UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE COMPUTACIÓN

Especificación y semántica de tipos de datos replicados

Tesis presentada para optar al título de Doctor de la Universidad de Buenos Aires en el área Ciencias de la Computación

Lic. Christian Roldán

Director: Dr. Hernán Claudio Melgratti
Consejero de Estudios: Dr. Víctor Braberman

Lugar de Presentación: Departamento de Computación, FCEyN

Buenos Aires, Noviembre 2020

Resumen

Especificación y semántica de tipos de datos replicados

La infraestructura para almacenes de datos replicados ofrece un *throughput* (o latencia) bajo y la capacidad de seguir respondiendo ante la presencia de fallas. Sin embargo, un resultado conocido por la sigla CAP [1] (*Consistency, Availability y Partition Tolerance*) establece que no es posible garantizar alta disponibilidad, tolerancia a fallas y consistencia (fuerte) simultáneamente. Como consecuencia, una de estas tres propiedades debe ser descartada. En la actualidad, muchas bases de datos replicadas, como *Dynamo* [2] o Cassandra [3] se construyen relajando la noción de consistencia y ofreciendo versiones más débiles de consistencia, conocidas generalmente como *Consistencia Eventual* [4]. La consistencia eventual garantiza que cualquier escritura será entregada a cada réplica del sistema y que tarde o temprano todas las réplicas convergerán a un mismo estado. Sin embargo, esto significa también, que (i) las réplicas mostrarán inconsistencia de datos (o anomalías) hasta que todas las réplicas lleguen al mismo estado y que (ii) todas las réplicas deben utilizar la misma política de resolución de conflictos ante la presencia de escrituras concurrentes con el fin de garantizar la convergencia.

Las sistemas replicados adoptan diferentes estrategias para lidiar con inconsistencias temporales y resolver conflictos entre escrituras concurrentes, por ejemplo el uso de tipos de datos replicados (RDTs) [5, 6]. Por lo tanto, la solución seleccionada impacta en las propiedades aseguradas por el sistema, es decir, en el tipo de inconsistencias o anomalías que las aplicaciones permiten y observan. Dichas anomalías se caracterizan generalmente en términos de modelos de consistencia (como *monotonic-read* y consistencia causal) definidas axiomáticamente restringiendo las computaciones que pueden ocurrir en un sistema replicado [7, 8, 9].

Esta tesis contribuye al desarrollo de modelos de programación y técnicas de análisis para construir aplicaciones que lidien con nociones de consistencia débil. Presentamos una técnica de especificación para datos replicados cuya presentación será denotacional, para luego, movernos a una descripción categorial. Esta caracterización funcional es una propuesta alternativa a los enfoques operacionales actuales que se basan en historias abstractas [10, 11, 12, 13]. Brindamos, por lo tanto, una caracterización directa sobre la correctitud de las implementaciones de los RDTs basándonos en una relación de simulación entre los estados de una implementación concreta y los estados abstractos determinados por nuestra especificación [14].

Siguiendo el enfoque denotacional, asociamos a los tipos de datos replicados objetos matemáticos que representan su significado (es decir, definen dominios). En nuestra visión, estos objetos son funtores de **PIDag**$(\mathcal{L})$ en **SPath**$(\mathcal{L})$, donde **PIDag**$(\mathcal{L})$ es la categoría de los Labelled Directed Acyclic Graphs (LDAGs) y morfismos que reflejan el pasado y **SPath**$(\mathcal{L})$ es la categoría de conjuntos de caminos (u ordenes posibles sobre las operaciones) y morfismos entre conjuntos de caminos (que denota los ordenes admisibles). Damos una caracterización de tipos de datos replicados en término de las propiedades que los funtores gozan respecto de la flechas de **PIDag**$(\mathcal{L})$ (por ejemplo, en término de la preservación de pullback/pushout). Como es usual con las semánticas funtoriales, la presentación categorial resultante de la actividad precedente posibilita el desarrollo de operadores adecuados para composición de especificaciones.

Por último, estudiamos la semántica de programas corriendo sobre sistemas replicados, es decir, desarrollamos una caracterización alternativa para modelos de consistencia débil que son lo suficientemente general como para capturar algunos de los modelos de consistencia conocidos [15]. Mostramos algunas instanciaciones en nuestro framework y probaremos que las caracterizaciones obtenidas son *sound* y *complete*. Más aún, mostramos que a partir de este modelo se desprende una semántica denotacional de programas.

Abstract

Specification and semantic of replicated data types

Replicated data storages provide low throughput (or latency) and the ability to serve requests even in the presence of communication failures. However, the CAP Theorem [1] states that it is not possible to guarantee simultaneously high availability, fault tolerance and consistency (strong). Hence, one of these three properties must be discarded. Today's popular data storages, such as Dynamo [2] or Cassandra [3] are built on the top of replicated data storage systems by relaxing consistency and offer weaker notions of consistency, called *Eventual Consistency* [4]. Roughly, eventual consistency guarantees that all updates will be delivered to the all the replicas, which will eventually converge to the same state. However that also means (i) replicas temporarily exhibit some discrepancies as long as they eventually converge to the same state, and (ii) all replicas have to use the same policy to solve conflict among concurrent updates in order to achieve the same state.

Storages adopt different strategies to cope with temporary inconsistencies and to resolve conflicts among concurrent updates, such as Replicated data types (RDTs) [5, 6]. Therefore, the selected solution impacts on the properties ensured by a store, i.e., on the kind of inconsistencies or anomalies that are allowed to happened and observed by applications. Such anomalies are characterised in terms of consistency models (such as monotonic-read and causal consistency) defined axiomatically by constraining the performed computations from a data store [7, 8].

This thesis contributes to the development of programming models and analysis techniques suitable for the development of applications that rely on weak consistent, replicated stores. We develop a denotational specification technique for replicated data (such as counters, registers and sets), and then move to a categorical presentation. This functional characterisation is an abstract counterpart of the previuous operational approaches, which rely on abstract histories. Addiotionaly, we provide a direct characterisation of the correct implementations of RDTs in terms of a simulation relation between the states of a concrete implementation and the abstract ones determined by the specification [14].

Following the denotational approach, we associate the types of replicated data with mathematical objects that represent their meaning (that is, define domains). In our view, these objects are functors from $\mathbf{PIDag}(\mathcal{L})$ into $\mathbf{SPath}(\mathcal{L})$, where $\mathbf{PIDag}(\mathcal{L})$ is the category of Labelled Directed Acyclic Graphs (LDAGs) and morphisms that reflect the

past and **SPath**$(\mathcal{L})$ is the category of path-sets (or possible orders of operations) and morphisms between path-sets (denoting the admissible orders). We give a characterisation of replicated data types in terms of the properties that functors enjoy with respect to **PIDag**$(\mathcal{L})$ (e.g., in terms of the preservation of pullback / pushout). As it is standard in classical results of functorial semantics, the categorical presentation will enable us to develop suitable operators for the composition of specifications.

Finally, we study the semantics of programs running on replicated systems, i.e., we develop a parametric characterisation of weak consistent models general enough to capture well-known consistency models [15]. This allow us to prove the correctness of the programs working with different weak consistency models. We show some instantiation of this framework and prove that the obtained characterisations are *sound* and *complete*. Moreover we are able to obtain a denotational semantic of programs.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

Muchas de las aplicaciones Web que utilizamos hoy en día, son desarrolladas sobre bases de datos geo-replicadas (Cassandra[3], Dynamo [2], Spanner [16]). Una base de datos geo-replicada es un repositorio de datos donde un mismo dato se puede almacenar en distintos servidores (o réplicas) ubicados geográficamente en lugares distintos. Sin embargo, un resultado teórico conocido por su siglas CAP (*Consistency, Availability* y *Partition Tolerance*) [1], asegura que no es posible construir sistemas que simultáneamente brinden disponibilidad, tolerancia a fallas y consistencia (fuerte). En consecuencia, los sistemas geo-replicados necesitan renunciar a una de estas tres propiedades. Puesto que ningún sistema distribuido puede evitar fallas de comunicación, la elección termina siendo renunciar a la disponibilidad o a la consistencia. Elegir la consistencia por sobre la disponibilidad significa empobrecer la experiencia del usuario (también conocida como "user-expericence"), por lo tanto, la mayoría de los sistemas modernos se desarrollan ofreciendo alta disponibilidad a expensas de la sacrificar consistencia. Las consecuencias de esto son que las réplicas temporalmente mostrarán discrepancias (inconsistencias o anomalías) hasta que tarde o temprano converjan a un mismo estado [4]. Esta noción de consistencia débil es también conocida como *Consistencia Eventual*.

La consistencia eventual es un marco de trabajo en el que: (i) cada nodo o réplica mantiene una versión local del dato, (ii) los clientes realizan operaciones de lectura y escritura sobre las diferentes réplicas, (iii) cada operación realizada por un cliente es manejada por una de las réplicas, y (iv) las escrituras son propagadas asincrónicamente a aquellos réplicas que se encuentran disponibles de forma tal que estos puedan actualizar su propio estado. En particular, una operación de lectura se responde según el estado local de la réplica que maneja esa operación. Análogamente cualquier operación de escritura se ejecuta sobre el estado local de una réplica. Si bien este marco de trabajo resulta ser claro, sus implicancias no son inmediatas. Dado que los cambios se propagan asincrónicamente y diferentes réplicas pueden estar atendiendo escrituras concurrentes, cada réplica es responsable de resolver conflictos localmente. Por ejemplo, considere un sistema replicado

que almacena el tipo de datos `Registro` correspondiente a una celda de memoria, que se puede leer y a la que se le puede escribir números enteros (`k`) a través de las operaciones `rd` y `wr(k)` respectivamente. Inicialmente cada nodo del sistema contiene una copia del registro. En un escenario replicado, el valor obtenido cuando se lee un registro después de encontrarse con dos escrituras concurrentes `wr(1)` y `wr(2)` (es decir, escrituras que fueron ejecutadas en diferentes réplicas) se ve afectado por la manera en que las escrituras son propagadas sobre las diferentes réplicas. Es posible que el valor de dicha lectura sea (i) indefinido (cuando la lectura se realiza en una tercer réplica que no ha recibido ninguna operación de escritura), (ii) 1 o (iii) 2. Básicamente, el valor de retorno depende de las escrituras vistas por la operación de lectura. Elegir el valor de retorno es sencillo cuando un `rd` ve únicamente una sola operación de escritura. Sin embargo, no es tan sencillo si una lectura es realizada sobre una réplica que conoce ambas escrituras dado que todas las réplicas deben elegir de forma consistente el mismo valor sobre las valores disponibles y así tarde o temprano (*eventually*) converger a un mismo estado. Una estrategia muy común es *last-write-wins* [2, 3, 8], es decir, la última escritura debe ser elegida cuando se observan varias escrituras concurrentes. La estrategia implícitamente asume que todas las operaciones en un sistema pueden ser ordenadas, por ejemplo utilizando relojes (*timestamp*) que marcan el momento en que ocurrieron las operaciones, como una relación de orden total.

La literatura distingue dos maneras de replicación asincrónica: (i) la *basada en estados* ó (ii) la *basada en operaciones* [5]. Ejecutar una escritura en una replicación basada en estados, significa modificar el estado de la réplica que recibe la escritura. Como mencionamos anteriormente, la réplica en algún momento, envía su estado local a alguna otra réplica, que deberá combinar el estado recibido con su propio estado. De esta manera, cada escritura finalmente llega a cada réplica, ya sea directa o indirectamente. Alternativamente, en un estilo de replicación basado en operaciones, se replican las operaciones. Las ventajas del primer enfoque con respecto al segundo, es que en general el tiempo de convergencia es más rápido ya que hay escrituras que se propagan indirectamente. Sin embargo, los mensajes entre réplicas suelen ser más grandes que en el enfoque basado en operaciones.

Los sistemas geo-replicados adoptan diferentes estrategias para alcanzar consistencia eventual, y estas estrategias impactan directamente en las garantías que estos sistemas ofrecen, es decir, el tipo de inconsistencia que pueden ocurrir [5, 7, 9, 17, 18]. Algunas garantías de consistencia pueden ser por ejemplo *read-your-writes*, *monotonic-read* y *causal consistency* [7]. El modelo *read-your-writes* especifica que cuando un cliente lea una base de datos siempre verá los efectos de sus escrituras precedentes. No obstante, este modelo asegura que las operaciones de lectura que ejecuta un cliente resultan ser observaciones de la base de datos consistentes a las escrituras previamente realizadas, incluso existiendo réplicas potencialmente inconsistentes. Las anomalías son caracterizadas en términos de los modelos de consistencia. Un modelo de consistencia, básicamente, especifica un contrato entre el programador y el sistema, asegurando un determinado comportamiento por cada operación ejecutada en el sistema.

Un enfoque para trabajar con consistencia eventual consiste en dejar que las bases

de datos resuelvan automáticamente conflictos entre escrituras concurrentes cuando las réplicas intentan converger [5, 19, 20], mientras que otras bases de datos optan por dejar el problema a las aplicaciones (como por ejemplo, el caso de Casandra)[9, 21]. El diseño e implementación de tipos de datos que permiten asegurar convergencia se los conoce como tipos de datos replicados o por su acrónimo en inglés RDTs (*Replicated Data Types*). Por ejemplo, los *tipos de datos libres de conflicto y conmutativos* [5, 6] permiten la resolución automática de conflictos utilizando el *timestamp* de las operaciones. Diferentes líneas de investigación relacionadas a los RDTs proponen cómo llevar el problema de especificar e implementar tipos de datos replicados introduciendo políticas concretas para resolver conflictos, como por ejemplo, la estrategia mencionada anteriormente *last-write-wins*.

El otro enfoque se basa en la construcción de frameworks que le permita a las aplicaciones ser quienes decidan y establezcan los niveles de consistencia a alcanzar, desacoplando a las bases de datos de cualquier tipo de decisión relacionada con la consistencia eventual [9, 17, 18, 21, 22]. De esta forma, cuando existe un conflicto entre réplicas, el programador es el encargado de especificar cómo resolverlos. QUELEA [9] es un modelo de programación que utiliza lenguajes por contratos para declarar y verificar las propiedades de consistencia que los programas deben conseguir. El programador usa un lenguaje de contratos para especificar de manera axiomática el conjunto de ejecuciones que son permitidas sobre una operación. Otra línea de investigación relacionada a la construcción de frameworks es el *Global Sequence Protocol* (GSP) [17, 18, 21], un modelo operacional que describe el comportamiento de las aplicaciones corriendo sobre bases de datos replicadas, donde los updates se propagan asincrónicamente.

Estos enfoques comparten una visión general, definir formalismos que sean aplicables sobre sistemas que lidian con consistencia eventual. Sin embargo, hasta hoy no es claro cómo responder preguntas sencillas como: (i) ¿los requerimientos de mi aplicación me permiten trabajar con algún tipo de consistencia eventual? (ii) ¿puedo usar un tipo de dato replicado implementado para el sistema X en un sistema diferente Y? (iii) ¿cuál es la semántica de un sistema que combina dos formas de consistencia eventual distinta? Este tipo de preguntas, que no fueron respondidas en la literatura, han sido el puntapié inicial para este trabajo de tesis.

1.2. Contribuciones

El trabajo en esta tesis contribuye al desarrollo de modelos de programación y técnicas de análisis que se adecuen al desarrollo de aplicaciones basados en modelos de consistencia débil, en particular sistemas geo-replicados, estableciendo nuevos modelos de especificación que sean fundacionales en el área de teoría de la concurrencia. El objetivo propuesto es la definición de abstracciones que otorguen mayor flexibilidad a la hora de razonar e implementar programas sobre infraestructuras geo-replicadas. Este trabajo está divido en dos partes:

- Tipos de datos replicados (PRIMERA PARTE)

- Modelos de Consistencia (SEGUNDA PARTE)

Esta tesis está organizada en dos partes. La PRIMERA PARTE, centrada en los tipos de datos replicados, constituye un paso inicial hacia la consolidación de un enfoque denotacional de especificación para tipos de datos replicados. En particular, en el capítulo 3 presentamos una técnica que contrasta con los trabajos existentes el área, en el cual las especificaciones son exclusivamente operacionales [23]. En los enfoques tradicionales, los tipos de datos replicados se especifican en términos de dos relaciones: (i) visibilidad y (ii) arbitración. Sin embargo, en este trabajo presentamos la noción de estado para un tipo de dato replicado de manera funcional, es decir, que a partir de una función sobre la visibilidad puedan explicarse las arbitraciones admitidas.

Esta descripción denotacional, trae la oportunidad de trabajar con un enfoque functorial que permitirá utilizar, por ejemplo, operadores de composición de especificaciones para datos replicados, instrumento que aún carece el área. Por lo tanto, en el capítulo 4 proponemos una caracterización categorial para especificar RDTs [24]. Definimos las categorías para las relaciones de visibilidad y arbitración, mostrando la existencia de límites y colímites, caracterizando especificaciones para RDTs como functores entre ambas categorías que preservan estas estructuras adicionales.

Finalizamos la primera parte, con el capítulo 5 donde presentamos la noción de correctitud para algunas implementaciones de RDTs respecto a nuestras especificaciones. Para esto, nos basamos en la construcción de una relación de simulación entre estados concretos y abstractos, los últimos descriptos por las especificaciones introducidas en capítulo 3. Mostramos que la correctitud de las implementaciones son preservadas por la composición paralela, lo que nos permite asegurar que para estudiar la correctitud de una implementación alcanza con el caso que hay una única réplica.

La SEGUNDA PARTE propone una modelo operacional para sistemas replicados con consistencia débil, el cual es lo suficientemente general para poder capturar algunos de los principales modelos de consistencia ofrecidos en la literatura [7]. En particular, en el capítulo 6 desarrollamos un framework definido paramétricamente por tres predicados: *skip*, *read-consistency* y *write-consistency*. Mostramos que, dado un modelo de consistencia particular `C`, nuestro framework operacional es capaz de generar todas las trazas permitidas por `C` instanciando adecuadamente los tres predicados. Desarrollamos las definiciones para nuestros tres predicados capturando algunos de los modelos bien conocidos como las garantías de sesión (o *session guarantees*), causalidad y consistencia fuerte. Además, probamos que nuestra caracterización operacional para cada uno de estos modelos es *sound* y *complete*. Por último, en el capítulo 7, mostramos que la semántica operacional propuesta en 6 induce una semántica denotacional para programas que interactúan con sistemas replicados.

1.3. Origen de los capítulos

Algunos de los resultados en esta tesis han sido presentados y publicados en algunas conferencias. Si bien este documento presenta mejoras con respecto al material publicado,

las ideas principales de estos resultan ser las mismas. Por lo tanto, a continuación damos las referencias correspondientes.

- A CATEGORICAL ACCOUNT FOR THE SPECIFICATION OF REPLICATED DATA TYPE publicado en 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019), Schloss Dagstuhl- Leibniz-Zentrum fuer Informatik, 2019.
- ON THE SEMANTICS AND IMPLEMENTATION OF REPLICATED DATA TYPES publicado en *Science of Computer Programming* 167 (2018) 91-113.
- A DENOTATIONAL VIEW OF REPLICATED DATA TYPES publicado en: J. Jacquet, M. Massink (Eds.), COORDINATION 2017, Vol. 10319 of Lecture Notes in Computer Science, Springer, 2017, pp. 138-156.

Capítulo 2

Preliminares

2.1. Nociones básicas y Símbolos

Esta sección resume nociones básicas usadas en la tesis. En general, usamos notaciones estándar, es decir, usadas comúnmente en la literatura, libros de texto, etc. Es importante remarcar que aquellas notaciones más específicas serán presentadas a medida que sean necesarias.

Conjuntos. Informalmente, un conjunto es una colección de objetos, llamados *elementos*, que tiene la propiedad que dado un objeto cualquiera, se puede decidir si ese objeto es un elemento del conjunto o no. Para notar los conjuntos usaremos letras mayúsculas. Se dice que cada elemento $\mathtt{a}$ de un conjunto $\mathtt{A}$ *pertenece* al conjunto $\mathtt{A}$, y se nota $\mathtt{a} \in \mathtt{A}$. Dados dos conjuntos $\mathtt{A}$ y $\mathtt{B}$ decimos que $\mathtt{A}$ *está contenido* en $\mathtt{B}$ o también que $\mathtt{A}$ *es un subconjunto* de $\mathtt{B}$ si cada elemento de $\mathtt{A}$ es un elemento de $\mathtt{B}$, es decir, $\mathtt{a} \in \mathtt{A} \implies \mathtt{a} \in \mathtt{B}$. En tal caso, escribimos $\mathtt{A} \subseteq \mathtt{B}$. Escribimos $|\mathtt{A}|$ para denotar el *cardinal* de $\mathtt{A}$, es decir, la cantidad de elementos distintos que tiene $\mathtt{A}$. La notación $\mathtt{S}_\top$ será utilizada para extender el conjunto $\mathtt{S}$ con un elemento no perteneciente a $\mathtt{A}$, es decir, $\mathtt{A}_\top = \mathtt{A} \cup \{\top\}$ tal que $\top \notin \mathtt{A}$. Usaremos $\emptyset$ para denotar el conjunto vacío. Finalmente, el *producto cartesiano* de $\mathtt{A}$ con $\mathtt{B}$ se nota $\mathtt{A} \times \mathtt{B}$, es el conjunto de pares ordenados

$$\mathtt{A} \times \mathtt{B} = \{(\mathtt{a}, \mathtt{b}) : \mathtt{a} \in \mathtt{A}, \mathtt{b} \in \mathtt{B}\}$$

Relaciones. Sean $\mathtt{A}$ y $\mathtt{B}$ conjuntos. Decimos que $\mathtt{R}$ es una relación de $\mathtt{A}$ en $\mathtt{B}$ si $\mathtt{R}$ es un subconjunto de $\mathtt{A} \times \mathtt{B}$. Si $\mathtt{R}$ es una relación de $\mathtt{A}$ en $\mathtt{B}$ también escribiremos $\mathtt{aRb}$ en lugar de $(\mathtt{a}, \mathtt{b}) \in \mathtt{R}$. Una relación binaria $\mathtt{R}$ sobre el conjunto $\mathtt{A}$, es un subconjunto $\mathtt{R} \subseteq \mathtt{A} \times \mathtt{A}$. Decimos que $\mathtt{R}$ es una relación en $\mathtt{A}$ o *relación binaria* si $\mathtt{R}$ es una relación de $\mathtt{A}$ en $\mathtt{A}$, es decir, un subconjunto de $\mathtt{A} \times \mathtt{A}$. Dada dos relaciones $\mathtt{R}_1 \subseteq \mathtt{A} \times \mathtt{B}$ y $\mathtt{R}_2 \subseteq \mathtt{B} \times \mathtt{C}$, la composición de $\mathtt{R}_1$ y $\mathtt{R}_2$ es definida como $\mathtt{R}_1;\mathtt{R}_2 = \{(\mathtt{a}, \mathtt{c}) | (\mathtt{a}, \mathtt{b}) \in \mathtt{R}_1 \wedge (\mathtt{b}, \mathtt{c}) \in \mathtt{R}_2\}$. Usamos $\mathtt{R}^{-1}$ para denotar la relación inversa de $\mathtt{R}$ y $\textsc{Id}$ para representar la relación identidad. Fig. 2.1 describe algunas propiedades bien conocidas sobre relaciones binarias. Sea $\mathtt{R}$ una relación binaria sobre el conjunto $\mathtt{A}$. La clausura de $\mathtt{R}$ es la menor relación que contiene a $\mathtt{R}$

PROPIEDAD	DEFINICIÓN
simétrica	aRb implica bRa
antisimétrica	aRb y bRa implica $a = b$
reflexiva	aRa
irreflexiva	$\neg$(aRa)
transitiva	aRb y bRc implica aRc
total	$a \neq b$ implica aRb ó bRa
downward totality	aRc y bRc implica aRb ó bRa

Figura 2.1: Definición sobre relaciones binarias R

y cumple una propiedad dada. Tales propiedades pueden ser transitividad, reflexividad o simetría, en cuyo caso la clausura se llama, respectivamente, clausura transitiva, reflexiva o simétrica. Cada una de estas clausuras $C(\mathtt{R})$ verifica:

- $\mathtt{R} \subseteq C(\mathtt{R})$;
- $C(\mathtt{R})$ es transitiva (reflexiva, simétrica);
- Si $\mathtt{R}'$ es una relación transitiva (reflexiva, simétrica) tal que $\mathtt{R}' \subseteq \mathtt{R}$, entonces $C(\mathtt{R}) \subseteq \mathtt{R}'$

Usaremos $\mathtt{R}^+$ y $\mathtt{R}^*$ para notar a la clausura transitiva de R y a la clausura transitiva reflexiva de R. Usamos $\mathtt{R}|_{\mathtt{A}}$ para restringir R sobre los elementos de A, es decir, $\mathtt{R}|_{\mathtt{A}} = \mathtt{R} \cap (\mathtt{A} \times \mathtt{A})$.

Definición 2.1.1 (Downward closed)**.** *Dado un subconjunto* A *de* B*, es decir,* $\mathtt{A} \subseteq \mathtt{B}$*, decimos que* A *es* downward closed *si*

$$\forall \mathtt{a} \in \mathtt{A}.\forall \mathtt{b} \in \mathtt{B}.\mathtt{bRa} \implies \mathtt{b} \in \mathtt{A}.$$

Vamos a referirnos con $\lfloor \mathtt{a} \rfloor$ cuando queramos describir el conjunto más chico downward-closed que incluye $\mathtt{a} \in \mathtt{A}$.

Observación 1. *Una relación binaria* R *sobre un conjunto* A *es un grafo dirigido simple.*

Órdenes. Una relación $\leq$ definida en A, es un **orden parcial** (o simplemente un **orden**) si es *reflexiva, transitiva* y *antisimétrica*. Se dice que $(\mathtt{A}, \leq)$ es un conjunto parcialmente ordenado o **poset**. Diremos que un orden parcial es estricto y lo notamos $<$, si es *irreflexivo*. Se dice que los elementos $\mathtt{a}, \mathtt{b} \in \mathtt{A}$ son *comparables* si se cumple

$$\mathtt{a} \leq \mathtt{b} \text{ ó } \mathtt{b} \leq \mathtt{a}$$

Si cada par de elementos en un conjunto parcialmente ordenado son comparables, se dice que el conjunto es **totalmente ordenado**. Notar que $\leq$ denota la clausura reflexiva

de $< \cup \text{ID}$. Por último, un **orden prefijo** es una relación de orden que cumple con las propiedades: *reflexiva*, *transitiva*, *antisimétrica* y *downward totality*.

Escribiremos $- < \mathtt{a}$ (y similarmente $- \leq \mathtt{a}$, $- <^+ \mathtt{a}$ y $- <^* \mathtt{a}$) para denotar la preimagen de $\mathtt{a}$, es decir, $- < \mathtt{a} = \{\mathtt{b} \mid \mathtt{b} < \mathtt{a}\}$. Usamos el operador $max(<)$ para proyectar el elemento *maximal* de $<$. Un elemento se dice maximal si no precede a ningún elemento. En una relación de orden total, el maximal es único.

Grafos. Un grafo es un par $\mathtt{G} = (\mathtt{A}, \mathtt{R})$ donde $\mathtt{A}$ es un conjunto finito no vacío cuyos elementos se llaman vértices o nodos y $\mathtt{R}$ es una relación binaria cuyos elementos se llaman aristas. Un grafo *dirigido* es aquel en el que todas sus aristas tienen sentido o dirección, es decir, $\mathtt{R}$ no es simétrica. Sea $\mathcal{L}$ un conjunto de etiquetas. Un *Grafo acíclico, dirigido y etiquetado* (LDAG) es una tripla $\mathtt{G} = \langle \mathtt{E_G}, \prec_\mathtt{G}, \lambda_\mathtt{G} \rangle$ tal que $(\mathtt{E_G}, \prec_\mathtt{G})$ es un grafo y $\lambda_\mathtt{G} : \mathtt{E_G} \to \mathcal{L}$ es una función de etiquetado. Escribiremos $\mathbb{G}(\mathcal{L})$ y $\mathbb{P}(\mathcal{L})$ respectivamente para denotar a los conjuntos de todos los LDAGs y *caminos* sobre $\mathcal{L}$. Usaremos $\mathtt{G}$ para referir a un elemento de $\mathbb{G}(\mathcal{L})$ y $\mathtt{P}$ para un elemento de $\mathbb{P}(\mathcal{L})$. Más aún, escribiremos $<_\mathtt{P}$ en lugar de $\prec_\mathtt{P}$ para resaltar que los caminos son ordenes totales. Diremos que $\mathtt{P} = \langle \mathtt{E_P}, <_\mathtt{P}, \lambda_\mathtt{P} \rangle$ es un camino sobre $\mathtt{E}$ si $\mathtt{E_P} = \mathtt{E}$ y escribimos $\mathbb{P}(\mathtt{E}, \lambda)$ para $\{\mathtt{P} \mid \mathtt{P} \text{ es un camino sobre } \mathtt{E} \text{ y } \lambda_\mathtt{P} = \lambda\}$. En general, omitimos el subíndice $\mathtt{G}$ (o $\mathtt{P}$) cuando hagamos referencia a elementos de $\mathtt{G}$ (o $\mathtt{P}$, respectivamente) siempre y cuando no se preste a confusión. Escribiremos ε para el LDAG vacío, es decir, definido tal que $\mathtt{E}_\varepsilon = \emptyset$.

Ejemplo 2.1.1. *Considere el siguiente conjunto de etiquetas $\mathcal{L}$ que describen las operaciones para un registro de 1 bit (*1-bit register*)*

$$\mathcal{L} = \{\langle \mathtt{rd}, 0\rangle, \langle \mathtt{rd}, 1\rangle, \langle \mathtt{wr}(0), \mathtt{ok}\rangle, \langle \mathtt{wr}(1), \mathtt{ok}\rangle\}$$

Cada etiqueta es una tupla $\langle \mathtt{o}, \mathtt{v} \rangle$ donde $\mathtt{o}$ denota una operación y $\mathtt{v}$ su valor de retorno. Asumimos de aquí en adelante, que realizar operaciones de escritura, en este caso un escritura, siempre será realizado satisfactoriamente y por lo tanto su valor de respuesta será ok *Además, considere el* LDAG *sobre $\mathcal{L}$ definido como $\mathtt{G}_1 = \langle \{\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3\}, \prec, \lambda \rangle$, dónde $\prec = \{(\mathtt{e}_1, \mathtt{e}_3), (\mathtt{e}_2, \mathtt{e}_3)\}$ and λ es un conjunto tal que $\lambda(\mathtt{e}_1) = \langle \mathtt{wr}(0), \mathtt{ok}\rangle$, $\lambda(\mathtt{e}_2) = \langle \mathtt{wr}(1), \mathtt{ok}\rangle$, $\lambda(\mathtt{e}_3) = \langle \mathtt{rd}, 0\rangle$. Una representación gráfica $\mathtt{G}_1$ se muestra en Fig. 2.2a. Notar que en lugar de representar los eventos, simplemente escribimos sus etiquetas dado que la función de etiquetado es inyectiva. Fig. 2.2b representa el* LDAG *$\mathtt{G}_2$, donde $\prec_{\mathtt{G}_2}$ es vacío. Notar que ni $\mathtt{G}_1$ ni $\mathtt{G}_2$ son caminos ya que estos no son órdenes totales. $\mathtt{P}_1$ en Fig. 2.2c es un* LDAG *que además es un camino. De aquí en más, usaremos flechas sin punta cuando deseemos representar caminos, evitando además, dibujar transiciones que se obtienen por transitividad como se muestra en Fig. 2.2d. Todos* LDAGs *en Fig. 2.2 pertenecen a $\mathbb{G}(\mathcal{L})$, pero sólo $\mathtt{P}_1$ está en $\mathbb{P}(\mathcal{L})$.*

Funciones. Sean $\mathtt{A}$ y $\mathtt{B}$ conjuntos. Diremos que una relación $\mathtt{f}$ de $\mathtt{A}$ en $\mathtt{B}$ es una *función* si para cada $\mathtt{a} \in \mathtt{A}$ existe un único $\mathtt{b} \in \mathtt{B}$ tal que $(\mathtt{a}, \mathtt{b}) \in \mathtt{f}$. Escribimos $\mathtt{f} : \mathtt{A} \to \mathtt{B}$ para indicar que $\mathtt{f}$ es una función de $\mathtt{A}$ en $\mathtt{B}$, y denotaremos por $\mathtt{f}(\mathtt{a})$ al único $\mathtt{b} \in \mathtt{B}$ tal que $(\mathtt{a}, \mathtt{b}) \in \mathtt{f}$.

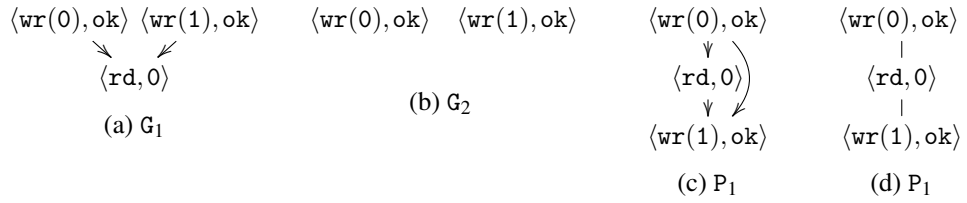


Figura 2.2: Dos LDAGs y dos caminos.

Llamamos *dominio* de $\mathtt{f}$ al conjunto $\mathtt{A}$ e *imagen* de $\mathtt{f}$ al conjunto $\mathtt{B}$. Una función parcial $\mathtt{f}$ es una función que puede estar indefinida para algunos (tal vez ninguno; tal vez todos) sus argumentos. Estas serán escritas como $\mathtt{f} : \mathtt{A} \rightharpoonup \mathtt{B}$. Para una función parcial, definimos el dominio de $\mathtt{f}$ como: $\mathit{dom}\ \mathtt{f} = \{\mathtt{a} \in \mathtt{A} \mid \exists \mathtt{b} \in \mathtt{B}.\ \mathtt{f}(\mathtt{a}) = \mathtt{b}\}$. Escribiremos $\mathtt{f}(\mathtt{a}) = \bot$ cuando $\mathtt{a} \notin \mathit{dom}\ \mathtt{f}$. Por lo tanto, el símbolo $\bot$ significa que la función $\mathtt{f}$ no está definida para el argumento $\mathtt{a}$.

2.2. Sistema de Transiciones Etiquetadas

Como es usual, describimos el comportamiento de un sistema por medio de *Sistemas de Transiciones Etiquetadas* (por sus siglas en inglés LTS). Esta sección reproduce definiciones de [25] que son bases para algunos resultados que esta tesis propone.

Definición 2.2.1 (Sistema de Transiciones)**.** *Un* sistema de transiciones etiquetadas (LTS - labelled transition system) *es una terna* $\mathbf{P} = (\mathtt{S}, \mathcal{L}, \rightarrow)$ *donde:*

- $\mathtt{S} = \{P, Q, \ldots\}$ *es un conjunto de estados;*
- $\mathcal{L} = \{\ell, \ell', \ldots\}$ *es un conjunto de acciones o etiquetas;*
- $\rightarrow \subseteq \mathtt{S} \times \mathcal{L} \times \mathtt{S}$ *es la relación de transición entre estados.*

Un sistema de transiciones etiquetadas $\mathbf{P} = (\mathtt{S}, \mathcal{L}, \rightarrow)$ permite modelar y capturar el comportamiento de un sistema de la siguiente manera:

- el conjunto $\mathtt{S}$ representa el espacio de todos los estados posibles que el sistema puede alcanzar;
- el conjunto $\mathcal{L}$ de etiquetas representa las acciones que el sistema puede ejecutar, o los eventos a los que puede responder;
- las transiciones en $\rightarrow$ representan la manera en que ocurren las computaciones del sistema, es decir, indican cómo se progresa de un estado a otro en el sistema a través de la ejecución de acciones.

Generalmente escribiremos $P \xrightarrow{\ell} Q$ en lugar de $(P,\ell,Q) \in \rightarrow$. De esta manera, $P \xrightarrow{\ell} Q$ significa que un sistema que está en un estado P evoluciona a otro estado Q realizando una acción a. Podemos ver que la relación de transición modela el comportamiento operacional de un sistema, por esto, este estilo de modelar el comportamiento es usualmente llamado semántica operacional. Un LTS es generalmente representado como un grafo dirigido con raíz y etiquetado en las aristas.

Consideraremos una acción especial $\tau \notin \mathcal{A}$ que representa el comportamiento interno de un proceso. Definimos $\mathcal{L}_\tau = \mathcal{L} \cup \{\tau\}$.

Una manera de describir a los sistemas es a través de su comportamiento. La propuesta más simple de dar semántica a un LTS es a partir de las ejecuciones que estos admiten, y esto es justamente la semántica de trazas, entendiendo una traza como una secuencia de etiquetas que representa la ejecución de un sistema. La semántica formal de LTS nos permite poder comparar el comportamientos de diferentes LTS. El instrumento estándar para comparar cuando dos sistemas son equivalentes, es la noción de simulación. Intuitivamente, diremos que un sistema **P** puede simular a otro sistema **P′** si cada acción que **P** realiza puede ser también realizada por **P′**. A continuación se define formalmente la noción de simulación débil utilizada durante el trabajo de tesis.

