108                      + 0.5 * R2 * (RED+2-D4)/NFails: (D4'=D4+1) & (f'=!f);
109     [] f & D5<RED+2 -> 0.5 * R2 * (RED+2-D5)/NFails: (D5'=D5+1)
110                      + 0.5 * R2 * (RED+2-D5)/NFails: (D5'=D5+1) & (f'=!f);
111     [] f & D6<RED+2 -> 0.5 * R2 * (RED+2-D6)/NFails: (D6'=D6+1)
112                      + 0.5 * R2 * (RED+2-D6)/NFails: (D6'=D6+1) & (f'=!f);
113 endmodule
114
115 label "reference" = true;
116 label "goal" = (P1=0) | (P2=0)
117              | (C1=0) | (C2=0)
118              | (D1<=2) | (D2<=2) | (D3<=2) | (D4<=2) | (D5<=2) | (D6<=2);
```

A.6. Cola tándem

Modelo IOSA de una cola tándem en un entorno de tiempo continuo, usado para producir los resultados presentados en la Subsección 4.6.2.

```
1  const int c = 8;  // Capacity of both queues
2
3  // The following values were taken from Marnix Garvels' PhD Thesis:
4  // The splitting method in rare event simulation, p. 85.
5  const int lambda = 3;  // rate(-> q1         )
6  const int    mu1 = 2;  // rate(   q1 -> q2   )
7  const int    mu2 = 6;  // rate(         q2 ->)
8
9  // The following values are in p. 61 of the same work:
10 // const int lambda = 1;
11 // const int    mu1 = 4;
12 // const int    mu2 = 2;
13
14 module Arrivals
15     clk0: clock;  // External arrivals ~ Exponential(lambda)
16     [P0!] @ clk0 -> (clk0'= exponential(lambda));
17 endmodule
18
19 module Queue1
20     q1: [0..c];
21     clk1: clock;  // Queue1 processing ~ Exponential(mu1)
```

```
22      // Packet arrival
23      [P0?] q1 == 0          -> (q1'= q1+1) & (clk1'= exponential(mu1));
24      [P0?] q1  > 0 & q1 < c -> (q1'= q1+1);
25      [P0?] q1 == c          -> ;
26      // Packet processing
27      [P1!] q1 == 1 @ clk1 -> (q1'= q1-1);
28      [P1!] q1  > 1 @ clk1 -> (q1'= q1-1) & (clk1'= exponential(mu1));
29  endmodule
30
31  module Queue2
32      q2: [0..c] init 1;
33      clk2: clock;  // Queue2 processing ~ Exponential(mu2)
34      // Packet arrival
35      [P1?] q2 == 0          -> (q2'= q2+1) & (clk2'= exponential(mu2));
36      [P1?] q2  > 0 & q2 < c -> (q2'= q2+1);
37      [P1?] q2 == c          -> ;
38      // Packet processing
39      [P2!] q2 == 1 @ clk2 -> (q2'= q2-1);
40      [P2!] q2  > 1 @ clk2 -> (q2'= q2-1) & (clk2'= exponential(mu2));
41  endmodule
42
43  properties
44      P( q2 > 0 U q2 == c )  // transient
45      S( q2 == c )           // steady-state
46  endproperties
```

A.7. Cola tándem (alternativa)

Modelo PRISM de una cola tándem en un entorno de tiempo continuo, usado para producir los resultados presentados en la Subsección 4.6.2. Esta cola, modelada en el lenguaje de entrada de PRISM, es equivalente a la que se describe usando la sintaxis de modelado IOSA en la Apéndice A.6.

```
1   ctmc
2
3   const int c;                // Queues capacity
4   const double lambda = 3;    // rate(--> q1           )
5   const double    mu1 = 2;    // rate(    q1 --> q2    )
6   const double    mu2 = 6;    // rate(           q2 -->)
7
8   module Arrival
9       // External packet arrival
10      [P0] true -> lambda: true;
11  endmodule
12
13  module Queue1
14      q1: [0..c] init 0;
15      // Packet arrival
16      [P0] q1<c -> 1: (q1'=q1+1);
17      [P0] q1=c -> 1: true;
18      // Packet processing
19      [P1] q1>0 -> mu1: (q1'=q1-1);
20  endmodule
21
22  module Queue2
23      q2: [0..c] init 1;
24      // Packet arrival
```

```
    [P1] q2<c -> 1: (q2'=q2+1);
    [P1] q2=c -> 1: true;
    // Packet processing
    [P2] q2>0 -> mu2: (q2'=q2-1);
endmodule
```

A.8. Cola tándem triple

Modelo IOSA de una cola tándem triple no markoviana, usado para producir los resultados presentados en la Subsección 4.6.3.

```
// The following values were extracted from José Villén-Altamirano,
// RESTART simulation of networks of queues with Erlang service times,
// Winter Simulation Conference, 2009, pp. 1146-1154.
// This model corresponds to the system described in Section 4.1

const int a = 2;    // Service time shape parameter ('alpha', all queues)
const int b1 = 3;   // Service time scale parameter ('beta1', Queue1)
const int b2 = 4;   // Service time scale parameter ('beta2', Queue2)
const int b3 = 6;   // Service time scale parameter ('beta3', Queue3)
const int L = 7;    // Threshold occupancy (Queue3)
const int c = L+5;  // Queues capacity (all queues)

// Combinations tested in J. V-A's article:
//     L  alpha beta1 beta2 beta3
//  A) 18   2     3     4     6
//  B) 13   3    4.5    6     9
//  C) 20   2     6     4     6
//  D) 16   3     9     6     9
//  E) 24   2    10     8     6
//  F) 21   3    15    12     9
//
// Those values of 'L' yield rare events of probability ~ 10-15.
// Alternatively the following values yield rare events ~ 10-9:
// L = (A:11, B:7, C:11, D:9, E:14, F:12)

module Arrivals
    clk0: clock;  // External arrivals ~ Exponential(1)
    [P0!] @ clk0 -> (clk0'= exponential(1));
endmodule

module Queue1
    q1: [0..c];
    clk1: clock;  // Queue1 processing ~ Erlang(alpha;beta1)
    // Packet arrival
    [P0?] q1 == 0          -> (q1'= q1+1) & (clk1'= erlang(a,b1));
    [P0?] q1  > 0 & q1 < c -> (q1'= q1+1);
    [P0?] q1 == c          -> ;
    // Packet processing
    [P1!] q1 == 1 @ clk1 -> (q1'= q1-1);
    [P1!] q1  > 1 @ clk1 -> (q1'= q1-1) & (clk1'= erlang(a,b1));
endmodule

module Queue2
    q2: [0..c];
    clk2: clock;  // Queue2 processing ~ Erlang(alpha;beta2)
    // Packet arrival
    [P1?] q2 == 0          -> (q2'= q2+1) & (clk2'= erlang(a,b2));
```

```
48      [P1?] q2  > 0 & q2 < c -> (q2'= q2+1);
49      [P1?] q2 == c          -> ;
50      // Packet processing
51      [P2!] q2 == 1 @ clk2 -> (q2'= q2-1);
52      [P2!] q2  > 1 @ clk2 -> (q2'= q2-1) & (clk2'= erlang(a,b2));
53  endmodule
54
55  module Queue3
56      q3: [0..c];
57      clk3: clock;  // Queue3 processing ~ Erlang(alpha;beta3)
58      // Packet arrival
59      [P2?] q3 == 0          -> (q3'= q3+1) & (clk3'= erlang(a,b3));
60      [P2?] q3  > 0 & q3 < c -> (q3'= q3+1);
61      [P2?] q3 == c          -> ;
62      // Packet processing
63      [P3!] q3 == 1 @ clk3 -> (q3'= q3-1);
64      [P3!] q3  > 1 @ clk3 -> (q3'= q3-1) & (clk3'= erlang(a,b3));
65  endmodule
66
67  properties
68      S( q3 >= L )  // steady-state
69  endproperties
```

A.9. Sistema de colas con rupturas

Versión resumida del modelo IOSA para una cola con rupturas, usado para producir los resultados presentados en la Subsección 4.6.4.