Definición 2.2.2 (Simulación Débil)**.** *Sea el LTS* $\mathbf{P} = (\mathtt{S}, \mathcal{L}_\tau, \rightarrow)$. *Una* simulación *es una relación binaria* $\mathcal{R} \subseteq \mathtt{S} \times \mathtt{S}$ *que satisface para todo* $\ell \in \mathcal{L}_\tau$*:*

$$\textit{Si } P\mathcal{R}Q \textit{ y } P \xrightarrow{\ell} P' \textit{ entonces } \exists Q', Q \xrightarrow{\ell} Q' \textit{ y } P'\mathcal{R}Q'$$

2.3. Categorías

Definición 2.3.1 (Categoría)**.** *Una categoría* $\mathscr{C}$ *es:*

- *una colección de objetos, llamada* $Ob(\mathscr{C})$*;*
- *una colección de flechas, morfismos, o homomorfismos, llamada* $mor(\mathscr{C})$*;*
- *Toda flecha* $\mathtt{f}$ *tiene asociado un origen o dominio* $dom(\mathtt{f})$ *y un destino o codominio* $cod(\mathtt{f})$*. Escribimos* $\mathtt{f} : A \rightarrow B$ *ó* $A\mathtt{f}B$ *para indicar que* $dom(\mathtt{f}) = A$ *y* $cod(\mathtt{f}) = B$*;*
- *Dadas* $\mathtt{f} : A \rightarrow B$ *y* $\mathtt{g} : B \rightarrow C$ *hay una flecha* $\mathtt{f};\mathtt{g} : A \rightarrow C$*, llamada la composición de* $\mathtt{f}$ *y* $\mathtt{g}$*. La composición de flechas es asociativa.*
- *Para todo objeto A existe una flecha* $id_A : A \rightarrow A$ *llamada la flecha identidad de A. Esta flecha es* neutra *para la composición.*

Notación. $Hom_{\mathscr{C}}(A,B) = \{\mathtt{f} \in mor(\mathscr{C}) : dom(\mathtt{f}) = A, cod(\mathtt{f}) = B\}$.

Sean A, B objetos de una categoría $\mathscr{C}$. Un morfismo $\mathtt{f} : B \rightarrow C$ de una categoría es un *monomorfismo* (o simplemente mono) si para todo morfismo $\mathtt{g} : A \rightarrow B$ y $\mathtt{h} : A \rightarrow B$ tales

que $\mathtt{g};\mathtt{f} = \mathtt{h};\mathtt{f}$, entonces $\mathtt{g} = \mathtt{h}$. Un *isomorfimo* es una flecha $\mathtt{f} : A \to B$ tal que existe una flecha $\mathtt{g} : B \to A$ tal que $\mathtt{f};\mathtt{g} = id_A$ y $\mathtt{g};\mathtt{f} = id_B$. Se suele decir que $\mathtt{f}$ es iso.

Dada una categoría $\mathscr{C}$ se define la categoría $\mathscr{C}^{op}$ (la categoría opuesta o dual de $\mathscr{C}$) tal que los objetos de $\mathscr{C}^{op}$ son los mismos que los de $\mathscr{C}$, y las flechas de $\mathscr{C}^{op}$ también, salvo que su orientación está invertida. Formalmente, se define de la siguiente manera

Definición 2.3.2 (Categoría Dual)**.** *Sea $\mathscr{C}$ una categoría, su categoría dual $\mathscr{C}^{op}$ es aquella que cumple*

- $Ob(\mathscr{C}^{op}) = Ob(\mathscr{C})$*;*
- $mor(\mathscr{C}^{op}) = mor(\mathscr{C})$*;*
- *para toda flecha* $\mathtt{f}$, $dom^{op}(\mathtt{f}) = cod(\mathtt{f})$ *y* $cod^{op}(\mathtt{f}) = dom(\mathtt{f})$*;*
- *para todo par de flechas* $\mathtt{f} : Hom_{\mathscr{C}}(A,B)$ *y* $\mathtt{g} : Hom_{\mathscr{C}}(B,C)$ $\mathtt{f};^{op}\mathtt{g} = \mathtt{g};\mathtt{f}$*;*
- *para todo objeto A,* $id_A^{op} = id_A$.

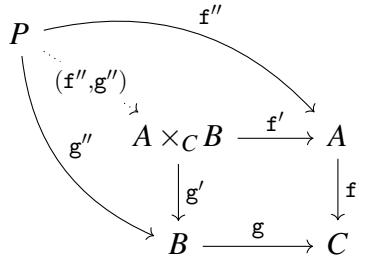
Un objecto $0 \in Ob(\mathscr{C})$ se dice *inicial* si $\forall A \in Ob(\mathscr{C}), \exists! 0 \to A$. La definición dual es la de objeto *terminal*. Un objeto $1 \in Ob(\mathscr{C})$ se dice *terminal* si $\forall A \in Ob(\mathscr{C}), \exists! A \to 1$.

Otra construcción importante en categorías es la de *pullback*.

Definición 2.3.3 (Pullback)**.** *Dados dos morfismos* $\mathtt{f} : A \to C$ *y* $\mathtt{g} : B \to C$*, entonces el* pullback *de A y B sobre C es un objeto* $A \times_C B$ *con morfismos* $\mathtt{f}' : A \times_C B \to A$ *y* $\mathtt{g}' : A \times_C B \to B$ *que forman parte del diagrama conmutativo*

$$\begin{array}{ccc} A \times_C B & \xrightarrow{\mathtt{f}'} & A \\ \downarrow{\scriptstyle \mathtt{g}'} & & \downarrow{\scriptstyle \mathtt{f}} \\ B & \xrightarrow{\mathtt{g}} & C \end{array}$$

Además se pide la siguiente propiedad universal: si P es otro objeto junto con morfismos $P \to A$ *y* $P \to A$ *que conmutan con los morfismos* $\mathtt{f}$ *y* $\mathtt{g}$*, entonces existe un único morfismo* $V \to A \times_C B$ *tal que el siguiente diagrama es conmutativo:*



Esta construcción es importante ya que se puede comprobar que una categoría tiene objeto terminal y todos los pullbacks si y sólo si tiene todos los ecualizadores y productos finitos[26]. Objeto terminal, pullbacks, ecualizadores y productos son ejemplos particulares de construcciones más generales que reciben el nombre de *límites*

El *pushout* es la construcción dual de un *pullback*, y un resultado análogo al mencionado se obtiene para el concepto general: el de *colímite*.

Para relacionar diferentes categorías, sirve la noción de funtor. Intuitivamente los funtores son los morfismos entre categorías que preservan la estructura.

Definición 2.3.4 (Funtores)**.** *Sean $\mathscr{C}$, $\mathscr{D}$ dos categorías. Un* funtor $\mathtt{F} : \mathscr{C} \to \mathscr{D}$ *asigna:*

- *a cada objeto $A \in Ob(\mathscr{C})$, un objeto $\mathtt{F}(A) \in Ob(\mathscr{D})$;*
- *a cada morfismo $\mathtt{f} : A \to B$ en $mor(\mathscr{C})$, un morfismo $\mathtt{F}(\mathtt{f}) : \mathtt{F}(A) \to \mathtt{F}(B)$ en $mor(\mathscr{D})$ tal que:*
  - *para todo $A \in Ob(\mathscr{C})$, $\mathtt{F}(id_A) = id_{\mathtt{F}(A)}$;*
  - *para todos $\mathtt{f}, \mathtt{g} \in mor(\mathscr{C})$ tales que tenga sentido la composición $\mathtt{f};\mathtt{g}$, se tiene $\mathtt{F}(\mathtt{f};\mathtt{g}) = \mathtt{F}(\mathtt{f});\mathtt{F}(\mathtt{g})$.*

Observación 2. *Para cualquier categoría concreta, existe un funtor a* **Set***, llamado* forgetful funtor*, que simplemente ignora u olvida la estructura y devuelve el conjunto subyacente.*

Definición 2.3.5 (Subcategoría)**.** *Dada una categoría $\mathscr{C}$, una subcategoría $\mathscr{D}$ cumple: las flechas de $\mathscr{D}$ son flechas de $\mathscr{C}$, con las mismas identidades, composición, dominio y codominio. Es decir, tal que:*

- *los objetos de $\mathscr{D}$ son objetos de $\mathscr{C}$, $\forall X \in Ob(\mathscr{D})$ se tiene $X \in Ob(\mathscr{C})$;*
- *las flechas de $\mathscr{D}$ son flechas de $\mathscr{C}$, $Hom_{\mathscr{D}}(X,Y) \subseteq Hom_{\mathscr{C}}(X,Y)$ para cualquier $X, Y \in Ob(\mathscr{D})$;*
- *la composición de morfismos en $\mathscr{D}$ es la misma que en $\mathscr{C}$.*

Hay una funtor evidente de $\mathscr{D}$ en $\mathscr{C}$, llamado *funtor de inclusión*.

Finalmente, la última construcción importante a mencionar es la de la categoría *coma*.

Definición 2.3.6 (Categoría Coma)**.** *Sean A, B y C categorías, y dos funtores $\mathtt{S} : A \to C$, $\mathtt{T} : B \to C$ (por las siglas en inglés* Source *y* Target *respectivamente). Definimos entonces la categoría coma, $\mathtt{S} \downarrow \mathtt{T}$, como aquella que tiene:*

- *como objetos, todas las tuplas $(\alpha, \beta, \mathtt{f})$ donde α es un objeto de A, β es un objeto de B, y $\mathtt{f} : \mathtt{S}(A) \to \mathtt{T}(B)$ es un morfismo de C;*
- *como morfismos de $(\alpha, \beta, \mathtt{f})$ a $(\alpha', \beta', \mathtt{f}')$, los pares $(\mathtt{g}, \mathtt{h})$ donde $\mathtt{g} : \alpha \to \alpha'$ y $\mathtt{h} : \beta \to \beta'$ son morfismos de A y B respectivamente, tales que el siguiente diagrama conmute:*

$$\begin{array}{ccc} \mathtt{S}(\alpha) & \xrightarrow{\mathtt{S}(\mathtt{g})} & \mathtt{S}(\alpha') \\ \downarrow{\scriptstyle \mathtt{f}'} & & \downarrow{\scriptstyle \mathtt{f}'} \\ \mathtt{T}(\beta) & \xrightarrow{\mathtt{T}(\mathtt{h})} & \mathtt{T}(\beta) \end{array}$$

- *los morfismos se componen definiendo* $(\mathtt{g},\mathtt{h});(\mathtt{g}',\mathtt{h}')$ *como* $(\mathtt{g};\mathtt{g}',\mathtt{h};\mathtt{h})$ *en aquellos casos en los que la expresión esté bien definida.*
- *el morfismo identidad en un objeto* $(\alpha,\beta,\mathtt{f})$ *será* (id_α, id_β).

2.4. Grafo de Eventos

Con el fin de razonar sobre ejecuciones de sistemas cuyo estado se encuentro replicada, usaremos *grafos de eventos*. Los grafos de eventos permitirán ilustrar las computaciones que ocurren en este tipo de sistemas, representando información de la ejecución a través de vértices, atributos y relaciones.

- *Vértices* representan los eventos o computaciones que ocurrieron en determinado momento de una ejecución.
- *Atributos* es una decoración sobre los eventos agregando información adicional, por ejemplo, el nombre de la operación, el valor de retorno, etc.
- *Relaciones* describe órdenes sobre el conjunto de eventos, capturando fenómenos distintos como ser, cuando un evento ocurre antes que otro.

Definición 2.4.1 (Grafo de Eventos)**.** *Un grafo de eventos es una tupla* $(\mathtt{E}, d_1, \dots\ d_n)$ *donde* $\mathtt{E}$ *es un conjunto de eventos,* $n \geq 1$*, y cada* d_i *son atributos o relaciones sobre* $\mathtt{E}$*.*

2.5. Tipos de datos replicados

En esta sección resumimos el enfoque clásico para especificar RDTs, propuesto en [8], adoptado sucesivamente en varios trabajos como [7, 14, 9, 27, 28].

La manera clásica para razonar sobre un dato es a través de sus operaciones, que generalmente dependen de su estado. Asumimos que en un cierto momento, el dato se encuentra en un estado particular, y su valor es conocido a través de una operación de lectura. Cuando se realiza una operación de escritura el estado del dato se modifica. En sistemas que ofrecen semántica de *una única-copia*, es decir, dan la ilusión que sólo hay un estado actual de los datos, la semántica de un tipo de dato se especifica como una tupla $\mathcal{S} = (\Sigma, \sigma_0, \delta)$ donde Σ es el conjunto de estados, cuyo valor inicial es $\sigma_0 \in \Sigma$ y δ es una función que dada una operación y un estado, devuelve una tupla con el valor de retorno y un nuevo estado. Considere el tipo de datos `Contador`, el cual ofrece operaciones de incremento y lectura (`inc`, `rd` respectivamente) sobre un entero cuyo valor inicial es `0`. Una operación de lectura retorna la cantidad de incrementos vistos por la operación de lectura. De manera análoga al ejemplo Ej. 2.1.1, se asocia `ok` al valor de retorno de cada operación de escritura. Entonces,

$$\mathcal{S}_{ctr} = (\mathbb{N}_0, 0, \delta) \text{ con } \delta(\mathtt{o},\mathtt{n}) = \begin{cases} (\mathtt{n},\mathtt{n}) & \text{si } \mathtt{o} = \mathtt{rd} \\ (\mathtt{n}+1,\mathtt{ok}) & \text{si } \mathtt{o} = \mathtt{inc} \end{cases}$$

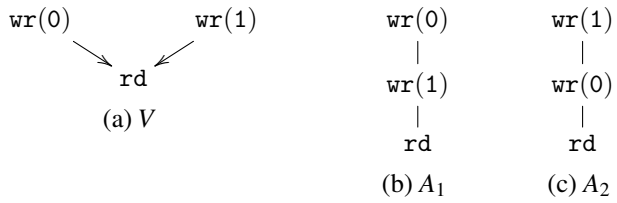


Figura 2.3: Escenario para el tipo de dato replicado `Registro`

En entornos replicados, donde los sistemas son conocidos como *eventually consistent*, las operaciones sobre un mismo dato pueden ser ejecutadas en réplicas distintas, por lo tanto, los usuarios pueden observar, temporalmente, discrepancias sobre dicho estado. En consecuencia, la especificación de un tipo de dato requiere una granularidad más fina en la descripción del dato ya que las operaciones son aplicadas sobre una vista parcial del estado del dato. En la solución propuesta por Buckhardt at al.[8], una instancia de un tipo de dato replicado se representa en términos de una estructura definida sobre un conjunto de eventos E, cada uno de los cuales representa la ejecución de una determinada operación sobre el tipo de dato. Concretamente, esta estructura es un grafo de eventos llamada *contexto*. Un contexto describe un posible estado de un RDT y consiste, básicamente, en un grafo acíclico dirigido llamado *visibilidad* y una relación de orden total, llamada *arbitración*, ambas relaciones definidas sobre un conjunto eventos La relación de visibilidad, explica las causas de cada resultado, mientras que la relación de arbitración, ordena todos los eventos de manera total. Considere la relación de visibilidad V en Fig. 2.3a y la arbitración A_1 and A_2 en Fig. 2.3b y Fig. 2.3c, respectivamente. El significado de `rd` es definido de forma tal, que $\texttt{rd}(V,A_1) = \texttt{1}$ y $\texttt{rd}(V,A_2) = \texttt{0}$. Remarcamos que todo enfoque operacional, se describe como una especificación *funcional*, es decir, una especificación donde para cada operación realizada sobre un par de relaciones (visibilidad y arbitración), existe un único valor de retorno. De esta forma, para cumplir con esto, las especificaciones operacionales deben establecer políticas concretas para resolver conflictos.

A continuación definimos formalmente un contexto de operación.

Definición 2.5.1 (Contexto)**.** *Un* contexto *de operación es un grafo de eventos finito* $\mathcal{C} = (\texttt{E}, \texttt{OP}, \texttt{VIS}, \texttt{AR})$ *donde*

- E *es un conjunto de eventos finito;*
- $\texttt{OP} : \texttt{E} \rightarrow$ *O asocia una operación a cada evento;*
- VIS *es un grafo acíclico dirigido*
- AR *es un orden total*

Considere el contexto $\mathcal{C} = (\{\texttt{e}_0, \texttt{e}_1, \texttt{e}_2\}, \texttt{OP}, \texttt{VIS}, \texttt{AR})$ donde cada evento se corresponde a una operación para un `Registro`. Definimos $\texttt{OP} = \{(\texttt{e}_0, \texttt{wr}(\texttt{1})), (\texttt{e}_1, \texttt{wr}(\texttt{2})), (\texttt{e}_2, \texttt{rd})\}$. Finalmente, $\texttt{VIS} = V$ y $\texttt{AR} = A_1$ definidas en Fig. 2.3a y Fig. 2.3b respectivamente.

Finalmente, la manera de especificar un tipo de dato replicado es a través de una función $\mathcal{F}$ que toma una operación o y un contexto de operación $\mathcal{C}$ y devuelve un valor.

Definición 2.5.2 (Tipo de Dato Replicado)**.** *Un* Tipo de Dato Replicado *(*RDT*) es una función* $\mathcal{F} : \mathcal{O} \times \mathcal{C} \to \mathcal{V}$.

Con el fin de clarificar, ilustraremos las definiciones con ejemplos sobre tipos de datos bien-conocidos.

Ejemplo 2.5.1. *Generalizamos el tipo de datos* `Contador` *a una enfoque replicado, especificando como valor de retorno de una operación de lectura, la cantidad de operaciones de incrementos que ocurrieron en el contexto:*

$$\mathcal{F}_{ctr}(\texttt{inc}, \_) = \texttt{ok}$$
$$\mathcal{F}_{ctr}(\texttt{rd}, (\texttt{E}, \texttt{OP}, \_, \_)) = |\{\texttt{e} \mid \texttt{e} \in \texttt{E} \text{ y } \texttt{OP}(\texttt{e}) = \texttt{inc}\}|$$

Es importante mencionar que este es el tipo de datos más simple ya que las operaciones de incremento conmutan. Por lo tanto, no es necesario resolver conflictos. Concretamente, $\mathcal{F}_{ctr}$ *no depende ni de* `VIS` *y ni de* `AR`.

Ejemplo 2.5.2. *Un* `Registro` *representa una celda de memoria que puede ser leída o escrita utilizando respectivamente las operaciones:* `rd` *y* `wr`. *Asumiremos que el valor inicial de un registro es el valor indefinido* $\perp$. *Consideraremos el caso generalizado de* 1-bit register*, es decir, donde la cantidad de bits es un N arbitrario. Luego* $\mathcal{F}_{lwwR}$ *da la semántica de un registro que adopta la estrategia* last-write-wins*, es decir, tiene como resultado el valor correspondiente a la última operación de escritura ejecutada según el contexto de operación* $\mathcal{C}$. *Escribimos* $\mathsf{W}(\mathcal{C})$ *para denotar el conjunto de eventos de un contexto* $\mathcal{C}$ *asociados a operaciones de escrituras, es decir,* $\mathsf{W}(\texttt{E}, \texttt{OP}, \_, \_) = \{\texttt{e} \in \texttt{E} \mid \texttt{OP}(\texttt{e}) = \texttt{wr}(\texttt{v})\}$.

$$\mathcal{F}_{lwwR}(\texttt{wr}(\texttt{v}), \_) = \texttt{ok}$$
$$\mathcal{F}_{lwwR}(\texttt{rd}, (\texttt{E}, \texttt{OP}, \_, \texttt{AR})) = \begin{cases} \perp & \textit{si } \mathsf{W}(\texttt{E}, \texttt{OP}, \_, \_) = \emptyset \\ \texttt{v} & \textit{si } \texttt{OP}(\textit{max}(\texttt{AR}|_{\mathsf{W}})) = \texttt{v} \end{cases}$$

Notar que `E` *es un conjunto finito y* `AR` *una relación de orden total, por lo tanto* $max_{\texttt{AR}}$ *devuelve un único evento maximal.*

Ejemplo 2.5.3. *Considere el tipo de dato* `Set`*, el cual provee (entre otras) las operaciones* `add`*,* `rem` *and* `lookup` *para agregar, remover y examinar los elementos dentro de un* `Set` *respectivamente. En la literatura se han propuesto distintas alternativas para resolver conflictos ante la presencia de operaciones concurrentes de agregación y eliminación de un mismo elemento (ver [6] para una discusión detallada). La siguiente especificación utiliza la política* add-win-set*, donde una adición siempre gana contra una eliminación concurrente. Para esto, decimos que un elemento está en el conjunto si ha sido agregado por alguna operación que no fue reemplazada por una operación de eliminación*

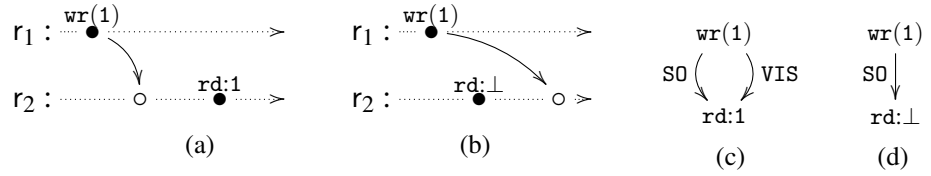


Figura 2.4: Ejemplo motivacional

posterior.

$$\mathcal{F}_{awSet}(\texttt{add}(\_),\_) = \texttt{ok}$$
$$\mathcal{F}_{awSet}(\texttt{rem}(\_),\_) = \texttt{ok}$$
$$\mathcal{F}_{awSet}(\texttt{lookup},(\texttt{E},\texttt{OP},\texttt{VIS},\_)) = \{\texttt{e} \mid \texttt{e} \in \texttt{E}\ y\ \texttt{OP}(\texttt{e}) = \texttt{add}(\texttt{k})\ y \\ (\nexists \texttt{e}' \in \texttt{E}\ \textit{tal que}\ \texttt{OP}(\texttt{e}') = \texttt{rem}(\texttt{k})\ y\ (\texttt{e},\texttt{e}') \in \texttt{VIS})\}$$

2.6. Consistencia

El nivel de consistencia que brindan los tipos de datos replicados no es lo suficientemente fuerte como para escribir programas de forma correcta, ya que, quienes escriben aplicaciones, pueden observar muchas anomalías. Dichas anomalías o inconsistencias se caracterizan en términos de modelos de consistencia que pueden variar de un sistema a otro (ver [7] para una descripción completa sobre las diferentes elecciones que hay a la hora de diseñar este tipo de aplicaciones).

Un modelo de consistencia se define usualmente de manera axiomática [15, 7, 29, 12, 9, 30, 28] como un predicado que caracteriza las ejecuciones permitidas por dicho modelo. Las ejecuciones se describen de manera abstracta como la combinación de varias relaciones, mientras que los modelos de consistencia se expresan como invariantes que satisfacen las ejecuciones. Por ejemplo, las ejecuciones en figuras 2.4a y 2.4b se definen en términos del *session order* (SO), que ordena totalmente las operaciones en una sesión; y la *visibilidad* (VIS) introducida en Def. 2.6.1, que describe la forma en que los efectos de las operaciones son visibles para otras operaciones Las ejecuciones en figuras 2.4a y 2.4b son representadas por SO y VIS como se muestran respectivamente en figuras 2.4c y 2.4d. El modelo de consistencia *read-your-write* es definido como el predicado $\texttt{SO} \subseteq \texttt{VIS}$. Por lo tanto, un sistema que asegura *read-your-write* no admite la ejecución en figura 2.4b.

A continuación se define de manera formal una *ejecución abstracta* usada para describir el comportamiento observable de un sistema.

Definición 2.6.1 (Ejecución abstracta)**.** *Una ejecución abstracta es un grafo de eventos* $\mathcal{A} = (\texttt{E},\texttt{OP},\texttt{RVAL},\texttt{SO},\texttt{VIS},\texttt{AR})$ *tal que*

- $\texttt{E}$ *es un conjunto de eventos;*
- $\texttt{OP} : \texttt{E} \to O$*;*

- $\texttt{RVAL} : \texttt{E} \to \mathcal{V}$;
- $\texttt{SO} \subseteq \texttt{E} \times \texttt{E}$ *es una unión de ordenes totales;*
- $\texttt{VIS} \subseteq \texttt{E} \times \texttt{E}$ *un grafo acíclico dirigido;*
- $\texttt{AR} \subseteq \texttt{E} \times \texttt{E}$ *un orden total.*

Una ejecución abstracta es un grafo de eventos que captura la misma información que un contexto, pero adicionalmente agrega (i) una función que asocia a cada evento el valor de respuestay (ii) el orden en una sesión o *session order*. Una sesión, conocida muchas veces como proceso o cliente, representa una secuencia de operaciones. El orden en una sesión es una relación de orden, que describe el orden en que se realizaron las computaciones de un cliente o sesión. Si proyectamos las operaciones de una sesión particular, diremos que es una relación total.

Definición 2.6.2 (Valor de Retorno Consistente)**.** *Dado un tipo de dato replicado $\mathcal{F}$, diremos que el valor de retorno es consistente siempre que se cumpla el siguiente predicado:*

$$\textsc{rval}(\mathcal{F}) : \forall \texttt{e} \in \texttt{E}. \quad \texttt{RVAL}(\texttt{e}) = \mathcal{F}(\texttt{OP}(\texttt{e}), \texttt{contexto}(\mathcal{A}, \texttt{e}))$$

donde *contexto* es definido de la siguiente manera:

Definición 2.6.3 (Contexto)**.** *Sea $\mathcal{A} = (\texttt{E}, \texttt{OP}, \texttt{RVAL}, \texttt{SO}, \texttt{VIS}, \texttt{AR})$ una ejecución abstracta y* e *un evento perteneciente a* E*. Luego,*

$$\texttt{contexto}(\mathcal{A}, \texttt{e}) = \mathcal{A}|_{\texttt{VIS}^{-1}(\texttt{e}), \texttt{OP}, \texttt{VIS}, \texttt{AR}}$$

Notar que $\texttt{contexto}(\mathcal{A}, \texttt{e})$ denota un contexto (ver Def. 2.5.1), y $\mathcal{A}|_{\texttt{VIS}^{-1}(\texttt{e}), \texttt{OP}, \texttt{VIS}, \texttt{AR}}$ es un abuso de notación que restringe la ejecución abstracta $\mathcal{A}$ con los eventos conocidos por e, es decir, $\mathcal{A}|_{\texttt{VIS}^{-1}(\texttt{e})} = (\texttt{E}|_{\texttt{VIS}^{-1}(\texttt{e})}, \texttt{OP}, \texttt{VIS}, \texttt{AR})$, y además proyecta OP, VIS y AR.

Definición 2.6.4 (Garantías de consistencia)**.** *Una garantías de consistencia* C *es un predicado sobre una ejecución abstracta $\mathcal{A}$.*

Por lo tanto, un modelo de consistencia es la combinación de garantías de consistencia que permite asegurar que bajo ciertas condiciones, el orden de las operaciones es preservado.

Los axiomas RYW, MR, MW y WFR en la Fig. 2.6 formalizan las *garantías de sesión*, asegurando que los resultados de las operaciones realizadas por una sesión (o cliente) son consistentes con la visión parcial que esta sesión tiene sobre el estado del sistema. La garantía de *causalidad* se cumple siempre que se satisfagan los axiomas CV y CA en la Fig. 2.6. El axioma CV garantiza que una operación siempre conoce a todas las operaciones que la preceden causalmente. Finalmente, garantizar consistencia fuerte es pedir que la visibilidad sea igual a la arbitración y el orden en que se ejecutan las operaciones sean consistentes con la arbitración. En el capítulo 6 se analizarán cada una de estas.

READYOURWRITES (RYW):	$\texttt{SO} \subseteq \texttt{VIS}$
MONOTONICREAD (MR):	$\texttt{VIS};\texttt{SO} \subseteq \texttt{VIS}$
MONOTONICWRITE (MW):	$\texttt{SO} \subseteq \texttt{AR}$
WRITESFOLLOWREAD (WFR):	$\texttt{VIS};\texttt{SO} \subseteq \texttt{AR}$
CAUSALVISIBILITY (CV):	$(\texttt{SO} \cup \texttt{VIS})^+ \subseteq \texttt{VIS}$
CAUSALARBITRATION (CA):	$(\texttt{SO} \cup \texttt{VIS})^+ \subseteq \texttt{AR}$
CAUSALITY (CC):	CAUSALVISIBILITY $\wedge$ CAUSALARBITRATION
SINGLEORDER:	$\texttt{VIS} = \texttt{AR}$
SEQUENTIAL:	SINGLEORDER $\wedge$ READYOURWRITES
CAUSALCONSISTENCY($\mathcal{F}$):	CAUSALITY $\wedge$ RVAL($\mathcal{F}$)
SEQUENTIALCONSISTENCY($\mathcal{F}$):	SEQUENTIAL $\wedge$ RVAL($\mathcal{F}$)

Figura 2.5: Modelos de Consistencia

Parte I

Tipos de Datos Replicados

Capítulo 3

Especificación Funcional de Tipos de Datos Replicados

A pesar de que la forma de especificar RDTs es conceptualmente clara, este enfoque presenta algunas cuestiones un tanto ambiguas. Concretamente:

(i) ¿es posible especificar un tipo de dato sin utilizar una estrategia particular para resolver conflictos, proponiendo así una noción de *subespecificación* y *refinamiento*?,

(ii) dada una visibilidad, ¿es posible arbitrar las operaciones en cualquier orden?, y finalmente

(iii) ¿cómo se construye la visibilidad?, ¿puede una visibilidad más larga ser explicada en términos de visibilidades más cortas?.

En este capítulo, nos corremos del enfoque clásico que se utiliza para interpretar el estado de un tipo de dato replicado. En nuestra perspectiva, los RDTs van a ser vistos como funciones que asignan grafos de visibilidad (ahora llamados *configuraciones*) en conjunto de arbitraciones admisibles, es decir, todas las posibles ejecuciones que podrían explicar una configuración. En particular, una configuración asociada al conjunto vacío de arbitraciones representa una configuración inalcanzable, es decir, una configuración que no puede ser explicada en términos de ninguna arbitración. La propuesta de una visión más abstracta sobre los RDTs nos permite remarcar algunos supuestos implícitos compartidos por la mayoría de los enfoques operacionales. En particular, caracterizamos los enfoques operacionales, presentados en [5, 8], como una clase de especificaciones que satisfacen las siguientes tres propiedades: además del requerimiento evidente de ser *funcional* (es decir, determinística y total), deben ser *coherentes* (es decir, estados más grandes se explican como la composición de los más pequeños) y *saturadas* (por ejemplo, una operación no observada se puede ordenar en cualquier posición, incluso antes de los eventos que ve). Mostramos que la caracterización funcional nos permite introducir de forma elegante los conceptos de *subespecificación* y *refinamiento*, nociones estándar en la especificación de tipos de datos. Además, este cambio de visión nos permite desarrollar una formulación en términos de teoría de categorías, el cual será presentado en el Capítulo 4.

3.1. Operaciones sobre LDAG

En esta sección mencionaremos algunas operaciones sobre grafos acíclicos dirigidos y etiquetados, las cuales resultan ser imprescindibles para dar una caracterización de los tipos de datos replicados. Contaremos con un conjunto enumerable $\mathtt{E}$ de eventos $\mathtt{e}, \mathtt{e}', \ldots, \mathtt{e}_1, \ldots$ y un conjunto enumerable $\mathcal{L}$ de etiquetas $\ell, \ell', \ldots, \ell_1, \ldots$

Definición 3.1.1 (Restricción y Extensión)**.** *Sea* $\mathtt{G} = \langle \mathtt{E}, \prec, \lambda \rangle$ *y* $\mathtt{E}' \subseteq \mathtt{E}$*. Definimos*

- $\mathtt{G}|_{\mathtt{E}'} = \langle \mathtt{E}', \prec|_{\mathtt{E}'}, \lambda|_{\mathtt{E}'} \rangle$ *como la* restricción de $\mathtt{G}$ sobre $\mathtt{E}'$*;*
- $\mathtt{G}^{\ell}_{\mathtt{E}'} = \langle \mathtt{E}_\top, \prec \cup (\mathtt{E}' \times \{\top\}), \lambda[\top \mapsto \ell] \rangle$ *como la* extensión de $\mathtt{G}$ sobre $\mathtt{E}'$ con ℓ.

Intuitivamente, $\mathtt{G}^{\ell}_{\mathtt{E}'}$ es el resultado de agregar a la relación de visibilidad $\mathtt{G}$, un nuevo evento que ve a todos los eventos en $\mathtt{E}'$. Omitiremos el subíndice $\mathtt{E}'$ en $\mathtt{G}^{\ell}_{\mathtt{E}'}$ cuando $\mathtt{E}' = \mathtt{E}$. Haciendo abuso de notación utilizaremos la operación de restricción sobre conjuntos de LDAGs. Sea $\mathcal{X} \subseteq \mathbb{G}(\mathcal{L})$, entonces, $\mathcal{X}|_{\mathtt{E}} = \{\mathtt{G}|_{\mathtt{E}} \mid \mathtt{G} \in \mathcal{X}\}$.

Ejemplo 3.1.1. *Considere los* LDAG*s* $\mathtt{G}_1$ *y* $\mathtt{G}_2$ *del Ej. 2.1.1, representados en Fig. 2.2a y Fig. 2.2b. Notar que,* $\mathtt{G}_2 = \mathtt{G}_1|_{-\prec \mathtt{e}_3}$ *y* $\mathtt{G}_1$ *es isomórfico a* $\mathtt{G}_2{}^{\langle \mathtt{rd},0 \rangle}$*. De hecho,* $\mathtt{G}_1$ *puede ser obtenido de* $\mathtt{G}_2$ *agregando un nuevo nodo etiquetado con* $\langle \mathtt{rd}, 0 \rangle$*, de esa manera el nuevo nodo estará relacionado con cada nodo en* $\mathtt{G}_2$ *a través de* $\prec$*.*

El próximo paso es caracterizar la compatibilidad entre LDAGs que significa bajo qué condiciones se pueden combinar o unir LDAGs y caminos. Daremos una definición para caracterizar cuando dos o más LDAGs son *compatibles* y un operador que permite combinarlos. Sin embargo primero necesitamos considerar un tipo de morfismos para LDAG.

Definición 3.1.2 (Morfismos Past-reflecting)**.** *Un morfismo* LDAG $\mathtt{f}$ *de* $\mathtt{G}_1$ *a* $\mathtt{G}_2$*, escrito como* $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$*, es un mapeo* $\mathtt{f} : \mathtt{E}_{\mathtt{G}_1} \to \mathtt{E}_{\mathtt{G}_2}$ *tal que*

$$\lambda_{\mathtt{G}_1} = \mathtt{f}; \lambda_{\mathtt{G}_2} \text{ y } \mathtt{e} \prec_{\mathtt{G}_1} \mathtt{e}' \text{ implica } \mathtt{f}(\mathtt{e}) \prec_{\mathtt{G}'} \mathtt{f}(\mathtt{e}')$$

Un morfismo injectivo LDAG $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$ *es* past-reflecting *si se cumple*

- $\mathtt{f}(\mathtt{e}) \prec_{\mathtt{G}_2} \mathtt{f}(\mathtt{e}')$ *implica* $\mathtt{e} \prec_{\mathtt{G}_1} \mathtt{e}'$*;*
- $\bigcup_{\mathtt{e} \in \mathtt{E}_{\mathtt{G}_1}} \mathtt{f}(\mathtt{e})$ *es* downward closed*.*

Definición 3.1.3 (Compatibilidad)**.** *Dos* LDAG*s* $\mathtt{G}_1$ *y* $\mathtt{G}_2$ *son* compatibles *si*

- $\mathtt{e} \in \mathtt{E}_{\mathtt{G}_1} \cap \mathtt{E}_{\mathtt{G}_2}$ *implica* $\lambda_1(\mathtt{e}) = \lambda_2(\mathtt{e})$*;*
- *los morfismos inyectivos* $\mathtt{G}_\mathtt{i} \to \mathtt{G}_1 \cup \mathtt{G}_2$ *para* $i \in \{1,2\}$ *son past-reflecting.*

Definición 3.1.4 (Unión Compatible)**.** *Sean* $\mathtt{G}_1$ *y* $\mathtt{G}_2$ *dos* LDAG*s compatibles. Entonces, definimos el* operador de unión compatible*, notado como* $\sqcup$*, de la siguiente manera*

$$\mathtt{G}_1 \sqcup \mathtt{G}_2 = (\mathtt{E}_{\mathtt{G}_1} \cup \mathtt{E}_{\mathtt{G}_2}, \prec_{\mathtt{G}_1} \cup \prec_{\mathtt{G}_2}, \lambda) \text{ tal que } \lambda = \begin{cases} \lambda_{\mathtt{G}_1}(\mathtt{e}) & \text{si } \mathtt{e} \in \mathtt{E}_{\mathtt{G}_1} \\ \lambda_{\mathtt{G}_2}(\mathtt{e}) & \text{si } \mathtt{e} \in \mathtt{E}_{\mathtt{G}_2} \end{cases}$$

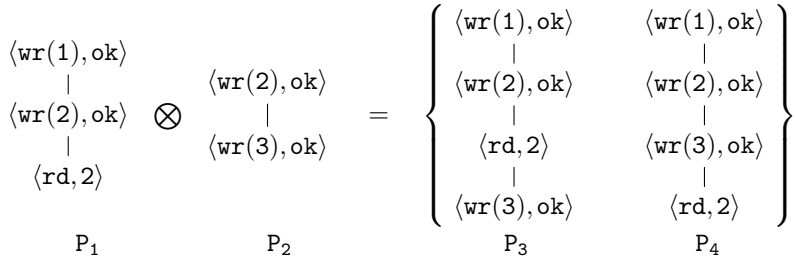


Figura 3.1: Producto entre dos caminos.

Notar que la operación $\sqcup$ sobre LDAGs es idempotente, asociativa, y commutativa.

El siguiente operador nos permite combinar varios caminos. Esta definición jugará un rol central en nuestra caracterización tanto para la descripción categorial como para describir la correctitud de los tipos de datos replicados.

Definición 3.1.5 (Producto)**.** *Sea $\mathcal{X} = \{\langle \mathtt{E}_i, <_i, \lambda_i \rangle\}_i$ un conjunto de caminos. El producto de $\mathcal{X}$ es*

$$\bigotimes \mathcal{X} = \{ Q \mid Q \text{ es un camino sobre } \bigcup_i \mathtt{E}_i \text{ y } Q|_{\mathtt{E}_i} \in \mathcal{X} \}$$

Intuitivamente, el producto de caminos es análogo a la composición paralela de máquinas de estado, donde los elementos en común deben respetar el mismo orden y el resto pueden intercalarse libremente. Es inmediato mostrar que $\otimes$ es asociativo y commutativo. En consecuencia, usaremos libremente $\otimes$ sobre conjuntos de conjuntos de caminos. Por conveniencia, utilizamos $\otimes$ como un operador infijo.

Notar que si $\mathtt{G}_1$ y $\mathtt{G}_2$ son compatibles, entonces el producto $\mathtt{P}_1 \otimes \mathtt{P}_2$ es bien-etiquetado para cada par $\mathtt{P}_1 \in \mathbb{S}(\mathtt{G}_1)$ y $\mathtt{P}_2 \in \mathbb{S}(\mathtt{G}_2)$ (es decir, eventos que pertenecen a ambos $\mathtt{E}_{\mathtt{G}_1}$ y $\mathtt{E}_{\mathtt{G}_2}$ tienen la misma etiqueta) porque cada camino es construido del conjunto de eventos (y las etiquetas correspondientes) al LDAG asociado.

Ejemplo 3.1.2. *Considere los caminos $\mathtt{P}_1$ y $\mathtt{P}_2$ in Fig. 3.1, tales que comparten el evento etiquetado $\langle \mathtt{wr}(2), \mathtt{ok} \rangle$. Su producto tiene dos posibles caminos $\mathtt{P}_3$ y $\mathtt{P}_4$, cada uno contiene el elemento común entre $\mathtt{P}_1$ y $\mathtt{P}_2$ y preservan el orden relativo de los elementos de los caminos originales. Notar que el producto es vacío cuando los caminos son ordenes incompatibles. Por ejemplo, $\mathtt{P}_3$ y $\mathtt{P}_4$ tienen el mismo conjunto de elementos pero los órdenes entre los elementos son incompatibles, por lo tanto $\mathtt{P}_3 \otimes \mathtt{P}_4 = \emptyset$.*

3.2. Especificaciones Funcionales

En esta sección introducimos una noción denotacional para especificar tipos de datos replicados e ilustraremos su aplicación definiendo tipos de datos bien conocidos.

Definición 3.2.1 (Especificación Funcional)**.** *Una* especificación *funcional* $\mathbb{S}$ *es una función* $\mathbb{S} : \mathbb{G}(\mathcal{L}) \to 2^{\mathbb{P}(\mathcal{L})}$ *tal que* $\mathbb{S}(\varepsilon) = \{\varepsilon\}$ *y* $\forall \mathtt{G}.\ \mathbb{S}(\mathtt{G}) \in 2^{\mathbb{P}(\mathtt{E_G}, \lambda_\mathtt{G})}$.

Una especificación funcional $\mathbb{S}$ asigna un LDAG (es decir, una relación de visibilidad ahora conocida como *configuración*[1]) a un conjunto de caminos (es decir, sus arbitraciones admisibles). La intuición detrás de la definición es que un $\mathtt{G}$ puede ser explicado a partir de las distintas ejecuciones de $\mathbb{S}(\mathtt{G})$. Notar que $\mathtt{P} \in \mathbb{S}(\mathtt{G})$ es un camino sobre $\mathtt{E_G}$, y en consecuencia es un orden total sobre los eventos de $\mathtt{G}$. Sin embargo, no requerimos que $\mathtt{P}$ sea un orden topológico de $\mathtt{G}$, es decir, $\prec_\mathtt{G} \subseteq <_\mathtt{P}$ puede no satisfacerse. Aunque algunas especificaciones en la literatura consideran solo arbitraciones/caminos que incluyen a la visibilidad [12, 13, 18, 27, 31], nuestra definición se ajusta a presentaciones como en [8, 15, 32, 33], donde las arbitraciones no preservan la visibilidad. Dado que nuestro enfoque es independiente de esta elección, adoptaremos la presentación más general. Queremos hacer notar que es posible definir $\mathbb{S}$ de manera tal que $\mathbb{S}(\mathtt{G}) = \emptyset$ para algún $\mathtt{G}$, esto significa que $\mathbb{S}$ prohibe configuraciones $\mathtt{G}$ (más detalle en Ej. 3.2.1 descripto abajo). Por conveniencias técnicas, vamos a imponer que $\mathbb{S}(\varepsilon) = \{\varepsilon\}$ y no permitiremos $\mathbb{S}(\varepsilon) = \emptyset$: una especificación no puede prohibir a una configuración vacía, la cual denota el estado inicial de un tipo de dato.

A continuación, mostraremos especificaciones para tipos de datos bien conocidos.

Ejemplo 3.2.1 (Contador)**.** *Considere la especificación del tipo de dato replicado* `Contador` *presentado en Ej. 2.5.1. En nuestro caso consideramos el conjunto de etiquetas* $\mathcal{L} = \{\langle \mathtt{inc}, \mathtt{ok}\rangle\} \cup (\{\mathtt{rd}\} \times \mathbb{N})$. *Entonces, la especificación de un* `Contador` *está dada por* $\mathbb{S}_{Ctr}$ *definida tal que*

$$\mathtt{P} \in \mathbb{S}_{Ctr}(\mathtt{G}) \Leftrightarrow \forall \mathtt{e} \in \mathtt{E_G}.\forall \mathtt{k}.\lambda(\mathtt{e}) = \langle \mathtt{rd}, \mathtt{k}\rangle \textit{ implica } \mathtt{k} = |\{\mathtt{e}' \mid \mathtt{e}' \prec_\mathtt{G} \mathtt{e} \textit{ y } \lambda(\mathtt{e}') = \langle \mathtt{inc}, \mathtt{ok}\rangle\}|$$

Una configuración $\mathtt{G}$ *tiene arbitraciones admisibles (es decir,* $\mathbb{S}_{Ctr}(\mathtt{G}) \neq \emptyset$*) solo cuando cada evento* $\mathtt{e}$ *en* $\mathtt{G}$ *es asociado a una operación de lectura que tiene como valor de retorno un entero* $\mathtt{k}$ *que se corresponde con el número de incrementos que preceden a* $\mathtt{e}$ *en* $\mathtt{G}$*. Ilustraremos dos casos para la definición de* $\mathbb{S}_{Ctr}$ *en Fig. 3.2. Mientras la configuración en Fig. 3.2a tiene arbitraciones admisibles, la figura Fig. 3.2b no, ya que el único evento etiquetado por* `rd` *devuelve* `0` *cuando este es precedido por un único incremento. En otras palabras, no hay ejecución posible que genere la configuración representada en Fig. 3.2b. Notar que* $\mathbb{S}_{Ctr}$ *no impone restricciones en el orden* $<_\mathtt{P}$*. De hecho, un camino* $\mathtt{P} \in \mathbb{S}_{Ctr}(\mathtt{G})$ *no necesariamente es un orden topológico de* $\mathtt{G}$*, por ejemplo, considere el camino de más a la derecha en el conjunto de caminos de la Fig. 3.2a.*

Ejemplo 3.2.2 (Registro *Last-Write-Wins*)**.** *Presentamos la especificación del tipo de dato replicado* `Registro`*, introducido en Ej. 2.5.2. Consideramos el conjunto de etiquetas* $\mathcal{L} = \{\langle \mathtt{wr}(\mathtt{k}), \mathtt{ok}\rangle \mid \mathtt{k} \in \mathbb{N}\} \cup (\{\mathtt{rd}\} \times \mathbb{N} \cup \{\bot\})$ *como el conjunto de etiquetas de un*

[1] Dado que la visibilidad describe la configuración de un sistema usaremos generalmente el término *configuración* para referirnos a la visibilidad.

$$\mathbb{S}_{Ctr}\left(\begin{array}{c}\langle \texttt{inc},\texttt{ok}\rangle \\ \downarrow \\ \langle \texttt{rd},1\rangle\end{array}\right) = \left\{\begin{array}{cc}\langle \texttt{inc},\texttt{ok}\rangle & \langle \texttt{rd},1\rangle \\ | & | \\ \langle \texttt{rd},1\rangle & \langle \texttt{inc},\texttt{ok}\rangle\end{array}\right\} \qquad \mathbb{S}_{Ctr}\left(\begin{array}{c}\langle \texttt{inc},\texttt{ok}\rangle \\ \downarrow \\ \langle \texttt{rd},0\rangle\end{array}\right) = \emptyset$$

(a) (b)

Figura 3.2: Especificación de un `Contador`.

`Registro` *cuya política de resolución de conflictos es* last-write-wins*. Luego, el tipo de datos registro es especificado por la función* $\mathbb{S}_{lwwR}$ *definida de manera tal que*

$$\texttt{P} \in \mathbb{S}_{lwwR}(\texttt{G}) \Leftrightarrow$$

$$\forall \texttt{e} \in \texttt{E}_{\texttt{G}}.\begin{cases} \lambda(\texttt{e}) = \langle \texttt{rd},\bot\rangle \textit{ implica } \forall \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\forall \texttt{k}.\lambda(\texttt{e}') \neq \langle \texttt{wr}(\texttt{k}),\texttt{ok}\rangle & (3.1) \\ \forall \texttt{k}.\lambda(\texttt{e}) = \langle \texttt{rd},\texttt{k}\rangle \textit{ implica } \exists \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\lambda(\texttt{e}') = \langle \texttt{wr}(\texttt{k}),\texttt{ok}\rangle \ \ y & \\ \qquad \forall \texttt{e}'' \prec_{\texttt{G}} \texttt{e}.\ \texttt{e}' <_{\texttt{P}} \texttt{e}'' \textit{ implica } \forall \texttt{k}'.\lambda(\texttt{e}'') \neq \langle \texttt{wr}(\texttt{k}'),\texttt{ok}\rangle & (3.2) \end{cases}$$

Un LDAG `G` *tiene arbitraciones admisibles solo cuando cada evento asociado a una operación de lectura retorna el último valor escrito. La condición 3.1 dice que una operación de lectura devuelve el valor indefinido* $\bot$ *cuando este no ve ninguna escritura. La condición 3.2 establece que una lectura asociada a el evento* `e` *retorna un valor entero* `k` *cuando este evento conozca una operación* $\texttt{e}'$ *que haya escrito el valor* `k`*. En ese caso, cualquier arbitración posible* `P` *debe ordenar* $\texttt{e}'$ *como el elemento maximal (según* $<_{\texttt{P}}$*) entre todas las operaciones de escritura vistas por* `e`*.*

Ejemplo 3.2.3 (Registro Genérico)**.** *Un* `Registro Genérico` *es similar al tipo de datos* `Registro` *pero su especificación no establece una estrategia particular para resolver conflictos. Especificamos este tipo de datos a través de la función* $\mathbb{S}_{gR}$ *definida de manera tal que*

$$\texttt{P} \in \mathbb{S}_{gR}(\texttt{G}) \Leftrightarrow$$

$$\forall \texttt{e} \in \texttt{E}_{\texttt{G}}.\begin{cases} \lambda(\texttt{e}) = \langle \texttt{rd},\bot\rangle \textit{ implica } \forall \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\forall \texttt{k}.\lambda(\texttt{e}') \neq \langle \texttt{wr}(\texttt{k}),\texttt{ok}\rangle & (3.3) \\ \forall \texttt{k}.\lambda(\texttt{e}) = \langle \texttt{rd},\texttt{k}\rangle \textit{ implica } \exists \texttt{e}' \prec_{\texttt{G}} \texttt{e}.\lambda(\texttt{e}') = \langle \texttt{wr}(\texttt{k}),\texttt{ok}\rangle \ \ y & \\ \forall \texttt{e}''.\forall \texttt{k}''.\lambda(\texttt{e}'') = \langle \texttt{rd},\texttt{k}''\rangle \ y - \prec_{\texttt{G}} \texttt{e} = - \prec_{\texttt{G}} \texttt{e}'' \textit{ implica } \texttt{k} = \texttt{k}'' & (3.4) \end{cases}$$

La especificación es análoga a $\mathbb{S}_{lwwR}$ *en Ej. 3.2.2 pero difiere en la segunda condición. El valor de retorno asociado a un evento* `e`*, correspondiente a una operación de lectura, es alguno de los valores escritos por las operaciones de escritura vistas por* `e`*. Notar que en este caso, la especificación no determina cuál es el valor de retorno. Vamos a requerir entonces que todas las operaciones de lectura con las mismas causas (es decir,* $- \prec_{\texttt{G}} \texttt{e} = - \prec_{\texttt{G}} \texttt{e}''$*) tengan el mismo resultado. Dado que está condición debe ser satisfecha por cualquier configuración admisible* `G`*, esta asegura convergencia.*

Ejemplo 3.2.4 (Add-Win-Set)**.** *La especificación del tipo de dato replicado* `Set`*, presentado en Ej. 2.5.3, utiliza el siguiente conjunto de etiquetas*

$$\mathcal{L} = \{\langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle \mid \mathtt{k} \in \mathbb{N}\} \cup \{\langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\rangle \mid \mathtt{k} \in \mathbb{N}\} \cup (\{\mathtt{lookup}\} \times 2^{\mathbb{N}} \cup \{\bot\})$$

Entonces, la especificación de un `Set` *cuya política de resolución de conflictos es* add-win-set*. está dada por la función* $\mathbb{S}_{awSet}$ *definida tal que*

$$\begin{array}{c} \mathtt{P} \in \mathbb{S}_{awSet}(\mathtt{G}) \\ \Leftrightarrow \\ \forall \mathtt{e} \in \mathtt{E_G}.\forall \mathtt{S} \in 2^{\mathbb{N}}. \quad \lambda(\mathtt{e}) = \langle \mathtt{lookup}, \mathtt{S}\rangle \textit{ implica} \\ \mathtt{S} = \{\mathtt{k} \mid \exists \mathtt{e}' \in \mathtt{E_G}.\mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \textit{ y } \lambda(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle \textit{ y} \\ \forall \mathtt{e}''.\mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}'' \prec_{\mathtt{G}} \mathtt{e} \textit{ implica } \lambda(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\rangle\} \end{array}$$

Un grafo de visibilidad `G` *tiene arbitraciones admisibles solo cuando cada evento* `e` *en* `G` *asociado a una operación de lectura tiene como valor de retorno un conjunto* `S` *donde las adiciones ganan contra las remociones concurrentes del mismo elemento. La condición asegura que los elementos de* `S` *son números naturales tal que cada natural fue agregado a través de una operación de adición y no fue alcanzado por una operación de una remoción sobre el mismo elemento.*

Observación 3. *El requerimiento de convergencia es importante ya que estamos identificando las condiciones mínimas para asegurar la correctitud de una especificación sin hacer ninguna suposición sobre la arbitración o el camino elegido. De hecho, convergencia puede ser probada para especificaciones a las que llamaremos* determinísticas *(es decir, especificaciones para las cuales el valor de retorno de cada operación es únicamente determinado por la relación de visibilidad y arbitración, este será formalizado en Sección 3.4.2), cuyas instancias son por ejemplo descriptas en Ej. 3.2.1 y Ej. 3.2.2. En consecuencia, nuestra propuesta propone un enfoque alternativo a propuestas como [8, 15, 9], donde la convergencia es asegurada automáticamente dado que las especificaciones son determínisticas (como mostramos formalmente en Sección 3.4.3).*

3.3. Refinamiento

El refinamiento es un enfoque estándar en la especificación de tipos de datos[34, 35], ya que permite una organización jerárquica que va de descripciones abstractas a implementaciones concretas. El principal beneficio del refinamiento se encuentra en el hecho de que las aplicaciones pueden ser desarrolladas y pensadas en términos de tipos de datos abstractos, los cuales ocultan detalles de implementación y dejan algunos grados de libertad para las implementaciones. Considere la especificación $\mathbb{S}_{gR}$ de un `Registro genérico` introducido en Ej. 3.2.3, la cual requiere alguna política para resolución de conflictos que asegure convergencia. Por el contrario, la especificación $\mathbb{S}_{lwwR}$ en Ej. 3.2.2 establece explícitamente que las escrituras concurrentes deben resolverse adoptando la

política *last-write-wins*. Dado que esta última asegura la convergencia, nos gustaría pensar a $\mathbb{S}_{lwwR}$ como un refinamiento de $\mathbb{S}_{gR}$. Caracterizamos el refinamiento en nuestro framework de la siguiente manera.

Definición 3.3.1 (Refinamiento)**.** *Sean* $\mathbb{S}_1, \mathbb{S}_2$ *dos especificaciones. Diremos que* $\mathbb{S}_1$ refina a $\mathbb{S}_2$*, escrito como* $\mathbb{S}_1 \sqsubseteq \mathbb{S}_2$*, si* $\forall \mathtt{G}.\ \mathbb{S}_1(\mathtt{G}) \subseteq \mathbb{S}_2(\mathtt{G})$.

Ejemplo 3.3.1. *Es fácil probar que* $\mathtt{P} \in \mathbb{S}_{lwwR}(\mathtt{G})$ *implica* $\mathtt{P} \in \mathbb{S}_{gR}(\mathtt{G})$ *para todo* $\mathtt{G}$*. Por lo tanto,* $\mathbb{S}_{lwwR}$ *es un refinamiento de* $\mathbb{S}_{gR}$*.*

Ejemplo 3.3.2. *A continuación presentamos una especificación del tipo de datos* `Set` *que no fija una política de resolución de conflictos. Dicha especificación está dada por la función* $\mathbb{S}_{Set}$ *definida de manera tal que*

$$\mathtt{P} \in \mathbb{S}_{Set}(\mathtt{G}) \ \ sii \ \ \forall \mathtt{e} \in \mathtt{E_G}.\lambda(\mathtt{e}) = \langle \mathtt{lookup}, \mathtt{S} \rangle \ implica \\ B_\mathtt{e} \subseteq \mathtt{S} \subseteq A_\mathtt{e} \ y \ \mathtt{Conv_{e,S}}$$

donde

$$A_\mathtt{e} = \{\mathtt{k} \mid \exists \mathtt{e}' \in \mathtt{E_G}.\mathtt{e}' \prec_\mathtt{G} \mathtt{e} \ and \ \lambda(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok} \rangle\}$$
$$B_\mathtt{e} = A_\mathtt{e} \setminus \{\mathtt{k} \mid \exists \mathtt{e}' \in \mathtt{E_G}.\mathtt{e}' \prec_\mathtt{G} \mathtt{e} \ and \ \lambda(\mathtt{e}') = \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok} \rangle\}$$
$$\mathtt{Conv_{e,S}} = \forall \mathtt{e}' \in \mathtt{E_G}.\forall \mathtt{S}' \in 2^{\mathbb{N}}.\lambda(\mathtt{e}') = \langle \mathtt{lookup}, \mathtt{S}' \rangle \ y \ - \prec_\mathtt{G} \mathtt{e} = - \prec_\mathtt{G} \mathtt{e}' \\ implica \ \mathtt{S} = \mathtt{S}'$$

El conjunto $A_\mathtt{e}$ *contiene los elementos agregados al (y posiblemente sacados del) conjunto visto por* `e` *mientras que* $B_\mathtt{e}$ *contiene aquellos elementos agregados que* `e` *no ve quitados. Por lo tanto, la condición* $B_\mathtt{e} \subseteq \mathtt{S} \subseteq A_\mathtt{e}$ *establece que* `lookup` *devuelve un conjunto con todos los elementos agregados que nunca fueron quitados (es decir, en* $B_\mathtt{e}$*). Sin embargo, el valor de retorno* `S` *puede contener elementos que han sido agregados y quitados (la elección se deja sin especificar). Análogamente a la especificación de* $\mathbb{S}_{gR}$ *en Ej. 3.2.3,* $\mathtt{Conv_{e,S}}$ *asegura convergencia.*

Otra política para resolver conflictos en la implementación del RDT `Set`*, conocida como* `2P-Set`*, establece que las adiciones de elementos que fueron previamente sacados no tienen efecto. Un refinamiento de* `Set` *que implementa esta política de resolución de conflictos puede especificarse a través de una función* $\mathbb{S}_{2P\text{-}Set}$ *definida de manera tal que*

$$\mathtt{P} \in \mathbb{S}_{2P\text{-}Set}(\mathtt{G}) \ \ sii \ \ \forall \mathtt{e} \in \mathtt{E_G}.\forall \mathtt{S} \in 2^{\mathbb{N}}.\lambda(\mathtt{e}) = \langle \mathtt{lookup}, \mathtt{S} \rangle \ implica \ \mathtt{S} = \mathtt{B_e}$$

Es inmediato notar que $\mathbb{S}_{2P\text{-}Set}$ *es un refinamiento de* $\mathbb{S}_{Set}$*. Análogamente, tomando la función* $\mathbb{S}_{Add\text{-}Win\text{-}Set}$ *en Ej. 3.2.4, podemos ver que* $\mathbb{S}_{Add\text{-}Win\text{-}Set}$ *es un refinamiento también de* $\mathbb{S}_{Set}$*.Otras políticas pueden ser especificadas análogamente.*

3.4. Correspondencia con el enfoque clásico

En esta sección mostraremos que nuestra noción de especificación generaliza las nociones tradicionales de RDTs § 2.5. Comenzaremos parafraseando la definición de RDT

(dada en Capítulo 2, Def. 2.5.2) con las nociones introducidas en este capítulo con el fin de facilitar la comparación entre las especificaciones operacionales y las especificaciones funcionales. Luego caracterizaremos distintos tipos de especificaciones funcionales para luego mostrar la correspondencia que hay entre el enfoque clásico y nuestras especificaciones funcionales.

3.4.1. Revisitación del enfoque clásico

Como hemos mencionado, el significado de cada operación de un RDT se especifica en términos de un contexto, escrito como $\mathcal{C}$, el cual puede verse como un par $\langle \mathtt{G}, \mathtt{P} \rangle$ tal que $\mathtt{P} \in \mathbb{P}(\mathtt{E_G}, \lambda_\mathtt{G})$. Escribimos $\mathbb{C}(\mathcal{L})$ para los contextos sobre $\mathcal{L}$. Entonces, la descripción operacional de RDTs en § 2.5 puede ser formulada de la siguiente manera.

Definición 3.4.1 (Tipo de Dato Replicado)**.** *Un* Tipo de Dato Replicado *(*RDT*) es una función* $\mathcal{F} : O \times \mathbb{C}(O) \to \mathcal{V}$.

Esto significa que para cualquier configuración $\mathtt{G}$ y arbitración $\mathtt{P}$, la especificación $\mathcal{F}$ indica que el resultado de ejecutar la operación $\mathtt{op}$ sobre $\mathtt{G}$ y $\mathtt{P}$, es $\mathcal{F}(\mathtt{o}, \langle \mathtt{G}, \mathtt{P} \rangle)$.

Ejemplo 3.4.1. *El tipo de dato* `Contador` *introducido en Ej. 2.5.1 y especificado en Ej. 3.2.1 es definido como una función* $\mathcal{F}_{ctr}$ *que en lugar de tomar una operación y un contexto, toma una operación y un par* $\langle \mathtt{G}, \mathtt{P} \rangle$ *tal que*

$$\mathcal{F}_{ctr}(\mathtt{inc}, \langle \mathtt{G}, \mathtt{P} \rangle) = \mathtt{ok}$$
$$\mathcal{F}_{ctr}(\mathtt{rd}, \langle \mathtt{G}, \mathtt{P} \rangle) = \#\{\mathtt{e} \mid \mathtt{e} \in \mathtt{G} \text{ y } \lambda(\mathtt{e}) = \mathtt{inc}\}$$

Dado un contexto $\langle \mathtt{G}, \mathtt{P} \rangle$ en $\mathbb{C}(O \times \mathcal{V})$, podemos verificar si el valor asociado a cada operación coincide con la definición particular del RDT que estamos considerando a través de la noción de *valor de retorno consistente* (ver Def. 2.6.2). Con el fin de relacionar contextos con y sin valores de retorno, usaremos la siguiente notación: dados $\mathtt{G} \in \mathbb{G}(O \times \mathcal{V})$, con $\overline{\mathtt{G}} \in \mathbb{G}(O)$ denotaremos el LDAG obtenido de proyectar las etiquetas de $\mathtt{G}$ de la forma obvia, es decir, quitando la segunda componente de cada etiqueta.

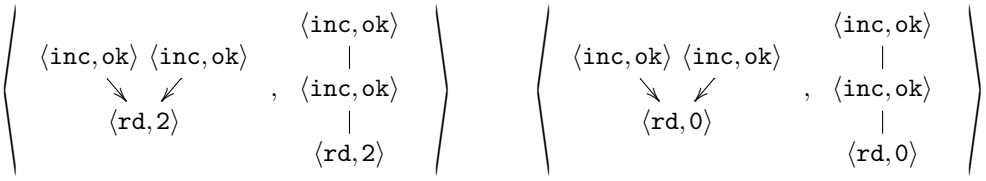
Definición 3.4.2 (RVAL Consistente)**.** *Sea* $\mathcal{F}$ *un* RDT *y* $\langle \mathtt{G}, \mathtt{P} \rangle \in \mathbb{C}(O \times \mathcal{V})$ *un contexto. Diremos que* $\mathcal{F}$ *es* RVAL Consistente *(*RVAL*) sobre* $\mathtt{G}$ *y* $\mathtt{P}$*, escrito como* $\text{RVAL}(\mathcal{F}, \mathtt{G}, \mathtt{P})$ *si*

$$\forall \mathtt{e} \in \mathtt{E_G}.\lambda(\mathtt{e}) = \langle \mathtt{o}, \mathtt{v} \rangle \text{ implica } \mathcal{F}(\mathtt{o}, \overline{\mathtt{G}}|_{-\prec \mathtt{e}}, \overline{\mathtt{P}}|_{-\prec \mathtt{e}}) = \mathtt{v}$$

Más aún, definimos

$$\text{PRVAL}(\mathcal{F}, \mathtt{G}) = \{\mathtt{P} \mid \text{RVAL}(\mathcal{F}, \mathtt{G}, \mathtt{P})\}$$

Ejemplo 3.4.2. *Considere el* RDT $\mathcal{F}_{ctr}$ *que se muestra en Ej. 3.4.1. El contexto en Fig. 3.3a es* RVAL Consistente *mientras que el descripto en Fig. 3.3b no lo es porque* $\mathcal{F}_{ctr}$ *requiere que* `rd` *devuelva el número de operaciones de* `inc` *vistas, la cual en este caso debería ser* `2` *y no* `0`.



(a) Consistente. (b) No consistente.

Figura 3.3: RVAL para $\mathcal{F}_{ctr}$.

3.4.2. Propiedades de las especificaciones

Ahora estudiamos algunas propiedades sobre las especificaciones funcionales que nos serán de utilidad para establecer la correspondencia con las especificaciones del enfoque clásico. Vamos a caracterizar una subclase adecuada de especificaciones que precisamente se corresponden a los RDTs. En esta sección, pondremos nuestra atención en especificaciones sobre el conjunto de etiquetas $O \times \mathcal{V}$, es decir, $\mathbb{S} : \mathbb{G}(O \times \mathcal{V}) \to 2^{\mathbb{P}(O \times \mathcal{V})}$.

Comenzamos con especificaciones que explican una configuración $\mathtt{G}$ a partir de explicaciones más chicas, es decir, a partir combinar las arbitraciones asociadas a los prefijos de $\mathtt{G}$.

Definición 3.4.3 (Especificaciones Coherente)**.** *Sea $\mathbb{S}$ una especificación. Diremos que $\mathbb{S}$ es* coherente *si*

$$\forall \mathtt{G}.\ \mathbb{S}(\mathtt{G}) = \bigotimes_{\mathtt{e} \in \mathtt{E_G}} \mathbb{S}(\mathtt{G}|_{-\prec^* \mathtt{e}})$$

Una especificación coherente $\mathbb{S}$ es una especificación donde las arbitraciones asociadas a una determinada configuración $\mathtt{G}$ (es decir, el conjunto de caminos $\mathbb{S}(\mathtt{G})$) se obtiene componiendo las arbitraciones asociadas a cada una de sus sub-configuraciones $\mathtt{G}|_{-\prec^* \mathtt{e}}$.

Ejemplo 3.4.3. *La especificación en Ej. 3.2.1, Ej. 3.2.2 y Ej. 3.2.3 son todas coherentes, ya que están definidas en términos de restringir los* LDAG*s correspondientes. Esto puede ser probado aplicando la definición en Def. 3.4.3. Considere, por ejemplo, la especificación del tipo de dato* `Contador` *en Ej. 3.2.1. Si $\mathtt{P} \in \mathbb{S}_{Ctr}(\mathtt{G})$ entonces*

$$\forall \mathtt{e} \in \mathtt{E_G}.\forall \mathtt{k}.\lambda(\mathtt{e}) = \langle \mathtt{rd}, \mathtt{k} \rangle \textit{ implica } \mathtt{k} = |\{\mathtt{e}' \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \textit{ y } \lambda(\mathtt{e}') = \langle \mathtt{inc}, \mathtt{ok} \rangle\}| \tag{3.5}$$

Sea $\overline{\mathtt{e}} \in \mathtt{E_G}$ y $\mathtt{G}_{\overline{\mathtt{e}}} = \mathtt{G}|_{-\prec^* \overline{\mathtt{e}}}$. Claramente la propiedad 3.5 se satisface $\forall \mathtt{e} \in \mathtt{E}_{\mathtt{G}_{\overline{\mathtt{e}}}}$. En consecuencia $\mathtt{P}|_{-\prec^* \overline{\mathtt{e}}} \in \mathbb{S}_{Ctr}(\mathtt{G}_{\overline{\mathtt{e}}}) = \mathbb{S}_{Ctr}(\mathtt{G}|_{-\prec^* \overline{\mathtt{e}}})$ y por consiguiente $\mathtt{P} \in \bigotimes_{\overline{\mathtt{e}} \in \mathtt{E_G}} \mathbb{S}_{Ctr}(\mathtt{G}|_{-\prec^* \overline{\mathtt{e}}})$. Por definición de $\mathtt{G}_{\overline{\mathtt{e}}}$, $\mathtt{k} = |\{\mathtt{e}' \mid \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e} \textit{ y } \lambda(\mathtt{e}') = \langle \mathtt{inc}, \mathtt{ok} \rangle\}|$ y consequentemente $\mathtt{P} \in \mathbb{S}_{Ctr}(\mathtt{G})$.

Ahora, considere una especificación $\mathbb{S}$ tal que se cumplen las igualdades descriptas en Fig. 3.4. $\mathbb{S}$ no es coherente porque las arbitraciones para el LDAG *en Fig. 3.4b debe contener todos los posibles ordenamientos para los caminos asociados a sus subconfiguraciones (representadas en Fig. 3.4a), es decir,*

$$\mathbb{S}\Big(\ \langle \mathtt{o}_1, \mathtt{v}_1 \rangle \quad \langle \mathtt{o}_2, \mathtt{v}_2 \rangle\ \Big) = \mathbb{S}(\langle \mathtt{o}_1, \mathtt{v}_1 \rangle) \otimes \mathbb{S}(\langle \mathtt{o}_2, \mathtt{v}_2 \rangle) = \left\{ \begin{array}{cc} \langle \mathtt{o}_1, \mathtt{v}_1 \rangle & \langle \mathtt{o}_2, \mathtt{v}_2 \rangle \\ | & | \\ \langle \mathtt{o}_2, \mathtt{v}_2 \rangle & \langle \mathtt{o}_1, \mathtt{v}_1 \rangle \end{array} \right\}$$

$$\mathbb{S}\Big(\ \langle o_1, v_1\rangle\ \Big) = \Big\{\ \langle o_1, v_1\rangle\ \Big\}$$
$$\mathbb{S}\Big(\ \langle o_2, v_2\rangle\ \Big) = \Big\{\ \langle o_2, v_2\rangle\ \Big\}$$

(a)

$$\mathbb{S}\Big(\ \langle o_1, v_1\rangle \quad \langle o_2, v_2\rangle\ \Big) = \left\{ \begin{array}{c} \langle o_1, v_1\rangle \\ | \\ \langle o_2, v_2\rangle \end{array} \right\}$$

(b)

Figura 3.4: Una especificación no-coherente.

El siguiente resultado establece que el valor de retorno asociado a cada evento de una arbitración para la especificación clásica de RDT es siempre *coherente*.

Lema 1. *Sea* $\mathcal{F}$ *una* RDT *y* G *un* LDAG*. Entonces*

$$\textsc{prval}(\mathcal{F}, G) = \bigotimes_{e \in E_G} \textsc{prval}(\mathcal{F}, G|_{-\prec^* e}).$$

Demostración. Sea $P \in \bigotimes_{e \in E_G} \textsc{prval}(\mathcal{F}, G|_{-\prec^* e})$, esto es:

$$\forall e \in E_G.\ \ P|_{-\prec^* e} \in \textsc{prval}(\mathcal{F}, G|_{-\prec^* e})$$

Por definición de RVAL *Consistente* (Def. 3.4.2), esto es equivalente a

$$\forall e \in E_G.\ \forall e' \in E_{G|_{-\prec^* e}}.\ \lambda(e') = \langle o', v'\rangle \textit{ implica } \mathcal{F}(o', (\overline{G}|_{-\prec^* e})\Big|_{-\prec e'}, (\overline{P}|_{-\prec^* e})\Big|_{-\prec e'}) = v'$$

Notar que $(\overline{G}|_{-\prec^* e})\Big|_{-\prec e'}$ coincide con $\overline{G}|_{-\prec e'}$, y lo mismo para $\overline{P}$. Entonces, el resultado vale ya que la fórmula a continuación coincide con

$$\forall e \in E_G.\ \lambda(e) = \langle o, v\rangle \textit{ implica } \mathcal{F}(o, \overline{G}|_{-\prec e}, \overline{P}|_{-\prec e}) = v$$

□

A continuación proporcionamos un resultado técnico para especificaciones coherentes, el cual será necesario para probar que una especificación coherente induce un funtor entre las categorías de las configuraciones y los conjuntos de arbitraciones.

Lema 2. *Sea* $\mathbb{S}$ *una especificación coherente y* $E \subseteq E_G$*. Si* E *es* downward closed*, entonces* $\mathbb{S}(G)|_E \subseteq \mathbb{S}(G|_E)$.

Demostración. Sea E un conjunto *downward closed*, que equivale a requerir $E = \bigcup_{e \in E} \lfloor e \rfloor$, por lo tanto, para todo $e \in E$ tenemos $(G|_E)|_{\lfloor e \rfloor} = G|_{\lfloor e \rfloor}$. Usando esto, y la definición de coherencia, tenemos

$$\mathbb{S}(G)|_E = (\bigotimes_{e \in E_G} \mathbb{S}(G|_{\lfloor e \rfloor}))\Big|_E \tag{3.6}$$

$$\mathbb{S}(G|_E) = \bigotimes_{e \in E} \mathbb{S}(G|_{\lfloor e \rfloor}) \tag{3.7}$$

Notar que $(\bigotimes_{e \in E_G} \mathbb{S}(G|_{\lfloor e \rfloor}))\Big|_E \subseteq \bigotimes_{e \in E} \mathbb{S}(G|_{\lfloor e \rfloor})$ porque todo camino en $(\bigotimes_{e \in E_G} \mathbb{S}(G|_{\lfloor e \rfloor}))\Big|_E$ es un camino en $\bigotimes_{e \in E} \mathbb{S}(G|_{\lfloor e \rfloor})$ ya que 3.6 restringe más caminos que 3.7. □

$$\mathbb{S}\Big(\ \langle \mathtt{inc}, \mathtt{ok}\rangle\ \Big) = \Big\{\ \langle \mathtt{inc}, \mathtt{ok}\rangle\ \Big\} \qquad \mathbb{S}\left(\begin{array}{c}\langle \mathtt{rd}, 0\rangle \\ \downarrow \\ \langle \mathtt{inc}, \mathtt{ok}\rangle\end{array}\right) = \left\{\begin{array}{c}\langle \mathtt{rd}, 0\rangle \\ | \\ \langle \mathtt{inc}, \mathtt{ok}\rangle\end{array}\right\}$$
$$\mathbb{S}\Big(\ \langle \mathtt{rd}, 0\rangle\ \Big) = \Big\{\ \langle \mathtt{rd}, 0\rangle\ \Big\}$$

Figura 3.5: Una especificación no saturada

Una segunda propiedad se refiere a la *saturación*. Intuitivamente, una especificación saturada permite arbitrar en cualquier posición a un elemento maximal de una configuración. Primero introduciremos la noción de saturación de un camino.

Definición 3.4.4 (Saturación de Camino)**.** *Sea* $\mathtt{P}$ *un camino y* ℓ *una etiqueta. Escribimos formalmente* $\mathtt{sat}(\mathtt{P}, \ell)$ *para representar al conjunto de caminos que se obtienen saturando* $\mathtt{P}$ *con* ℓ*. Lo definimos como*

$$\mathtt{sat}(\mathtt{P}, \ell) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathtt{E}_{\mathtt{P}^\ell}, \lambda_{\mathtt{P}^\ell})\ y\ \mathtt{Q}|_{\mathtt{E}_\mathtt{P}} = \mathtt{P}\}$$

Un camino $\mathtt{P}$ saturado con la etiqueta ℓ genera el conjunto de todos los caminos que se obtienen agregando a $\mathtt{P}$ un nuevo evento etiquetado con ℓ en cualquier posición de $\mathtt{P}$. Análogamente, una especificación saturada asegura que cada evento de una configuración $\mathtt{G}$ puede ser arbitrado en cualquier posición de los caminos de $\mathbb{S}(\mathtt{G})$.

Definición 3.4.5 (Especificación Saturada)**.** *Sea* $\mathbb{S}$ *una especificación. Diremos que,* $\mathbb{S}$ *es saturada si*

$$\forall \langle \mathtt{G}, \mathtt{P}\rangle, \mathtt{E}, \ell.\ \mathtt{P} \in \mathbb{S}(\mathtt{G}^\ell_\mathtt{E})\Big|_{\mathtt{E}_\mathtt{G}}\ \textit{implica}\ \mathtt{sat}(\mathtt{P}, \ell) \subseteq \mathbb{S}(\mathtt{G}^\ell_\mathtt{E})$$

Ejemplo 3.4.4. *Las especificaciones en Ej. 3.2.1, Ej. 3.2.2 y Ej. 3.2.3 son todas saturadas ya que un nuevo evento* $\mathtt{e}$ *etiquetado con* ℓ *aparece arbitrado en cualquier posición de los caminos de* $\mathbb{S}(\mathtt{G}^\ell_\mathtt{E})\Big|_{\mathtt{E}_\mathtt{G}}$*. De hecho, las dos especificaciones en Ej. 3.2.1 y Ej. 3.2.3 no usan ninguna información sobre la arbitración, mientras que la especificación Ej. 3.2.2 restringe arbitraciones solo para eventos que no son maximales. Fig. 3.5 muestra una especificación que no es saturada ya que no permite arbitrar al evento más grande o maximal (el evento etiquetado por* $\langle \mathtt{inc}, \mathtt{ok}\rangle$*) como la primera operación de un camino. Esta es una especificación coherente aunque no es saturada.*

La tercera propiedad que estudiamos es la de ser *funcional*, es decir, en cualquier estado en que el que se encuentre una réplica se puede realizar una operación y el valor de retorno es único. Una especificación es *funcional* si cumple las propiedades de ser *total* y *determinística*.