```
1  // The following values were extracted from Kroese & Nicola,
2  // Efficient estimation of overflow probabilities in queues
3  // with breakdowns, Performance Evaluation, 36-37, 1999, pp. 471-484.
4  // This model corresponds to the system described in Section 4.4
5
6  // Sources of Type 1
7  const int lambda1 = 3;  // Production rate
8  const int alpha1  = 3;  // Repair     rate
9  const int beta1   = 2;  // Fail       rate
10
11 // Sources of Type 2
12 const int lambda2 = 6;  // Production rate
13 const int alpha2  = 1;  // Repair     rate
14 const int beta2   = 4;  // Fail       rate
15
16 // Server
17 const int mu = 100;     // Processing rate
18 const int delta = 4;    // Repair     rate
19 const int gama = 3;     // Fail       rate
20
21 // Buffer capacity: 40, 60, 80, 100, 120, 140, 160
22 const int K = 120;
23
24 ////////////////////////////////////////////////////////////////////
25 //
26 // Type 1 Sources | Total: 5
27 //                | Intially on: 0
28
29 module T1S1
30     on11: bool init false;
```

```
31      clkF11: clock;  // Type 1 sources Failures   ~ exp(beta1)
32      clkR11: clock;  // Type 1 sources Repairs    ~ exp(alpha1)
33      clkP11: clock;  // Type 1 sources Production ~ exp(lambda1)
34      // Breakdowns
35      []  on11 @ clkF11 -> (on11'= false) &
36                           (clkR11'= exponential(alpha1));
37      [] !on11 @ clkR11 -> (on11'= true) &
38                           (clkF11'= exponential(beta1)) &
39                           (clkP11'= exponential(lambda1));
40      // Production
41      [p11!]  on11 @ clkP11 -> (clkP11'= exponential(lambda1));
42  endmodule
```

⋮

```
122 module T1S5
123     on15: bool init false;
124     clkF15: clock;  // Type 1 sources Failures   ~ exp(beta1)
125     clkR15: clock;  // Type 1 sources Repairs    ~ exp(alpha1)
126     clkP15: clock;  // Type 1 sources Production ~ exp(lambda1)
127     // Breakdowns
128     []  on15 @ clkF15 -> (on15'= false) &
129                          (clkR15'= exponential(alpha1));
130     [] !on15 @ clkR15 -> (on15'= true) &
131                          (clkF15'= exponential(beta1)) &
132                          (clkP15'= exponential(lambda1));
133     // Production
134     [p15!]  on15 @ clkP15 -> (clkP15'= exponential(lambda1));
135 endmodule
136
137 ////////////////////////////////////////////////////////////////////////
138 //
139 // Type 2 Sources | Total: 5
140 //                | Intially on: 1
141
142 module T2S1
143     on21: bool init true;
144     clkF21: clock;  // Type 2 sources Failures   ~ exp(beta2)
145     clkR21: clock;  // Type 2 sources Repairs    ~ exp(alpha2)
146     clkP21: clock;  // Type 2 sources Production ~ exp(lambda2)
147     // Breakdowns
148     []  on21 @ clkF21 -> (on21'= false) &
149                          (clkR21'= exponential(alpha2));
150     [] !on21 @ clkR21 -> (on21'= true) &
151                          (clkF21'= exponential(beta2)) &
152                          (clkP21'= exponential(lambda2));
153     // Production
154     [p21!]  on21 @ clkP21 -> (clkP21'= exponential(lambda2));
155 endmodule
```

⋮

```
203 module T2S5
204     on25: bool init false;
205     clkF25: clock;  // Type 2 sources Failures   ~ exp(beta2)
206     clkR25: clock;  // Type 2 sources Repairs    ~ exp(alpha2)
207     clkP25: clock;  // Type 2 sources Production ~ exp(lambda2)
208     // Breakdowns
209     []  on25 @ clkF25 -> (on25'= false) &
210                          (clkR25'= exponential(alpha2));
211     [] !on25 @ clkR25 -> (on25'= true) &
212                          (clkF25'= exponential(beta2)) &
```

```
213                          (clkP25'= exponential(lambda2));
214     // Production
215     [p25!]  on25 @ clkP25 -> (clkP25'= exponential(lambda2));
216 endmodule
217
218 ////////////////////////////////////////////////////////////////////
219 //
220 // Buffered server | Keeps track of 'overflow' and 'reset'
221 //                 | Translated from bluemoon's homonymous model
222
223 module BufferedServer
224     buf: [0..K] init 1;
225     clkF: clock;  // Server Failure    ~ exp(gama)
226     clkR: clock;  // Server Repair     ~ exp(delta)
227     clkP: clock;  // Server Processing ~ exp(mu)
228     on:    bool init false;  // Server on?
229     reset: bool init false;
230     // Server failure and recovery
231     []  on @ clkF -> (on'= false) &
232                      (clkR'= exponential(delta));
233     [] !on @ clkR -> (on'= true)  &
234                      (clkF'= exponential(gama)) &
235                      (clkP'= exponential(mu));
236     // Buffer out (dequeueing by server processing)
237     [] on & buf  > 1 @ clkP -> (buf'= buf-1) &
238                                (clkP'= exponential(mu));
239     [] on & buf == 1 @ clkP -> (buf'= buf-1) &
240                                (reset'= true);
241     // Buffer in (enqueueing by sources production)
242     [p11?] buf == 0  -> (buf'= buf+1) & (clkP'= exponential(mu));
243     [p11?] 0 < buf & buf < K  -> (buf'= buf+1);
244     [p11?] buf == K  -> ;
          ⋮
277     [p25?] buf == 0  -> (buf'= buf+1) & (clkP'= exponential(mu));
278     [p25?] 0 < buf & buf < K  -> (buf'= buf+1);
279     [p25?] buf == K  -> ;
280 endmodule
281
282 properties
283     P( !reset U buf == K )  // transient
284 endproperties
```

A.10. Sistema de base de datos con redundancia

Versión resumida de un modelo IOSA para una base de datos con redundancia 2. Usamos estos modelos para producir los resultados presentados en la Subsección 4.6.5. El número de componentes del sistema se incrementa al aumentar la redundacia.

```
1 const int PF = 50;   // Processors'  mean time to failure
2 const int CF = 50;   // Controllers' mean time to failure
3 const int DF = 150;  // Disks'       mean time to failure
4
5 ////////////////////////////////////////////////////////////////////
6 //
7 // Disk clusters | Num clusters: 6
```

```
//               | Redundancy per cluster: 4
//               | Mean time to failure: DF
//               | Num failures to breakdown per cluster: 2

module Disk11
    d11f: bool init false;  // Disk failed?
    d11t: [1..2];           // Failure type
    d11cF1: clock;          // Type 1 failure ~ exp(1/(DF*2))
    d11cF2: clock;          // Type 2 failure ~ exp(1/(DF*2))
    d11cR1: clock;          // Repair for type 1 failure ~ exp(1.0)
    d11cR2: clock;          // Repair for type 2 failure ~ exp(0.5)
    [] !d11f          @ d11cF1 -> (d11f'= true)  &
                                  (d11t'= 1)     &
                                  (d11cR1'= exponential(1.0));
    [] !d11f          @ d11cF2 -> (d11f'= true)  &
                                  (d11t'= 2)     &
                                  (d11cR2'= exponential(0.5));
    [] d11f & d11t==1 @ d11cR1 -> (d11f'= false) &
                                  (d11cF1'= exponential(1/(DF*2))) &
                                  (d11cF2'= exponential(1/(DF*2)));
    [] d11f & d11t==2 @ d11cR2 -> (d11f'= false) &
                                  (d11cF1'= exponential(1/(DF*2))) &
                                  (d11cF2'= exponential(1/(DF*2)));
endmodule
```

⋮

```
module Disk64
    d64f: bool init false;  // Disk failed?
    d64t: [1..2];           // Failure type
    d64cF1: clock;          // Type 1 failure ~ exp(1/(DF*2))
    d64cF2: clock;          // Type 2 failure ~ exp(1/(DF*2))
    d64cR1: clock;          // Repair for type 1 failure ~ exp(1.0)
    d64cR2: clock;          // Repair for type 2 failure ~ exp(0.5)
    [] !d64f          @ d64cF1 -> (d64f'= true)  &
                                  (d64t'= 1)     &
                                  (d64cR1'= exponential(1.0));
    [] !d64f          @ d64cF2 -> (d64f'= true)  &
                                  (d64t'= 2)     &
                                  (d64cR2'= exponential(0.5));
    [] d64f & d64t==1 @ d64cR1 -> (d64f'= false) &
                                  (d64cF1'= exponential(1/(DF*2))) &
                                  (d64cF2'= exponential(1/(DF*2)));
    [] d64f & d64t==2 @ d64cR2 -> (d64f'= false) &
                                  (d64cF1'= exponential(1/(DF*2))) &
                                  (d64cF2'= exponential(1/(DF*2)));
endmodule

////////////////////////////////////////////////////////////////
//
// Controllers | Num types: 2
//             | Redundancy per type: 2
//             | Mean time to failure: CF

module Controller11
    c11f: bool init false;  // Controller failed?
    c11t: [1..2];           // Failure type
    c11cF1: clock;          // Type 1 failure ~ exp(1/(CF*2))
    c11cF2: clock;          // Type 2 failure ~ exp(1/(CF*2))
    c11cR1: clock;          // Repair for type 1 failure ~ exp(1.0)
    c11cR2: clock;          // Repair for type 2 failure ~ exp(0.5)
```