Definición 3.4.6 (Especificaciones Totales)**.** *Sea* $\mathbb{S}$ *una especificación. Decimos que* $\mathbb{S}$ *es total si*

$$\forall \langle \mathtt{G}, \mathtt{P}\rangle, \mathtt{E}, \mathtt{o}.\ \exists \mathtt{G}_1,\ \mathtt{v}.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \wedge \overline{\mathtt{P}} \in \overline{\mathbb{S}((\mathtt{G}_1)^{\langle \mathtt{o}, \mathtt{v}\rangle}_\mathtt{E})}\Big|_{\mathtt{E}_{\mathtt{G}_1}}$$

Intuitivamente, una especificación es total cuando cada operación de un tipo de dato se puede realizar en cualquier configuración. Mas específicamente, considere el contexto $\langle \overline{\mathtt{G}}, \overline{\mathtt{P}} \rangle$ como una representación del estado de una réplica. Queremos remarcar que a diferencia de $\mathtt{G}$ y $\mathtt{P}$, que son etiquetadas por $O \times \mathcal{V}$, $\overline{\mathtt{G}}$ y $\overline{\mathtt{P}}$ no tienen información de los valores de retorno de cada operación (es decir, sus etiquetas son tomadas de O). Por lo tanto, la propiedad de totalidad nos dice que siempre podemos tomar una representación equivalente del estado (es decir, $\langle \mathtt{G}_1, \mathtt{P} \rangle$ en lugar de $\langle \mathtt{G}, \mathtt{P} \rangle$) y extenderlo con una operación obteniendo una configuración que tiene arbitraciones admisibles. Esto es que $\overline{\mathtt{P}} \in \overline{\mathbb{S}((\mathtt{G}_1)_{\mathtt{E}}^{\langle \mathtt{o},\mathtt{v} \rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}}$ para algún valor de retorno $\mathtt{v}$.

Es importante remarcar que una especificación total no evita que una operación admita más de un valor de retorno para una determinada configuración, es decir, $\mathtt{v}$ en Definición 3.4.8 no necesariamente es único. Por ejemplo, considerar el tipo de dato `Registro Genérico` presentado en Ej. 3.2.3, para el cual la operación `rd` puede retornar cualquier valor previamente escrito. A pesar de ser total, la especificación es no determinística ya que existe más de un valor de retorno que explica una misma configuración. Por el contrario, una especificación es determinística si toda operación ejecutada sobre una configuración admite a lo sumo un valor de retorno. A continuación, definimos la noción de especificación determinística.

Definición 3.4.7 (Especificación Determinística)**.** *Sea $\mathbb{S}$ una especificación. Decimos que $\mathbb{S}$ es determinística si*

$$\forall \mathtt{G}, \mathtt{E}, \mathtt{o}, \mathtt{v}, \mathtt{v}'.\ \mathtt{v} \neq \mathtt{v}' \textit{ implica } \overline{\mathbb{S}(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{o},\mathtt{v} \rangle})}\Big|_{\mathtt{E}_{\mathtt{G}}} \cap \overline{\mathbb{S}(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{o},\mathtt{v}' \rangle})}\Big|_{\mathtt{E}_{\mathtt{G}}} = \emptyset$$

Intuitivamente estamos diciendo que si extendemos una configuración $\mathtt{G}$ con una operación $\mathtt{o}$ cuyo valor de retorno puede ser $\mathtt{v}$ o $\mathtt{v}'$, necesariamente las arbitraciones que explican el valor de retorno $\mathtt{v}$ tienen un orden diferente para explicar el valor de retorno $\mathtt{v}'$.

Diremos que $\mathbb{S}$ es *débilmente determinística* si la propiedad se cumple para $\mathtt{E} = \emptyset$.

Una noción más débil para el determinismo podría permitir que el resultado de agregar una operación dependa también del camino admisible dado. Decimos que una especificación $\mathbb{S}$ es *value-deterministic* si

$$\forall \mathtt{G}, \mathtt{E}, \mathtt{o}, \mathtt{v}, \mathtt{v}'.\ \mathtt{v} \neq \mathtt{v}' \wedge\ \mathtt{G} \neq \varepsilon \textit{ implica } \mathbb{S}(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{o},\mathtt{v} \rangle})\Big|_{\mathtt{E}_{\mathtt{G}}} \cap \mathbb{S}(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{o},\mathtt{v}' \rangle})\Big|_{\mathtt{E}_{\mathtt{G}}} = \emptyset$$

Ejemplo 3.4.5. *Fig. 3.6 muestra una especificación* value-deterministic. *Aunque una operación de lectura que sigue un incremento puede retornar dos valores diferentes, tales diferencias pueden ser explicadas por las computaciones anteriores: en un caso, el incremento ocurre mientras en el otro falla. Sin embargo, la especificación no es determinística porque admite una secuencia de operaciones que pueden ser decoradas con diferentes valores de retorno.*

Observación 4. *Observamos que una especificación no determinística no implica una resolución de conflictos no determinística, pero permite una sub-especificación.*

$$\mathbb{S}\left(\begin{array}{c}\langle \texttt{inc},\texttt{ok}\rangle \\ \downarrow \\ \langle \texttt{rd},1\rangle\end{array}\right)=\left\{\begin{array}{c}\langle \texttt{inc},\texttt{ok}\rangle \\ | \\ \langle \texttt{rd},1\rangle\end{array}\right\} \quad \mathbb{S}\left(\begin{array}{c}\langle \texttt{inc},\texttt{fail}\rangle \\ \downarrow \\ \langle \texttt{rd},\bot\rangle\end{array}\right)=\left\{\begin{array}{c}\langle \texttt{inc},\texttt{fail}\rangle \\ | \\ \langle \texttt{rd},\bot\rangle\end{array}\right\}$$

(a) (b)

Figura 3.6: Especificación value-deterministic y coherente

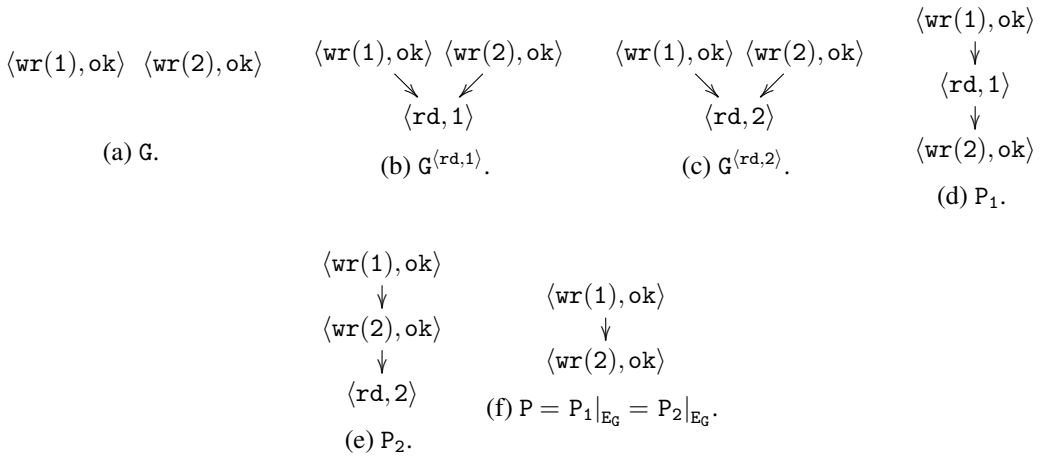


Figura 3.7: Registro Genérico.

Ejemplo 3.4.6. *Podemos corroborar que la especificación definida en el ejemplo Ej. 3.2.1 y Ej. 3.2.2 son determinísticas. Para Ej. 3.2.1 razonamos de la siguiente manera. El caso para* $\texttt{o} = \texttt{inc}$ *es inmediato porque el único valor de retorno posible es* ok*. Cuando* $\texttt{o} = \texttt{rd}$*, por definición de* $\mathbb{S}_{Ctr}$ *(Ej. 3.2.1) concluímos que* $\mathbb{S}_{Ctr}(\texttt{G}_{\texttt{E}}^{\langle \texttt{rd},\texttt{v}\rangle}) \neq \emptyset$ *sólo cuando* $\texttt{v} = |\{\texttt{e} \mid \texttt{e} \in \texttt{E}\ y\ \lambda(\texttt{e}) = \langle \texttt{inc},\texttt{ok}\rangle\}|$*. Consecuentemente, para cada* $\texttt{v}' \neq \texttt{v}$*, vale que* $\mathbb{S}_{Ctr}(\texttt{G}_{\texttt{E}}^{\langle \texttt{rd},\texttt{v}'\rangle}) = \emptyset$ *se mantiene, por lo tanto,* $\mathbb{S}_{Ctr}$ *es determinítica. Para Ej. 3.2.2, podemos realizar un razonamiento análogo.*

Contrariamente, la especificación del RDT `Registro Genérico` *dada en Ej. 3.2.3 no es ni siquiera value-deterministic. Es suficiente considerar una configuración* G *con dos valores diferentes, como se muestra en Fig. 3.7a. Considere dos extensiones* $\texttt{G}^{\langle \texttt{rd},1\rangle}$ *y* $\texttt{G}^{\langle \texttt{rd},2\rangle}$ *representadas en Fig. 3.7b y Fig. 3.7c y los dos caminos* $\texttt{P}_1$ *y* $\texttt{P}_2$ *en Fig. 3.7d y Fig. 3.7e. Por definición de* $\mathbb{S}_{gR}$*, podemos concluir que* $\texttt{P}_1 \in \mathbb{S}_{gR}(\texttt{G}^{\langle \texttt{rd},1\rangle})$ *y* $\texttt{P}_2 \in \mathbb{S}_{gR}(\texttt{G}^{\langle \texttt{rd},2\rangle})$*. El camino* P *en Fig. 3.7f se corresponde a ambos caminos:* $\texttt{P}_1|_{\texttt{E}_\texttt{G}}$ *y* $\texttt{P}_2|_{\texttt{E}_\texttt{G}}$*. Consecuentemente,* $\mathbb{S}_{gR}(\texttt{G}^{\langle \texttt{rd},1\rangle})\Big|_{\texttt{E}_\texttt{G}} \cap \mathbb{S}_{gR}(\texttt{G}^{\langle \texttt{rd},2\rangle})\Big|_{\texttt{E}_\texttt{G}} \neq \emptyset$*.*

Similarmente, la especificación para el RDT `Set` *presentada en Ej. 3.3.2 es no determinístico.*

El siguiente lema establece un criterio para el determinismo.

Lema 3. *Sea* $\mathbb{S}$ *una especificación coherente y determinística, entonces,*

$$\forall \mathtt{G}_1, \mathtt{G}_2.\ \overline{\mathtt{G}_1} = \overline{\mathtt{G}_2} \textit{ implica } \mathtt{G}_1 = \mathtt{G}_2 \vee\ \overline{\mathbb{S}(\mathtt{G}_1)} \cap \overline{\mathbb{S}(\mathtt{G}_2)} = \emptyset$$

Demostración. Considere $\mathtt{G}_1$, $\mathtt{G}_2$ tal que $\overline{\mathtt{G}_1} = \overline{\mathtt{G}_2}$ y $\mathtt{G}_1 \neq \mathtt{G}_2$. Vamos a demostrar que $\overline{\mathbb{S}(\mathtt{G}_1)} \cap \overline{\mathbb{S}(\mathtt{G}_2)} = \emptyset$ vale. Dado que $\mathtt{G}_1 \neq \mathtt{G}_2$ existe un evento $\bar{\mathtt{e}}$ tal que

$$\mathtt{G}_1|_{-\prec^+\bar{\mathtt{e}}} = \mathtt{G}_2|_{-\prec^+\bar{\mathtt{e}}} \text{ y } \lambda_1(\bar{\mathtt{e}}) = \langle \mathtt{o}, \mathtt{v}_1 \rangle, \lambda_2(\bar{\mathtt{e}}) = \langle \mathtt{o}, \mathtt{v}_2 \rangle \text{ para } \mathtt{v}_1 \neq \mathtt{v}_2$$

Sea $\mathtt{G} = \mathtt{G_i}|_{-\prec^+\bar{\mathtt{e}}}$. Por determinismo, tenemos que $\overline{\mathbb{S}(\mathtt{G}^{\lambda_1(\bar{\mathtt{e}})}_{-\prec\bar{\mathtt{e}}})}\Big|_{\mathtt{E_G}} \cap \overline{\mathbb{S}(\mathtt{G}^{\lambda_2(\bar{\mathtt{e}})}_{-\prec\bar{\mathtt{e}}})}\Big|_{\mathtt{E_G}} = \emptyset$, y equivalentemente que $\overline{\mathbb{S}(\mathtt{G}_1|_{-\prec^*\bar{\mathtt{e}}})}\Big|_{-\prec^+\bar{\mathtt{e}}} \cap \overline{\mathbb{S}(\mathtt{G}_2|_{-\prec^*\bar{\mathtt{e}}})}\Big|_{-\prec^+\bar{\mathtt{e}}} = \emptyset$.

Ahora, asumir que existe $\mathtt{P_i} \in \mathbb{S}(\mathtt{G_i})$ tal que $\overline{\mathtt{P}_1} = \overline{\mathtt{P}_2}$, entonces por coherencia sabemos que $\forall \mathtt{e}.\ \mathtt{P_i}|_{-\prec^*\mathtt{e}} \in \mathbb{S}(\mathtt{G_i}|_{-\prec^*\mathtt{e}})$. En consecuencia, $\forall \mathtt{e}.\ \mathtt{P_i}|_{-\prec^+\mathtt{e}} \in \mathbb{S}(\mathtt{G_i}|_{-\prec^*\mathtt{e}})\Big|_{-\prec^+\mathtt{e}}$, lo cual contradice el hecho de que $\mathtt{e} = \bar{\mathtt{e}}$. □

Entonces, si dos configuraciones tienen las mismas operaciones con valores diferentes, entonces sus caminos admisibles son todos diferentes incluso si ignoramos los valores de retorno.

Luego consideramos una última propiedad que garantiza algún tipo de localidad adicional para la noción de coherencia.

Definición 3.4.8 (Especificación Local)**.** *Una especificación* $\mathbb{S}$ *es* local *si*

$$\forall \mathtt{G}, \mathtt{E}, \mathtt{o}, \mathtt{v}.\ \exists \mathtt{G}_1.\ \overline{\mathtt{G}}|_{\mathtt{E}} = \overline{\mathtt{G}_1} \ \wedge\ \overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}\rangle}_{\mathtt{E}})}\Big|_{-\preccurlyeq\top} \subseteq \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{o},\mathtt{v}\rangle})}$$

Intuitivamente, la localidad establece que los caminos admisibles de una configuración extendida están restringidos: dada una configuración $\mathtt{G}$, su extensión con una operación $\mathtt{o}$ con respecto a los eventos $\mathtt{E}$ pueden ser explicados por la sub-configuración de $\mathtt{G}$ que solo contiene aquellos eventos, es decir, $\overline{\mathtt{G}}|_{\mathtt{E}} = \overline{\mathtt{G}_1}$. Tal comportamiento está dado por $\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle})$. Decimos que una especificación es *localmente funcional* si esta es funcional y local.

Lema 4. *Sea* $\mathbb{S}$ *una especificación local y débilmente determinística. Entonces* $\mathbb{S}$ *es determinística.*

Demostración. Por *localidad*, sabemos que para cualquier $\mathtt{G}$ y $\mathtt{E}$ existe $\mathtt{G}_1$ tal que para todo $\mathtt{o}, \mathtt{v}$, tenemos $\overline{\mathtt{G}}|_{\mathtt{E}} = \overline{\mathtt{G}_1} \ \wedge\ \overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}\rangle}_{\mathtt{E}})}\Big|_{-\preccurlyeq\top} \subseteq \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{o},\mathtt{v}\rangle})}$.

Si $\mathbb{S}$ no es determinística, entonces para algún $\mathtt{o}, \mathtt{v}, \mathtt{v}'$ tal que $\mathtt{v} \neq \mathtt{v}'$ tenemos

$$\overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}\rangle}_{\mathtt{E}})}\Big|_{\mathtt{E_G}} \cap\ \overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}'\rangle}_{\mathtt{E}})}\Big|_{\mathtt{E_G}} \neq \emptyset$$

En consecuencia,

$$\overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}\rangle}_{\mathtt{E}})}\Big|_{-\prec\top} \cap\ \overline{\mathbb{S}(\mathtt{G}^{\langle \mathtt{o},\mathtt{v}'\rangle}_{\mathtt{E}})}\Big|_{-\prec\top} \neq \emptyset$$

y por lo tanto tenemos

$$\overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \cap \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}'\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \neq \emptyset$$

el cual contradice con ser débilmente determinística para $\mathtt{G}_1$. □

3.4.3. Correspondencia entre RDTs y Especificaciones

A continuación haremos foco en la relación que hay entre nuestra noción de especificación funcional, introducida en Definición 3.2.1, y la descripción operacional de RDTs, introducidos en § 2.5 y formalizados en Definición 3.4.1 en términos de LDAGs. Específicamente, caracterizamos una subclase adecuada de especificaciones que precisamente se corresponden al enfoque clásico para especificar RDTs. Primero comenzaremos introduciendo la correspondencia de RDTs a especificaciones.

Definición 3.4.9 (RDTs como Especificaciones)**.** *Sea $\mathcal{F}$ un* RDT*. Escribimos $\mathcal{S}(\mathcal{F})$ para la especificación asociada con $\mathcal{F}$, definida como*

$$\mathcal{S}(\mathcal{F})(\mathtt{G}) = \textsc{prval}(\mathcal{F},\mathtt{G})$$

El siguiente resultado muestra que cada RDTse corresponde con un especificación coherente, funcional y saturada.

Lema 5. *Para todo* RDT *$\mathcal{F}$, $\mathcal{S}(\mathcal{F})$ es coherente, localmente funcional, y saturado.*

Demostración. Probamos las cuatro propiedades de $\mathcal{S}(\mathcal{F})$.

- Coherente. Inmediato por Lema 1.
- Saturado. Tenemos que probar que

  $$\forall\langle \mathtt{G},\mathtt{P}\rangle,\mathtt{E},\langle \mathtt{op},\mathtt{v}\rangle.\ \mathtt{P} \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})\Big|_{\mathtt{E}_{\mathtt{G}}} \text{ implica } \mathtt{sat}(\mathtt{P},\langle \mathtt{op},\mathtt{v}\rangle) \subseteq \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})$$

  Dado que $\mathtt{P} \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})\Big|_{\mathtt{E}_{\mathtt{G}}}$, entonces hay $\mathtt{P}_1 \in \mathtt{sat}(\mathtt{P},\langle \mathtt{op},\mathtt{v}\rangle) \cap \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})$.

  $$\begin{aligned}
  \mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) &\overset{\text{Def. 3.4.9}}{\equiv} \mathtt{P}_1 \in \textsc{prval}(\mathcal{F},\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) \\
  &\overset{\text{Def. 3.4.2}}{\equiv} \textsc{rval}(\mathcal{F},\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle},\mathtt{P}_1) \\
  &\overset{\text{Def. 3.4.2}}{\equiv} \textsc{rval}(\mathcal{F},\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}\Big|_{\mathtt{E}_{\mathtt{G}}},\mathtt{P}_1|_{\mathtt{E}_{\mathtt{G}}}) \wedge \\
  &\qquad \mathcal{F}(\mathtt{op},\overline{\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}\Big|_{-\prec\top}},\overline{\mathtt{P}_1}|_{-\prec\top}) = \mathtt{v} \\
  &= \textsc{rval}(\mathcal{F},\mathtt{G},\mathtt{P}) \wedge \\
  &\qquad \mathcal{F}(\mathtt{op},\overline{\mathtt{G}}|_{\mathtt{E}},\overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}
  \end{aligned}$$

  Luego, para cada $\mathtt{P}_2 \in \mathtt{sat}(\mathtt{P},\langle \mathtt{op},\mathtt{v}\rangle)$ notar que $\overline{\mathtt{P}_2}|_{\mathtt{E}} = \overline{\mathtt{P}_1}|_{\mathtt{E}}$ y, consecuentemente, $\mathcal{F}(\mathtt{op},\overline{\mathtt{G}}|_{\mathtt{E}},\overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}$ se mantiene. Por lo tanto, $\mathtt{P}_2 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})$ se mantiene y el resultado se mantiene.

- Total. Asumamos que existe $\mathtt{G}$, $\mathtt{P}$, $\mathtt{E}$, y $\mathtt{op}$ tal que

$$\forall \mathtt{G}_1, \mathtt{v}.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \text{ implies } \overline{\mathtt{P}} \notin \overline{\mathcal{S}(\mathcal{F})((\mathtt{G}_1)_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \tag{3.8}$$

  Más aún, sin perder generalidad podemos asumir que $\mathtt{G}$ es un LDAG minimal que satisface (3.8), es decir, que para todo $\mathtt{G}'$ estrictamente contenido en $\mathtt{G}$ y para todo $\mathtt{P}' \in \mathcal{S}(\mathcal{F})(\mathtt{G}')$ tenemos

$$\forall \mathtt{E}', \mathtt{op}'.\ \exists \mathtt{G}'_1, \mathtt{v}'.\ \overline{\mathtt{G}'} = \overline{\mathtt{G}'_1} \wedge \overline{\mathtt{P}'} \in \overline{\mathcal{S}(\mathcal{F})((\mathtt{G}'_1)_{\mathtt{E}}^{\langle \mathtt{op}',\mathtt{v}'\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}'_1}}$$

  Ahora, consideremos $\mathtt{G}'$, $\mathtt{E}'$, y $\mathtt{op}'$ tal que $\overline{\mathtt{G}} = (\overline{\mathtt{G}'})_{\mathtt{E}'}^{\mathtt{op}'}$ y sea $\mathtt{P}' = \mathtt{P}|_{\mathtt{E}_{\mathtt{G}'}}$. Dado que probamos que $\mathcal{S}(\mathcal{F})$ es saturada, tenemos

$$\exists \langle \mathtt{G}'_1, \mathtt{P}'_1\rangle,\ \mathtt{v}'.\ \overline{\mathtt{G}'} = \overline{\mathtt{G}'_1} \wedge \overline{\mathtt{P}'} = \overline{\mathtt{P}'_1} \wedge \mathtt{sat}(\mathtt{P}'_1, \langle \mathtt{op}', \mathtt{v}'\rangle) \subseteq \mathcal{S}(\mathcal{F})((\mathtt{G}'_1)_{\mathtt{E}'}^{\langle \mathtt{op}',\mathtt{v}'\rangle})$$

  Entonces, tomemos $\mathtt{G}_1 = (\mathtt{G}'_1)_{\mathtt{E}'}^{\langle \mathtt{op}',\mathtt{v}'\rangle}$ y $\mathtt{P}_1 \in \mathtt{sat}(\mathtt{P}'_1, \langle \mathtt{op}', \mathtt{v}'\rangle)$ tal que $\overline{\mathtt{P}} = \overline{\mathtt{P}_1}$ y $\mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_1)$. Primero, notar que $\overline{\mathtt{G}} = \overline{\mathtt{G}_1}$ porque $\overline{\mathtt{G}} = (\overline{\mathtt{G}'})_{\mathtt{E}'}^{\mathtt{op}'}$, $\overline{\mathtt{G}'} = \overline{\mathtt{G}'_1}$, y $\mathtt{G}_1 = (\mathtt{G}'_1)_{\mathtt{E}'}^{\langle \mathtt{op}',\mathtt{v}'\rangle}$. Ahora mostramos que para cada $\mathtt{E}$ y $\mathtt{op}$ existe $\mathtt{v}$ tal que $\mathtt{P}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle} \in \mathcal{S}(\mathcal{F})((\mathtt{G}_1)_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})$, el cual se contradice con la suposición (3.8). Dados $\mathtt{E}$ y $\mathtt{op}$, tomar $\mathtt{v}$ tal que $\mathcal{F}(\mathtt{op}, \overline{\mathtt{G}_1}|_{\mathtt{E}}, \overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}$ (este *v* existe por la definición de $\mathcal{F}$). Siguiendo el mismo razonamiento que para saturación, tenemos

$$\begin{aligned}
\mathtt{P}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle} \in \mathcal{S}(\mathcal{F})((\mathtt{G}_1)_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) &\overset{\text{Def. 3.4.9}}{\equiv} \mathtt{P}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle} \in \textsc{prval}(\mathcal{F}, (\mathtt{G}_1)_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) \\
&\overset{\text{Def. 3.4.2}}{\equiv} \ldots \\
&= \textsc{rval}(\mathcal{F}, \mathtt{G}_1, \mathtt{P}_1) \wedge \\
&\quad\ \mathcal{F}(\mathtt{op}, \overline{\mathtt{G}_1}|_{\mathtt{E}}, \overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}
\end{aligned}$$

  Dado que $\textsc{rval}(\mathcal{F}, \mathtt{G}_1, \mathtt{P}_1)$ coincide con $\mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_1)$, el resultado se mantiene. □

- Determinística. Sea $\mathtt{G}$, $\mathtt{E}$, $\mathtt{op}$, $\mathtt{v}_1$, y $\mathtt{v}_2$ tal que $\mathtt{v}_1 \neq \mathtt{v}_2$ y

$$\overline{\mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_1\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}}} \cap \overline{\mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_2\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}}} \neq \emptyset$$

  Entonces, existen los caminos $\mathtt{P}_1$, $\mathtt{P}_2$ tal que $\mathtt{P}_i \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_i\rangle})$ y $\overline{\mathtt{P}_1}|_{\mathtt{E}_{\mathtt{G}}} = \overline{\mathtt{P}_2}|_{\mathtt{E}_{\mathtt{G}}}$. Por la definición de $\mathcal{S}(\mathcal{F})$ (ver Def. 3.4.9), la primer condición es equivalente a $\mathtt{P}_i \in \textsc{prval}(\mathcal{F}, \mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_i\rangle})$ y por lo tanto a $\textsc{rval}(\mathcal{F}, \mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_i\rangle}, \mathtt{P}_i)$. Por definición de *valor de retorno consistente* (Def. 3.4.2), tenemos que

$$\mathcal{F}(\mathtt{op}, \overline{\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_i\rangle}}\Big|_{-\prec\top}, \overline{\mathtt{P}_i}|_{-\prec\top}) = \mathtt{v}_i$$

y en consecuencia

$$\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}_i\rangle}\Big|_{-\prec\top} = \mathtt{G}|_{\mathtt{E}} \text{ y } \mathtt{P}_\mathtt{i}|_{-\prec\top} = \mathtt{P}_\mathtt{i}|_{\mathtt{E}}$$

lo que resulta ser una contradicción.

- Local. Para cada $\mathtt{P}$ tal que $\overline{\mathtt{P}} \in \overline{\mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})}\Big|_{-\preccurlyeq\top}$ existe $\mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle})$ tal que $\overline{\mathtt{P}} = \overline{\mathtt{P}_1}|_{-\preccurlyeq\top}$. Entonces, siguiendo el mismo razonamiento que para saturación, tenemos:

$$\begin{aligned}\mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) &\overset{\text{Def. 3.4.9}}{\equiv} \mathtt{P}_1 \in \mathrm{PRVAL}(\mathcal{F}, \mathtt{G}_{\mathtt{E}}^{\langle \mathtt{op},\mathtt{v}\rangle}) \\ &\overset{\text{Def. 3.4.2}}{\equiv} \forall \mathtt{e} \in \mathtt{E}_\mathtt{G}.\ \lambda(\mathtt{e}) = \langle \mathtt{op}', \mathtt{v}'\rangle \textit{ implica} \\ &\qquad \mathcal{F}(\mathtt{op}', \overline{\mathtt{G}}|_{-\prec\mathtt{e}}, \overline{\mathtt{P}_1}|_{-\prec\mathtt{e}}) = \mathtt{v}' \\ &\quad \wedge\ \mathcal{F}(\mathtt{op}, \overline{\mathtt{G}}|_{\mathtt{E}}, \overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}\end{aligned}$$

Dado que $\mathcal{F}$ es una función, para cada $\mathtt{G}_1$ tal que $\overline{\mathtt{G}}|_{\mathtt{E}} = \overline{\mathtt{G}_1}$ se cumple:

$$\forall \mathtt{e} \in \mathtt{E}_{\mathtt{G}_1}, \mathtt{op}'.\exists \mathtt{v}''.\mathcal{F}(\mathtt{op}', \overline{\mathtt{G}_1}|_{-\prec\mathtt{e}}, \overline{\mathtt{P}_1}|_{-\prec\mathtt{e}}) = \mathtt{v}''$$

Por consecuencia, por coherencia, existe $\mathtt{P}_2 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_1)$ tal que $\overline{\mathtt{P}_1}|_{\mathtt{E}_{\mathtt{G}_1}} = \overline{\mathtt{P}_2}$. Más aún, $\mathcal{F}(\mathtt{op}, \overline{\mathtt{G}}|_{\mathtt{E}}, \overline{\mathtt{P}_1}|_{\mathtt{E}}) = \mathtt{v}$ y por saturación, esto implica que $\mathtt{sat}(\mathtt{P}_2, \langle \mathtt{op}, \mathtt{v}\rangle) \subseteq \mathcal{S}(\mathcal{F})(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle})$. En consecuencia, $\overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle})}$.

□

La correspondencia inversa, es decir de especificaciones a RDTs se define a continuación.

Definición 3.4.10 (Especificaciones como RDTs)**.** *Sea* $\mathbb{S}$ *una especificación. Escribimos* $\mathbb{F}(\mathbb{S})$ *para el* RDT *asociado con* $\mathbb{S}$*, definido como*

$$\mathbb{F}(\mathbb{S})(\mathtt{o}, \overline{\mathtt{G}}, \overline{\mathtt{P}}) = \mathtt{v} \textit{ si } \exists \mathtt{G}_1.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \wedge \overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{o},\mathtt{v}\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}}$$

Notar que $\mathbb{F}(\mathbb{S})$ puede no estar bien definida para algunas especificaciones, por ejemplo cuando $\mathbb{S}$ es no determinística. El siguiente lema establece las condiciones suficientes para garantizar que $\mathbb{F}(\mathbb{S})$ está bien definida.

Lema 6. *Para toda especificación coherente, funcional y saturada* $\mathbb{S}$*,* $\mathbb{F}(\mathbb{S})$ *está bien definida.*

Demostración. Dado que $\mathbb{S}$ es total, existe al menos un valor para cada tripla $\mathtt{op}$, $\overline{\mathtt{G}}$, y $\overline{\mathtt{P}}$. Asumamos que hay más de dos valores, es decir, existe $\mathtt{op}$, $\mathtt{G}_\mathtt{i}$, y $\mathtt{v}_i$ tal que

$$\overline{\mathtt{G}} = \overline{\mathtt{G}_1} = \overline{\mathtt{G}_2} \wedge \mathtt{v}_1 \neq \mathtt{v}_2 \wedge \overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_\mathtt{i}{}^{\langle \mathtt{op},\mathtt{v}_\mathtt{i}\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_\mathtt{i}}}$$

Dado que $\mathbb{S}$ es determinística y $\overline{\mathtt{G}_1} = \overline{\mathtt{G}_2}$, pueden ocurrir dos casos por Lem. 3

- $\mathtt{G}_1 = \mathtt{G}_2$. Por hipótesis tenemos que $\overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_i\langle \mathtt{op}, \mathtt{v}_i\rangle)}\Big|_{\mathtt{E}_{\mathtt{G}_i}}$, y por lo tanto

$$\overline{\mathbb{S}(\mathtt{G}_1\langle \mathtt{op}, \mathtt{v}_1\rangle)}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \cap \overline{\mathbb{S}(\mathtt{G}_1\langle \mathtt{op}, \mathtt{v}_2\rangle)}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \neq \emptyset$$

  De nuevo, por ser $\mathbb{S}$ determinística, se llega a $\mathtt{v}_1 = \mathtt{v}_2$.

- $\overline{\mathbb{S}(\mathtt{G}_1)} \cap \overline{\mathbb{S}(\mathtt{G}_2)} = \emptyset$. Dado que $\mathbb{S}$ es coherente y saturada y por hipótesis $\overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_i\langle \mathtt{op}, \mathtt{v}_i\rangle)}\Big|_{\mathtt{E}_{\mathtt{G}_i}}$, tenemos que $\overline{\mathtt{P}} \in \overline{\mathbb{S}(\mathtt{G}_i)}$, lo que nos lleva a una contradicción.

□

Presentamos el siguiente resultado auxiliar que será usado para la prueba del Teorema 1.

Lema 7. *Sea $\mathbb{S}$ una especificación coherente, funcional y saturada y $\langle \mathtt{G}, \mathtt{P}\rangle$ un contexto. Si*

$$\forall \mathtt{e} \in \mathtt{E}_{\mathtt{G}}.\ \lambda(\mathtt{e}) = \langle \mathtt{op}, \mathtt{v}\rangle \textit{ implica } \exists \mathtt{G}_1.\ \overline{\mathtt{G}}|_{-\prec \mathtt{e}} = \overline{\mathtt{G}_1} \wedge \overline{\mathtt{P}}|_{-\prec \mathtt{e}} \in \overline{\mathbb{S}(\mathtt{G}_1\langle \mathtt{op}, \mathtt{v}\rangle)}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \quad (3.9)$$

entonces $\forall \mathtt{e} \in \mathtt{E}_{\mathtt{G}}.\ \mathtt{P}|_{-\prec^* \mathtt{e}} \in \mathbb{S}(\mathtt{G}|_{-\prec^* \mathtt{e}})$.

Más aún, sea $\mathbb{S}$ una especificación local y coherente. Entonces la vuelta se cumple.

Demostración.

($\Leftarrow$ Sea $\bar{\mathtt{e}}$ tal que $\lambda(\bar{\mathtt{e}}) = \langle \mathtt{op}, \mathtt{v}\rangle$. Por hipótesis $\mathtt{P}|_{-\prec^* \bar{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^* \bar{\mathtt{e}}})$, por coherencia $\mathtt{P}|_{-\prec^+ \bar{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^+ \bar{\mathtt{e}}})$, y por localidad

$$\exists \mathtt{G}_1.\ (\overline{\mathtt{G}}|_{-\prec^+ \bar{\mathtt{e}}})\Big|_{-\prec \bar{\mathtt{e}}} = \overline{\mathtt{G}_1} \wedge \overline{\mathbb{S}((\mathtt{G}|_{-\prec^+ \bar{\mathtt{e}}})^{\langle \mathtt{op}, \mathtt{v}\rangle}_{-\prec \bar{\mathtt{e}}})}\Big|_{-\preccurlyeq \bar{\mathtt{e}}} \subseteq \overline{\mathbb{S}(\mathtt{G}_1\langle \mathtt{op}, \mathtt{v}\rangle)}$$

Esto se vuelve equivalente a

$$\exists \mathtt{G}_1.\ \overline{\mathtt{G}}|_{-\prec \bar{\mathtt{e}}} = \overline{\mathtt{G}_1} \wedge \overline{\mathbb{S}(\mathtt{G}|_{-\prec^* \bar{\mathtt{e}}})}\Big|_{-\preccurlyeq \bar{\mathtt{e}}} \subseteq \overline{\mathbb{S}(\mathtt{G}_1\langle \mathtt{op}, \mathtt{v}\rangle)}$$

y de nuevo por hipótesis $\mathtt{P}|_{-\prec^* \bar{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^* \bar{\mathtt{e}}})$ el resultado se mantiene.

$\Rightarrow$) Esta prueba se sigue por contradicción. Asumamos que existe $\bar{\mathtt{e}} \in \mathtt{E}_{\mathtt{G}}$ tal que la ecuación (3.9) se mantiene pero $\mathtt{P}|_{-\prec^* \bar{\mathtt{e}}} \notin \mathbb{S}(\mathtt{G}|_{-\prec^* \mathtt{e}})$. Sin perder generalidad, asumamos que $\mathtt{e}$ es minimal, es decir, para todo $\mathtt{e}'$ tal que $\mathtt{e}' \prec^+ \bar{\mathtt{e}}$ tenemos

$$\mathtt{P}|_{-\prec^* \mathtt{e}'} \in \mathbb{S}(\mathtt{G}|_{-\prec^* \mathtt{e}'})$$

Por coherencia, tenemos

$$\mathtt{P}|_{-\prec^+ \bar{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^+ \bar{\mathtt{e}}})$$

Asumiendo $\lambda(\bar{\mathtt{e}}) = \langle \mathtt{op}, \mathtt{v}\rangle$, por totalidad se cumple que

$$\exists \mathtt{G}_1, \mathtt{v}_1.\ \overline{\mathtt{G}_1} = \overline{\mathtt{G}}|_{-\prec^+ \bar{\mathtt{e}}} \wedge \overline{\mathtt{P}}|_{-\prec^+ \bar{\mathtt{e}}} \in \overline{\mathbb{S}((\mathtt{G}_1)^{\langle \mathtt{op}, \mathtt{v}_1\rangle}_{-\prec \bar{\mathtt{e}}})}\Big|_{-\prec^+ \bar{\mathtt{e}}}$$

Ahora, por saturación y coherencia tenemos que $\overline{\mathtt{P}}|_{-\prec^+\overline{\mathtt{e}}} \in \overline{\mathbb{S}(\mathtt{G}_1)}$, y por lo tanto por determinismo $\mathtt{G}_1 = \mathtt{G}|_{-\prec^+\overline{\mathtt{e}}}$, entonces se mantiene

$$\exists \mathtt{v}_1.\ \overline{\mathtt{P}}|_{-\prec^+\overline{\mathtt{e}}} \in \overline{\mathbb{S}((\mathtt{G}|_{-\prec^+\overline{\mathtt{e}}})^{\langle \mathtt{op},\mathtt{v}_1\rangle}_{-\prec\overline{\mathtt{e}}})}\Big|_{-\prec^+\overline{\mathtt{e}}}$$

Ahora, por saturación y (3.9)

$$\exists \mathtt{G}_2.\ (\overline{\mathtt{G}}|_{-\prec^+\overline{\mathtt{e}}})\Big|_{-\prec\overline{\mathtt{e}}} = \overline{\mathtt{G}_2} \ \wedge\ \overline{\mathtt{P}}|_{-\prec\overline{\mathtt{e}}} \in \overline{\mathbb{S}((\mathtt{G}_2)^{\langle \mathtt{op},\mathtt{v}_1\rangle})}\Big|_{-\prec\overline{\mathtt{e}}}$$

Sin embargo, aplicando (3.9) a $\mathtt{G}$ dice que

$$\exists \mathtt{G}_3.\ \overline{\mathtt{G}}|_{-\prec\overline{\mathtt{e}}} = \overline{\mathtt{G}_3} \ \wedge\ \overline{\mathtt{P}}|_{-\prec\overline{\mathtt{e}}} \in \overline{\mathbb{S}((\mathtt{G}_3)^{\langle \mathtt{op},\mathtt{v}\rangle})}\Big|_{-\prec\overline{\mathtt{e}}}$$

y por saturación y determinismo $(\mathtt{G}_2)^{\langle \mathtt{op},\mathtt{v}_1\rangle} = (\mathtt{G}_3)^{\langle \mathtt{op},\mathtt{v}\rangle}$, entonces $\mathtt{v} = \mathtt{v}_1$.