```
507     [] !c11f          @ c11cF1 -> (c11f'= true)  &
508                                   (c11t'= 1)     &
509                                   (c11cR1'= exponential(1.0));
510     [] !c11f          @ c11cF2 -> (c11f'= true)  &
511                                   (c11t'= 2)     &
512                                   (c11cR2'= exponential(0.5));
513     [] c11f & c11t==1 @ c11cR1 -> (c11f'= false) &
514                                   (c11cF1'= exponential(1/(CF*2))) &
515                                   (c11cF2'= exponential(1/(CF*2)));
516     [] c11f & c11t==2 @ c11cR2 -> (c11f'= false) &
517                                   (c11cF1'= exponential(1/(CF*2))) &
518                                   (c11cF2'= exponential(1/(CF*2)));
519 endmodule
      ⋮
583 ////////////////////////////////////////////////////////////////
584 //
585 // Processors | Num types: 2
586 //            | Redundancy per type: 2
587 //            | Mean time to failure: PF
588
589 module Processor11
590     p11f: bool init false;  // Processor failed?
591     p11t: [1..2];           // Failure type
592     p11cF1: clock;          // Type 1 failure ~ exp(1/(PF*2))
593     p11cF2: clock;          // Type 2 failure ~ exp(1/(PF*2))
594     p11cR1: clock;          // Repair for type 1 failure ~ exp(1.0)
595     p11cR2: clock;          // Repair for type 2 failure ~ exp(0.5)
596     [] !p11f          @ p11cF1 -> (p11f'= true)  &
597                                   (p11t'= 1)     &
598                                   (p11cR1'= exponential(1.0));
599     [] !p11f          @ p11cF2 -> (p11f'= true)  &
600                                   (p11t'= 2)     &
601                                   (p11cR2'= exponential(0.5));
602     [] p11f & p11t==1 @ p11cR1 -> (p11f'= false) &
603                                   (p11cF1'= exponential(1/(PF*2))) &
604                                   (p11cF2'= exponential(1/(PF*2)));
605     [] p11f & p11t==2 @ p11cR2 -> (p11f'= false) &
606                                   (p11cF1'= exponential(1/(PF*2))) &
607                                   (p11cF2'= exponential(1/(PF*2)));
608 endmodule
      ⋮
672 properties
673   S( (d11f & d12f) | (d11f & d13f) | (d11f & d14f) |  // Disk cl. #1
674      (d12f & d13f) | (d12f & d14f) | (d13f & d14f) |
675      (d21f & d22f) | (d21f & d23f) | (d21f & d24f) |  // Disk cl. #2
676      (d22f & d23f) | (d22f & d24f) | (d23f & d24f) |
677      (d31f & d32f) | (d31f & d33f) | (d31f & d34f) |  // Disk cl. #3
678      (d32f & d33f) | (d32f & d34f) | (d33f & d34f) |
679      (d41f & d42f) | (d41f & d43f) | (d41f & d44f) |  // Disk cl. #4
680      (d42f & d43f) | (d42f & d44f) | (d43f & d44f) |
681      (d51f & d52f) | (d51f & d53f) | (d51f & d54f) |  // Disk cl. #5
682      (d52f & d53f) | (d52f & d54f) | (d53f & d54f) |
683      (d61f & d62f) | (d61f & d63f) | (d61f & d64f) |  // Disk cl. #6
684      (d62f & d63f) | (d62f & d64f) | (d63f & d64f) |
685      (c11f & c12f) |  // Controllers type 1
686      (c21f & c22f) |  // Controllers type 2
687      (p11f & p12f) |  // Processors type 1
688      (p21f & p22f) )  // Processors type 2
```

```
689 endproperties
```

A.11. Tuberías de petróleo o sistema C(k,n: F)

Versión resumida de un modelo IOSA para un sistema C(k, n: F) no markoviano con reparaciones, para $n = 20$ y $k = 3$. Usamos estos modelos para producir los resultados presentados en la Subsección 4.6.6.. El número de componentes del sistema se incrementa al aumentar n, pero es independiente de k.

```
1  // These distributions are used in Section 4.1 of José Villén-
2  // Altamirano: RESTART simulation of non-Markov consecutive-k-
3  // out-of-n:F repairable systems, Reliability Engineering and
4  // System Safety, Vol. 95, Issue 3, 2010, pp. 247-254:
5  //  - Repair time ~ Lognormal(1.21,0.8)
6  //  - Nodes lifetime ~ Exponential(lambda) or Rayleigh(sigma)
7  //                     for (lambda,sigma) in {(0.001 ,  798.000),
8  //                                            (0.0003, 2659.615),
9  //                                            (0.0001, 7978.845)}
10
11 module BE_pipe1
12     c_fail1:   clock;
13     c_repair1: clock;
14     inform1: [0..2] init 0;  // 0 idle, 1 inform fail, 2 inform repair
15     broken1: [0..2] init 0;  // 0 operational, 1 broken, 2 under repair
16     // failing (by itself)
17     [fpipe1!] broken1==0 & inform1==0 @ c_fail1 -> (inform1'= 1) &
18                                                    (broken1'= 1);
19     [fail1!!] inform1 == 1 -> (inform1'= 0);
20     // reparation (with repairman)
21     [repair1??] broken1==1 & inform1==0
22                 -> (broken1'= 2) &
23                    (c_repair1'= lognormal(1.21,0.8));
24     [rpipe1!] broken1 == 2 @ c_repair1 -> (inform1'= 2) &
25                                           (broken1'= 0) &
26                                           (c_fail1'= rayleigh(729));
27     [repaired1!!] inform1 == 2 -> (inform1'= 0);
28 endmodule
```

⋮

```
370 module BE_pipe20
371     c_fail20:   clock;
372     c_repair20: clock;
373     inform20: [0..2] init 0;  // 0 idle, 1 inform fail, 2 inform repair
374     broken20: [0..2] init 0;  // 0 operational, 1 broken, 2 under repair
375     // failing (by itself)
376     [fpipe20!] broken20==0 & inform20==0 @ c_fail20 -> (inform20'= 1) &
377                                                        (broken20'= 1);
378     [fail20!!] inform20 == 1 -> (inform20'= 0);
379     // reparation (with repairman)
380     [repair20??] broken20==1 & inform20==0
381                  -> (broken20'= 2) &
382                     (c_repair20'= lognormal(1.21,0.8));
383     [rpipe20!] broken20 == 2 @ c_repair20 -> (inform20'= 2) &
384                                              (broken20'= 0) &
385                                              (c_fail20'= rayleigh(729));
386     [repaired20!!] inform20 == 2 -> (inform20'= 0);
387 endmodule
```