En consecuencia, tenemos

$$\overline{\mathtt{P}}|_{-\prec^+\overline{\mathtt{e}}} \in \overline{\mathbb{S}(\mathtt{G}|_{-\prec^*\overline{\mathtt{e}}})}\Big|_{-\prec^+\overline{\mathtt{e}}}$$

y por saturación se cumple que

$$\overline{\mathtt{P}}|_{-\prec^*\overline{\mathtt{e}}} \in \overline{\mathbb{S}(\mathtt{G}|_{-\prec^*\mathtt{e}})}$$

y por hipótesis

$$\mathtt{P}|_{-\prec^+\overline{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^+\mathtt{e}})$$

tenemos que

$$\mathtt{P}|_{-\prec^*\overline{\mathtt{e}}} \in \mathbb{S}(\mathtt{G}|_{-\prec^*\mathtt{e}})$$

el cual contradice la hipótesis. □

Los siguientes dos resultados muestran que los RDTs son una clase particular de especificaciones, y por lo tanto, proporcionan una caracterización completamente abstracta de la definición clásica que ofrecen los RDTs.

Teorema 1. *Para cada especificación $\mathbb{S}$ que es coherente, localmente funcional y saturada, $\mathbb{S} = \mathcal{S}(\mathbb{F}(\mathbb{S}))$. $\mathcal{F} = \mathbb{F}(\mathcal{S}(\mathcal{F}))$, $\mathbb{F}(\mathbb{S}) = \mathcal{F}$, $\mathcal{S}(\mathcal{F}) = \mathbb{S}$*

Demostración. Tenemos que probar que para cualquier $\mathtt{G}$ se mantiene $\mathbb{S}(\mathtt{G}) = \mathcal{S}(\mathbb{F}(\mathbb{S}))(\mathtt{G})$.

$$\begin{aligned}
\mathtt{P} \in \mathcal{S}(\mathbb{F}(\mathbb{S}))(\mathtt{G}) &\overset{\text{Def. 3.4.9}}{\equiv} \mathtt{P} \in \textsc{prval}(\mathbb{F}(\mathbb{S}), \mathtt{G}) \\
&\overset{\text{Def. 3.4.2}}{\equiv} \textsc{rval}(\mathbb{F}(\mathbb{S}), \mathtt{G}, \mathtt{P}) \\
&\overset{\text{Def. 3.4.2}}{\equiv} \forall \mathtt{e} \in \mathtt{E}_\mathtt{G}.\ \lambda(\mathtt{e}) = \langle \mathtt{op}, \mathtt{v}\rangle \ \textit{implica} \\
&\qquad\qquad \mathbb{F}(\mathbb{S})(\mathtt{op}, \overline{\mathtt{G}}|_{-\prec\mathtt{e}}, \overline{\mathtt{P}}|_{-\prec\mathtt{e}}) = \mathtt{v} \\
&\overset{\text{Def. 3.4.10}}{\equiv} \forall \mathtt{e} \in \mathtt{E}_\mathtt{G}.\ \lambda(\mathtt{e}) = \langle \mathtt{op}, \mathtt{v}\rangle \ \textit{implica} \\
&\qquad\qquad \exists \mathtt{G}_1.\ \overline{\mathtt{G}}|_{-\prec\mathtt{e}} = \overline{\mathtt{G}_1} \ \wedge\ \overline{\mathtt{P}}|_{-\prec\mathtt{e}} \in \overline{\mathbb{S}(\mathtt{G}_1{}^{\langle \mathtt{op},\mathtt{v}\rangle})}\Big|_{\mathtt{E}_{\mathtt{G}_1}} \\
&\overset{\text{Lem. 7}}{\equiv} \forall \mathtt{e} \in \mathtt{E}_\mathtt{G}.\ \mathtt{P}|_{-\prec^*\mathtt{e}} \in \mathbb{S}(\mathtt{G}|_{-\prec^*\mathtt{e}}) \\
&\overset{\text{Def. 3.1.5}}{\equiv} \mathtt{P} \in \textstyle\bigotimes_{\mathtt{e}\in\mathtt{E}_\mathtt{G}} \mathbb{S}(\mathtt{G}|_{-\prec^*\mathtt{e}}) \\
&\overset{\text{coh}}{\equiv} \mathtt{P} \in \mathbb{S}(\mathtt{G})
\end{aligned}$$

□

Teorema 2. *Para cada* RDT $\mathcal{F}$, $\mathcal{F} = \mathbb{F}(\mathcal{S}(\mathcal{F}))$.

Demostración. Probaremos que $\mathcal{F}(\mathtt{op}, \overline{\mathtt{G}}, \overline{\mathtt{P}}) = \mathbb{F}(\mathcal{S}(\mathcal{F}))(\mathtt{op}, \overline{\mathtt{G}}, \overline{\mathtt{P}})$ para cualquier $\langle \overline{\mathtt{G}}, \overline{\mathtt{P}} \rangle$ y $\mathtt{op}$.

$$
\begin{array}{ll}
& \mathbb{F}(\mathcal{S}(\mathcal{F}))(\mathtt{op}, \overline{\mathtt{G}}, \overline{\mathtt{P}}) = \mathtt{v} \\
\overset{\text{Def. 3.4.10}}{\equiv} & \exists \langle \mathtt{G}_1, \mathtt{P}_1 \rangle.\ \overline{\mathtt{G}} = \overline{\mathtt{G}_1} \wedge \overline{\mathtt{P}} = \overline{\mathtt{P}_1} \wedge \mathtt{P}_1 \in \mathcal{S}(\mathcal{F})(\mathtt{G}_1^{\langle \mathtt{op}, \mathtt{v} \rangle})\Big|_{\mathtt{E}_{\mathtt{G}_1}} \\
\overset{\text{sat}}{\Longrightarrow} & \exists \langle \mathtt{G}_1,\ \mathtt{P}_1 \rangle.\ \overline{\mathtt{G}_1} = \overline{\mathtt{G}} \wedge \overline{\mathtt{P}} = \overline{\mathtt{P}_1} \wedge \mathtt{P}_1^{\langle \mathtt{op}, \mathtt{v} \rangle} \in \mathcal{S}(\mathcal{F})(\mathtt{G}_1^{\langle \mathtt{op}, \mathtt{v} \rangle}) \\
\overset{\text{Def. 3.4.10}}{\equiv} & \ldots \wedge \mathtt{P}_1^{\langle \mathtt{op}, \mathtt{v} \rangle} \in \mathrm{PRVAL}(\mathcal{F}, \mathtt{G}_1^{\langle \mathtt{op}, \mathtt{v} \rangle}) \\
\overset{\text{Def. 3.4.2}}{\equiv} & \ldots \wedge \mathrm{RVAL}(\mathcal{F}, \mathtt{G}_1^{\langle \mathtt{op}, \mathtt{v} \rangle}, \mathtt{P}_1^{\langle \mathtt{op}, \mathtt{v} \rangle}) \\
\overset{\text{Def. 3.4.2}}{\equiv} & \ldots \wedge \mathrm{RVAL}(\mathcal{F}, \mathtt{G}_1, \mathtt{P}_1) \wedge \mathcal{F}(\mathtt{op}, \overline{(\mathtt{G}_1)^{\langle \mathtt{op}, \mathtt{v} \rangle}\Big|_{-\prec \top}}, \overline{(\mathtt{P}_1)^{\langle \mathtt{op}, \mathtt{v} \rangle}\Big|_{-\prec \top}}) = \mathtt{v} \\
\equiv & \ldots \wedge \mathcal{F}(\mathtt{op}, \overline{\mathtt{G}_1}, \overline{\mathtt{P}_1}) = \mathtt{v} \\
\Longrightarrow & \mathcal{F}(\mathtt{op}, \overline{\mathtt{G}}, \overline{\mathtt{P}}) = \mathtt{v}
\end{array}
$$

Entonces el resultado se mantiene ya que $\mathcal{F}$ y $\mathbb{F}(\mathcal{S}(\mathcal{F}))$ son funciones totales. □

La caracterización anterior implica que hay tipos de datos que no pueden especificarse operacionalmente con los RDTs. Considere por ejemplo las especificaciones de los RDTs `Registro Genérico` y un `Set`, tal que se presentan en Ej. 3.2.3 y Ej. 3.3.2. Como se señaló en Ej. 3.4.6, estas especificaciones no son determinísticas y, por lo tanto, no se pueden expresar como RDTs.

Capítulo 4

Especificaciones como funtores

En este capítulo desarrollamos y presentamos una caracterización categorial para especificaciones funcionales presentadas en el capítulo anterior. Para esto, vamos a definir una categoría para: (i) las configuraciones, llamada **PIDag**$(\mathcal{L})$, y (ii) para los conjuntos de arbitraciones, llamada **SPath**$(\mathcal{L})$. Luego, una especificación que cumple la propiedad de ser *coherente* será un funtor que asigna objetos y morfismos de **PIDag**$(\mathcal{L})$ en **SPath**$(\mathcal{L})$. Intuitivamente, vamos a capturar cómo la configuración de un sistema evoluciona a partir de ejecutar una operación o al sincronizarse una réplica con otra y esto lo explicaremos con un nivel de abstracción más alto. Cuando una configuración crece, el conjunto de arbitraciones que explican una configuración también crece, y esto estará dado a partir de la existencia de morfismos tanto en **PIDag**$(\mathcal{L})$ como en **SPath**$(\mathcal{L})$.

Concretamente, en la categoría **PIDag**$(\mathcal{L})$ los objetos son grafos acíclicos, dirigidos y etiquetados y *pr-morfismos* inyectivos, es decir, morfismos que preservan etiquetas y que reflejan los arcos dirigidos, y la categoría **SPath**$(\mathcal{L})$ es un conjuntos de ordenes totales etiquetados y *ps-morfismos*, es decir, morfismos entre conjuntos de arbitraciones. Un ps-morfimo $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ de un conjunto de arbitraciones $\mathcal{X}_1$ a un conjunto de arbitraciones $\mathcal{X}_2$ establece que cualquier orden en $\mathcal{X}_2$ puede ser obtenido extendiendo algún orden total de $\mathcal{X}_1$. Informalmente hablando, una especificación coherente habla de aquellos RDTs, tales que, los arbitrajes asociados a una configuración se pueden obtener extendiendo las arbitraciones asociadas a configuraciones "más pequeñas" y se corresponden a lo que se llama RVAL *Consistente* para RDTs en Def. 3.4.2. Finalmente, al establecer la biyección entre funtores y especificaciones, observamos la preservación de colímites y pullbacks binarios.

4.1. Categorías de base

Usaremos el par $\langle \mathcal{E}, \rho \rangle$ para denotar una relación ρ sobre un conjunto finito $\mathcal{E}$, con el fin de tener siempre a $\mathcal{E}$ de forma explícita. Recordamos además que usamos $\lfloor \mathtt{e} \rfloor$ cuando queremos describir el conjunto *downward-closed* más chico que incluye $\mathtt{e} \in \mathcal{E}$ (ver Def. 2.1.1).

Definición 4.1.1 (Morfismos (Relaciones Binarias))**.** *Un morfismo (sobre relaciones binarias)* $\mathtt{f} : \langle \mathcal{E}, \rho \rangle \to \langle T, \gamma \rangle$ *es una función* $\mathtt{f} : \mathcal{E} \to T$ *tal que*

$$\forall \mathtt{e}, \mathtt{e}' \in \mathcal{E}.\ \mathtt{e}\ \rho\ \mathtt{e}' \text{ implica } \mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$$

Un morfismo $\mathtt{f} : \langle \mathcal{E}, \rho \rangle \to \langle T, \gamma \rangle$ *es* past-reflecting *(abreviado como pr-morfismos) si*

$$\forall \mathtt{e} \in \mathcal{E}, \mathtt{t} \in T.\ \mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e}) \text{ implica } \exists \mathtt{e}' \in \mathcal{E}.\ \mathtt{e}'\ \rho\ \mathtt{e} \wedge \mathtt{t} = \mathtt{f}(\mathtt{e}')$$

Notar que tanto los morfismos sobre relaciones como los *pr-morfismos* son cerrados respecto de la composición de morfismos. Denotamos como **Bin** la categoría de relaciones y sus morfismos y **PBin** la sub-categoría de pr-morfismos.

Lema 8 (Caracterización de pr-morfismos)**.** *Sea* $\mathtt{f} : \langle \mathcal{E}, \rho \rangle \to \langle T, \gamma \rangle$ *un morfismo. Si*

1. $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ *implica* $\mathtt{e}\ \rho\ \mathtt{e}'$*, y*
2. $\bigcup_{\mathtt{e} \in \mathcal{E}} \mathtt{f}(\mathtt{e})$ *es* downward closed*,*

entonces $\mathtt{f}$ *es un pr-morfismo. Además, si* $\mathtt{f}$ *es inyectivo, entonces vale la recíproca.*

Demostración.

$\Rightarrow$) Tomemos $\mathtt{e} \in \mathcal{E}$ y $\mathtt{t} \in T$. Si $\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e})$, entonces por (2), existe $\mathtt{e}' \in \mathcal{E}$ tal que $\mathtt{t} = \mathtt{f}(\mathtt{e}')$. Luego, por (1), $\mathtt{f}(\mathtt{e}')\ \gamma\ \mathtt{f}(\mathtt{e})$ implica $\mathtt{e}'\ \rho\ \mathtt{e}$.

$\Leftarrow$) Por definición de pr-morfismo $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ implica $\exists \bar{\mathtt{e}} \in \mathcal{E}.\ \bar{\mathtt{e}}\ \rho\ \mathtt{e}' \wedge \mathtt{f}(\mathtt{e}) = \mathtt{f}(\bar{\mathtt{e}})$. Dado que $\mathtt{f}$ es inyectiva, $\bar{\mathtt{e}} = \mathtt{e}$ y en consecuencia $\mathtt{e}\ \rho\ \mathtt{e}'$. Entonces, sea $\mathcal{T} = \bigcup_{\mathtt{e} \in \mathcal{E}} \mathtt{f}(\mathtt{e})$. Vamos a mostrar que

$$\forall \mathtt{t} \in T, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\gamma\ \mathtt{t}' \text{ implica } \mathtt{t} \in \mathcal{T}$$

La prueba sigue por absurdo. Asumir que $\exists \mathtt{t} \in T, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\ \gamma\ \mathtt{t}' \wedge \mathtt{t} \notin \mathcal{T}$. Por definición de $\mathcal{T}, \exists \mathtt{e} \in \mathcal{E}$ tal que $\mathtt{f}(\mathtt{e}) = \mathtt{t}'$. Dado que $\mathtt{f}$ es un pr-morfismo, entonces

$$\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e}) \text{ implica } \exists \mathtt{e}' \in \mathcal{E}.\ \mathtt{e}'\ \rho\ \mathtt{e} \wedge \mathtt{t} = \mathtt{f}(\mathtt{e}')$$

Por lo tanto, $\mathtt{t} = \mathtt{f}(\mathtt{e}') \in \mathcal{T}$, el cual contradice la suposición $\mathtt{t} \notin \mathcal{T}$. □

Como **Bin** tiene límites y colímites finitos, los cuales se calculan puntualmente como en la categoría de **Set**. La estructura interesante, y que estudiaremos a continuación es **PBin**.

Proposición 1 (Propiedades de **PBin**)**.** *El funtor de inclusión* **PBin** $\to$ **Bin** *refleja colímites finitos y pullbacks binarios.*

En otras palabras, dado que **Bin** tiene límites y colímites finitos, entonces existen en **PBin** los pullbacks binarios y los colímites finitos. Estos pueden calcularse como en **Bin**. Sin embargo, en **PBin** no existe objeto final, ya que, a diferencia de **Bin** no existen pr-morfismos desde cualquier relación al singleton.

Monos en **Bin** son sólo morfismos cuya función subyacente es inyectiva, y similarmente en **PBin**, entonces el funtor de inclusión los preserva (y refleja).

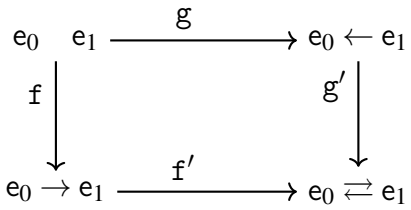


Figura 4.1: Pushout en *Dag* no existe

Lema 9 (Monos bajo pushouts)**.** *Los pushouts en* **Bin** *(y en consecuencia en* **PBin***) preservan monos.*

A continuación se introducen las nociones de relaciones etiquetadas. Considere un *forgetful funtor* $\mathtt{U_r} : \mathbf{Bin} \to \mathbf{Set}$ y $\mathtt{U_p} : \mathbf{PBin} \to \mathbf{Set}$, este último se puede descomponer como dos funtores. uno la inclusión $\mathbf{PBin} \to \mathbf{Bin}$ y el otro $\mathbf{Bin} \to \mathbf{Set}$. Dado un conjunto de etiquetas $\mathcal{L}$, considere las categorías coma $\mathbf{Bin}(\mathcal{L}) = \mathtt{U_r} \downarrow \mathcal{L}$ y $\mathbf{PBin}(\mathcal{L}) = \mathtt{U_p} \downarrow \mathcal{L}$, por lo tanto, los colímites finitos y pullbacks binarios siempre existen y son esencialmente computados como en **Bin**.

Explícitamente, un objeto en $\mathtt{U_r} \downarrow \mathcal{L}$ es una tripla $(\mathcal{E}, \rho, \lambda)$ donde $\lambda : \mathcal{E} \to \mathcal{L}$ es la función de etiquetado. Un morfismo que preserva etiquetas $(\mathcal{E}, \rho, \lambda) \to (\mathcal{E}', \rho', \lambda')$ es un morfismo $\mathtt{f} : (\mathcal{E}, \rho) \to (\mathcal{E}', \rho')$ tal que $\forall \mathtt{s} \in \mathcal{E}.\ \lambda(\mathtt{s}) = \lambda'(\mathtt{f}(\mathtt{s}))$. Además, colímites finitos y pullbacks binarios existen y son calculados como en **Bin**. Similarmente, propiedades para objetos.y los objetos $\mathtt{U_p} \downarrow \mathcal{L}$ se cumplen.

4.2. Categoría para Configuraciones

A continuación se introduce la categoría de las configuraciones utilizada para definir la sintaxis y la semántica de las especificaciones.

Definición 4.2.1 (**PDag**)**. PDag** *es la subcategoría completa de* **PBin** *cuyos objetos son grafos dirigidos acíclicos y los morfismos son pr-morfismos.*

En otras palabras, los objetos de **PDag** son relaciones cuya clausura transitiva son órdenes parciales *estrictos*.

Observación 5. *La subcategoría completa de* **Bin** *cuyos objetos son grafos dirigidos acíclicos no son adecuados para nuestro propósito, ya que por ejemplo, no admite pushouts, ni siquiera cuando los morfismos son monos. En Fig. 4.2 mostramos que aunque los morfismos son monos, el pushout no se preserva ya que se puede al generarse un ciclo se rompe la condición de ser un grafo acíclico. Por lo tanto, trabajaremos con pr-morfismos.*

Proposición 2 (Propiedades de **PDag**)**.** *El funtor de inclusión* **PDag** $\to$ **PBin** *refleja colímites finitos y pullbacks binarios.*

Ahora reducimos aún más las flechas, de **PDag**($\mathcal{L}$) a *monic*. Intuitivamente, solo nos interesa lo que sucede si agregamos más eventos a las relaciones de visibilidad. Por lo tanto, consideramos la subcategoría **PIDag**($\mathcal{L}$) de grafos acíclicos, dirigidos y monic pr-morfismos.

Lema 10 (Monos bajo pushouts, 2)**.** *Pushouts en* **PDag**($\mathcal{L}$) *preservan monos.*

4.3. Categoría para Conjunto de Arbitraciones

Para estudiar la categoría cuyos objetos son conjuntos de arbitraciones, introducimos la categoría para *caminos* o arbitraciones, es decir, relaciones que son órdenes totales.

Definición 4.3.1 (**Path**)**. Path** *es la full sub-categoría de* **Bin** *cuyos objetos son caminos.*

Notar que definir **Path** con pr-morfismos sería muy restrictivo, ya que existe un pr-morfismo entre dos caminos si y solo si un camino es un prefijo de otro.

Proposición 3 (Propiedades de **Path**)**.** *El funtor de inclusión* **Path** $\to$ **Bin** *refleja colímites finitos.*

Como con relaciones, vamos a considerar las *categorías coma* con el fin de capturar caminos y grafos etiquetados. En particular, vamos a usar los *forgetful funtors* $\mathtt{U}_{\mathtt{rp}}$: **Path** $\to$ **Set** y $\mathtt{U}_{\mathtt{pd}}$: **PDag** $\to$ **Set**: para el conjunto de etiquetas $\mathcal{L}$. Vamos a denotar **PDag**$(\mathcal{L}) = \mathtt{U}_{\mathtt{rp}} \downarrow \mathcal{L}$ y **Path**$(\mathcal{L}) = \mathtt{U}_{\mathtt{pd}} \downarrow \mathcal{L}$. Una vez más, colímites finitos y pullbacks binarios siempre existen y son esencialmente capturados como en **Bin**.

A continuación estudiamos algunos operadores para trabajar con la categoría de conjuntos de caminos.

Definición 4.3.2 (Saturación de Camino)**.** *Sea* $\mathtt{P}$ *un camino y* $\mathtt{f} : (\mathtt{E_P}, \lambda_{\mathtt{P}}) \to (\mathtt{E}, \lambda)$ *una función que preserva etiquetas. La saturación de* $\mathtt{P}$ *según* $\mathtt{f}$ *se define como*

$$\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathtt{E}, \lambda) \text{ y } \mathtt{f} \text{ induce un morfismo } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

La noción de saturación se extiende a un conjunto de caminos $\mathcal{X} \subseteq \mathbb{P}(\mathtt{E}, \lambda)$ *como*

$$\bigcup_{\mathtt{P} \in \mathcal{X}} \mathtt{sat}(\mathtt{P}, \mathtt{f})$$

Notar que si $\mathtt{f}$ no fuera inyectiva $\mathtt{f}$, podría pasar que $\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \emptyset$.

Ejemplo 4.3.1. *Considere la función inyecta y que preserva etiquetas* $\mathtt{f}$ *con dominio* $\{\langle \mathtt{wr}(1), \mathtt{ok} \rangle, \langle \mathtt{wr}(2), \mathtt{ok} \rangle\}$ *y codominio* $\{\langle \mathtt{wr}(1), \mathtt{ok} \rangle, \langle \mathtt{wr}(2), \mathtt{ok} \rangle, \langle \mathtt{rd}, 2 \rangle\}$. *Entonces, tenemos*

$$\mathtt{sat}\left(\left\{\begin{array}{c}\langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle\end{array}\right\}, \mathtt{f}\right) = \left\{\begin{array}{ccc}\langle \mathtt{wr}(1), \mathtt{ok} \rangle & \langle \mathtt{wr}(1), \mathtt{ok} \rangle & \langle \mathtt{rd}, 2 \rangle \\ | & | & | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle & \langle \mathtt{rd}, 2 \rangle & \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | & | & | \\ \langle \mathtt{rd}, 2 \rangle & \langle \mathtt{wr}(2), \mathtt{ok} \rangle & \langle \mathtt{wr}(2), \mathtt{ok} \rangle\end{array}\right\}$$

Intuitivamente, la saturación agrega $\langle \mathtt{rd}, 2 \rangle$ *a el camino original de todas las formas posibles, preservando el orden original de los eventos.*

Definición 4.3.3 (Retracción de Camino)**.** *Sea* $\mathtt{Q}$ *un camino y* $\mathtt{f} : \mathtt{E} \to \mathtt{E_Q}$ *una función. La retracción de* $\mathtt{Q}$ *según* $\mathtt{f}$ *es definida como*

$$\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\mathtt{E}, \lambda) \text{ y } \mathtt{f} \text{ induce un morfismo } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

La noción de retracción es extendida al conjunto de caminos $\mathcal{X} \subseteq \mathbb{P}(\mathtt{E}, \lambda)$ *como*

$$\bigcup_{\mathtt{Q} \in \mathcal{X}} \mathtt{ret}(\mathtt{Q}, \mathtt{f})$$

Notar que λ está caracterizada tomando la restricción de $\lambda_{\mathtt{Q}}$ sobre el mapping que esta hace. Además si $\mathtt{f}$ es inyectiva, $\mathtt{ret}(\mathtt{Q}, \mathtt{f})$ es un singleton, y si $\mathtt{f}$ es una inclusión, entonces $\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \mathtt{Q}|_{\mathtt{E}}$.

Ahora podemos considerar la relación que existe entre saturación y retracción de caminos.

Lema 11. *Sea* $\mathcal{X}_1 \subseteq \mathbb{P}(\mathtt{E}_1, \lambda_1)$ *un conjunto de caminos y* $\mathtt{f} : (\mathtt{E}_1, \lambda_1) \to (\mathtt{E}_2, \lambda_2)$ *una función que preserva etiquetas. Entonces* $\mathcal{X}_1 \subseteq \mathtt{ret}(\mathtt{sat}(\mathcal{X}_1, \mathtt{f}), \mathtt{f})$*. Si* $\mathtt{f}$ *es inyectiva, entonces* $\mathcal{X}_1 = \mathtt{ret}(\mathtt{sat}(\mathcal{X}_1, \mathtt{f}), \mathtt{f})$.

Lema 12. *Sea* $\mathcal{X}_2 \subseteq \mathbb{P}(\mathtt{E}_2, \lambda_2)$ *un conjunto de caminos y* $\mathtt{f} : \mathtt{E}_1 \to \mathtt{E}_2$ *una función. Entonces* $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathtt{ret}(\mathcal{X}_2, \mathtt{f}), \mathtt{f})$.

Diremos que una función inyectiva $\mathtt{f}$ es *saturada* con respecto a $\mathcal{X}_2$ si se cumple $\mathcal{X}_2 = \mathtt{sat}(\mathtt{ret}(\mathcal{X}_2, \mathtt{f}), \mathtt{f})$.

Ejemplo 4.3.2. *Considere los conjuntos de caminos* $\mathcal{X}_1$ *y* $\mathcal{X}_2$ *y el pr-morfismo* $\mathtt{f}$ *definido a continuación*

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \end{array} \right\} \qquad \mathcal{X}_2 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{rd}, 2 \rangle \end{array} \right\} \qquad \mathtt{f}: \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \end{array} \to \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{rd}, 2 \rangle \end{array}$$

la función subyacente $\mathtt{f}$ *(definida en Ejemplo 4.3.1)* no *es saturada con respecto a* $\mathcal{X}_2$ *porque*

$$\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{rd}, 2 \rangle \end{array} \right\} \neq \mathtt{sat}(\mathtt{ret}(\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{rd}, 2 \rangle \end{array} \right\}, \mathtt{f}), \mathtt{f}) = \mathtt{sat}(\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \end{array} \right\}, \mathtt{f})$$

Ahora podemos explotar el concepto de saturación para obtener la definición de la categoría de conjunto de arbitraciones.

Definición 4.3.4 (ps-morfismo)**.** *Sea $\mathcal{X}_1 \subseteq \mathbb{P}(\mathtt{E}_1, \lambda_1, \lambda)$ y $\mathcal{X}_2 \subseteq \mathbb{P}(\mathtt{E}_2, \lambda_2, \lambda)$ un conjunto de caminos. Un path-set morfismo (abreviado como ps-morfismo) $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ es una función $\mathtt{f} : (\mathtt{E}_1, \lambda_1) \to (\mathtt{E}_2, \lambda_2)$ que preserva etiquetas y además satisface $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$.*

Intuitivamente, hay un ps-morfismo desde el conjunto de caminos $\mathcal{X}_1$ al conjunto de caminos $\mathcal{X}_2$ si cualquier camino en $\mathcal{X}_2$ se puede obtener agregando eventos a algún camino en $\mathcal{X}_1$. Esta noción captura la idea de que las arbitraciones asociadas a las configuraciones más grandes se obtienen extendendiendo caminos asociados a configuraciones mas pequeñas (o subconfiguraciones).

Ejemplo 4.3.3. *Considere los siguientes tres conjuntos y la función* f *del ejemplo 4.3.1*

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle \end{array} \right\} \quad \mathcal{X}_2 = \left\{ \begin{array}{cc} \langle \mathtt{wr}(1), \mathtt{ok} \rangle & \langle \mathtt{wr}(1), \mathtt{ok} \rangle \\ | & | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle , & \langle \mathtt{rd}, 2 \rangle \\ | & | \\ \langle \mathtt{rd}, 2 \rangle & \langle \mathtt{wr}(2), \mathtt{ok} \rangle \end{array} \right\} \quad \mathcal{X}_3 = \left\{ \begin{array}{cc} \langle \mathtt{wr}(1), \mathtt{ok} \rangle & \langle \mathtt{wr}(2), \mathtt{ok} \rangle \\ | & | \\ \langle \mathtt{wr}(2), \mathtt{ok} \rangle , & \langle \mathtt{rd}, 2 \rangle \\ | & | \\ \langle \mathtt{rd}, 2 \rangle & \langle \mathtt{wr}(1), \mathtt{ok} \rangle \end{array} \right\}$$

Entonces, f *induce un ps-morfismo $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ porque $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$ (este último se muestra en 4.3.1). Por el contrario, no hay un ps-morfismo de $\mathcal{X}_1$ a $\mathcal{X}_3$ ya que el camino de más a la derecha de $\mathcal{X}_3$ no puede obtenerse extendiendo algún camino de $\mathcal{X}_1$ con un evento etiquetado por $\langle \mathtt{rd}, 2 \rangle$.*

Definición 4.3.5 (**SPath**)**. SPath**$(\mathcal{L})$ *es la categoría cuyos objetos son conjuntos de caminos etiquetados sobre $\mathcal{L}$ y las flechas son ps-morfismos.*

Proposición 4 (Propiedades de **SPath**)**.** *La categoría* **SPath**$(\mathcal{L})$ *tiene colímites finitos sobre monos y pullbacks binarios.*

Demostración. (Estricto) objeto inicial. El (único) objeto inicial es $\langle \emptyset, \{\varepsilon\}, \emptyset \rangle$, con $\varepsilon \in \mathbb{P}(\emptyset, \emptyset)$ el camino vacío. Sea $\mathcal{X} \subseteq \mathbb{P}(\mathtt{E}, \lambda)$ y $! \colon \emptyset \to \mathtt{E}$ la única función con dominio $\emptyset$ y codominio E. Tenemos una función $! \colon (\emptyset, \emptyset) \to (\mathtt{E}, \lambda)$ tal que $\mathcal{X} \subseteq \mathtt{sat}(\{\varepsilon\}, !) = \mathbb{P}(\mathtt{E}, \lambda, \lambda)$.

Binary Pushouts. Sean $\mathcal{X}, \mathcal{X}_1$, y $\mathcal{X}_2$ conjuntos de caminos y $\mathtt{f}_i : \mathcal{X} \to \mathcal{X}_i$ ps-morfismos. Considere las funciones subyacentes $\mathtt{f}_i : \mathtt{E} \to \mathtt{E}_i$ y su pushout $\mathtt{f}'_i : \mathtt{E}_i \to \mathtt{E}_1 +_{\mathtt{E}} \mathtt{E}_2$ en la categoría de conjuntos: este induce un pushout $\mathtt{f}'_i : \mathcal{X}_i \to \mathtt{sat}(\mathcal{X}_1, \mathtt{f}'_1) \cap \mathtt{sat}(\mathcal{X}_2, \mathtt{f}'_2)$ en **SPath**$(\mathcal{L})$.

Pullbacks binarios. Sean $\mathcal{X}, \mathcal{X}_1$, y $\mathcal{X}_2$ conjuntos de caminos y $\mathtt{f}_i : \mathcal{X}_i \to \mathcal{X}$ ps-morfismos. Considere la función subyacente $\mathtt{f}_i : \mathtt{E}_i \to \mathtt{E}$ y su pullback $\mathtt{f}'_i : \mathtt{E}_1 \times_{\mathtt{E}} \mathtt{E}_2 \to \mathtt{E}$ en la categoría de conjuntos: este induce un pullback $\mathtt{f}'_i : \mathtt{ret}(\mathcal{X}_1, \mathtt{f}'_1) \cup \mathtt{ret}(\mathcal{X}_2, \mathtt{f}'_2) \to \mathcal{X}_i$ en **SPath**$(\mathcal{L})$. □

La caracterización anterior de los pushouts vale por el hecho de que consideramos las funciones inyectivas. La siguiente resultado instancia la caracterización anterior para el caso en que las funciones subyacente a los ps-morfismos sean inclusiones y justifica la operación de producto de caminos introducida en Def. 3.1.5.

Lema 13. *Sea $\mathtt{f}_i : \mathcal{X} \to \mathcal{X}_i$ ps-morfismos tal que las funciones subyacentes $\mathtt{f}_i : \mathtt{E} \to \mathtt{E}_i$ son inclusiones y $\mathtt{E} = \mathtt{E}_1 \cap \mathtt{E}_2$. Entonces su pushout está dado por $\mathtt{f}'_i : \mathcal{X}_i \to \mathcal{X}_1 \otimes \mathcal{X}_2$.*

Demostración. Por definición de producto,

$$\mathcal{X}_1 \otimes \mathcal{X}_2 = \{\mathtt{P} \mid \mathtt{P} \text{ es un camino sobre } \bigcup_i \mathtt{E}_i \text{ y } \mathtt{P}|_{\mathtt{E}_i} \in \mathcal{X}_i\}$$

Notar también que

$$\mathtt{sat}(\mathcal{X}_i, \mathtt{f}'_{\mathtt{i}}) = \bigcup_{\mathtt{Q} \in \mathcal{X}_i} \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\bigcup_i \mathtt{E}_i, \bigcup_i \lambda_i, \lambda)$$

y $\mathtt{f}'_1$ induce un morfismo sobre un camino $\mathtt{f}'_{\mathtt{i}} : \mathtt{P} \to \mathtt{Q}$. Dado que $\mathtt{f}'_{\mathtt{i}}$ es una inclusión, la última condición es igual a $\mathtt{P}|_{\mathtt{E}_i} = \mathtt{Q}$, y por lo tanto la propiedad se mantiene. □

Ejemplo 4.3.4. *Considere los siguientes ps-morfismos*

$$\left\{\begin{array}{c}\langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle\end{array}\right\} \leftarrow \left\{\begin{array}{c}\langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle\end{array}\right\} \rightarrow \left\{\begin{array}{cc}\langle \mathtt{wr}(1), \mathtt{ok}\rangle & \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle & \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{rd}, 1\rangle & \langle \mathtt{rd}, 1\rangle\end{array}\right\}$$

entonces, el pushout está dado por los siguientes ps-morfismos

$$\left\{\begin{array}{c}\langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle\end{array}\right\} \rightarrow \left\{\begin{array}{cc}\langle \mathtt{wr}(1), \mathtt{ok}\rangle & \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle & \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{rd}, 1\rangle & \langle \mathtt{rd}, 2\rangle \\ | & | \\ \langle \mathtt{rd}, 2\rangle & \langle \mathtt{rd}, 1\rangle\end{array}\right\} \leftarrow \left\{\begin{array}{cc}\langle \mathtt{wr}(1), \mathtt{ok}\rangle & \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle & \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | & | \\ \langle \mathtt{rd}, 1\rangle & \langle \mathtt{rd}, 1\rangle\end{array}\right\}$$

Una propiedad análoga se mantiene para pullbacks. Sea $\mathtt{f}_i : \mathcal{X}_i \to \mathcal{X}$ un ps-morfimo tal que las funciones subyacentes son inclusiones: el pullback está dado por $\mathtt{f}'_i : \bigcup_i \mathcal{X}_i|_{\mathtt{E}_1 \cap \mathtt{E}_2} \to \mathcal{X}_i$. En particular, el cuadrado a continuación es a la vez un pullback y un pushout.

$$\begin{array}{ccc} \bigcup_i \mathcal{X}_i|_{\mathtt{E}_1 \cap \mathtt{E}_2} & \hookrightarrow & \mathcal{X}_1 \\ \downarrow & & \downarrow \\ \mathcal{X}_2 & \hookrightarrow & \mathcal{X}_1 \otimes \mathcal{X}_2 \end{array}$$

4.4. Especificaciones como funtores

En esta sección mostraremos que las especificaciones coherentes inducen funtores entre la categoría de las configuraciones y la categoría de los conjuntos de caminos que preservar la estructura categorial relevante (soundness) y, a la inversa, que una cierta clase de funtores (básicamente, aquellos que conservan colímites finitos y pullbacks binarios) inducen especificaciones coherentes (completitud). Finalmente, demostraremos que estas funciones entre funtores y especificaciones son mutuamente inversas, estableciendo una correspondencia uno a uno (up-to isomorfismo).