```
388
389 module Repairman
390     xs[20] : bool init false;  // Array of booleans
391     busy   : bool init false;
392     // Register a failure
393     [ fail1?? ] -> (xs[0]'= true);
                ⋮
411     [ fail20?? ] -> (xs[19]'= true);
412     // Begin a repair
413     [ repair1!! ] busy == false & fsteq(xs,true) == 0
414                   -> (busy'= true);
                ⋮
450     [ repair20!! ] busy == false & fsteq(xs,true) == 19
451                    -> (busy'= true);
452     // Finish a repair
453     [ repaired1?? ] -> (busy'= false) & (xs[0]'= false);
                ⋮
471     [ repaired20?? ] -> (busy'= false) & (xs[19]'= false);
472 endmodule
473
474 properties
475   S( ( broken1>0 & broken2>0 & broken3>0 ) |
476      ( broken2>0 & broken3>0 & broken4>0 ) |
                           ⋮
491      ( broken17>0 & broken18>0 & broken19>0 ) |
492      ( broken18>0 & broken19>0 & broken20>0 ) )
493 endproperties
```

Apéndice: Teoría de la medida

B

Algunos conceptos fundamentales de teoría de la medida son compendiados aquí. Estos conceptos son útiles para comprender las definiciones y resultados que presentamos en el Apéndice C, y también para entender los aspectos más teóricos de las Subsecciones 2.3.3 y 3.3.4 y de la Sección 4.4.

Todos los resultados en este apéndice son presentados sin demostraciones. El lector interesado puede encontrar una introducción completa a la teoría de la medida en textos clásicos como [Bre68] y el más moderno [Dur10]. El tutorial online de N. Vaillant, disponible en `www.probability.net`, también es muy recomendado.

En lo que sigue Ω denotará un conjunto arbitrario y 2^Ω será el conjunto de todas sus partes. Si $A \in 2^\Omega$ entonces A^c denotará su complemento, i.e. $A \cap A^\mathsf{c} = \varnothing$ y $A \uplus A^\mathsf{c} = \Omega$. Los bloques básicos sobre los que se construye la teoría de la medida son las estructuras algebraicas conocidas como σ-álgebras:

Definición 23 (σ-álgebra)**.** Una *σ-álgebra* sobre Ω es una colección $\mathscr{F} \subseteq 2^\Omega$ que satisface: $\Omega \in \mathscr{F}$, $A \in \mathscr{F} \Rightarrow A^\mathsf{c} \in \mathscr{F}$, y para cualquier familia numerable de subconjuntos de Ω, digamos $\{\Omega_i\}_{i\in\mathbb{N}}$, su unión numerable también se encuentra en la σ-álgebra, viz. $\bigcup_{i\in\mathbb{N}} \Omega_i \in \mathscr{F}$.

Si $\mathscr{F}$ es una σ-álgebra sobre Ω, el par $(\Omega, \mathscr{F})$ se denota un *espacio medible*. Las *σ-álgebras triviales* de Ω son $\{\varnothing, \Omega\}$ y 2^Ω. Los elementos de $\mathscr{F}$ en un espacio medible $(\Omega, \mathscr{F})$ son llamados *conjuntos medibles* de la σ-álgebra.

Cualquier colección $\mathscr{C} \subseteq 2^\Omega$ puede ser transformada en una σ-álgebra, que denotaremos $\sigma(\mathscr{C})$, incluyendo en $\sigma(\mathscr{C})$ los complementos y uniones numerables de los conjuntos originalmente en $\mathscr{C}$. De esta forma uno *genera* una σ-álgebra a partir de una colección arbitraria de subconjuntos de Ω. Este concepto tiene una definición alternativa que damos a continuación.

Definición 24. Sea $\mathscr{C} \subseteq 2^\Omega$, entonces la *$\sigma$-álgebra generada por $\mathscr{C}$* es la intersección de todas las σ-álgebras que contienen a $\mathscr{C}$, y se denota $\sigma(\mathscr{C})$, i.e.

$$\sigma(\mathscr{C}) \doteq \bigcap \left\{ \mathscr{F} \subseteq 2^\Omega \;\middle|\; \mathscr{C} \subseteq \mathscr{F} \wedge \mathscr{F} \text{ es } \sigma\text{-álgebra} \right\}.$$

Los elementos de $\mathscr{C}$ son llamados *generadores*.

Propiedad 11. *Sea $\mathscr{C} \subseteq 2^\Omega$, entonces $\sigma(\mathscr{C})$ es una σ-álgebra, y de hecho es la menor σ-álgebra que contiene a $\mathscr{C}$.*

La Definición 24 provee los medios para generar σ-álgebras a partir de colecciones arbitrarias de subconjuntos de Ω. En particular, también se puede generar una nueva σ-álgebra a partir de otras σ-álgebras.

La noción que buscamos introducir es análoga a la del producto cartesiano, la cual para conjuntos $\{\Omega_i\}_{i=1}^n$ y colecciones correspondientes $\{\mathscr{C}_i\}_{i=1}^n$ defina un *rectángulo* $A \subseteq \prod_{i=1}^n \Omega_i$ como cualquier conjunto $A = A_1 \times A_2 \times \cdots \times A_n = \prod_{i=1}^n A_i$ t.q. $A_i \in \mathscr{C}_i \cup \{\Omega_i\}$ para todo $i = 1, \ldots, n$. Para obtener la σ-álgebra producto es necesario trabajar con *rectángulos medibles*.

Definición 25. Sea $\mathcal{C} = \{(\Omega_i, \mathscr{F}_i)\}_{i=1}^n$ una colección finita de espacios medible. Un *rectángulo medible* es cualquier rectángulo de $\prod_{i=1}^n \Omega_i$ conformado por conjuntos medibles de $\{\mathscr{F}_i\}_{i=1}^n$.
La *σ-álgebra producto* de $\mathcal{C}$, denotada $\bigotimes_{i=1}^n \mathscr{F}_i$, es la σ-álgebra generada por los rectángulos medibles, viz.

$$\bigotimes_{i=1}^{n} \mathscr{F}_i \doteq \sigma\left(\left\{\prod_{i=1}^{n} A_i \,\middle|\, A_i \in \mathscr{F}_i \text{ para todo } i = 1, 2, \ldots, n\right\}\right).$$

Como el producto es finito también escribiremos $\mathscr{F}_1 \otimes \mathscr{F}_2 \otimes \cdots \otimes \mathscr{F}_n = \bigotimes_{i=1}^n \mathscr{F}_i$.

Hasta ahora hemos tratado *aspectos estructurales* de la medibilidad. Sin embargo, el nombre mismo de la teoría proviene de una perspectiva más *dinámica* si se quiere, que involucra funciones actuantes sobre dichas estructuras. La teoría de la medida se ocupa de aquello que se puede *medir* (y de lo que no); en su núcleo yacen los conceptos de *medida* y *medida de probabilidad*.

Definición 26. Sea $\mathscr{C} \subseteq 2^\Omega$ t.q. $\varnothing \in \mathscr{C}$ y sea $\mu \colon \mathscr{C} \to [0, \infty)$. La función μ es una *medida en $\mathscr{C}$* si $\mu(\varnothing) = 0$ y si es σ-aditiva, i.e. para cualquier secuencia $\{A_i\}_{i\in\mathbb{N}}$ de elementos de $\mathscr{C}$ donde $\biguplus_{i\in\mathbb{N}} A_i \in \mathscr{C}$, μ satisface $\mu(\biguplus_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} \mu(A_i)$. Si a su vez $\mu(\Omega) = 1$ (y por ende $\mu \colon \mathscr{C} \to [0, 1]$), entonces μ es una *medida de probabilidad en $\mathscr{C}$*.

De esta forma, dado un espacio medible $(\Omega, \mathscr{F})$, una medida μ sobre $\mathscr{F}$, y un conjunto medible $A \in \mathscr{F}$, la medida de A según μ es $\mu(A) \in [0, \infty)$. La *medida Dirac de probabilidad* concentrada en $\{\omega\} \subseteq \Omega$, que denotaremos δ_ω, es la única medida de probabilidad t.q. $\delta_\omega(Q) = 1$ si $\omega \in Q$ y $\delta_\omega(Q) = 0$ en caso contrario, para todo $Q \in \mathscr{F}$.

La noción de medibilidad puede ser extendida para considerar el espacio de las funciones, lo que introduce el concepto de *funciones medibles* que definimos a continuación:

Definición 27. Sean $(\Omega_1, \mathscr{F}_1)$ y $(\Omega_2, \mathscr{F}_2)$ espacios medibles. Entonces $f \colon \Omega_1 \to \Omega_2$ es una *función medible* si la imagen inversa de todo conjunto medible de $\mathscr{F}_2$ es un conjunto medible de $\mathscr{F}_1$, i.e.

$$\forall B \in \mathscr{F}_2 \,.\, f^{-1}(B) \in \mathscr{F}_1 \,.$$

La calidad de medible de f en la definición previa es a veces denotada $f\colon (\Omega_1, \mathscr{F}_1) \to (\Omega_2, \mathscr{F}_2)$. A pesar de que la Definición 27 pueda parecer artificial, varios conceptos matemáticos estándar muy difundidos hacen uso de funciones medibles. La teoría de las probabilidades es un buen ejemplo, donde una función medible sobre un espacio probabilístico es comúnmente conocida como una *variable aleatoria*.

Este apéndice concluye introduciendo algunos conceptos que serán usados ampliamente en el Apéndice C. Dado un espacio medible $(\Omega, \mathscr{F})$, es usual denotar $\Delta(\Omega)$ al conjunto de todas las medidas de probabilidad sobre $\mathscr{F}$. Además existe una construcción estándar por [Gir82] para dotar a $\Delta(\Omega)$ con una σ-álgebra. $\Delta(\mathscr{F})$ denotará pues a la *σ-álgebra de Giry* generada por los conjuntos de medidas de probabilidad,

$$\Delta^B(Q) \doteq \{\, \mu\colon \mathscr{F} \to [0,1] \mid \mu(Q) \in B \,\},$$

donde $B \in \mathscr{B}([0,1])$ y $Q \in \mathscr{F}$. Aquí $\mathscr{B}(\Upsilon)$ es la *σ-álgebra de Borel* sobre el conjunto Υ, viz. la σ-álgebra generada por los conjuntos abiertos de Υ. La siguiente proposición enuncia que la σ-álgebra de Giry puede ser generada de manera numerable.

Proposición 12. *Sea $(\Omega, \mathscr{F})$ un espacio medible y denotemos $\Delta^{\geqslant p}(Q)$ al conjunto de medidas de probabilidad $\{\mu \mid \mu(Q) \geqslant p\}$ para $Q \in \mathscr{F}$ y $p \in [0,1]$. Entonces*