4.4.1. Soundness

En esta sección daremos los resultados de *soundness*. Para esto, primero exigimos un requerimiento de consistencia mínimo: una especificación asigna configuraciones isomórficas a conjuntos de caminos isomórficos, a través de un mismo isomorfismo entre eventos. Esto es, si existe un isomorfismo en **PDag** de $\mathtt{G}_1$ a $\mathtt{G}_2$ con una biyección subyacente $\mathtt{f} : \mathtt{E}_{\mathtt{G}_1} \to \mathtt{E}_{\mathtt{G}_2}$, entonces para toda especificación $\mathbb{S}$ hay un isomorfismo en $\mathbf{SPath}(\mathcal{L})$ de $\mathbb{S}(\mathtt{G}_1)$ a $\mathbb{S}(\mathtt{G}_2)$ con la misma función subyacente.

Proposición 5 (funtores inducidos por especificaciones)**.** *Una especificación coherente* $\mathbb{S}$ *induce un funtor* $\mathbb{M}(\mathbb{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$.

Demostración. Sea $\mathtt{G}$ un LDAG, definimos $\mathbb{M}(\mathbb{S})(\mathtt{G})$ como $\mathbb{S}(\mathtt{G})$ y para $\mathtt{f} : \mathtt{G} \to \mathtt{G}'$ definimos $\mathbb{M}(\mathbb{S})(\mathtt{f})$ como el ps-morfismo con la función inyectiva subyacente $\mathtt{f}\colon (\mathtt{E}_{\mathtt{G}}, \lambda_{\mathtt{G}}) \hookrightarrow (\mathtt{E}_{\mathtt{G}'}, \lambda_{\mathtt{G}'})$.

La demostración consiste en mostrar que $\mathtt{f}$ es un ps-morfismo de $\mathbb{S}(\mathtt{G})$ en $\mathbb{S}(\mathtt{G}')$, es decir,

$$\mathbb{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathbb{S}(\mathtt{G}), \mathtt{f})$$

y, dado que estamos considerando especificaciones que preservan isomorfismos, ponemos nuestra atención al caso donde $\mathtt{f}$ es una inclusión. Luego, tenemos: (i) $\mathtt{f}$ es un pr-morfismo, (ii) $\bigcup_{\mathtt{e} \in \mathtt{E}_{\mathtt{G}}} \mathtt{f}(\mathtt{e})$ es *downward-closed* en $\mathtt{G}'$ y (iii) $\mathbb{S}$ es coherente, entonces vale Lem. 2, esto es $\mathbb{S}(\mathtt{G}')|_{\mathtt{E}_{\mathtt{G}}} \subseteq \mathbb{S}(\mathtt{G}'|_{\mathtt{E}_{\mathtt{G}}}) = \mathbb{S}(\mathtt{G})$.

Considerar un camino $\mathtt{P} \in \mathbb{S}(\mathtt{G}')$. Dado que $\mathtt{P}|_{\mathtt{E}_{\mathtt{G}}} \in \mathbb{S}(\mathtt{G})$, tenemos $\mathtt{P} \in \mathtt{sat}(\mathbb{S}(\mathtt{G}), \mathtt{f})$, ya que saturación agrega eventos de todas las maneras posibles. Por lo tanto, concluimos que $\mathbb{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathbb{S}(\mathtt{G}), \mathtt{f})$. □

Es un hecho bien conocido que la categoría de conjuntos y funciones inyectivas carece de pushouts. Lo mismo también vale para $\mathbf{PIDag}(\mathcal{L})$. Sin embargo, recordemos que pushouts en $\mathbf{PDag}(\mathcal{L})$ preservan monos (Lema 10). Entonces, a continuación decimos que un funtor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ conserva débilmente los pushouts finitos (y, de hecho, los colímites finitos) si cualquier cuadrado commuta en $\mathbf{PIDag}(\mathcal{L})$ que es un pushout (a través del funtor de inclusión) en $\mathbf{PDag}(\mathcal{L})$ se asigna por $\mathbb{F}$ a un pushout en $\mathbf{SPath}(\mathcal{L})$.

Teorema 3. *Sea* $\mathbb{S}$ *una especificación coherente. El funtor inducido* $\mathbb{M}(\mathbb{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *preserva débilmente colímites finitos y preserva pullbacks binarios.*

Demostración. El objeto inicial es fácil dado que vale por construcción. Para los pushouts y pullbacks: dado que $\mathbb{S}$ es coherente, es inmediato por Lema 13. □

4.4.2. Completitud

En continuación presentamos los resultados completitud de este capítulo, mostrando (algunas alternativas sobre) cómo obtener una especificación a partir de un funtor.

Teorema 4. *Sea* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *un funtor tal que* $\mathbb{F}(\mathtt{G}) \subseteq \mathbb{P}(\mathtt{E_G}, \lambda_\mathtt{G}, \lambda)$. *Si* $\mathbb{F}$ *preserva débilmente colímites finitos y preserva pullbacks binarios,* $\mathbb{F}$ *induce una especificación coherente* $\mathbb{S}(\mathbb{F})$.

Demostración. Sea $\mathbb{S}(\mathbb{F})(\mathtt{G}) = \mathbb{F}(\mathtt{G})$. Vamos a demostrar que $\mathbb{F}(\mathtt{G})$ es coherente. Considerar el siguiente pushout en $\mathbf{PDag}(\mathcal{L})$

$$\begin{array}{ccc} \mathtt{G}|_{\lfloor e_1 \rfloor \cap \lfloor e_2 \rfloor} & \hookrightarrow & \mathtt{G}|_{\lfloor e_2 \rfloor} \\ \downarrow & & \downarrow \\ \mathtt{G}|_{\lfloor e_1 \rfloor} & \hookrightarrow & \mathtt{G}|_{\lfloor e_1 \rfloor \cup \lfloor e_2 \rfloor} \end{array} \tag{4.1}$$

Como $\mathbb{F}$ preserva pullbacks, y en consecuencia sobre monos, y preserva débilmente pushouts, el diagrama 4.1 es asociado por el funtor $\mathbb{F}$ al siguiente pushout en $\mathbf{SPath}(\mathcal{L})$

$$\begin{array}{ccc} \mathbb{F}(\mathtt{G}|_{\lfloor e_1 \rfloor \cap \lfloor e_2 \rfloor}) & \hookrightarrow & \mathbb{F}(\mathtt{G}|_{\lfloor e_2 \rfloor}) \\ \downarrow & & \downarrow \\ \mathbb{F}(\mathtt{G}|_{\lfloor e_1 \rfloor}) & \hookrightarrow & \mathbb{F}(\mathtt{G}|_{\lfloor e_1 \rfloor \cup \lfloor e_2 \rfloor}) \end{array} \tag{4.2}$$

donde las funciones subyacente entre los eventos son inclusiones. Por Lem. 13 tenemos que

$$\mathbb{F}(\mathtt{G}|_{\lfloor e_1 \rfloor \cup \lfloor e_2 \rfloor}) \simeq \mathbb{F}(\mathtt{G}|_{\lfloor e_1 \rfloor}) \otimes \mathbb{F}(\mathtt{G}|_{\lfloor e_2 \rfloor})$$

ya que $\mathtt{G} = \mathtt{G}|_{\bigcup_{e \in \mathtt{E_G}} \lfloor e \rfloor}$. Por asociatividad de pushout obtenemos coherencia ya que

$$\mathbb{F}(\mathtt{G}) \simeq \bigotimes_{e \in \mathtt{E_G}} \mathbb{F}(\mathtt{G}|_{\lfloor e \rfloor})$$

□

Capítulo 5

Correctitud de Tipos de Datos Replicados

En el capítulo 3 hemos probado la correspondencia entre una noción alternativa sobre especificaciones y la descripción clásica para modelar tipos de datos replicados. En este capítulo, argumentamos que nuestra definición es lo suficientemente flexible para razonar sobre la corrección de posibles implementaciones en términos de la noción clásica de *simulación* (ver Def. 2.2.2).

Dado que la noción de simulación es operacional, necesitamos darles a las especificaciones una interpretación operacional. Para eso, en la primera parte del capítulo nos concentramos en mostrar como una especificación funcional naturalmente da lugar a un LTS. Luego, dada una definición operacional de la implementación de un RDT, propuesto en [8], aplicamos la noción de simulación débil para mostrar la correctitud de dicha implementación. La demostración consiste primero en mostrar la correctitud sobre una única réplica y luego nos ocupamos de probar que también la implementación es correcta cuando hay múltiples réplicas.

5.1. De especificación a LTS

Las especificaciones funcionales tienen una interpretación implícita que es operacional.

Definición 5.1.1 (LTS para una réplica abstracta)**.** *Sea* $\mathbb{S}$ *una especificación. Entonces, el LTS* para una-réplica abstracta $\mathcal{T}_{\mathbb{S}} = (\mathtt{S}, \mathcal{L}, \rightarrow)$ *está definido de forma tal que:*

- $\mathtt{S} \subseteq \mathbb{G}(\mathcal{L}) \times \mathbb{P}(\mathcal{L})$ *y* $\langle \mathtt{G}, \mathtt{P} \rangle \in \mathcal{P}$ *si* $\mathtt{P} \in \mathbb{S}(\mathtt{G})$*;*
- $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}', \mathtt{P}' \rangle$ *es una transición siempre que* $\mathtt{G}' = \mathtt{G}^{\ell}$ *y* $\mathtt{P}'|_{\mathtt{E_G}} = \mathtt{P}$.

Un par $\langle \mathtt{G}, \mathtt{P} \rangle$ tal que $\mathtt{P} \in \mathbb{S}(\mathtt{G})$ abstractamente representa una computación admisible de acuerdo a $\mathbb{S}$. Más aún, $\mathbb{S}$ permite extender esta computación con un evento etiquetado

por ℓ, siempre que una especificación nos permita extender a $\mathtt{G}$ y a $\mathtt{P}$ con un evento fresco cuya etiqueta es ℓ.

Nuestro próximo paso es mostrar que una especificación puede ser equipada con la noción de composición paralela.

Escribimos $\langle \mathtt{G}_1 \sqcup \ldots \sqcup \mathtt{G}_n, \mathtt{P} \rangle$ para un estado replicado que consiste de *n* réplicas *compatibles* (de acuerdo a Def. 3.1.3), con $\langle \mathtt{G}_i, \mathtt{P}|_{\mathtt{E}_{\mathtt{G}_i}} \rangle$ denotando el estado de la réplica *i*.

Observación 6. *Tal como vimos en el capítulo anterior, se puede interpretar al* pushout *como la composición de distintos estados. Por lo tanto, describimos la evolución del sistema en términos de la evolución de una de las partes.*

Definición 5.1.2 (LTS para multiples réplicas)**.** *Sea* $\mathbb{S}$ *una especificación coherente. Entonces, la LTS* para multiples réplicas abstractas $\mathcal{T}_{\mathbb{S}}$ *es generada por la LTS para una réplica y la regla adicional que describimos a continuación.*

$$\frac{\langle \mathtt{G}_1, \mathtt{P}|_{\mathtt{E}_{\mathtt{G}_1}} \rangle \xrightarrow{\ell} \langle \mathtt{G}_1', \mathtt{P}_1' \rangle \qquad \mathtt{P}' \in \mathtt{P} \otimes \mathtt{P}_1'}{\langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle \xrightarrow{\ell} \langle \mathtt{G}_1' \sqcup \mathtt{G}_2, \mathtt{P}' \rangle} \text{(COMP)}$$

Regla (COMP) describe las computaciones asociadas a un estado replicado. Siempre que una parte de un estado replicado (denotado por $\langle \mathtt{G}_1, \mathtt{P}|_{\mathtt{E}_{\mathtt{G}_1}} \rangle$) evoluciona a $\langle \mathtt{G}_1', \mathtt{P}_1' \rangle$ por realizar la acción ℓ, entonces todo el sistema evoluciona al estado obtenido por componer la parte que no ha cambiado (en este caso, $\mathtt{G}_2$) y el nuevo estado $\langle \mathtt{G}_1', \mathtt{P}_1' \rangle$.

De hecho, los estados correspondientes a la unión compatible LDAGs se puede ver como la composición paralela, y la propiedad de *soundness* de la regla es asegurada con el siguiente resultado que establece que las transiciones de un estado compuesto se corresponde con las transiciones de sus componentes.

Lema 14. *Sea* $\mathbb{S}$ *una especificación coherente,* $\mathtt{G}_1$, $\mathtt{G}_2$ LDAG*s compatibles, y* $\mathtt{P} \in \mathbb{S}(\mathtt{G}_1 \sqcup \mathtt{G}_2)$ *un camino. Si* $\langle \mathtt{G}_1, \mathtt{P}|_{\mathtt{E}_{\mathtt{G}_1}} \rangle \xrightarrow{\ell} \langle \mathtt{G}_1', \mathtt{P}' \rangle$ *entonces* $\mathtt{P} \otimes \mathtt{P}' \subseteq \mathbb{S}(\mathtt{G}_1' \sqcup \mathtt{G}_2)$.

Demostración. Esta es inmediata por el hecho que $\mathbb{S}$ es coherente. □

5.1.1. Implementando una especificación

En esta subsección, repasamos la manera en que se describe la implementación de RDTs siguiendo la presentación de [8]. En particular, nos concentraremos en la implementación de los tipos de datos replicados *basada en estado*, en contrapropuesta a las implementaciones *basadas en operaciones* [5] (las diferencias fueron explicadas en § 1.1).

Siguiendo el enfoque en [8], describiremos una implementación de un tipo de dato en términos del comportamiento de una réplica y asumiremos que todas las réplicas tienen el mismo comportamiento. También consideraremos LTSs como los modelos operacionales de las réplicas. Sea Σ (cuyos estados se identifican como $\sigma, \sigma_0, \ldots$) el conjunto de los

$$\frac{\mathtt{k} = \Sigma_{\mathsf{s} \in dom(\mathsf{v})} \mathsf{v}(\mathsf{s})}{\langle \mathsf{r}, \mathsf{v} \rangle \xrightarrow{\mathtt{rd},\mathtt{k}} \langle \mathsf{r}, \mathsf{v} \rangle} \text{(READ)} \qquad \langle \mathsf{r}, \mathsf{v} \rangle \xrightarrow{\mathtt{inc},\mathtt{ok}} \langle \mathsf{r}, \mathsf{v}[\mathsf{r} \mapsto \mathsf{v}(\mathsf{r}) + 1] \rangle \text{(INC)}$$

$$\langle \mathsf{r}, \mathsf{v} \rangle \xrightarrow{\mathtt{send},\langle \mathsf{r},\mathsf{v} \rangle} \langle \mathsf{r}, \mathsf{v} \rangle \text{(SEND)} \qquad \langle \mathsf{r}, \mathsf{v} \rangle \xrightarrow{\mathtt{rcv},\langle \mathsf{r}_k,\mathsf{v}_k \rangle} \langle \mathsf{r}, \mathtt{max}\{\mathsf{v}, \mathsf{v}_k\} \rangle \text{(RCV)}$$

Figura 5.1: Implementación del tipo de dato CONTADOR

posibles estados de una réplica. Dada una especificación $\mathbb{S}$ definida sobre el conjunto de etiquetas $\mathcal{L}$, el conjunto de etiquetas para su LTS esta definido como

$$\mathcal{L}_{\mathbb{S}} = \mathcal{L} \cup (\{\mathtt{send}, \mathtt{rcv}\} \times \Sigma) \cup \{\tau\}$$

El conjunto $\mathcal{L}_{\mathbb{S}}$ es construido sobre las operaciones del tipo de dato (es decir, elementos en $\mathcal{L} = O \times \mathcal{V}$), con dos operaciones especiales $\langle \mathtt{send}, \sigma \rangle$ y $\langle \mathtt{rcv}, \sigma \rangle$ usadas para sincronizar los estados, y τ para transiciones internas.

El comportamiento de una réplica para una especificación $\mathbb{S}$ está dado por un LTS $\mathcal{C}_{\mathbb{S}}$ con etiquetas en $\mathcal{L}_{\mathbb{S}}$ que es *total*, es decir, tal que $\forall \sigma, \mathsf{o}.\ \exists v.\ \sigma \xrightarrow{\mathsf{o},v}$. Totalidad asegura que todas las operaciones en una implementación de un tipo de dato son no-bloqueantes, es decir, una réplica es capaz de realizar cualquier operación del tipo de dato en cualquier estado.

Ejemplo 5.1.1. *Considere el tipo de dato* `Contador` *en Ej. 3.2.1 y una implementación basada en estado presentada en [8], esta se define como*

- $\Sigma = \mathcal{R} \times (\mathcal{R} \mapsto \mathbb{N})$*, donde* $\mathcal{R}$ *es el conjunto de identificadores de réplicas;*
- $\mathcal{L} = \{\langle \mathtt{inc}, \mathtt{ok} \rangle\} \cup (\{\mathtt{rd}\} \times \mathbb{N})$ *es el conjunto de operaciones del tipo de dato;*
- $\xrightarrow{\ell}$ *está dado por las reglas de inferencia en Fig. 5.1.*

Los estados de una réplica son pares de la forma $\langle \mathsf{r}, \mathsf{v} \rangle$*, dónde* r *es el identificador de la réplica y* v *es un diccionario que mantiene la cantidad de incrementos realizados en cada réplica, es decir,* $\mathsf{v}(\mathsf{r}')$ *es el número de incrementos realizados sobre la réplica* r'*.*

En Fig. 5.1 mostramos las reglas de inferencia para las reglas de transición. La regla (READ) *describe el comportamiento de una réplica que maneja la operación de lectura que realiza un cliente. En este caso, la réplica retorna el valor* $\mathtt{k}$*, el cual se corresponde al número total de incrementos conocidos por todas las réplicas. Esta transición no cambia el estado de la réplica. En cambio, el estado cambia cuando se realiza una operación de incremento, como se describe en la regla* (INC)*. El nuevo estado registra el hecho de que* r *realizó otro incremento. Otro cambio de estado sucede al actualizar el estado local cuando se reciben cambios propagados por otras réplicas, como se describe en la regla*

(RCV). *Cuando se recibe un mensaje* $\langle \mathtt{r}_k, \mathtt{v}_k \rangle$*, la réplica actualiza su diccionario local con* $\mathtt{max}\{\mathtt{v}, \mathtt{v}_k\}$*, el cual se define de manera tal que*

$$\forall \mathtt{s}.\mathtt{max}\{\mathtt{v}, \mathtt{v}_k\}(\mathtt{s}) = \mathtt{max}\{\mathtt{v}(\mathtt{s}), \mathtt{v}_k(\mathtt{s})\}$$

La pregunta que surge es si el comportamiento de una réplica descripto en Ej. 5.1.1 es una implementación correcta con respecto a la especificación en Ej. 3.2.1. Direccionaremos este problema en dos pasos: primero de todo, vamos a mostrar que una sóla réplica es una implementación correcta del tipo de dato (§ 5.1.2); luego, analizaremos el caso de combinar el comportamiento de varias réplicas (§ 5.1.3).

5.1.2. Vinculando una especificación con el comportamiento de una réplica

En esta sección, vamos a dar un criterio para probar formalmente la correctitud de la implementación de una réplica. Más precisamente, proponemos la correspondencia entre una especificación y el comportamiento de una réplica a partir de una relación de simulación débil.

Definición 5.1.3 (Implementación)**.** *Sea* $\mathbb{S}$ *una especificación,* $\mathcal{T}_\mathbb{S}$ *el LTS de una réplica (o multiples réplicas) abstracta, y* $\mathcal{C}_\mathbb{S}$ *una implementación.* $\mathcal{I}_\mathbb{S} \subseteq \mathcal{T}_\mathbb{S} \times \mathcal{C}_\mathbb{S}$ *es una relación de implementación si* $(\sigma, \langle \mathtt{G}, \mathtt{P} \rangle) \in \mathcal{I}_\mathbb{S}$ *entonces para todo* σ'*,* σ''*,* $\mathtt{o}$ *y* $\mathtt{v}$ *se cumple que*

1. *si* $\sigma \xrightarrow{\mathtt{o},\mathtt{v}} \sigma'$ *entonces* $\exists \mathtt{G}', \mathtt{P}'$ *tal que* $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\mathtt{o},\mathtt{v}} \langle \mathtt{G}', \mathtt{P}' \rangle$ *y* $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}_\mathbb{S}$*;*
2. *si* $\sigma \xrightarrow{\mathtt{rcv},\sigma'} \sigma''$ *entonces* $\exists \mathtt{G}', \mathtt{P}', \mathtt{P}''$ *tal que* $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}_\mathbb{S}$ *y* $\mathtt{P}'' \in \mathtt{P} \otimes \mathtt{P}'$ *y* $(\sigma'', \langle \mathtt{G} \sqcup \mathtt{G}', \mathtt{P}'' \rangle) \in \mathcal{I}_\mathbb{S}$*;*
3. *if* $\sigma \xrightarrow{\mathtt{send},\sigma'} \sigma$ *entonces* $\exists \mathtt{G}', \mathtt{P}'$ *tal que* $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}_\mathbb{S}$ *y* $\mathtt{G} \sqcup \mathtt{G}' = \mathtt{G}$ *y* $\mathtt{P} \otimes \mathtt{P}' = \{\mathtt{P}\}$*;*
4. *si* $\sigma \xrightarrow{\tau} \sigma'$ *entonces* $(\sigma', \langle \mathtt{G}, \mathtt{P} \rangle) \in \mathcal{I}_\mathbb{S}$.

Escribiremos $\sim^o_\mathbb{S}$ *para denotar a la mayor relación de implementación con respecto a el LTS de una-réplica abstracta de la especificación* $\mathbb{S}$*, y* $\sim^m_\mathbb{S}$ *para las multi-réplicas.*

Cualquier par $(\sigma, \langle \mathtt{G}, \mathtt{P} \rangle)$ en la relación $\mathcal{I}_\mathbb{S}$ establece que el estado σ es explicado en términos de la visibilidad $\mathtt{G}$ y la arbitración $\mathtt{P}$, el cual es admitido por la especificación (es decir, $\mathtt{P} \in \mathbb{S}(\mathtt{G})$). Más aún, hay una correspondencia cercana entre la evolución de σ y $\langle \mathtt{G}, \mathtt{P} \rangle$, la cual es establecida en los ítems 1–4. El ítem 1 predica sobre las transiciones correspondientes a las operaciones del tipo de dato. Básicamente, si la réplica realiza la operación $\mathtt{o}$ que produce el resultado $\mathtt{v}$, entonces la especificación $\mathbb{S}$ permite que $\mathtt{G}$ y $\mathtt{P}$ sean extendidos con el evento correspondiente, lo que se expresa formalmente con la transición $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\mathtt{o},\mathtt{v}} \langle \mathtt{G}', \mathtt{P}' \rangle$ formalizado en Def. 5.1.1 and Def. 5.1.2.

El ítem 2 se refiere a aquellas transiciones en las que una réplica recibe actualizaciones (es decir, escrituras) propagadas por otras réplicas. En una implementación basada en

estado, el contenido del mensaje `rcv` es un estado computado por otra réplica. Por esta razón, vamos a requerir que el estado recibido σ' esté relacionado a una configuración admitida por la especificación, es decir, $(\sigma', \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}_{\mathbb{S}}$. Más aún, $\langle \mathtt{G}', \mathtt{P}' \rangle$ debe ser consistente con la historia ya vista por la réplica, es decir, la historia común entre $\langle \mathtt{G}, \mathtt{P} \rangle$ y $\langle \mathtt{G}', \mathtt{P}' \rangle$ debe coincidir. Esto es asegurado requeriendo que la unión de dos configuraciones, es decir, $\mathtt{G} \sqcup \mathtt{G}'$, y el producto de dos caminos, es decir, $\mathtt{P}'' \in \mathtt{P} \otimes \mathtt{P}'$, estén definidos. Bajo las condiciones mencionadas arriba, el estado σ'' se obtiene como la combinación de σ y σ'. Por lo tanto, se debe corresponder con la combinación de $\langle \mathtt{G}, \mathtt{P} \rangle$ y $\langle \mathtt{G}', \mathtt{P}' \rangle$, es decir, $(\sigma'', \langle \mathtt{G} \sqcup \mathtt{G}', \mathtt{P}'' \rangle) \in \mathcal{I}_{\mathbb{S}}$.

El ítem 3 establece que una réplica solo propague mensajes que lleven información del estado actual. Notar que σ' puede contener información sobre algunos eventos del estado actual. Esto se establece formalmente imponiendo las las condiciones $\mathtt{G} \sqcup \mathtt{G}' = \mathtt{G}$ y $\mathtt{P} \otimes \mathtt{P}' = \{\mathtt{P}\}$ (y será útil en § 5.1.3). El ítem 4 es auto explicativo.

A continuación, vamos a referirnos a la correctitud de una implementación sobre una-réplica ó multi-réplicas cuando el LTS abstracto $\mathcal{T}_{\mathbb{S}}$ en la definición anterior sea una-réplica o una multiple-réplica, respectivamente.

Ejemplo 5.1.2. *Podemos mostrar que la réplica definida en el ejemplo 5.1.1 es una implementación correcta para la implementación sobre una-réplica del tipo de dato* `Contador` *(descripto en Ej. 3.2.1) mostrando que la siguiente relación satisface las condiciones de la Def. 5.1.3 (el detalle de las pruebas puede encontrarse en el Apéndice A.1).*

$$\mathcal{I} = \{(\langle \mathtt{r}, \mathtt{v} \rangle, \langle \mathtt{G}, \mathtt{P} \rangle) \mid \mathtt{r} \in \mathit{dom}(\mathtt{v}) \text{ y existe } \{\mathtt{E}_i\}_{r_i \in \mathit{dom}(\mathtt{v})} \text{ partición de } \mathtt{E}_{\mathtt{G}} \\ \text{tal que } \forall \mathtt{r}_i \in \mathit{dom}(\mathtt{v}).\mathtt{v}(\mathtt{r}_i) = |\{\mathtt{e} \mid \mathtt{e} \in \mathtt{E}_i \text{ and } \lambda(\mathtt{e}) = \langle \mathtt{inc}, \mathtt{ok} \rangle\}|\}$$

Ejemplo 5.1.3. *En este ejemplo vamos a mostrar que la implementación optimizada de* `OR-set` *presentada en [8] es correcta con respecto a la especificación dada en Ej. 3.3.2. La implementación de una réplica está dada por*

- $\Sigma = \mathcal{R} \times (\mathcal{R} \mapsto \mathbb{N}) \times ((\mathcal{V} \times \mathcal{R}) \mapsto \mathbb{N})$*;*
- $\mathcal{L} = \{\langle \mathtt{add}(\mathtt{k}), \mathtt{ok} \rangle, \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok} \rangle \mid \mathtt{k} \in \mathcal{V}\} \cup (\{\mathtt{lookup}\} \times 2^{\mathcal{V}})$*;*
- $\rightarrow$ *dada por las reglas de inferencia en Fig. 5.2.*

Los estados son triplas $\langle \mathtt{r}, \mathtt{V}, \mathtt{W} \rangle$*, donde* $\mathtt{r} \in \mathcal{R}$ *es el identificador de cada réplica. El diccionario* $\mathtt{V}$ *asocia cada réplica con un número de versión y significa que* $\mathtt{r}$ *está sincronizado con la versión* $\mathtt{V}(\mathtt{r}')$ *de la réplica* $\mathtt{r}'$*. El diccionario* $\mathtt{W}$ *indica para cada réplica* $\mathtt{r}'$ *y elemento* $\mathtt{k}$*, la versión conocida mas nueva de* $\mathtt{r}'$ *en la que el elemento* $\mathtt{k}$ *pertenece al conjunto:* $\mathtt{W}(\mathtt{k}, \mathtt{r}') = 0$ *si* $\mathtt{k}$ *ha sido agregado o ha sido agregado y quitado. La definición asume que vale el siguiente invariante sobre los estados* $\mathtt{W}(\mathtt{k}, \mathtt{r}) \leq \mathtt{V}(\mathtt{r})$ *para todo* $\mathtt{r}$ *y* $\mathtt{k}$*.*

Ahora discutiremos las reglas en Fig. 5.2. La regla (ADD) *describe el comportamiento de una réplica* $\mathtt{r}$ *que maneja la operación de agregar un elemento* $\mathtt{k}$ *al conjunto. En este caso,* $\mathtt{r}$ *cambia su estado local (i) creando una nueva versión, es decir, la entrada* $\mathtt{V}(\mathtt{r})$ *es actualizada a* $\mathtt{V}(\mathtt{r}) + 1$ *para reflejar que hay una nueva versión de* $\mathtt{r}$*, y (ii) registrando que*

$$\text{(ADD)}\quad \frac{\mathtt{V}' = \mathtt{V}[\mathtt{r} \mapsto \mathtt{V}(\mathtt{r}) + 1] \qquad \mathtt{W}' = \mathtt{W}[(\mathtt{k},\mathtt{r}) \mapsto \mathtt{V}(\mathtt{r}) + 1]}{\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \xrightarrow{\mathtt{add}(\mathtt{k}),\mathtt{ok}} \langle \mathtt{r},\mathtt{V}',\mathtt{W}'\rangle}$$

$$\text{(REM)}\quad \frac{\mathtt{W}' = \mathtt{W}[\forall \mathtt{s} \in \mathcal{R}.\ (\mathtt{k},\mathtt{s}) \mapsto 0]}{\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \xrightarrow{\mathtt{rem}(\mathtt{k}),\mathtt{ok}} \langle \mathtt{r},\mathtt{V},\mathtt{W}'\rangle} \qquad \text{(LOOKUP)}\quad \frac{\mathtt{S} = \{\mathtt{k} \mid \exists \mathtt{s} \in \mathcal{R}.\ \mathtt{W}(\mathtt{k},\mathtt{s}) > 0\}}{\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \xrightarrow{\mathtt{lookup},\mathtt{S}} \langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle}$$

$$\text{(SEND)}\quad \langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \xrightarrow{\mathtt{send},\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle} \langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \qquad \text{(RECEIVE)}\quad \langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle \xrightarrow{\mathtt{rcv},\langle \mathtt{r}',\mathtt{V}',\mathtt{W}'\rangle} \langle \mathtt{r},\mathtt{max}\{\mathtt{V},\mathtt{V}'\},(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')\rangle$$

Figura 5.2: Implementación del tipo de dato OR-SET

se agrega el elemento k *en la nueva versión de* r*, es decir, la entrada* $\mathtt{W}(\mathtt{k},\mathtt{r})$ *se actualiza con* $\mathtt{V}(\mathtt{r}) + 1$*. La regla* (REM) *describe sacar el elemento* k*. En este caso, para toda* $\mathtt{s} \in \mathcal{R}$*,* $\mathtt{W}(\mathtt{k},\mathtt{s})$ *se actualiza a 0, indicando la eliminación del elemento* k*. La regla* (LOOKUP) *explica cuando se realiza la operación* lookup*: la réplica* r *devuelve el conjunto* S *con los elementos que están presentes en al menos una de las versiones conocidas de las réplicas.*

La regla (SEND) *es inmediata. Durante la sincronización, la cual se describe en la regla* (RECEIVE)*, los diccionarios* V *y* W *se actualizan con la información contenida en el mensaje recibido. Se asume que los diccionarios recibidos* $\mathtt{V}'$ *y* $\mathtt{W}'$ *son consistentes, esto es,* $\mathtt{W}'(\mathtt{k},\mathtt{r}_i) \leq \mathtt{V}'(\mathtt{r}_i)$ *para todo* k*. Para* V*, el nuevo estado mantiene la versión mayor de cada réplica, es decir,* $\mathtt{max}\{\mathtt{V},\mathtt{V}'\}$*. La combinación de* W *y* $\mathtt{W}'$ *es más interesante ya que esta maneja los conflictos que surgen por operaciones concurrentes sobre el mismo elemento. Hay un conflicto entre* W *y* $\mathtt{W}'$ *para* $(\mathtt{k},\mathtt{s})$ *cuando un diccionario indica que* k *está presente en la réplica* s *y el otro dice que no, es decir,* $\mathtt{W}(\mathtt{k},\mathtt{s}) > 0$ *y* $\mathtt{W}'(\mathtt{k},\mathtt{s}) = 0$ *ó viceversa.*

*En caso de conflictos, estos se resuelven usando la información en los diccionarios*V *y* $\mathtt{V}'$*. Formalmente, esto se refleja en el operador* $\oplus$*, definido a continuación*

$$(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{s}) = \begin{cases} 0 & \textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{s}) \ \vee \textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{s}) \\ \mathtt{max}\{\mathtt{W}(\mathtt{k},\mathtt{s}),\mathtt{W}'(\mathtt{k},\mathtt{s})\} & \textit{caso contrario} \end{cases}$$

donde $\textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{s}) = (\mathtt{W}(\mathtt{k},\mathtt{s}) = 0 \wedge \mathtt{W}'(\mathtt{k},\mathtt{s}) \leq \mathtt{V}(\mathtt{s}))$ *es el predicado que caracterisa el hecho de que* k *ha sido quitado de la réplica* s *de acuerdo a los diccionarios* W*,* $\mathtt{W}'$ *y* V*.*

Es posible demostrar que una réplica es correcta con respecto a la especificación en Ej. 3.3.2 mostrando que la siguiente relación satisface las condiciones en Def. 5.1.3

$$\mathcal{I} = \{(\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle, \langle \mathtt{G},\mathtt{P}\rangle) \mid \textit{ existe } f : \mathtt{E}_{\mathtt{G}} \to \mathcal{R} \textit{ tal que } \\ \forall \mathtt{k}.(\exists \mathtt{r} \in \mathcal{R}.\mathtt{W}(\mathtt{k},\mathtt{r}) > 0 \iff \exists\, \mathtt{S}.\ \mathbb{S}_{\textit{Or-Set}}(\mathtt{G}^{\langle \mathtt{lookup},\mathtt{S}\rangle}) \neq \emptyset \ \wedge\ \mathtt{k} \in \mathtt{S})\}$$

Correctitud y Refinamiento Es fácil notar que la correctitud es preservada por refinamiento, es decir, si $\mathbb{S}$ es un refinamiento de $\mathbb{S}'$ y una implementación I es correcta con

$$\frac{\sigma_1 \xrightarrow{\ell} \sigma_1'}{\sigma_1 \| \sigma_2 \xrightarrow{\ell} \sigma_1' \| \sigma_2} \text{(PARL)} \qquad \frac{\sigma_1 \xrightarrow{\texttt{send},\sigma} \sigma_1' \quad \sigma_2 \xrightarrow{\texttt{rcv},\sigma} \sigma_2'}{\sigma_1 \| \sigma_2 \xrightarrow{\tau} \sigma_1' \| \sigma_2'} \text{(COMM)}$$

Figura 5.3: Comunicación sincrónica de réplicas (reglas simétricas son omitidas)

respecto a $\mathbb{S}$, entonces I es correcta con respecto $\mathbb{S}'$. Esto se deduce del hecho de que $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\mathtt{o},\mathtt{v}} \langle \mathtt{G}', \mathtt{P}' \rangle$ en $\mathbb{S}$ implica $\langle \mathtt{G}, \mathtt{P} \rangle \xrightarrow{\mathtt{o},\mathtt{v}} \langle \mathtt{G}', \mathtt{P}' \rangle$ en $\mathbb{S}'$. En consecuencia, podemos concluir que la implementación en Ej. 5.1.3 también es un implementación correcta de la especificación no determinística de $\mathbb{S}_{Set}$ introducida en Ej. 3.3.2.

5.1.3. Sobre el comportamiento de múltiples réplicas

Una implementación completa de un tipo de dato se obtiene poniendo varias réplicas en paralelo. Además, usaremos un operador de composición que establece la forma en que se sincronizan diferentes réplicas. Su definición depende del modelo de comunicación elegido (sincrónico, asincrónico, broadcast, ...).

Como final del capítulo, vamos mostrar que la composición paralela de varias réplicas (cada una individualmente correcta), es una correcta implementación de un tipo de dato. Hacemos notar que nos enfocamos en especificaciones coherentes y comenzamos considerando el modelo estándar de comunicación sincrónica, que se define a continuación.

Definición 5.1.4 (Implementación Sincrónica)**.** *Sea $\mathbb{S}$ una especificación y $\mathcal{C}_\mathbb{S}$ una implementación. Entonces la extensión* sincrónica *$\mathcal{C}_\mathbb{S}^\sigma$ de $\mathcal{C}_\mathbb{S}$ se obtiene agregando una operación binaria $\|$ para la composición parallela de estados y extendiendo la relación de transición con las reglas adicionales en Fig. 5.3 (donde se omiten las versiones simétricas).*

Ahora queremos llegar a la conclusión de que siempre que tengamos una implementación $\mathcal{C}_\mathbb{S}$ para una réplica correcta, entonces la extensión sincrónica $\mathcal{C}_\mathbb{S}^\sigma$ que es multiples réplicas, también es correcta. La intuición es que se describe el estado de la composición paralela de las réplicas. en términos de las configuraciones asociadas con cada uno de los componentes.

El siguiente resultado es suficiente para nuestro propósito, ya que establece que la simulación para una especificación coherente es cerrada bajo composición paralela sincrónica, y se basa en Lema 14.

Lema 15. *Sea $\mathbb{S}$ una especificación coherente, $\mathcal{C}_\mathbb{S}$ una implementación de una-réplica que es correcta y σ_0, σ_1 dos estados de $\mathcal{C}_\mathbb{S}$. Si $\sigma_i \sim_\mathbb{S}^o \langle \mathtt{G_i}, \mathtt{P_i} \rangle$ para $i \in \{1,2\}$, entonces $\sigma_1 \| \sigma_2 \sim_\mathbb{S}^m \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle$ para cualquier $\mathtt{P} \in \mathtt{P}_1 \otimes \mathtt{P}_2$.*

Demostración. Mostramos que la siguiente relación satisface Def. 5.1.3.

$$I = \{(\sigma_1 \| \sigma_2, \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle) \mid \sigma_1 \sim_\mathbb{S}^o \langle \mathtt{G}_1, \mathtt{P}_1 \rangle \text{ y } \sigma_2 \sim_\mathbb{S}^o \langle \mathtt{G}_2, \mathtt{P}_2 \rangle \text{ and } \mathtt{P} \in \mathtt{P}_1 \otimes \mathtt{P}_2\}$$

$$\text{(PARL)}\ \frac{\sigma_1 \xrightarrow{\ell} \sigma_1'}{\sigma_1 \mid_{\mathtt{B}} \sigma_2 \xrightarrow{\ell} \sigma_1' \mid_{\mathtt{B}} \sigma_2} \qquad \text{(SENDL)}\ \frac{\sigma_1 \xrightarrow{\mathtt{send},\sigma} \sigma_1'}{\sigma_1 \mid_{\mathtt{B}} \sigma_2 \xrightarrow{\tau} \sigma_1' \mid_{\mathtt{B}\cdot\sigma} \sigma_2} \qquad \text{(RECEIVEL)}\ \frac{\sigma_1 \xrightarrow{\mathtt{rcv},\sigma} \sigma_1'}{\sigma_1 \mid_{\sigma\cdot\mathtt{B}} \sigma_2 \xrightarrow{\tau} \sigma_1' \mid_{\mathtt{B}} \sigma_2}$$

Figura 5.4: Comunicación de réplicas con búfer(reglas simétricas son omitidas)

Procedemos por análisis de casos sobre la transición de $\sigma_1 \parallel \sigma_2$.

- (PARL). Dado $\sigma_1 \sim^o_{\mathbb{S}} \langle \mathtt{G}_1, \mathtt{P}_1 \rangle$, tenemos que $\langle \mathtt{G}_1, \mathtt{P}|_{\mathtt{E}_{\mathtt{G}_1}} \rangle \xrightarrow{\ell} \langle \mathtt{G}_1', \mathtt{P}_1' \rangle$. Entonces, el caso vale por Lem. 14.

- (COMM). Dado que $\sigma_1 \xrightarrow{\mathtt{send},\sigma} \sigma_1'$ y $\sigma_1 \sim^o_{\mathbb{S}} \langle \mathtt{G}_1, \mathtt{P}_1 \rangle$, tenemos que (i) $\sigma_1' = \sigma_1$, (ii) $\sigma \sim^o_{\mathbb{S}} \langle \mathtt{G}', \mathtt{P}' \rangle$, (iii) $\mathtt{G}_1 \sqcup \mathtt{G}' = \mathtt{G}_1$, and (iv) $\mathtt{P}_1 \otimes \mathtt{P}' = \mathtt{P}_1$. Adicionalmente, $\sigma_2 \xrightarrow{\mathtt{rcv},\sigma} \sigma_2'$ implica que existe $\mathtt{G}'', \mathtt{P}''$ tal que (v) $\sigma \sim^o_{\mathbb{S}} \langle \mathtt{G}'', \mathtt{P}'' \rangle$, (vi) $\mathtt{P}_2' \in \mathtt{P}_2 \otimes \mathtt{P}''$ y (vii) $\sigma_2' \sim^o_{\mathbb{S}} \langle \mathtt{G}_2 \sqcup \mathtt{G}'', \mathtt{P}_2' \rangle$. Por (ii), (v) y (vii) concluimos que $\sigma_2' \sim^o_{\mathbb{S}} \langle \mathtt{G}_2 \sqcup \mathtt{G}', \mathtt{P}_2' \rangle$. Más aún, $\mathtt{G}_1 \sqcup (\mathtt{G}_2 \sqcup \mathtt{G}') = \mathtt{G}_1 \sqcup \mathtt{G}_2$ dado que vale (iii). Por lo tanto, $(\sigma_1' \parallel \sigma_2', \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle) \in I$.

□

Luego, obtenemos inmediatamente el resultado deseado.

Proposición 6. *Sea $\mathbb{S}$ una especificación coherente, y $\mathcal{C}_{\mathbb{S}}$ una implementación que es una réplica correcta. Entonces, $\mathcal{C}_{\mathbb{S}}^{\sigma}$ es múltiples réplicas, también es correcta.*

Demostración. Esta se sigue como en el lema anterior, mostrando que $\sim^m_{\mathbb{S}}$ es preservado por la composición en paralelo $\parallel$ de estados. □

Ejemplo 5.1.4. *El resultado anterior nos permite concluir que la implementación que consiste de la composición paralela de réplicas que se comportan como se ha descripto en Ej. 5.1.1 es correcta con respecto a la especificación de $\mathbb{S}_{\mathtt{Ctr}}$ en Ej. 3.2.1, que es coherente (ver Ej. 3.4.3). Del mismo modo, podemos concluir que la composición paralela de varias réplicas para* `OR-Set` *en Ej. 5.1.3 es una implementación correcta.*

Observamos que diferentes modelos de comunicación pueden acomodarse de manera análoga. Podemos considerar la implementación asincrónica $\mathcal{C}_{\mathbb{S}}^{\alpha}$ obtenida al agregar una familia de operadores $\mid_{\mathtt{B}}$ definidos en Fig. 5.4, donde $\mathtt{B}$ denota una *cola* FIFO. Formalmente, representamos $\mathtt{B}$ como una secuencia $\sigma_1 \cdot \ldots \cdot \sigma_n$ de estados y escribimos ε para la secuencia vacía. La regla (SENDL) dice que el estado σ enviado por una réplica se agrega al final del búfer. Simétricamente, la regla (RECIBIRL) establece que una réplica consume elementos a la cabeza de la cola. También en este caso, podemos demostrar que se preserva la corrección de la implementación.

Lema 16. *Sea $\mathbb{S}$ una especificación coherente, $\mathcal{C}_{\mathbb{S}}$ una implementación para una-réplica que es correcta, y σ_0, σ_1 dos estados de $\mathcal{C}_{\mathbb{S}}$. Si $\sigma_i \sim^o_{\mathbb{S}} \langle \mathtt{G}_i, \mathtt{P}_i \rangle$ para $i \in \{1,2\}$, entonces $\sigma_1 \mid_{\varepsilon} \sigma_2 \sim^m_{\mathbb{S}} \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P} \rangle$ para cualquier $\mathtt{P} \in \mathtt{P}_1 \otimes \mathtt{P}_2$.*

Demostración. La prueba es análoga a Lem. 15 mostrando que la siguiente relación satisface Def. 5.1.3.

$$I = \{(\sigma_1 \mid_{\mathtt{B}} \sigma_2, \langle \mathtt{G}_1 \sqcup \mathtt{G}_2, \mathtt{P}\rangle) \mid \sigma_1 \sim^{o}_{\mathbb{S}} \langle \mathtt{G}_1, \mathtt{P}_1\rangle \text{ y } \sigma_2 \sim^{o}_{\mathbb{S}} \langle \mathtt{G}_2, \mathtt{P}_2\rangle \text{ y } \mathtt{P} \in \mathtt{P}_1 \otimes \mathtt{P}_2 \\ \text{y } \forall \sigma \in \mathtt{B}.\exists \mathtt{G}.(\sigma \sim^{o}_{\mathbb{S}} \langle \mathtt{G}, \mathtt{P}\rangle \text{ y } (\mathtt{G}_1 \sqcup \mathtt{G}_2) \sqcup \mathtt{G} \text{ definida})\}$$

□

Luego, obtenemos inmediatamente el siguiente resultado.

Proposición 7. *Sea $\mathbb{S}$ una especificación coherente, $\mathcal{C}_{\mathbb{S}}$ una implementación que es una-réplica correcta, Entonces, $\mathcal{C}^{\alpha}_{\mathbb{S}}$ es multi-réplica correcta.*

Demostración. Esta se sigue como en el lema anterior, mostrando que $\sim^{m}_{\mathbb{S}}$ es preservada por la composición paralela $|_{\mathtt{B}}$ de estados. □

Parte II

Modelos de Consistencia

Capítulo 6

Semánticas para Sistemas Débilmente Replicados

En la primera parte de este trabajo nos propusimos desarrollar una contrapropuesta a las especificaciones clásicas para tipos de datos replicados. Sin embargo, más allá de las diferencias presentadas y estudiadas en Capítulo 3, encontramos que ambas especificaciones tienen un aspecto en común, y este es: el ser declarativas, es decir, no hacen referencia al comportamiento interno que tiene una base de datos geo-replicada que implementa un RDT. Si bien en el Capítulo 5 probamos que las implementaciones para RDT s, propuestas en [8], son correctas con respecto a nuestras especificaciones, esto nos lleva a entender que lidiar con detalles de implementación requiere otro nivel de abstracción.

La literatura[12, 21, 36], propone diferentes alternativas para implementar sistemas replicados, por lo tanto, nuestro objetivo ahora es describir el comportamiento de una base de datos geo-replicada capturando de manera abstracta detalles de implementación que impactan directamente en las garantías de consistencia que estos sistemas ofrecen. Inspirado en trabajos precedentes sobre modelos de memoria débil [37, 38, 39, 40, 41, 42, 43], proponemos un marco operacional abstracto para sistemas débilmente consistentes, donde se obtienen modelos concretos como instancias particulares de uno general. Más específicamente, el estado de una base de datos replicada σ se describe como un orden parcial sobre el conjunto de escrituras $u_0, \ldots, u_n$ realizadas en el sistema. El comportamiento de un sistema se define formalmente en términos de un LTS paramétrico en los siguientes tres predicados (abstractos):

- *skip* y *read-consistency* regulan lecturas. Skip dice si una operación de lectura puede saltear algunas escrituras ejecutadas sobre el sistema; por ejemplo, cuando son realizadas por diferentes sesiones. Read-consistency establece las condiciones que tiene que tener el estado del sistema cuando se ejecuta una operación de lectura; por ejemplo, puede ser usado para prohibir una lectura sobre una réplica que no contiene todas las escrituras ejecutadas previamente por la misma sesión, como en el caso de consistencia de *read-your-write*.

- *write-consistency* es análogo a read-consistency pero establece las condiciones que

permiten una escritura particular. Write-consistency también se usa para controlar la forma en que se realizan las escrituras cuando se propagan cambios entre las distintas réplicas.

Vamos a mostrar que nuestro modelo operacional es lo suficientemente general como para capturar algunos de los modelos de consistencia conocidos sobre varios objetos, como los que proporcionan (i) garantías de sesión, es decir,*read-your-writes*, *monotonic-reads*, *monotonic writes* y *writes-follow-reads*; y (ii) *causal visibility*, *causal arbitration* y *sequential consistency*. Probamos que nuestras caracterizaciones son correctas al exhibir una correspondencia uno-a-uno entre las ejecuciones del modelo operacional y los admitidos por las caracterizaciones axiomáticas en [15]. *Soundness* se obtiene al mostrar que cada computación del modelo operacional preserva el invariante de consistencia correspondiente a la caracterización axiomática. Por el contrario, obtenemos *completitud* al mostrar que cualquier ejecución permitida por una definición axiomática puede ser realmente alcanzada por el modelo operacional correspondiente.

Es importante remarcar que nuestra caracterización operacional no establece ni sugiere una implementación para un modelo de consistencia. Particularmente, nos alejamos de los detalles concretos para favorecer una descripción compacta y operacional del comportamiento de un sistema replicado.

Remarcamos que a lo largo de este capítulo, trabajaremos con el estilo de propagación *basado en estados*, asumiendo que el sistema replicado implementa el tipo de datos `Registro`, introducido en el Ej. 2.5.2.

6.1. Semántica operacional para sistemas replicados

Introducimos un marco operacional para sistemas débilmente consistentes, el cual se utilizará, luego, para obtener la semántica operacional de modelos de consistencia bien conocidos.

6.1.1. Sintaxis

Vamos a considerar a los conjuntos $\mathcal{S}$ de sesiones $\mathtt{s}$, $\mathtt{s}'$, ...; $\mathcal{R}$ de réplicas $\mathtt{r}$, $\mathtt{r}'$, ...; $\mathcal{X}$ de objetos $\mathtt{x}$, $\mathtt{y}$, $\mathtt{z}$ ...; y $\mathcal{V}$ de valores $\mathtt{v}$,$\mathtt{v}'$,$\mathtt{v}''$, ...

Representaremos de manera abstracta al estado de una base de datos a través de una secuencia σ que contiene a todas las escrituras realizadas por las diferentes sesiones. Formalmente, el dominio de las escrituras está dado por $\mathbb{U} = \mathcal{X} \times \mathcal{V} \times \mathcal{S} \times \mathcal{R} \times 2^{\mathcal{S}} \times 2^{\mathcal{R}}$ y un estado σ es un elemento de $\mathbb{U}^*$. Escribimos $[\mathtt{x} \hookleftarrow \mathtt{v}]^{\mathtt{S},\mathtt{R}}_{\mathtt{s},\mathtt{r}}$, en lugar de $(\mathtt{x},\mathtt{v},\mathtt{s},\mathtt{r},\mathtt{S},\mathtt{R}) \in \mathbb{U}$, para denotar una operación de escritura en el cual la sesión $\mathtt{s}$ ejecuta una escritura del objeto $\mathtt{x}$ con el valor $\mathtt{v}$ sobre la replica $\mathtt{r}$. Además, decoramos cada operación de escritura con dos conjuntos: el conjunto $\mathtt{R} \in 2^{\mathcal{R}}$ que contiene las réplicas que conocen esa escritura en particular (esto es, el conjunto de réplicas que han recibido la escritura) mientras que $\mathtt{S} \in 2^{\mathcal{S}}$ es el conjunto que contiene las sesiones que ya han leído la escritura.

Sean $\mathsf{u}, \mathsf{u}', \mathsf{u}''$ elementos de $\mathbb{U}$, escribimos $\mathsf{u.x}$ para proyectar la primera componente de u, es decir, el objeto que fue escrito. Similarmente $\mathsf{u.v}$, $\mathsf{u.s}$, $\mathsf{u.r}$, $\mathsf{u.S}$ y $\mathsf{u.R}$ denota las proyecciones correspondientes al resto de las componentes. El operador $\_[\_ \mapsto \_]$ actualiza el valor de un componente en una tupla, por ejemplo, $\mathsf{u}[\mathtt{R} \mapsto \emptyset]$ denota la tupla u reemplazando el valor de $\mathsf{u.R}$ por $\emptyset$.

Como es usual, escribimos ε para la secuencia vacia y '$-\cdot-$' para concatenar secuencias. Usamos $\mathsf{u} \in \sigma$ para denotar $\sigma = \sigma_0 \cdot \mathsf{u} \cdot \sigma_1$.

Definición 6.1.1 (Equivalencia de estados)**.** *La equivalencia de estados* $\equiv$ *es una relación de congruencia sobre* $\mathbb{U}^*$ *tal que* $\mathsf{u} \cdot \mathsf{u}' \equiv \mathsf{u}' \cdot \mathsf{u}$ *sii* $\mathsf{u.x} \neq \mathsf{u}'.\mathsf{x} \vee \mathsf{u.R} \cap \mathsf{u}'.\mathtt{R} = \emptyset$.

Intuitivamente, se pueden intercambiar dos escrituras siempre que modifiquen diferentes objetos o sean conocidas por partes desconectadas del sistema (es decir, su orden relativo no está decidido).

Definición 6.1.2 (Estado arbitrado)**.** *Un estado* σ *está arbitrado, escrito* $\mathtt{arb}(\sigma)$*, sii* $\forall \sigma'.\sigma' \equiv \sigma \implies \sigma' = \sigma$.

6.1.2. Semántica operacional

La semántica operacional para un sistema con datos replicados está dada por un LTS sobre $\mathbb{U}^*$. El conjunto de etiquetas β está dado por la siguiente gramática:

$$\alpha \;::=\; [\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathsf{s},\mathsf{r}} \mid (\mathtt{v} \leftarrowtail \mathtt{x})_{\mathsf{s},\mathsf{r}} \qquad\qquad \beta \;::=\; \tau \mid \alpha$$

Una etiqueta α corresponde a la interacción de una sesión con una réplica : $[\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathsf{s},\mathsf{r}}$ denota una escritura mientras $(\mathtt{v} \leftarrowtail \mathtt{x})_{\mathsf{s},\mathsf{r}}$ representa una operación de lectura. El significado de $[\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathsf{s},\mathsf{r}}$ es análogo al de una escritura ejecutada en una réplica excepto del hecho que estas no están decoradas con las componentes $\mathtt{S}$ y $\mathtt{R}$. Haciendo abuso de notación, vamos a escribir $[\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathsf{s},\mathsf{r}} \in \mathbb{U}$. La etiqueta $(\mathtt{v} \leftarrowtail \mathtt{x})_{\mathsf{s},\mathsf{r}}$ denota una lectura: la sesión s lee el objeto $\mathtt{x}$ de la réplica r y obtiene el valor $\mathtt{v}$. El conjunto de todas las lecturas se define como $\mathbb{R} = \mathcal{X} \times \mathcal{V} \times \mathcal{S} \times \mathcal{R}$. Vamos a considerar rd, rd', ... para identificar elementos de $\mathbb{R}$. Como es usual, τ denota a una acción interna no-observable usada para modelar la sincronización entre réplicas.

Los modelos de consistencia se garantizan a partir de instanciar un modelo general que ofrece tres predicados paramétricos. La semántica operacional para un sistema replicado se define en términos de los siguientes tres predicados.

- $\lhd \subseteq \mathbb{U} \times \mathbb{R}$ (*skip*): escribimos $\mathsf{u} \lhd \mathsf{rd}$ cuando rd puede ignorar a u. También escribimos $\lhd$ para la extensión del predicado skip a una secuencia de escrituras, es decir, $\sigma \lhd \mathsf{rd}$ iff $\forall \mathsf{u} \in \sigma.\ \mathsf{u} \lhd \mathsf{rd}$.

- $\circledcirc \subseteq \mathbb{U}^* \times \mathbb{R}$ (*read-consistency*): denotamos con $\sigma \circledcirc \mathsf{rd}$ el hecho que rd puede leer σ. Este predicado se utiliza para capturar el hecho de que una sesión puede realizar una lectura sobre una réplica particular cuando se cumplen las condiciones impuestas por el modelo de consistencia.

- $\circledcirc \subseteq \mathbb{U}^* \times \mathbb{U}$ (*write-consistency*): denotamos con $\sigma \circledcirc u$ el hecho que u es una escritura válida del estado σ. Análogo al predicado anterior, *write-consistency* es usado para capturar las condiciones que permiten a una sesión realizar una escritura sobre una réplica particular.

La semántica operacional se define como la mínima relación que satisface las reglas de inferencia de Fig. 6.1. El comportamiento de una lectura está dado por las reglas (READ) y (READ EMPTY). Por la regla (READ), una operación de lectura rd devuelve un valor que fue escrito por una escritura particular u en el sistema. En este caso, rd y u hacen referencia al mismo objeto (es decir, $u.x = rd.x$) y tiene el mismo valor (es decir, $u.v = rd.v$). La lectura rd puede ser ejecutada solo si esta puede ignorar todas las escrituras que preceden a u (es decir, $\sigma_1 \upharpoonleft rd$), y ser consistente con el estado del sistema (es decir, $(\sigma_0 \cdot u \cdot \sigma_1) \circledcirc rd$). Luego de realizar rd, el estado del sistema se actualiza registrando que rd ha leído todos las escrituras que pertenecen a la réplica u.r. La notación $\sigma[u'.S \mapsto u'.S \cup \{rd.s\}]_{u'.r=u.r}$ denota el estado obtenido de σ por medio de sustituir cualquier $u' \in \sigma$ tal que $u'.r = u.r$ con $u'[S \mapsto u'.S \cup \{rd.s\}]$. La regla (READ EMPTY) maneja el caso en el cual la operación de lectura rd ignora todas las escrituras en el sistema σ (es decir, $\sigma \upharpoonleft rd$ se mantiene); consecuentemente, su valor de retorno es indefinido (es decir, $rd.v = \perp$).

Por (regla UPDATE), una sesión s puede actualizar un objeto x sobre una réplica r sólo si la escritura en cuestión satisface el predicado *write-consistency*. En tal caso, la nueva escritura es agregada al sistema.

Finalmente, la regla (PROPAGATION) describe la propagación asincrónica de escrituras entre las distintas réplicas. Esta se alcanza agregando réplicas a la decoración R de una escritura, es decir, la componente R se cambia por $R \cup \{r\}$. La primer premisa en la regla indica que *write-consistency* regula la forma en la que las escrituras son propagadas. Un efecto colateral de una propagación es que algunas operaciones son ordenadas definitivamente ya que no pueden ser más intercambiadas. El resto de las premisas de la regla aseguran que la propagación induce un *orden prefijo* sobre las escrituras, es decir, el pasado de cada escritura es totalmente ordenado (insistiremos esto en Sec. 6.2). Regla (STR) permite el reorden de escrituras que no fueron arbitradas.

Como es usual, escribimos $\Longrightarrow$ para $(\xrightarrow{\tau,})^*$ y $\overset{\alpha}{\Longrightarrow}$ para $\Longrightarrow\xrightarrow{\alpha,}\Longrightarrow$. Tambien, abreviamos $\overset{\alpha_1}{\Longrightarrow} \ldots \overset{\alpha_k}{\Longrightarrow}$ con $\overset{\alpha_1 \ldots \alpha_k}{\Longrightarrow}$.

6.1.3. Concretizando sistemas débilmente consistentes

Consideremos ahora cómo codificar diferentes modelos de consistencia, instanciando los predicados descritos en la sección anterior. Para empezar, consideraremos las *session guarantees* de [44], presentadas en Fig. 6.2.

La codificación de la garantía **Read Your Writes** (RYW) requiere que siempre que una sesión realiza una operación de escritura sobre un objeto x, entonces todos las operaciones de lecturas precedentes sobre el mismo objeto por la misma sesión *deben* ver los efectos de dicha escritura. Por ejemplo, si consideramos la siguiente sesión:

wr x v; **rd** x;

$$\frac{\sigma = \sigma_0 \cdot u \cdot \sigma_1 \qquad \sigma_1 \nwarrow \mathsf{rd} \qquad \sigma \circledcirc \mathsf{rd} \qquad u.x = \mathsf{rd}.x \qquad u.v = \mathsf{rd}.v}{\sigma \xrightarrow{\mathsf{rd},} \sigma[u'.S \mapsto u'.S \cup \{\mathsf{rd}.s\}]_{u'.r=u.r}} \text{(READ)}$$

$$\frac{\sigma \nwarrow \mathsf{rd} \qquad \sigma \circledcirc \mathsf{rd} \qquad \mathsf{rd}.v = \bot}{\sigma \xrightarrow{\mathsf{rd}} \sigma} \text{(READ-EMPTY)}$$

$$\frac{\sigma \circledcirc [x \leftarrow\!\shortmid v]^{\{\}\{r\}}_{s,r}}{\sigma \xrightarrow{[x \leftarrow\!\shortmid v]_{s,r},} \sigma \cdot [x \leftarrow\!\shortmid v]^{\{\}\{r\}}_{s,r}} \text{(UPDATE)}$$

$$\frac{\sigma_0 \circledcirc u[R \mapsto u.R \cup \{r\}] \quad \mathtt{arb}(\sigma_0 \cdot u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1) \quad \sigma_1 \cdot \sigma_2 \equiv \sigma_2 \cdot \sigma_1 \quad \forall u' \in \sigma_2.r \notin u'.R}{\sigma_0 \cdot u \cdot \sigma_1 \cdot \sigma_2 \xrightarrow{\tau,} \sigma_0 \cdot u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1 \cdot \sigma_2} \text{(PROPAGATION)}$$

$$\frac{\sigma \equiv \sigma_0 \quad \sigma_0 \xrightarrow{\beta,} \sigma_0' \quad \sigma_0' \equiv \sigma'}{\sigma \xrightarrow{\beta,} \sigma'} \text{(STR)}$$

Figura 6.1: Semántica operacional para un sistema replicado.

Read Your Writes (RYW)		**Monotonic Reads (MR)**	
$u \nwarrow \mathsf{rd}$ iff $(\mathsf{rd}.x \neq u.x \vee (\mathsf{rd}.s \neq u.s \wedge \mathsf{rd}.r \notin u.R))$	(6.1)	$u \nwarrow \mathsf{rd}$ iff $(u.x \neq \mathsf{rd}.x \vee (\mathsf{rd}.s \notin u.S \wedge \mathsf{rd}.r \notin u.R))$	(6.4)
$\sigma \circledcirc \mathsf{rd}$ iff $(\forall u \in \sigma.\ \mathsf{rd}.s = u.s \implies \mathsf{rd}.r \in u.R)$	(6.2)	$\sigma \circledcirc \mathsf{rd}$ iff $(\forall u \in \sigma.\ \mathsf{rd}.s \in u.S \implies \mathsf{rd}.r \in u.R)$	(6.5)
$\_ \circledcirc \_$ iff **true**	(6.3)	$\_ \circledcirc \_$ iff **true**	(6.6)
Monotonic Writes (MW)		**Writes follow Reads (WFR)**	
$\_ \nwarrow \_$ iff **true**	(6.7)	$\_ \nwarrow \_$ iff **true**	(6.10)
$\_ \circledcirc \_$ iff **true**	(6.8)	$\_ \circledcirc \_$ iff **true**	(6.11)
$\sigma \circledcirc u$ iff $(\forall u' \in \sigma.\ u.s = u'.s \implies u.R \subseteq u'.R)$	(6.9)	$\sigma \circledcirc u$ iff $(\forall u' \in \sigma.\ u.s \in u'.S \implies u.R \subseteq u'.R)$	(6.12)

Figura 6.2: Session guarantees

con solo escribir el valor v al objeto x, y asumiendo que no hay escrituras concurrentes en el sistema, el resultado de la operación sobre x es necesariamente el valor v. Notar que esto no es verdadero para todo sistema replicado. En particular, si una operación de escritura y de lectura son ejecutadas en réplicas diferentes (por ejemplo debido a la caída de un servidor), la lectura debería devolver un valor anterior. Para garantizar RYW, el sistema debe retrasar la operación de lectura hasta que la escritura sea ejecutada en la réplica sobre la que se hace la lectura.

La ecuación 6.1 en Fig. 6.2 establece que un read (rd) puede saltear una escritura (u) cuando ocurre que o

(i) las dos operaciones se realizan sobre objetos distintos, o

(ii) las operaciones son sobre sesiones diferentes y la réplica donde se realiza la lectura no ha recibido aún la escritura.

Causal Arbitration (CA)	**Causal Visibility (CV)**
$\_ \curvearrowleft \_$ iff **true** (6.13)	$u \curvearrowleft \mathsf{rd}$ iff $(u.x \neq \mathsf{rd}.x \vee (\mathsf{rd}.s \neq u.s \wedge \mathsf{rd}.s \notin u.\mathbb{S} \wedge \mathsf{rd}.r \notin u.\mathbb{R}))$ (6.16)
$\_ \circledcirc \_$ iff **true** (6.14)	$\sigma \circledcirc \mathsf{rd}$ iff $\forall u \in \sigma.(\mathsf{rd}.s = u.s \vee \mathsf{rd}.s \in u.\mathbb{S})$ (6.17) $\implies \mathsf{rd}.r \in u.\mathbb{R}$
$\sigma \circledcirc u$ iff $\forall u' \in \sigma.(u.s \in u'.\mathbb{S} \vee u.s = u'.s)$ (6.15) $\implies u.\mathbb{R} \subseteq u'.\mathbb{R}$	$\sigma \circledcirc u$ iff $\forall u' \in \sigma.(u.s \in u'.\mathbb{S} \vee u.s = u'.s)$ (6.18) $\implies u.\mathbb{R} \subseteq u'.\mathbb{R}$
Sequential Consistency (SC)	
$u \curvearrowleft \mathsf{rd}$ iff $u.x \neq \mathsf{rd}.x$ (6.19)	
$\sigma \circledcirc \mathsf{rd}$ iff $\forall u \in \sigma.\ \mathsf{rd}.r \in u.\mathbb{R}$ (6.20)	
$\sigma \circledcirc u$ iff $\forall u' \in \sigma.\ u.\mathbb{R} \subseteq u'.\mathbb{R}$ (6.21)	

Figura 6.3: Causalidad y Consistencia fuerte

Por otra parte, la condición 6.2, para *read-consistency*, requiere que todas las operaciones de lectura nuevas deban ver todas las escrituras realizadas previamente por la misma sesión. Esto se refleja por el hecho que la réplica de la operación de lectura ($\mathsf{rd}.r$) debe ser incluida en el conjunto $u.\mathbb{R}$ de todas las escrituras existentes por la misma sesión que se encuentran en la secuencia σ. Finalmente, no hay restricciones para realizar escrituras.

Considere ahora la codificación para la garantía de sesión llamada **Monotonic Reads** (MR). Esta garantía requiere que una vez que una escritura fue vista por una operación de lectura de una sesión, todas las operaciones de lectura subsiguientes sobre el mismo objeto de dicha sesión deben ver la escritura o un valor más nuevo. La restricción del *skip* en la ecuación 6.4 es casi idéntica a RYW, pero esta vez vamos asumir que una vez que una escritura es visible en una sesión, entonces cualquier lectura sobre un objeto no puede saltear dicha escritura. Más aún, *read-consistency* en la ecuación 6.5 requiere que una vez que una escritura se hace visible a una sesión, todas las réplicas que ejecuten la operación de lectura de dicha sesión deben ser consientes de la escritura también De nuevo, no hay restricciones para el predicado *write consistency*.

Las reglas **Monotonic Writes** (MW) y **Writes follow Reads** (WFR) son similares, y se las dejaremos al lector.

Consideremos ahora los modelos de consistencia más citados en la literatura, presentados en [15]. Estos son resumidos en Fig. 6.3. Considere la codificación del modelo **Causal Arbitration** (CA). En CA, un sistema debe asegurar que cada operación en una sesión s es ordenada después de todas las operaciones que son causalmente dependientes, las cuales incluyen todas las escrituras previamente ejecutadas por s y todas las escrituras vistas por las lecturas realizadas por s. Esto puede no ocurrir ya que en un sistema distribuido con datos replicados, una sesión puede leer un valor previamente escrito en un objeto sobre una réplica y luego escribir sobre una réplica diferente. Luego, dependiendo de la propagación de las escrituras, podría ocurrir que alguna réplica ordene la última

escritura antes que la escritura vista por la operación de lectura; en consecuencia, da la ilusión al que las escrituras fueron ejecutadas en el orden inverso. CA se alcanza en la ecuación 6.15, asegurando que cada escritura $\mathtt{u}$ en una sesión $\mathtt{s}$ se realiza sobre una réplica $\mathtt{u.r}$ que conoce las escrituras $\mathtt{u}'$ previamente ejecutadas por la misma sesión (es decir, $\mathtt{u.s} = \mathtt{u}'.\mathtt{s}$) y todas las escrituras que han sido previamente leídas por la misma sesión (es decir, $\mathtt{u.s} \in \mathtt{u}'.\mathtt{S}$).

La condición $\mathtt{u.R} \subseteq \mathtt{u}'.\mathtt{R}$ asegura que cualquier escritura es conocida por $\mathtt{u.r}$; más aún, todas las operaciones son totalmente ordenadas porque no pueden ser intercambiadas (por la equivalencia de estado).

Causal Visibility (CV) se comporta análogamente con respecto a las escrituras, pero además, impone restricciones sobre las lecturas, las cuales son esencialmente un combinación de RYW, MR, MW, y WFR. La definición de **Sequential Consistency** (SC) asegura que las lecturas y las escrituras son siempre realizadas sobre una réplica que conoce todas las escrituras del sistema, y en consecuencia, son totalmente ordenadas.

A veces, decoraremos las reducciones con un modelo de consistencia para referir a una instanciación particular de dicho modelo $\mathtt{C}$ escribimos $\xrightarrow{\beta}_{\mathtt{C}}$ y $\overset{\mathtt{t}}{\Longrightarrow}_{\mathtt{C}}$.

6.2. Correctitud de la caracterización operacional

En esta sección vamos a mostrar que los modelos presentados en las secciones anteriores de este capítulo son correctas con respecto a la caracterización presentada en la literatura basada en ejecuciones abstractas [15]. Una *ejecución abstracta* es una estructura definida sobre un conjunto de eventos $\mathcal{E}$ en términos de las dos relaciones presentadas en el Capítulo 3: *arbitración* y *visibilidad*, las cuales describen el orden relativo de los eventos y la propagación de los efectos. Eventos son asociados con operaciones sobre un store pero ignoran información sobre réplicas. El conjunto de operaciones en una ejecución abstracta está dada por la siguiente gramática; su significado es inmediato.

$$\delta \quad ::= \quad [\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s}} \mid (\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s}}$$

Escribimos $\alpha \approx \delta$ cuando δ es α luego de quitar información de la réplica, por ejemplo, $[\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s},\mathtt{r}} \approx [\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s}}$. Por abuso de notación, escribimos $[\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s}} \in \mathbb{U}$ y $(\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s}} \in \mathbb{R}$.

A continuación, adaptamos y refinamos la definición de *ejecución abstracta* presentada en Def. 2.6.1 de acuerdo a la sintaxis que utilizamos para describir el modelo de cómputo (operacional) para sistemas replicados.

Definición 6.2.1 (Ejecución abstracta)**.** *Redefinimos una ejecución abstracta como una tupla* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *donde:*

1. $\mathtt{OP} : \mathcal{E} \mapsto \mathbb{U} \cup \mathbb{R}$*;*
2. $\mathtt{VIS} \subseteq \mathcal{E} \times \mathcal{E}$ *es una relación acíclica ;*
3. $\mathtt{AR} \subseteq \mathcal{E} \times \mathcal{E}$ *es un* orden prefijo estricto *tal que* $\mathtt{VIS} \subseteq \mathtt{AR}$.

4. $\mathtt{SO} \subseteq \mathcal{E} \times \mathcal{E}$ *se define tal que* $\mathtt{SO} = \bigcup_{\mathsf{s} \in \mathcal{S}} \mathtt{SO}_\mathsf{s}$*; y* $\mathtt{SO}_\mathsf{s}$ *es un orden total sobre* $\{\mathsf{e} \mid \mathsf{e} \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}).\mathsf{s} = \mathsf{s}\}$ *para todo* s.[1]

Nuestra formalización se desvía de la original en [15] en algunos aspectos. Por razones de simplicidad, la información sobre el valor de retorno de una operación y la sesión que ejecuta una operación son embebidas en $\mathtt{OP}$ (es decir, $\mathtt{OP}(\mathsf{e}).\mathtt{v}$ y $\mathtt{OP}(\mathsf{e}).\mathsf{s}$) en lugar de tener dos funciones distintas. En consecuencia, una clase de equivalencia para los eventos en una misma sesión (*same session*) s es $\mathtt{SS}(\mathsf{s}) = \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \delta \wedge \delta.\mathsf{s} = \mathsf{s}\}$. En nuestro framework, $\mathtt{AR}$ no es necesariamente un orden total; nos permite describir situaciones en las cuales algunas escrituras no han sido arbitradas (por ejemplo, ante la presencia de particiones). Como consecuencia, $\mathtt{SO}$ no se puede obtener de $\mathtt{AR}$ y lo hacemos explícito. Dado que las operaciones son atómicas en nuestro modelo operacional, no consideramos operaciones que no terminan. Como en otros enfoques de la literatura [12, 13, 18], vamos a restringirnos a ejecuciones donde $\mathtt{VIS} \subseteq \mathtt{AR}$ se mantiene, es decir, un evento no puede ser arbitrado antes que los eventos que este vió.

6.2.1. Estados vs Ejecuciones abstractas

En esta sección vamos a mostrar cómo asociar una computación concreta del modelo operacional con la ejecución abstracta. Comenzamos presentando una operación para extender una ejecución abstracta $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ con un evento fresco e (es decir,, $\mathsf{e} \notin \mathcal{E}$) asociado con una operación δ que ve un conjunto de eventos que es *downward closed* $\mathtt{E} \subseteq \mathcal{E}$ tal que $\mathtt{AR}|_\mathtt{E}$ es un orden total ($\_|\_$ representado por la relación estándar de restricción).

$$\mathcal{A} \overset{\mathsf{e} \rhd \delta}{\oplus} \mathtt{E} \;=\; \langle \mathcal{E} \cup \{\mathsf{e}\},\;\; \mathtt{OP}[\mathsf{e} \mapsto \delta] \;\;,\mathtt{SO} \cup (\mathtt{SS}(\delta.\mathsf{s}) \times \{\mathsf{e}\}),\;\; \mathtt{VIS}' \;\;,\mathtt{AR} \cup (\mathtt{E} \times \{\mathsf{e}\}) \rangle$$

$$\mathtt{VIS}' = \begin{cases} \mathtt{VIS} \cup (\mathtt{E} \times \{\mathsf{e}\}) & \text{if } \mathtt{OP}(\mathsf{e}) \in \mathbb{R} \\ \mathtt{VIS} & \text{if } \mathtt{OP}(\mathsf{e}) \in \mathbb{U} \end{cases}$$

La extensión de $\mathcal{E}$ y $\mathtt{OP}$ son auto-explicativas; $\mathtt{SO}$ es extendido con los pares que tienen a e como maximal en la sesión $\delta.\mathsf{s}$. La extensión de $\mathtt{AR}$ es análoga pero extiende un subconjunto $\mathtt{E}$, que se encuentra ordenado totalmente, con un nuevo elemento maximal e. Hay dos casos para visibilidad: $\mathtt{VIS}$ es extendido registrando que e ve todos los eventos en $\mathtt{E}$ cuando e está asociado con una operación de lectura, en el caso contrario, $\mathtt{VIS}$ se mantiene sin cambiar. Elegimos, por simplicidad técnica, no registrar eventos vistos por operaciones de escrituras, observamos que las operaciones de escritura no se ven afectadas por operaciones de escrituras previas. El siguiente resultado muestra que $\_ \overset{\_ \rhd \_}{\oplus} \_$ genera ejecuciones abstractas.

Lema 17. *Si* $\mathcal{A}$ *es una ejecución abstracta, entonces* $\mathcal{A} \overset{\mathsf{e} \rhd \delta}{\oplus} \mathtt{E}$ *es una ejecución abstracta.*

[1] Observar que .s es una proyección mientras que s está cuantificada.