$$\Delta(\mathscr{F}) = \sigma\left(\{\Delta^p(Q) \,|\, p \in \mathbb{Q} \cap [0,1]\}\right).$$

Apéndice: LMP no deterministas

C

Los procesos de Markov etiquetados no deterministas (NLMP por sus siglas en inglés, [DSW12]) son el resultado de muchos esfuerzos por proveer a los *procesos de Markov etiquetados* de [Des99, DEP02] con no determinismo interno. NLMP sobresale entre otros intentos por lograr esta meta, dado que siguen la misma estrategia de Desharnais et al. en [DEP02], quienes se basan en las sólidas bases provistas por la teoría de la medida (ver el Apéndice B).

El objetivo general es extender las capacidades de modelado de los procesos de Markov con: un espacio continuo de estados; evolución temporal continua; no determinismo externo, i.e. gobernado por el entorno; y no determinismo interno, i.e. decidido por cada proceso. El formalismo de los procesos de Markov etiquetados cubre los tres primeros puntos, usando un conjunto etiquetado de acciones que codifican las interacciones con el entorno. Este formalismo define modelos reactivos donde diferentes transiciones de probabilidad se habilitan con cada acción. La incertidumbre es (sólo) considerada de manera probabilista.

Definición 28 (LMP,[DEP02])**.** Un *proceso de Markov etiquetado* (LMP por sus siglas en inglés) es una tupla $(S, \Sigma, \{\tau_a \mid a \in \mathcal{L}\})$ donde (S, Σ) es un espacio medible y, para cada acción $a \in \mathcal{L}$, la *función de transición de probabilidad* $\tau_a \colon S \to \Delta(S) \cup \{\mathbf{0}\}$ es una función medible, donde $\mathbf{0} \colon \Sigma \to [0, 1]$ denota a la *medida nula* t.q. $\mathbf{0}(Q) = 0$ para todo $Q \in \Sigma$.

El valor $\tau_a(s)(Q) \in [0, 1]$ representa la probabilidad de realizar una transición cayendo en algún estado de Q, provisto que el sistema se encuentra en el estado s y que la acción a ha sido aceptada. Por consiguiente, la transición de probabilidad en realidad es una probabilidad condicional, donde la probabilidad de Q está condicionada en que el sistema se encuentra en el estado s y que efectivamente reacciona a la acción a. Originalmente [Des99] permite que $\tau_a(s)$ sea una *medida de sub-probabilidad* (i.e. podría ocurrir que $\tau_a(s)(S) < 1$). En lugar de ello y siguiendo a [BDSW14], diremos que $\tau_a(s) = \mathbf{0}$ cuando la acción a es rechazada.

Los procesos de Markov etiquetados no deterministas fueron introducidos en [DSW12, DWTC09] como una generalización de los LMP para incluir no determinismo interno. Específicamente, permiten que transiciones de probabilidad etiquetadas con la misma acción salgan del mismo estado. Dos restricciones impuestas por el formalismo de NLMP lo diferencian de otras teorías que persiguen metas similares:

(a) la función de transición mapea estados a *conjuntos medibles* de medidas de probabilidad, y

(b) toda función de transición debe ser medible.

La restricción (a) está motivada por el uso de *schedulers* (también conocidos como *políticas*) para resolver el no determinismo. Permitir que el objetivo sean conjuntos de medidas arbitrarios haría que la teoría sufra de problemas de (no) medibilidad, concretamente la decisión para tomar acciones futuras podría no ser cuantificable. La restricción (b) está relacionada con el uso de operadores modales, como los que permiten los LMP. Tratar con funciones de transición no medibles podría volver imposible el medir ciertas trazas de ejecución. Para ver ejemplos ilustrativos que motivan estas restricciones sugerimos la lectura de [Wol12, BDSW14].

Definición 29 (NLMP,[DSW12])**.** Un *proceso de Markov etiquetado no determinista* (NLMP por sus siglas en inglés) es una tupla $(S, \Sigma, \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ donde:

- (S, Σ) es un espacio medible,
- para cada etiqueta $a \in \mathcal{L}$ la *función de transición no determinista* $\mathcal{T}_a \colon S \to \Delta(\Sigma)$ es medible.

Nótese que la noción de medibilidad para $\mathcal{T}_a$ requiere de la definición de una σ-álgebra sobre su codominio (la σ-álgebra de Giry $\Delta(\Sigma)$). Dicha definición es una construcción central del formalismo de los NLMP:

Definición 30 (σ-álgebra de impactos)**.** Sea (S, Σ) como en la Definición 29, entonces $H(\Delta(\Sigma))$ es la menor σ-álgebra que contiene a todos los conjuntos

$$H_\xi \doteq \{\zeta \in \Delta(\Sigma) \mid \zeta \cap \xi \neq \varnothing\}$$

para los conjuntos medibles $\xi \in \Delta(\Sigma)$.

En la Definición 30, H_ξ contiene a todos los conjuntos medibles que *impactan* contra el conjunto medible ξ. Obsérvese asimismo que $\mathcal{T}_a^{-1}(H_\xi)$ es el conjunto de todos los estados $s \in S$ que, a través de la etiqueta a, *impactan* contra el conjunto de medidas ξ. Por consiguiente, volviendo al análisis de la Definición 29, para cada acción $a \in \mathcal{L}$ la función de transición no determinista correspondiente $\mathcal{T}_a$ debe ser medible desde la σ-álgebra sobre los estados hacia la σ-álgebra de impactos de las medidas, i.e. $\mathcal{T}_a \colon (S, \Sigma) \to (\Delta(\Sigma), H(\Delta(\Sigma)))$.

Como cabría esperar, los LMP son un caso especial de los NLMP donde $\mathcal{T}_a$ es el conjunto unitario $\{\tau_a\}$ para toda etiqueta $a \in \mathcal{L}$. Claro está que esto requiere que las medidas de probabilidad individuales sean medibles en la σ-álgebra de Giry, viz. $\Delta(\Sigma)$ debe distinguir puntos. Si se satisface esta condición, puede probarse que $\mathcal{T}_a$ es medible sii τ_a también lo es [BDSW14].

A pesar de la estructura que la Definición 29 provee sobre el espacio de estados, la teoría aún puede sufrir de ciertos problemas de medibilidad derivados de un *uso inapropiado* (pero de todas formas *permitido* por la definición) de las etiquetas. Por esta razón los NLMP han sido extendidos en [Bud12] para darle estructura al conjunto de etiquetas $\mathcal{L}$. Así, en añadidura al espacio medible de estados, (S, Σ), hay un espacio medible de etiquetas, $(\mathcal{L}, \Lambda)$. *La* función de transición $\mathcal{T}$ resultante

mapea pues estados en conjuntos medibles de la σ-álgebra producto $\Lambda \otimes \Delta(\Sigma)$, y la σ-álgebra de impactos que hace a la medibilidad de $\mathcal{T}$ se define a través de rectángulos medibles.

Gran parte de la teoría de procesos de Markov etiquetados no deterministas está orientada al desarrollo de relaciones de bisimulación con diferentes grados de observabilidad. Bisimilitudes como se las define en [Mil80] para los LTS pueden extenderse sobre el mundo mucho más complejo de los LMP de varias formas. Dos nociones han sido desarrolladas por Desharnais et al., la primera de las cuales se define directamente sobre los estados [Des99]. Por ello esta definición puede separar sistemas potencialmente indistinguibles desde el punto de vista de Σ. Más adelante en [DDLP06] se define una relación de bisimulación "a través de eventos", que construye una concepción de equivalencia de comportamiento más a tono con la definición de LMP sobre el campo de la teoría de la medida. En alusión a la forma en que fueron definidas, [DDLP06] denota *bisimilitud de estado* a la primer relación (puntual), y *bisimilitud de evento* a la segunda relación.

Los NLMP poseen sus propias relaciones de bisimulación de evento y de estado, análogas a las de los LMP. Hay también una tercer noción, denotada *bisimulación de impacto* en [BDSW14], cuya granularidad de observación yace en algún punto entre las otras dos. Todas estas nociones coinciden bajo ciertas condiciones específicas. No obstante y en general la bisimilitud de estados es (estrictamente) la más fina y la bisimilitud de eventos es (estrictamente) la más gruesa [BDSW14]. Todas estas nociones también pueden ser aplicadas a los NLMP con estructura sobre las etiquetas [Bud12].

Bibliografía

[Bar84] Barendregt, H. P.: *The Lambda Calculus: Its Syntax and Semantics*. Sole Distributors for the U.S.A. And Canada, Elsevier Science Pub. Co., 1984.

[Bar14] Barbot, Benoît: *Acceleration for statistical model checking*. Tesis de Doctorado, École normale supérieure de Cachan, France, 2014. https://tel.archives-ouvertes.fr/tel-01149034.

[Bay70] Bayes, A. J.: *Statistical Techniques for Simulation Models*. Australian Computer Journal, 2(4):180–184, 1970.

[Bay72] Bayes, A. J.: *A Minimum Variance Sampling Technique for Simulation Models*. J. ACM, 19(4):734–741, 1972. http://doi.acm.org/10.1145/321724.321736.

[BCC+14] Brázdil, Tomás, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker y Mateusz Ujma: *Verification of Markov Decision Processes Using Learning Algorithms*. En *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volumen 8837 de *LNCS*, páginas 98–114. Springer, 2014, ISBN 978-3-319-11935-9. http://dx.doi.org/10.1007/978-3-319-11936-6_8.

[BDH15] Budde, Carlos E., Pedro R. D'Argenio y Holger Hermanns: *Rare Event Simulation with Fully Automated Importance Splitting*. En *EPEW 2015*, volumen 9272 de *LNCS*, páginas 275–290. Springer, 2015, ISBN 978-3-319-23266-9. http://dx.doi.org/10.1007/978-3-319-23267-6_18.

[BDH+17] Budde, Carlos E., Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges y Andrea Turrini: *JANI: Quantitative Model and Tool Interaction*. En Legay, Axel y Tiziana Margaria (editores): *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*, volumen 10206 de *LNCS*, páginas 151–168, 2017, ISBN 978-3-662-54579-9. http://dx.doi.org/10.1007/978-3-662-54580-5_9.

[BDHK06] Bohnenkamp, Henrik C., Pedro R. D'Argenio, Holger Hermanns y Joost Pieter Katoen: *MODEST: A Compositional Modeling Formalism for Hard and Softly Timed Systems*. IEEE Trans. Software Eng., 32(10):812–830, 2006.

[BDL+06] Behrmann, Gerd, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi y Martijn Hendriks: *UPPAAL 4.0*. En *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, páginas 125–126, Washington, DC, USA, 2006. IEEE Computer Society, ISBN 0-7695-2665-9. http://dx.doi.org/10.1109/QEST.2006.59.

[BDM17] Budde, Carlos E., Pedro R. D'Argenio y Raúl E. Monti: *Compositional Construction of Importance Functions in Fully Automated Importance Splitting*. En Puliafito, Antonio, Kishor S. Trivedi, Bruno Tuffin, Marco Scarpa, Fumio Machida y Javier Alonso (editores): *10th EAI International Conference on Performance Evaluation Methodologies and Tools VALUETOOLS 2016*. ACM, ACM, 2017, ISBN 978-1-63190-141-6. http://dx.doi.org/10.4108/eai.25-10-2016.2266501.

[BDSW14] Budde, Carlos E., Pedro R. D'Argenio, Pedro Sánchez Terraf y Nicolás Wolovick: *A Theory for the Semantics of Stochastic and Non-deterministic Continuous Systems*. En Remke, Anne y Mariëlle Stoelinga (editores): *ROCKS*, volumen 8453 de *LNCS*, páginas 67–86. Springer, 2014, ISBN 978-3-662-45488-6.

[Bel57] Bellman, Richard: *A Markovian decision process*. Informe técnico, DTIC Document, 1957.

[BFG+97] Bahar, R.I., E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo y F. Somenzi: *Algebric Decision Diagrams and Their Applications*. Formal Methods in System Design, 10(2):171–206, 1997, ISSN 1572-8102. http://dx.doi.org/10.1023/A:1008699807402.

[BFHH11] Bogdoll, Jonathan, Luis María Ferrer Fioriti, Arnd Hartmanns y Holger Hermanns: *Partial Order Methods for Statistical Model Checking and Simulation*. En *Formal Techniques for Distributed Systems - Joint 13th IFIP WG 6.1 International Conference, FMOODS 2011, and 31st IFIP WG 6.1 International Conference, FORTE 2011, Reykjavik, Iceland, June 6-9, 2011. Proceedings*, volumen 6722 de *LNCS*, páginas 59–74. Springer, 2011, ISBN 978-3-642-21460-8. http://dx.doi.org/10.1007/978-3-642-21461-5_4.

[BGC04] Baier, Christel, Marcus Größer y Frank Ciesinski: *Partial Order Reduction for Probabilistic Systems*. En *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands* [DBL04], páginas 230–239, ISBN 0-7695-2185-1. http://dx.doi.org/10.1109/QEST.2004.1348037.

[Bil12] Billingsley, P.: *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley, 2012, ISBN 9781118341919. https://books.google.com.ar/books?id=a3gavZbxyJcC.

[BK08] Baier, Christel y Joost Pieter Katoen: *Principles of Model Checking*. MIT press Cambridge, 2008.

[BKH99] Baier, Christel, Joost Pieter Katoen y Holger Hermanns: *Approximate Symbolic Model Checking of Continuous-Time Markov Chains*. En *Proceedings of the 10th International Conference on Concurrency Theory*, CONCUR '99, páginas 146–161, London, UK, 1999. Springer-Verlag, ISBN 3-540-66425-4. http://dl.acm.org/citation.cfm?id=646734.701464.

[BL02] Brinksma, Ed y Kim Guldstrand Larsen (editores): *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volumen 2404 de *LNCS*. Springer, 2002, ISBN 3-540-43997-8.

[Bre68] Breiman, L.: *Probability*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1968, ISBN 9780898712964. https://books.google.nl/books?id=ylyyoUQXkeAC.

[Bry86] Bryant, Randal E.: *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Trans. Comput., 35(8):677–691, Agosto 1986, ISSN 0018-9340. http://dx.doi.org/10.1109/TC.1986.1676819.

[Bud12] Budde, Carlos E.: *No-determinismo completamente medible en procesos probabilísticos continuos*. Tesis de Licenciatura, Universidad Nacional de Córdoba, Argentina, 2012.

[BvdP02] Blom, Stefan y Jaco van de Pol: *State Space Reduction by Proving Confluence*. En Brinksma, Ed y Kim Guldstrand Larsen [BL02], páginas 596–609, ISBN 3-540-43997-8. http://dx.doi.org/10.1007/3-540-45657-0_50.

[CAB05] Ching, J., S.K. Au y J.L. Beck: *Reliability estimation for dynamical systems subject to stochastic excitation using subset simulation with splitting*. Computer Methods in Applied

Mechanics and Engineering, 194(12–16):1557 – 1579, 2005, ISSN 0045-7825. http://www.sciencedirect.com/science/article/pii/S0045782504004050, Special Issue on Computational Methods in Stochastic Mechanics and Reliability Analysis.

[CDMFG12] Cérou, F., P. Del Moral, T. Furon y A. Guyader: *Sequential Monte Carlo for rare event estimation*. Statistics and Computing, 22(3):795–808, 2012, ISSN 1573-1375. http://dx.doi.org/10.1007/s11222-011-9231-6.

[CE82] Clarke, Edmund M. y E. Allen Emerson: *Design and synthesis of synchronization skeletons using branching time temporal logic*, páginas 52–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, ISBN 978-3-540-39047-3. http://dx.doi.org/10.1007/BFb0025774.

[CES86] Clarke, E. M., E. A. Emerson y A. P. Sistla: *Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications*. ACM Trans. Program. Lang. Syst., 8(2):244–263, Abril 1986, ISSN 0164-0925. http://doi.acm.org/10.1145/5397.5399.

[CFM+93] Clarke, Edmund M., M. Fujita, P. C. McGeer, K. McMillan, JC Y. Yang y X Zhao: *Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation*. 1993.

[CG07] Cérou, Frédéric y Arnaud Guyader: *Adaptive multilevel splitting for rare event analysis*. Stochastic Analysis and Applications, 25(2):417–443, 2007.

[CR65] Chow, Y. S. y Herbert Robbins: *On the Asymptotic Theory of Fixed-Width Sequential Confidence Intervals for the Mean*. The Annals of Mathematical Statistics, 36(2):457–462, Abril 1965. http://dx.doi.org/10.1214/aoms/1177700156.

[CW96] Clarke, Edmund M. y Jeannette M. Wing: *Formal Methods: State of the Art and Future Directions*. ACM Comput. Surv., 28(4):626–643, 1996. http://doi.acm.org/10.1145/242223.242257.

[dAKN+00] Alfaro, Luca de, Marta Kwiatkowska, Gethin Norman, David Parker y Roberto Segala: *Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation*, páginas 395–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, ISBN 978-3-540-46419-8. http://dx.doi.org/10.1007/3-540-46419-0_27.

[DBL04] *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The*

Netherlands. IEEE Computer Society, 2004, ISBN 0-7695-2185-1. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9341.

[DD09] Dean, Thomas y Paul Dupuis: *Splitting for rare event simulation: A large deviation approach to design and analysis*. Stochastic Processes and their Applications, 119(2):562 – 587, 2009, ISSN 0304-4149. http://www.sciencedirect.com/science/article/pii/S0304414908000598.

[DDLP06] Danos, Vincent, Josee Desharnais, François Laviolette y Prakash Panangaden: *Bisimulation and cocongruence for probabilistic systems*. Inf. Comput., 204(4):503–523, 2006. http://dx.doi.org/10.1016/j.ic.2005.02.004.

[DEP02] Desharnais, Josee, Abbas Edalat y Prakash Panangaden: *Bisimulation for Labelled Markov Processes*. Inf. Comput., 179(2):163–193, 2002.

[Des99] Desharnais, Josée: *Labelled markov processes*. Tesis de Doctorado, McGill University, Montréal, 1999.

[DHLS16] D'Argenio, Pedro R., Arnd Hartmanns, Axel Legay y Sean Sedwards: *Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata*. En *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*, volumen 9681 de *LNCS*, páginas 99–114. Springer, 2016, ISBN 978-3-319-33692-3. http://dx.doi.org/10.1007/978-3-319-33693-0_7.

[DJJL02] D'Argenio, Pedro R., Bertrand Jeannet, Henrik Ejersbo Jensen y Kim Guldstrand Larsen: *Reduction and Refinement Strategies for Probabilistic Analysis*. Volumen 2399 de *LNCS*, páginas 57–76. Springer, 2002, ISBN 3-540-43913-7. http://dx.doi.org/10.1007/3-540-45605-8_5.

[DJKV16] Dehnert, Christian, Sebastian Junges, Joost Pieter Katoen y Matthias Volk: *The Probabilistic Model Checker Storm*. arXiv preprint arXiv:1610.08713, 2016.

[DK05] D'Argenio, Pedro R. y Joost Pieter Katoen: *A theory of stochastic systems part I: Stochastic automata*. Inf. Comput., 203(1):1–38, 2005.

[DLM16] D'Argenio, Pedro R., Matías David Lee y Raúl E. Monti: *Input/Output Stochastic Automata - Compositionality and Determinism*. En *FORMATS 2016*, volumen 9884 de *LNCS*, páginas 53–68. Springer, Springer, 2016, ISBN 978-3-319-44877-0. http://dx.doi.org/10.1007/978-3-319-44878-7_4.

[DLST15] D'Argenio, Pedro, Axel Legay, Sean Sedwards y Louis-Marie Traonouez: *Smart sampling for lightweight verification of Markov decision processes*. STTT, 17(4):469–484, 2015. http://dx.doi.org/10.1007/s10009-015-0383-0.

[DN04] D'Argenio, Pedro R. y Peter Niebert: *Partial Order Reduction on Concurrent Probabilistic Programs*. En *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands* [DBL04], páginas 240–249, ISBN 0-7695-2185-1. http://dx.doi.org/10.1109/QEST.2004.1348038.

[DSW12] D'Argenio, Pedro R., Pedro Sánchez Terraf y Nicolás Wolovick: *Bisimulations for non-deterministic labelled Markov processes*. Mathematical Structures in Computer Science, 22(1):43–68, 2012.

[Dur10] Durrett, R.: *Probability: Theory and Examples*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2010, ISBN 9781139491136. https://books.google.com.ar/books?id=evbGTPhuvSoC.

[dVRLR09] Vega Rodrigo, Miguel de, Guy Latouche y Marie Ange Remiche: *Modeling bufferless packet-switching networks with packet dependencies*. Computer Networks, 53(9):1450 – 1466, 2009, ISSN 1389-1286. http://www.sciencedirect.com/science/article/pii/S1389128609000280.

[DWTC09] D'Argenio, Pedro R., Nicolás Wolovick, Pedro Sánchez Terraf y Pablo Celayes: *Nondeterministic Labeled Markov Processes: Bisimulations and Logical Characterization*. páginas 11–20. IEEE Computer Society, 2009, ISBN 978-0-7695-3808-2. http://dx.doi.org/10.1109/QEST.2009.17.

[Gar00] Garvels, Marnix Joseph Johann: *The splitting method in rare event simulation*. Tesis de Doctorado, Department of Computer Science, University of Twente, 2000.

[GHML11] Guyader, Arnaud, Nicolas Hengartner y Eric Matzner-Løber: *Simulation and Estimation of Extreme Quantiles and Extreme Probabilities*. Applied Mathematics & Optimization, 64(2):171–196, 2011, ISSN 1432-0606. http://dx.doi.org/10.1007/s00245-011-9135-z.

[GHSZ98] Glasserman, Paul, Philip Heidelberger, Perwez Shahabuddin y Tim Zajic: *A large deviations perspective on the efficiency of multilevel splitting*. IEEE Transactions on Automatic Control, 43(12):1666–1679, 1998.

[GHSZ99] Glasserman, Paul, Philip Heidelberger, Perwez Shahabuddin y Tim Zajic: *Multilevel splitting for estimating rare event probabilities*. Operations Research, 47(4):585–600, 1999.

[GI89] Glynn, Peter W. y Donald L. Iglehart: *Importance Sampling for Stochastic Simulations*. Management Science, 35(11):1367–1392, 1989. http://dx.doi.org/10.1287/mnsc.35.11.1367.

[Gir82] Giry, Michèle: *A categorical approach to probability theory*, páginas 68–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, ISBN 978-3-540-39041-1. http://dx.doi.org/10.1007/BFb0092872.

[GK98] Garvels, Marnix J.J. y Dirk P. Kroese: *A comparison of RESTART implementations*. En *Proceedings of the 1998 Winter Simulation Conference*, volumen 1, páginas 601–608. IEEE, 1998. http://doc.utwente.nl/18637/.

[GRT09] Glynn, Peter W., Gerardo Rubino y Bruno Tuffin: *Robustness Properties and Confidence Interval Reliability Issues*, páginas 63–84. En [RT09b], 2009, ISBN 0470772697, 9780470772690. http://dx.doi.org/10.1002/9780470745403.ch4.

[GSH+92] Goyal, A., P. Shahabuddin, P. Heidelberger, V. F. Nicola y P. W. Glynn: *A unified framework for simulating Markovian models of highly dependable systems*. IEEE Transactions on Computers, 41(1):36–51, Jan 1992, ISSN 0018-9340.

[GVOK02] Garvels, Marnix J. J., Jan Kees C. W. Van Ommeren y Dirk P. Kroese: *On the importance function in splitting simulation*. European Transactions on Telecommunications, 13(4):363–371, 2002, ISSN 1541-8251. http://dx.doi.org/10.1002/ett.4460130408.

[Har15] Hartmanns, Arnd: *On the analysis of stochastic timed systems*. Tesis de Doctorado, Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken, 2015. http://scidok.sulb.uni-saarland.de/volltexte/2015/6054.

[Hei95] Heidelberger, Philip: *Fast Simulation of Rare Events in Queueing and Reliability Models*. ACM Trans. Model. Comput. Simul., 5(1):43–85, 1995. http://doi.acm.org/10.1145/203091.203094.

[HHHK12] Hahn, Ernst Moritz, Arnd Hartmanns, Holger Hermanns y Joost Pieter Katoen: *A Compositional Modelling and Analysis Framework for Stochastic Hybrid Systems*. Formal Methods in System Design, 2012. ISSN: 0925-9856.

[HJ94] Hansson, Hans y Bengt Jonsson: *A logic for reasoning about time and reliability*. Formal Aspects of Computing, 6(5):512–535, 1994, ISSN 1433-299X. http://dx.doi.org/10.1007/BF01211866.

[HMZ+12] Henriques, David, João Martins, Paolo Zuliani, André Platzer y Edmund M. Clarke: *Statistical Model Checking for Markov Decision Processes*. En *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*, páginas 84–93. IEEE Computer Society, 2012, ISBN 978-1-4673-2346-8. http://dx.doi.org/10.1109/QEST.2012.19.

[JLS13] Jegourel, Cyrille, Axel Legay y Sean Sedwards: *Importance Splitting for Statistical Model Checking Rare Properties*. En *CAV 13*, volumen 8044 de *LNCS*, páginas 576–591. Springer, 2013, ISBN 978-3-642-39798-1. http://dx.doi.org/10.1007/978-3-642-39799-8_38.

[JLST15] Jégourel, Cyrille, Axel Legay, Sean Sedwards y Louis-Marie Traonouez: *Distributed Verification of Rare Properties with Lightweight Importance Splitting Observers*. CoRR, abs/1502.01838, 2015. http://arxiv.org/abs/1502.01838.

[JS06] Juneja, Sandeep y Perwez Shahabuddin: *Rare-event simulation techniques: An introduction and recent advances*. Handbooks in Operations Research and Management Science, 13:291–350, 2006.

[KH51] Kahn, Herman y Ted E. Harris: *Estimation of particle transmission by random sampling*. National Bureau of Standards applied mathematics series, 12:27–30, 1951.

[KN99] Kroese, Dirk P y Victor F Nicola: *Efficient estimation of overflow probabilities in queues with breakdowns*. Performance Evaluation, 36:471–484, 1999.

[KNP07] Kwiatkowska, M., G. Norman y D. Parker: *Stochastic Model Checking*. En *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volumen 4486 de *LNCS (Tutorial Volume)*, páginas 220–270. Springer, 2007.

[KNP11] Kwiatkowska, M., G. Norman y D. Parker: *PRISM 4.0: Verification of Probabilistic Real-time Systems*. En *CAV'11*, volumen 6806 de *LNCS*, páginas 585–591. Springer, 2011.

[Kon80] Kontoleon, J. M.: *Reliability Determination of a r-Successive-out-of-n:F System*. IEEE Transactions on Reliability, R-29(5):437–437, 1980, ISSN 0018-9529.

[LDB10] Legay, Axel, Benoît Delahaye y Saddek Bensalem: *Statistical Model Checking: An Overview*. En Barringer, Howard, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky y Nikolai Tillmann (editores): *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volumen 6418 de *LNCS*, páginas 122–135. Springer, 2010, ISBN 978-3-642-16611-2. http://dx.doi.org/10.1007/978-3-642-16612-9_11.

[LDT07] L'Ecuyer, Pierre, Valérie Demers y Bruno Tuffin: *Rare Events, Splitting, and quasi-Monte Carlo*. ACM Trans. Model. Comput. Simul., 17(2), Abril 2007, ISSN 1049-3301. http://doi.acm.org/10.1145/1225275.1225280.

[LK00] Law, A.M. y W.D. Kelton: *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2000, ISBN 9780070592926.

[LLGLT09] L'Ecuyer, Pierre, François Le Gland, Pascal Lezaud y Bruno Tuffin: *Splitting Techniques*, páginas 39–61. En [RT09b], 2009, ISBN 0470772697, 9780470772690. http://dx.doi.org/10.1002/9780470745403.ch3.

[LMT09] L'Ecuyer, Pierre, Michel Mandjes y Bruno Tuffin: *Importance Sampling in Rare Event Simulation*, páginas 17–38. En [RT09b], 2009, ISBN 0470772697, 9780470772690. http://dx.doi.org/10.1002/9780470745403.ch2.

[LST14] Legay, Axel, Sean Sedwards y Louis-Marie Traonouez: *Scalable Verification of Markov Decision Processes*. En *Software Engineering and Formal Methods - SEFM 2014, Grenoble, France, September 1-2, 2014, Revised Selected Papers*, volumen 8938 de *LNCS*, páginas 350–362. Springer, 2014, ISBN 978-3-319-15200-4. http://dx.doi.org/10.1007/978-3-319-15201-1_23.

[LT11] L'Ecuyer, Pierre y Bruno Tuffin: *Approximating zero-variance importance sampling in a reliability setting*. Annals of Operations Research, 189(1):277–297, 2011, ISSN 0254-5330. http://dx.doi.org/10.1007/s10479-009-0532-5.

[LZ00] Lam, Yeh y Yuan Lin Zhang: *Repairable consecutive-k-out-of-n: F system with Markov dependence*. Naval Research Logistics (NRL), 47(1):18–39, 2000, ISSN 1520-6750. http://dx.doi.org/10.1002/(SICI)1520-6750(200002)47:1<18::AID-NAV2>3.0.CO;2-B.

[Mil80] Milner, Robin: *A Calculus of Communicating Systems*, volumen 92 de *LNCS*. Springer, 1980, ISBN 3-540-10235-3. http://dx.doi.org/10.1007/3-540-10235-3.

[Mil89] Milner, Robin: *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989, ISBN 978-0-13-115007-2.

[Nor98] Norris, J.R.: *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998, ISBN 9780521633963.

[Par02] Parker, David Anthony: *Implementation of symbolic model checking for probabilistic systems*. Tesis de Doctorado, University of Birmingham, 2002.

[Pnu77] Pnueli, Amir: *The Temporal Logic of Programs*. En *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, páginas 46–57, Washington, DC, USA, 1977. IEEE Computer Society. http://dx.doi.org/10.1109/SFCS.1977.32.

[PTVF07] Press, William H., Saul A. Teukolsky, William T. Vetterling y Brian P. Flannery: *Numerical Recipes*. Cambridge University Press, 3ª edición, 2007, ISBN 9780521880688.

[RA14] Real Academia de la Lengua Española y Asociación de Academias de la Lengua Española: *Diccionario de la lengua española*. Planeta Publishing, 23ª edición, 2014, ISBN 978-8467041897. http://dle.rae.es/.

[RdBS16] Reijsbergen, Daniël, Pieter Tjerk de Boer y Werner Scheinhardt: *Hypothesis Testing for Rare-Event Simulation: Limitations and Possibilities*, páginas 16–26. Springer International Publishing, Cham, 2016, ISBN 978-3-319-47166-2. http://dx.doi.org/10.1007/978-3-319-47166-2_2.

[RdBSH13] Reijsbergen, Daniël, Pieter Tjerk de Boer, Werner Scheinhardt y Boudewijn Haverkort: *Automated Rare Event Simulation for Stochastic Petri Nets*. En *QEST 2013*, volumen 8054 de *LNCS*, páginas 372–388. Springer, 2013, ISBN 978-3-642-40195-4. http://dx.doi.org/10.1007/978-3-642-40196-1_31.

[RDDA15] Regis, Germán, Renzo Degiovanni, Nicolás D'Ippolito y Nazareno Aguirre: *Specifying Event-Based Systems with a Counting Fluent Temporal Logic*. En Bertolino, Antonia, Gerardo Canfora y Sebastian G. Elbaum (editores): *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, páginas 733–743. IEEE Computer Society, 2015, ISBN 978-1-4799-1934-5. http://dx.doi.org/10.1109/ICSE.2015.86.

[Ric06] Rice, J.A.: *Mathematical Statistics and Data Analysis*. Cengage Learning, 2006, ISBN 9781111793715. https://books.google.com.ar/books?id=7SI8AAAAQBAJ.

[RS15] Ruijters, Enno y Mariëlle Stoelinga: *Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools*. Computer Science Review, 15:29–62, 2015. http://dx.doi.org/10.1016/j.cosrev.2015.03.001.

[RT09a] Rubino, Gerardo y Bruno Tuffin: *Introduction to Rare Event Simulation*, páginas 1–13. En [RT09b], 2009, ISBN 9780470745403. http://dx.doi.org/10.1002/9780470745403.ch1.

[RT09b] Rubino, Gerardo y Bruno Tuffin: *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, Ltd, 2009, ISBN 0470772697, 9780470772690.

[SS07] Spiegel, Murray R. y Larry J. Stephens: *Schaum's Outline of Statistics*. McGraw-Hill, 2007, ISBN 0071485848. http://doi.contentdirections.com/mr/mgh.jsp?doi=10.1036/0071485848.

[Tso92] Tsoucas, Pantelis: *Rare events in series of queues*. Journal of Applied Probability, 29(1):168–175, July 1992. https://www.cambridge.org/core/article/div-class-title-rare-events-in-series-of-queues-div/680A9CE91F6CCCFD0902C86A8AE6E601.

[VA98] Villén-Altamirano, José: *RESTART method for the case where rare events can occur in retrials from any threshold*. International Journal of Electronics and Communications, 52:183–189, 1998.

[VA07a] Villén-Altamirano, José: *Importance Functions for RESTART Simulation of Highly-Dependable Systems*. Simulation, 83(12):821–828, 2007. http://dx.doi.org/10.1177/0037549707081257.

[VA07b] Villén-Altamirano, José: *Rare event RESTART simulation of two-stage networks*. European J. of Operational Research, 179(1):148–159, 2007. http://dx.doi.org/10.1016/j.ejor.2006.02.026.

[VA09] Villén-Altamirano, José: *RESTART Simulation of Networks of Queues with Erlang Service Times*. En *Winter Simulation Conference*, WSC '09, páginas 1146–1154. Winter Simulation Conference, 2009, ISBN 978-1-4244-5771-7. http://dl.acm.org/citation.cfm?id=1995456.1995616.

[VA10] Villén-Altamirano, José: *RESTART simulation of non-Markov consecutive-k-out-of-n: F repairable systems*. Rel. Eng. & Sys. Safety, 95(3):247–254, 2010. http://dx.doi.org/10.1016/j.ress.2009.10.005.

[VA14] Villén-Altamirano, José: *Asymptotic optimality of RESTART estimators in highly dependable systems*. Reliability Engineering & System Safety, 130:115 – 124, 2014, ISSN 0951-8320. http://www.sciencedirect.com/science/article/pii/S0951832014001227.

[Val90] Valmari, Antti: *A Stubborn Attack On State Explosion*. En Clarke, Edmund M. y Robert P. Kurshan (editores): *Computer Aided Verification, 2nd International Workshop, CAV '90, New Brunswick, NJ, USA, June 18-21, 1990, Proceedings*, volumen 531 de *LNCS*, páginas 156–165. Springer, 1990, ISBN 3-540-54477-1. http://dx.doi.org/10.1007/BFb0023729.

[VAMMGFC94] Villén-Altamirano, Manuel, A. Martínez-Marrón, J. Gamo y F. Fernández-Cuesta: *Enhancement of the accelerated simulation method RESTART by considering multiple thresholds*. En *Proc. 14th Int. Teletraffic Congress*, páginas 797–810, 1994.

[VAVA91] Villén-Altamirano, Manuel y José Villén-Altamirano: *RESTART: a method for accelerating rare event simulations*. En *Queueing, Performance and Control in ATM (ITC-13)*, páginas 71–76. Elsevier, 1991.

[VAVA02] Villén-Altamirano, Manuel y José Villén-Altamirano: *Analysis of restart simulation: Theoretical basis and sensitivity study*. European Transactions on Telecommunications, 13(4):373–385, 2002. http://dx.doi.org/10.1002/ett.4460130409.

[VAVA06] Villén-Altamirano, Manuel y José Villén-Altamirano: *On the efficiency of RESTART for multidimensional state systems*. ACM Transactions on Modeling and Computer Simulation (TOMACS), 16(3):251–279, 2006.

[VAVA11] Villén-Altamirano, Manuel y José Villén-Altamirano: *The Rare Event Simulation Method RESTART: Efficiency Analysis and Guidelines for Its Application*. En *Network Performance Engineering*, volumen 5233 de *LNCS*, páginas 509–547. Springer, 2011. http://dx.doi.org/10.1007/978-3-642-02742-0_22.

[VAVA13] Villén-Altamirano, Manuel y José Villén-Altamirano: *Rare event simulation of non-Markovian queueing networks using RESTART method*. Simulation Modelling Practice and Theory, 37:70 – 78, 2013, ISSN 1569-190X. http://www.sciencedirect.com/science/article/pii/S1569190X13000919.

[VSS95] Vanglabbeek, R.J., S.A. Smolka y B. Steffen: *Reactive, Generative, and Stratified Models of Probabilistic Processes*. Information and Computation, 121(1):59 – 80, 1995,

ISSN 0890-5401. http://www.sciencedirect.com/science/article/pii/S0890540185711236.

[Wei84] Weiser, Mark: *Program Slicing*. IEEE Trans. Software Eng., 10(4):352–357, 1984. http://dx.doi.org/10.1109/TSE.1984.5010248.

[Wil27] Wilson, Edwin B.: *Probable Inference, the Law of Succession, and Statistical Inference*. Journal of the American Statistical Association, 22(158):209–212, 1927, ISSN 01621459. http://www.jstor.org/stable/2276774.

[Wol12] Wolovick, Nicolás: *Continuous Probability and Nondeterminism in Labeled Transition Systems*. Tesis de Doctorado, Universidad Nacional de Córdoba, Argentina, 2012.

[XLL07] Xiao, Gang, Zhizhong Li y Ting Li: *Dependability estimation for non-Markov consecutive-k-out-of-n: F repairable systems by fast simulation*. Reliability Engineering & System Safety, 92(3):293 – 299, 2007, ISSN 0951-8320. //www.sciencedirect.com/science/article/pii/S0951832006000949, Selected Papers Presented at the Fourth International Conference on Quality and Reliability, ICQR 2005, Fourth International Conference on Quality and Reliability.

[Yi90] Yi, Wang: *Real-Time Behaviour of Asynchronous Agents*. En Baeten, Jos C. M. y Jan Willem Klop (editores): *CONCUR '90*, volumen 458 de *LNCS*, páginas 502–520. Springer, 1990, ISBN 3-540-53048-7.

[YS02] Younes, Håkan L. S. y Reid G. Simmons: *Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling*. En Brinksma, Ed y Kim Guldstrand Larsen [BL02], páginas 223–235, ISBN 3-540-43997-8. http://dx.doi.org/10.1007/3-540-45657-0_17.

[ZM12] Zimmermann, Armin y Paulo Maciel: *Importance function derivation for RESTART simulations of Petri nets*. En *9th Int. Workshop on Rare Event Simulation (RESIM 2012)*, páginas 8–15, 2012.

[ZRWCL16] Zimmermann, Armin, Daniël Reijsbergen, Alexander Wichmann y Andres Canabal Lavista: *Numerical Results for the Automated Rare Event Simulation of Stochastic Petri Nets*. En *11th Int. Workshop on Rare Event Simulation (RESIM 2016)*, páginas 1–10, Eindhoven, Netherlands, 2016.