Demostración.

Sea $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ una ejecución abstracta y $\mathtt{E}$ un conjunto de eventos downward-closed, $\mathtt{E} \subseteq \mathcal{E}$ tal que $\mathtt{AR}|_{\mathtt{E}}$ es un orden total. Entonces, definimos

$$\mathcal{A}' = \mathcal{A} \overset{\mathtt{e} \rhd (\mathtt{v} \leftarrow \mathtt{x})_{\mathtt{s}}}{\oplus} \mathtt{E} = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle$$

y queremos demostrar que la propiedad en Def. 6.2.1 se mantiene.

La prueba para los ítems de Def. 6.2.1 es inmediata, excepto por la condición 3. Por lo tanto, vamos a ver que $\mathtt{AR}' = \mathtt{AR} \cup (\mathtt{E} \times \{\mathtt{e}\}))$ es un orden prefijo estricto y $\mathtt{VIS}' \subseteq \mathtt{AR}'$. Notar que $\mathtt{AR}'$ es un orden prefijo si las siguientes propiedades se mantienen:

- (TRANSITIVIDAD). Esto es inmediato ya que $\mathtt{AR}$ y $\mathtt{E}$ son relaciones transitivas.
- (ANTISIMETRÍA). Se cumple porque $\mathtt{AR}'$ es una extensión de $\mathtt{AR}$ con un elemento maximal fresco $\mathtt{e}$.
- (DOWNWARD TOTALITY). Dado que $\mathtt{AR}$ es un orden prefijo, resta probar que la siguiente propiedad se mantiene:

  $$\forall \mathtt{e}', \mathtt{e}'' \in \mathtt{E}.\ (\mathtt{e}', \mathtt{e}) \in \mathtt{AR}' \ \wedge (\mathtt{e}'', \mathtt{e}) \in \mathtt{AR}' \implies (\mathtt{e}', \mathtt{e}'') \in \mathtt{AR}' \vee (\mathtt{e}'', \mathtt{e}') \in \mathtt{AR}'$$

  Esta propiedad se cumple porque $\mathtt{AR}|_{\mathtt{E}}$ es un orden total sobre $\mathtt{E}$ y $\mathtt{AR}|_{\mathtt{E}} \subseteq \mathtt{AR} \subseteq \mathtt{AR}'$. Finalmente, $\mathtt{VIS}' \subseteq \mathtt{AR}'$ se cumple inmediatamente ya que (i) $\mathtt{VIS} \subseteq \mathtt{AR}$ porque $\mathcal{A}$ es una ejecución abstracta, y (ii) $\mathtt{VIS}'$ es igual a $\mathtt{VIS}$ o $\mathtt{VIS} \cup (\mathtt{E} \times \{\mathtt{e}\})$ por definición de $\_ \overset{\_ \rhd \_}{\oplus} \_$.

□

Con el fin de vincular eventos de una ejecución abstracta con operaciones del sistema, vamos a presentar una versión de la semántica decorada, descripta en Fig. 6.1, la cual explícitamente asocia eventos con operaciones. En esta versión, un sistema es una secuencia de pares $\mathtt{e} \rhd \mathtt{u}$, en la cual todos los eventos son diferentes. Vamos a utilizar el siguiente conjunto de etiquetas

$$\lambda \ ::= \ \mathtt{e} \rhd \alpha \qquad\qquad \mu \ ::= \ \tau[\mathtt{e}_0, \mathtt{e}_1, \ldots, \mathtt{e}_k] \mid \lambda$$

donde $\mathtt{e} \rhd \alpha$ decora la acción α con el evento $\mathtt{e}$, y $\tau[\mathtt{e}_0, \mathtt{e}_1, \ldots, \mathtt{e}_k]$ representa una sincronización que ordena totalmente a los eventos $\mathtt{e}_0, \mathtt{e}_1, \ldots, \mathtt{e}_k$. De ahora en más, vamos a usar $[\mathtt{e}_0, \mathtt{e}_1, \ldots, \mathtt{e}_k]$ para denotar de manera más explícita los órdenes totales sobre los eventos $\mathtt{e}_0, \mathtt{e}_1, \ldots, \mathtt{e}_k$

La semántica decorada está dada por las reglas en Fig. 6.4; las primeras tres reglas imitan a las tres primeras en Fig. 6.1 pero adicionalmente asignan a un evento fresco la acción realizada. La regla de propagación es también análoga a la original pero la etiqueta en la transición mantiene un registro de la arbitración introducida por la propagación:

$$\frac{\sigma = \sigma_0 \cdot e' \rhd u \cdot \sigma_1 \qquad \sigma_1 \upharpoonleft rd \qquad \sigma \circledcirc rd \quad u.x = rd.x \qquad u.v = rd.v \qquad e \; \textit{fresco}}{\sigma \xrightarrow{e \rhd rd,} \sigma[u'.S \mapsto u'.S \cup \{rd.s\}]_{u'.r=u.r}} \text{(READ)}$$

$$\frac{\sigma \upharpoonleft rd \qquad \sigma \circledcirc rd \quad rd.v = \bot \quad e \; \textit{fresco}}{\sigma \xrightarrow{e \rhd rd} \sigma} \text{(READ-EMPTY)}$$

$$\frac{\sigma \circledcirc [x \leftarrowtail v]^{\{\}\{r\}}_{s,r} \qquad e \; \textit{fresco}}{\sigma \xrightarrow{[x \leftarrowtail v]_{s,r},} \sigma \cdot e \rhd [x \leftarrowtail v]^{\{\}\{r\}}_{s,r}} \text{(UPDATE)}$$

$$\frac{\mathtt{arb}(\sigma_0 \cdot e \rhd u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1) \qquad \sigma_0 \circledcirc u[R \mapsto u.R \cup \{r\}] \quad \sigma_1 \cdot \sigma_2 \equiv \sigma_2 \cdot \sigma_1 \quad \forall u' \in \sigma_2.r \notin u'.R}{\sigma_0 \cdot e \rhd u \cdot \sigma_1 \cdot \sigma_2 \xrightarrow{\tau[\widetilde{\sigma_0} \cdot e \cdot \widetilde{\sigma_1}],} \sigma_0 \cdot e \rhd u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1 \cdot \sigma_2} \text{(PROPAGATION)}$$

$$\frac{\sigma_0 \equiv \sigma \qquad \sigma \xrightarrow{\lambda,} \sigma' \qquad \sigma' \equiv \sigma'_0}{\sigma_0 \xrightarrow{\lambda,} \sigma'_0} \text{(STR)}$$

Figura 6.4: Semántica operacional decorada

el evento e es arbitrado después de los eventos en σ_0 y antes de los eventos en σ_1 ($\widetilde{\sigma}$ representa la secuencia de eventos en σ).

Ahora presentamos algunas nociones que son instrumentales para asociar sistemas replicados con ejecuciones abstractas. De aquí en más, escribimos $e < e'$ cuando $(e, e') \in \mathtt{AR}$.

Definición 6.2.2 (Estado bien-formado)**.** *Un estado σ es bien-formado, escrito $\mathtt{wf}(\sigma)$, si $\varepsilon \stackrel{t}{\Longrightarrow} \sigma$ para algún t. Escribimos Σ para el conjunto de estados bien-formados.*

La próxima definición da las condiciones para las cuales un estado bien-formado puede ser explicado en términos de una ejecución abstracta.

Definición 6.2.3 (Coherencia)**.** *Sea $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ una ejecución abstracta y σ un estado bien-formado. Diremos que $\mathcal{A}$ es coherente con respecto a σ, escrito $\langle \mathcal{A}, \sigma \rangle$, si:*

1. $|\sigma| = |\{e \in \mathcal{E} \;\; \mathtt{OP}(e) \in \mathbb{U}\}|$, *y*
2. $e \rhd u \in \sigma$ *sii* $e \in \mathcal{E} \wedge \mathtt{OP}(e) = u$, *y*
3. $e < e' \;\wedge\; \mathtt{OP}(e) = u \;\wedge\; \mathtt{OP}(e) = u'$ *sii* $\forall \sigma' \equiv \sigma, \sigma' = \sigma_0 \cdot (e \rhd u) \cdot \sigma_1 \cdot (e' \rhd u') \cdot \sigma_2$, *y*
4. *Para todo* r, $\mathtt{AR}|_{E_r}$ *con* $E_r = \{e \mid e \rhd u \; \in \sigma \wedge \; r \in u.R\}$ *es un orden total.*

Las primeras dos condiciones establecen una correspondencia uno-a-uno entre los eventos asociados y las operaciones de escritura. La tercer condición indica que dos escrituras son arbitradas en $\mathcal{A}$ si y solo si no pueden ser intercambiadas en σ. La última condición requiere que todos los eventos en cada réplica sean totalmente ordenados por AR.

$$\frac{\mathsf{e}\ \textit{fresco} \qquad \sigma \xrightarrow{\mathsf{e}\rhd[\mathtt{x}\leftarrow\mathtt{v}]_{\mathsf{s},\mathsf{r}},} \sigma'}{\langle\sigma,\mathcal{A}\rangle \overset{\cdot}{\rightarrow} \langle\sigma',\mathcal{A} \overset{\mathsf{e}\rhd[\mathtt{x}\leftarrow\mathtt{v}]_{\mathsf{s}}}{\oplus} \mathtt{E}_{\mathsf{r}}\rangle}\ \text{(A-UPDATE)} \qquad \frac{\mathsf{e}\ \textit{fresco} \qquad \sigma \xrightarrow{\mathsf{e}\rhd(\mathtt{v}\leftarrow\mathtt{x})_{\mathsf{s},\mathsf{r}},} \sigma'}{\langle\sigma,\mathcal{A}\rangle \overset{\cdot}{\rightarrow} \langle\sigma',\mathcal{A} \overset{\mathsf{e}\rhd(\mathtt{v}\leftarrow\mathtt{x})_{\mathsf{s}}}{\oplus} \mathtt{E}_{\mathsf{r}}\rangle}\ \text{(A-READ)}$$

$$\frac{\sigma \xrightarrow{\tau[\mathsf{e}_0,\mathsf{e}_1,\ldots,\mathsf{e}_k],} \sigma' \qquad \mathtt{AR}' = \mathtt{AR} \cup [\mathsf{e}_0,\mathsf{e}_1,\ldots,\mathsf{e}_k]}{\langle\sigma,\langle\mathcal{E},\mathtt{OP},\mathtt{SO},\mathtt{VIS},\mathtt{AR}\rangle\rangle \overset{\cdot}{\rightarrow} \langle\sigma',\langle\mathcal{E},\mathtt{OP},\mathtt{SO},\mathtt{VIS},\mathtt{AR}'\rangle\rangle}\ \text{(A-ARBITRATION)}$$

Figura 6.5: Computación de una ejecución abstracta (con $\mathtt{E}_{\mathsf{r}} = \{\mathsf{e} \mid \mathsf{e}\rhd\mathsf{u} \in \sigma \wedge\ \mathsf{r} \in \mathsf{u}.\mathtt{R}\}$)

El siguiente resultado caracteriza la correspondencia entre el evolución de un estado concreto y una ejecución abstracta.

Lema 18. *Sea $\mathcal{A} = \langle\mathcal{E},\mathtt{OP},\mathtt{SO},\mathtt{VIS},\mathtt{AR}\rangle$ una ejecución abstracta y σ un estado bien-formado tal que $\langle\mathcal{A},\sigma\rangle$. Luego,*

1. *Si $\sigma \xrightarrow{\mathsf{e}\rhd\alpha,} \sigma'$, entonces $\langle\mathcal{A} \overset{\mathsf{e}\rhd\delta}{\oplus} \mathtt{E}_{\mathsf{r}},\sigma'\rangle$ con $\alpha \approx \delta$, $\alpha.\mathsf{r} = \mathsf{r}$ y $\mathtt{E}_{\mathsf{r}} = \{\mathsf{e}' \mid \mathsf{e}\rhd\mathsf{u} \in \sigma \wedge \mathsf{r} \in \mathsf{u}.\mathtt{R}\}$.*
2. *Si $\sigma \xrightarrow{\tau[\mathsf{e}_0,\mathsf{e}_1,\ldots,\mathsf{e}_k],} \sigma'$, entonces $\langle\mathcal{A}',\sigma'\rangle$ donde $\mathcal{A}' = \langle\mathcal{E},\mathtt{OP},\mathtt{SO},\mathtt{VIS},\mathtt{AR}'\rangle$ con $\mathtt{AR}' = \mathtt{AR} \cup [\mathsf{e}_0,\mathsf{e}_1,\ldots,\mathsf{e}_k]$.*

El resultado anterior induce el sistema de transición en la Fig. 6.5, que explica la evolución sincronizada de σ y $\mathcal{A}$.

6.2.2. Correctitud de la caracterización operacional

Para cada modelo de consistencia mostramos que la semántica en la Fig. 6.5 genera todas las ejecuciones abstractas que corresponden a la caracterización de la Fig. 2.6 presentada en el Capítulo 2. Vamos a escribir $\emptyset$ para denotar la ejecución abstracta vacía, es decir, una ejecución definida sobre el conjunto de eventos vacío. Dada una ejecución abstracta $\mathcal{A}$, $\mathcal{U}$ (respectivamente, $\mathcal{R}$) denota la restricción del codominio de $\mathtt{OP}$ en $\mathbb{U}$ (respectivamente, $\mathbb{R}$), es decir, $\mathcal{U} = \{\mathsf{e} \mid \mathsf{e} \in \mathcal{E} \ \wedge\ \mathtt{OP}(\mathsf{e}) \in \mathbb{U}\}$ (respectivamente., $\mathcal{R} = \{\mathsf{e} \mid \mathsf{e} \in \mathcal{E} \ \wedge\ \mathtt{OP}(\mathsf{e}) \in \mathbb{R}\}$).

Los siguientes dos resultados muestran que la caracterización para RYW es correcta. Las pruebas pueden encontrarse en Apéndice B.

Lema 19 (Soundness RYW)**.** *Si $\langle\varepsilon,\emptyset\rangle \Rightarrow_{RYW} \langle\sigma,\mathcal{A}\rangle$ entonces $\mathcal{A} = \langle\mathcal{E},\mathtt{OP},\mathtt{SO},\mathtt{VIS},\mathtt{AR}\rangle$ y $\mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.*

Como remarcamos cuando comentamos la definición de $\_ \overset{\rhd}{\oplus} \_$ (figura 6.5), $\mathtt{VIS}$ solo mantiene registro de las escrituras vistas por operaciones de lectura. Por esta razón, calculamos la visibilidad que contienen $\mathtt{SO}|_{\mathcal{U}\times\mathcal{R}}$, las cuales son suficientes para capturar los

modelos de consistencia cuando las lecturas y escrituras son disjuntas, como ocurre en nuestro caso.

Lema 20 (Completeness RYW)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *una ejecución abstracta tal que* $\mathtt{SO} \subseteq \mathtt{VIS}$*. Entonces,*

$$\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle \textit{ tal que} \\ \langle \varepsilon, \emptyset \rangle \Rightarrow_{RYW} \langle \sigma, \mathcal{A}' \rangle,\ \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}},\ \mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}' \textit{ y } \mathtt{AR} \subseteq \mathtt{AR}'$$

La inclusión $\mathtt{AR} \subseteq \mathtt{AR}'$ es verdadera porque permitimos AR ser un prefix orden en lugar de un orden total. (Observamos que la condición colapsa a $\mathtt{AR} = \mathtt{AR}'$ cuando restringimos a órdenes totales.)

Se han desarrollado resultados análogos para el resto de los modelos.

Lema 21 (Soundness)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{\mathtt{C}} \langle \sigma, \mathcal{A} \rangle$ *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y*

$\mathtt{C} =$**MR** $(\mathtt{VIS}; \mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

$\mathtt{C} =$**MW** $\mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}$.

$\mathtt{C} =$**WFR** $\mathtt{VIS}; \mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}$.

$\mathtt{C} =$**CV** $(\mathtt{SO} \cup \mathtt{VIS})^{+}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

$\mathtt{C} =$**CA** $(\mathtt{SO} \cup \mathtt{VIS})^{+}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}$.

$\mathtt{C} =$**SC** $\mathtt{VIS} = \mathtt{AR}$ *y* $\mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$.

Lema 22 (Completeness)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *una ejecución abstracta tal que*

MR: $(\mathtt{VIS};\mathtt{SO}) \subseteq \mathtt{VIS}$. *Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A}' \rangle, \mathtt{SO} \subseteq \mathtt{AR}$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$

MW: $\mathtt{SO} \subseteq \mathtt{AR}$. *Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A}' \rangle, \mathtt{SO} \subseteq \mathtt{AR}$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$

WFR: $(\mathtt{VIS};\mathtt{SO}) \subseteq \mathtt{AR}$. *Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma, \mathcal{A}' \rangle, \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$, $\mathtt{VIS}'; \mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}'$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$

CV: $(\mathtt{SO} \cup \mathtt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$. *Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A}' \rangle, \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$, $(\mathtt{SO} \cup \mathtt{VIS}')^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$

CA: $(\mathtt{SO} \cup \mathtt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$. *Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CA} \langle \sigma, \mathcal{A}' \rangle, \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$, $(\mathtt{SO} \cup \mathtt{VIS}')^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{AR}'$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$

SC: $\mathtt{VIS} = \mathtt{AR}$, $\mathtt{SO} \subseteq \mathtt{VIS}$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$ *es total. Entonces,* $\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR} \rangle$ *tal que* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{SC} \langle \sigma, \mathcal{A}' \rangle$, $\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{VIS}' = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$ *es total.*

Capítulo 7

Hacia la Semántica Denotacional de Programas

La caracterización operacional introducida en el capítulo anterior no sugiere ninguna implementación particular para los modelos de consistencia estudiados en la literatura. Contrariamente, nos permite abstraernos de los detalles concretos de implementación con el fin de brindar una descripción compacta y operacional sobre el comportamiento de un sistema replicado.

A partir de la caracterización operacional anteriormente presentada, en este capítulo capturamos una semántica denotacional de programas que interactúan con sistemas replicados. Para esto, adaptamos y aplicamos el enfoque de Brookes[45], mostrando que la semántica denotacional de los programas es *adecuada*, es decir, la equivalencia de comportamiento se preserva para cualquier contexto. Como es costumbre, derivamos por razonamiento ecuacional, leyes estándar para programas.

7.1. Sintaxis de Programas

Consideramos un lenguaje imperativo, concurrente cuya sintaxis se describe a continuación.

$$
\begin{array}{lll}
E ::= & \textbf{rd}\ \mathtt{x} \mid \mathsf{tt} \mid \mathsf{ff} \mid \neg E \mid \ldots \\
C ::= & \textbf{wr}\ \mathtt{x}\ E \mid \textbf{skip} \mid C;C \mid \textbf{if}\ E\ \textbf{then}\ C\ \textbf{else}\ C \mid \textbf{while}\ E\ \textbf{do}\ C \\
P ::= & \{C\}_{\mathtt{s}} \mid P \parallel P
\end{array}
$$

El lenguaje para las expresiones E incluye operaciones de lectura **rd** x y tipos de datos básicos y operadores básicos: por simplicidad, solo mostramos constantes booleanas tt y ff y negación $\neg E$. La sintaxis para los comandos es estándar, usamos **wr** x E para escribir x con el resultado de la evaluación de E. El programa más simple P es $\{C\}_{\mathtt{s}}$, el cual denota un comando C corriendo sobre la sesión s. Programas pueden ser compuestos en paralelo. Escribimos $\mathtt{s}(P)$ para denotar el conjunto de sesiones identificadas en P. Asumimos que todas las sesiones identificadas en un programa deben ser diferentes, es decir, $\mathtt{s}(P_1) \cap \mathtt{s}(P_2) = \emptyset$ en $P_1 \parallel P_2$.

$$\text{(RD)}\ \frac{\sigma \xrightarrow{(\mathtt{v}\leftarrow\mathtt{x})_{\mathtt{s},\mathtt{r}},} \sigma'}{\{\mathbf{rd}\ \mathtt{x}\}_{\mathtt{s}},\sigma \to \{\mathtt{v}\}_{\mathtt{s}},\sigma'} \qquad \text{(NOTCTX)}\ \frac{\{E\}_{\mathtt{s}},\sigma \to \{E'\}_{\mathtt{s}},\sigma'}{\{\neg E\}_{\mathtt{s}},\sigma \to \{\neg E'\}_{\mathtt{s}},\sigma'} \qquad \text{(NOTTRUE)}\ \frac{}{\{\neg \mathtt{tt}\}_{\mathtt{s}},\sigma \to \{\mathtt{ff}\}_{\mathtt{s}},\sigma}$$

$$\text{(NOTFALSE)}\ \frac{}{\{\neg \mathtt{ff}\}_{\mathtt{s}},\sigma \to \{\mathtt{tt}\}_{\mathtt{s}},\sigma} \qquad \text{(WRCTXT)}\ \frac{\{E\}_{\mathtt{s}},\sigma \to \{E'\}_{\mathtt{s}},\sigma'}{\{\mathbf{wr}\ \mathtt{x}\ E\}_{\mathtt{s}},\sigma \to \{\mathbf{wr}\ \mathtt{x}\ E'\}_{\mathtt{s}},\sigma} \qquad \text{(WR)}\ \frac{\sigma \xrightarrow{[\mathtt{x}\leftarrow\mathtt{v}]_{\mathtt{s}},\mathtt{r},} \sigma'}{\{\mathbf{wr}\ \mathtt{x}\ \mathtt{v}\}_{\mathtt{s}},\sigma \to \{\mathbf{skip}\}_{\mathtt{s}},\sigma'}$$

$$\text{(SEQ-1)}\ \frac{\{C_1\}_{\mathtt{s}},\sigma \to \{C_1'\}_{\mathtt{s}},\sigma'}{\{C_1;C_2\}_{\mathtt{s}},\sigma \to \{C_1';C_2\}_{\mathtt{s}},\sigma'} \qquad \text{(SEQ-2)}\ \frac{}{\{\mathbf{skip};C\}_{\mathtt{s}},\sigma \to \{C\}_{\mathtt{s}},\sigma}$$

$$\text{(IFCTXT)}\ \frac{\{E\}_{\mathtt{s}},\sigma \to \{E'\}_{\mathtt{s}},\sigma'}{\{\mathbf{if}\ E\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\}_{\mathtt{s}},\sigma \to \{\mathbf{if}\ E'\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\}_{\mathtt{s}},\sigma'}$$

$$\text{(IF-TRUE)}\ \frac{}{\{\mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\}_{\mathtt{s}},\sigma \to \{C_1\}_{\mathtt{s}},\sigma} \qquad \text{(IF-FALSE)}\ \frac{}{\{\mathbf{if}\ \mathtt{ff}\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\}_{\mathtt{s}},\sigma \to \{C_2\}_{\mathtt{s}},\sigma}$$

$$\text{(WHILE)}\ \frac{}{\{\mathbf{while}\ E\ \mathbf{do}\ C\}_{\mathtt{s}},\sigma \to \{\mathbf{if}\ E\ \mathbf{then}\ (C;\mathbf{while}\ E\ \mathbf{do}\ C)\ \mathbf{else}\ \mathbf{skip}\}_{\mathtt{s}},\sigma}$$

$$\text{(SYNC)}\ \frac{\sigma \xrightarrow{\tau,} \sigma'}{P,\ \sigma \to P,\ \sigma'} \qquad \text{(PARL)}\ \frac{P_1,\ \sigma \to P_1',\ \sigma'}{P_1 \parallel P_2,\ \sigma \to P_1' \parallel P_2,\ \sigma'} \qquad \text{(PARR)}\ \frac{P_2,\ \sigma \to P_2',\ \sigma'}{P_1 \parallel P_2,\ \sigma \to P_1 \parallel P_2',\ \sigma'}$$

Figura 7.1: Semántica operacional de programas

La semántica operacional de programas está dada por un sistema estándar de transición sobre configuraciones $P \times \Sigma$. En Fig. 7.1 reportamos las reglas operacionales para interactuar con el sistema a través de las reglas (RD) y (WR), y para la sincronización, mediante la regla (SYNC). El resto de las primitivas son estándar. Remarcamos que la semántica es definida sobre estados bien-formados $\sigma \in \Sigma$ (ver Def. 6.2.2).

7.2. Semántica denotacional

Seguimos el enfoque de Brookes [45] para dar la semántica denotacional a programas. En consecuencia, la ejecución de un comando C que es interrumpido k veces es descripto por una secuencia finita $(\sigma_0,\sigma_0')(\sigma_1,\sigma_1')\ldots(\sigma_k,\sigma_k') \in \Sigma \times \Sigma$, llamada *transition trace*,

en la cual la interrupción iésima cambia el estado σ'_{i-1} en σ_i y la ejecución de C entre las interrupciones i^{th} y $i+1^{th}$ cambia el estado de σ_i a σ'_i. Más aún, interrupciones pueden solo realizar acciones sobre sesiones diferentes, desde una en la cual C corre porque las sesiones consistente de una secuencia de comandos. Por esta razón, adicionalmente vamos a requerir que la interrupción i^{th} para transformar σ'_{i-1} en σ_i por medio de las acciones sobre sesiones diferentes de C. Para un conjunto de sesiones identificadas como $\mathtt{S}$, escribimos $(\sigma_0,\sigma'_0)(\sigma_1,\sigma'_1)\dots(\sigma_k,\sigma'_k)_{\mathtt{S}}$ cuando σ'_i puede avanzar a σ_{i+1} realizando acciones con identificadores de sesión que no estén en $\mathtt{S}$, es decir, $\forall i.\sigma'_i \overset{\mathtt{t}_i}{\Longrightarrow}_{\neg\mathtt{S}} \sigma_{i+1}$. Adicionalmente consideramos trazas que comienzan sobre estados bien-formados. La función $\mathcal{T}[\![\_]\!]_{\mathtt{s}} : C \to \mathcal{P}(\Sigma\times\Sigma)^*$ definida a continuación asocia cada comando C con sus trazas completas C corriendo sobre la sesión $\mathtt{s}$:

$$\begin{aligned}\mathcal{T}[\![C]\!]_{\mathtt{s}} = \{(\sigma_0,\sigma'_0)\dots(\sigma_n,\sigma'_n)_{\{\mathtt{s}\}} \mid \quad & \wedge \quad \forall k\in[0,n-1].\\ & \{C_k\}_{\mathtt{s}},\sigma_k \to^* \{C_{k+1}\}_{\mathtt{s}},\sigma'_k \quad \wedge \quad C_0 = C \quad \wedge \quad C_n = \mathbf{skip}\}\end{aligned}$$

Sea $\alpha,\beta,\dots$ para identificar elementos de $(\Sigma\times\Sigma)^*$ y escribimos α en lugar de $\alpha_{\mathtt{S}}$ cuando el conjunto de sesiones identificadas no sea importante.

Definición 7.2.1. *Dado un conjunto de trazas T, la clausura de T bajo* stuttering *y* mumbling *es el conjunto más chico $T^\dagger$ que incluye T y satisface la siguientes reglas*

$$\frac{\alpha\beta_{\mathtt{S}} \in T^\dagger}{\alpha(\sigma,\sigma)\beta_{\mathtt{S}} \in T^\dagger}\ (\text{STUTTERING}) \qquad \frac{\alpha(\sigma,\sigma')(\sigma',\sigma'')\beta_{\mathtt{S}} \in T^\dagger}{\alpha(\sigma,\sigma'')\beta_{\mathtt{S}} \in T^\dagger}\ (\text{MUMBLING})$$

Lema 23. *$\mathcal{T}[\![C]\!]_{\mathtt{s}} = (\mathcal{T}[\![C]\!]_{\mathtt{s}})^\dagger$ para todo C.*

Demostración. Sea $\alpha = \alpha'(\sigma_j,\sigma'_j)$ y $\beta = (\rho_0,\rho'_0)\beta'$.

- STUTTERING. Si $\alpha\beta_{\{\mathtt{s}\}} \in T^\dagger$, entonces

  $$\{C_j\}_{\mathtt{s}},\sigma_j \to^* \{C_{j+1}\}_{\mathtt{s}},\sigma'_j \qquad \text{y} \qquad \{C_{j+1}\}_{\mathtt{s}},\rho_0 \to^* \{C_{j+1}\}_{\mathtt{s}},\rho'_0$$

  Más aún, $\{C_{j+1}\}_{\mathtt{s}},\sigma \to^* \{C_{j+1}\}_{\mathtt{s}},\sigma$ ya que por reflexividad de $\to^*$. Para todo traza bien definida $\alpha(\sigma,\sigma)\beta_{\{\mathtt{s}\}} \in T^\dagger$, tenemos $\sigma'_j \overset{s'}{\Rightarrow}_{\neg\{\mathtt{s}\}} \sigma$ y $\sigma \overset{s}{\Rightarrow}_{\neg\{\mathtt{s}\}} \rho_0$. Por lo tanto, $\alpha(\sigma,\sigma)\beta_{\{\mathtt{s}\}} \in T^\dagger$.

- MUMBLING. De $\alpha(\sigma,\sigma')(\sigma',\sigma'')\beta_{\{\mathtt{s}\}} \in T^\dagger$,

  $$\{C_{j+1}\}_{\mathtt{s}},\sigma \to^* \{C_{j+2}\}_{\mathtt{s}},\sigma' \qquad \text{y} \qquad \{C_{j+2}\}_{\mathtt{s}},\sigma' \to^* \{C_{j+3}\}_{\mathtt{s}},\sigma''$$

  Por transitividad de $\to^*$, $\{C_{j+1}\}_{\mathtt{s}},\sigma \to^* \{C_{j+3}\}_{\mathtt{s}},\sigma''$. En consecuencia $\alpha(\sigma,\sigma'')\beta_{\{\mathtt{s}\}} \in T^\dagger$

□

La siguiente operación en conjuntos de trazas es instrumental para la caracterización denotacional de $\mathcal{T}[\![\_]\!]_{\mathtt{s}}$. Sea T_1 y T_2 dos conjuntos de trazas cerrados, entonces su concatenación es definida como sigue

$$T_1 \overset{\mathtt{s}}{;} T_2 = \{\alpha\beta_{\{\mathtt{s}\}} \mid \alpha \in T_1,\ \beta \in T_2\}^{\dagger}$$

Concatenación es paramétrica con respecto a un identificador de sesión $\mathtt{s}$, la cual es importante para asegurar que el última estado en α puede avanzar a un primer estado en β ejecutando operaciones sobre sesiones diferentes de $\mathtt{s}$. La clausura de Kleene T^* sobre conjunto de trazas cerradas T es definida de la manera estándar como el conjunto más chico cerrado sobre stuttering, mumbling y concatenación, conteniendo a T y la traza vacía.

La caracterización denotacional de $\mathcal{T}[\![\_]\!]_{\mathtt{s}}$ está dada por las ecuaciones en figura 7.2. Para expresiones, tenemos $(\!|\_|\!)_{\mathtt{s}} : E \to \mathcal{P}((\Sigma \times \Sigma)^* \times \mathcal{V})$, las cuales asocian a cada expresión con un conjunto de pares, cada par representa una *transition trace* y el valor computado asociado a dicha traza. La definición sigue el enfoque estándar pero para **rd** $\mathtt{x}$, cuya evaluación puede tener efectos colaterales. Por ejemplo, $(\!|\_|\!)_{\mathtt{s}}$ establece que la evaluación de constantes booleanes $\mathtt{tt}$ y $\mathtt{ff}$ es atómica y no produce ningún efecto colateral sobre el sistema. La definición $(\!|\_|\!)_{\mathtt{s}}$ para **rd** $\mathtt{x}$ representa el caso en el que el valor de retorno no está definido y aquel en el que se evalúa a un valor escrito previamente. En el segundo caso, la evaluación también cambia σ en $\sigma[\mathtt{u}.\mathtt{S} \mapsto \mathtt{u}.\mathtt{S} \cup \{\mathtt{s}\}]_{\mathtt{r} \in \mathtt{u}.\mathtt{R}}$. Notar que la denotación de **rd** $\mathtt{x}$ (analogamente, que una de **wr** $\mathtt{x}$ E) es paramétrica con respecto al modelo de consistencia porque este es definido en términos de $\circledcirc$.

La denotación para una expresión compuesta es definido en términos de sus subexpresiones (ilustramos la definición de $\neg E$). Para una expresión booleana E, escribimos $[\![E]\!]_{\mathtt{s}}$ para el conjunto $\{\alpha \mid (\alpha, \mathtt{tt}) \in (\!|E|\!)_{\mathtt{s}}\}$.

Para programas, tenemos $[\![\_]\!]_{\mathtt{s}} : C \to \mathcal{P}((\Sigma \times \Sigma)^*)$, los cuales asocian a cada programa con su conjunto de transition traces; su definición es estándar.

Con el fin de establecer que las ecuaciones en Fig. 7.2 caracterizan $\mathcal{T}[\![\_]\!]_{\mathtt{s}}$, definimos el Lem. 25. Sin embargo, para la prueba, necesitamos la siguiente definición auxiliar para trazas de una expresión E.

$$\begin{aligned}\mathcal{T}[\![E]\!]_{\mathtt{s}} = \{((\sigma_0, \sigma_0') \ldots (\sigma_n, \sigma_n'), \mathtt{v})_{\mathtt{s}} \mid \quad & \mathtt{wf}(\sigma_0) \quad \wedge \quad \forall k \in [0, n-1]. \\ & \{E_k\}_{\mathtt{s}}, \sigma_k \to^* \{E_{k+1}\}_{\mathtt{s}}, \sigma_k' \quad \wedge \quad E_0 = E \quad \wedge \quad E_n = \mathtt{v}\}\end{aligned}$$

y el siguiente resultado auxiliar.

Lema 24. *Para cada programa E, $\mathcal{T}[\![E]\!]_{\mathtt{s}} = (\!|E|\!)_{\mathtt{s}}$.*

Demostración. Es inmediato por inducción en la estructura de E. □

Lema 25. *Para cada programa C, $\mathcal{T}[\![C]\!]_{\mathtt{s}} = [\![C]\!]_{\mathtt{s}}$.*

Demostración. Haremos inducción en la estructura de C.

$$
\begin{array}{lcl}
(\!|\mathtt{tt}|\!)_{\mathtt{s}} & = & \{((\sigma,\sigma),\mathtt{tt}) \mid \sigma \in \Sigma\}^{\dagger} \\
(\!|\mathtt{ff}|\!)_{\mathtt{s}} & = & \{((\sigma,\sigma),\mathtt{ff}) \mid \sigma \in \Sigma\}^{\dagger} \\
(\!|\mathbf{rd}\ \mathtt{x}|\!)_{\mathtt{s}} & = & \{((\sigma,\sigma[\mathtt{u.S} \mapsto \mathtt{u.S} \cup \{\mathtt{s}\}]_{\mathtt{r}\in\mathtt{u.R}}),\bot) \mid \sigma \in \Sigma,\ \sigma \upharpoonleft (\bot \leftarrowtail \mathtt{x})_{\mathtt{s,r}},\ \sigma \circledcirc (\bot \leftarrowtail \mathtt{x})_{\mathtt{s,r}}\}^{\dagger} \\
 & & \cup \\
 & & \{((\sigma,\sigma[\mathtt{u.S} \mapsto \mathtt{u.S} \cup \{\mathtt{s}\}]_{\mathtt{r}\in\mathtt{u.R}}),\mathtt{v}) \mid \sigma \in \Sigma,\ \sigma \circledcirc (\mathtt{v} \leftarrowtail \mathtt{x})_{\mathtt{s,r}},\ \sigma_1 \upharpoonleft \mathsf{rd}, \\
 & & \qquad \sigma = \sigma_0 \cdot \mathtt{u} \cdot \sigma_1,\ \mathtt{u.x} = \mathtt{x},\ \mathtt{u.v} = \mathtt{v}\}^{\dagger} \\
(\!|\neg E|\!)_{\mathtt{s}} & = & \{(\alpha,\overline{\mathtt{v}}) \mid (\alpha,\mathtt{v}) \in (\!|E|\!)_{\mathtt{s}}\},\ \text{where}\ \ \overline{\mathtt{tt}} = \mathtt{ff}, \overline{\mathtt{ff}} = \mathtt{tt} \\
[\![\mathbf{skip}]\!]_{\mathtt{s}} & = & \{(\sigma,\sigma) \mid \sigma \in \Sigma\}^{\dagger} \\
[\![\mathbf{wr}\ \mathtt{x}\ E]\!]_{\mathtt{s}} & = & \{\alpha(\sigma,\sigma\cdot[\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\}\{\mathtt{r}\}}_{\mathtt{s,r}})_{\{\mathtt{s}\}} \mid \sigma \in \Sigma,\ (\alpha,\mathtt{v}) \in (\!|E|\!)_{\mathtt{s}},\ \sigma \circledcirc [\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\mathtt{s}\}\{\mathtt{r}\}}_{\mathtt{s,r}}\}^{\dagger} \\
[\![C_1;C_2]\!]_{\mathtt{s}} & = & [\![C_1]\!]_{\mathtt{s}} \overset{\mathtt{s}}{;} [\![C_2]\!]_{\mathtt{s}} \\
[\![\mathbf{if}\ E\ \mathbf{then}\ C\ \mathbf{else}\ C']\!]_{\mathtt{s}} & = & [\![E]\!]_{\mathtt{s}} \overset{\mathtt{s}}{;} [\![C]\!]_{\mathtt{s}} \cup [\![\neg E]\!]_{\mathtt{s}} \overset{\mathtt{s}}{;} [\![C']\!]_{\mathtt{s}} \\
[\![\mathbf{while}\ E\ \mathbf{do}\ C]\!]_{\mathtt{s}} & = & ([\![E]\!]_{\mathtt{s}} \overset{\mathtt{s}}{;} [\![C]\!]_{\mathtt{s}})^{*} \overset{\mathtt{s}}{;} [\![\neg E]\!]_{\mathtt{s}}
\end{array}
$$

Figura 7.2: Semántica denotacional de expresiones y programas

- **skip**. $[\![\mathbf{skip}]\!]_{\mathtt{s}} = \{(\sigma,\sigma) \mid \sigma \in \mathbb{U}^*\}^{\dagger} = \mathcal{T}[\![\mathbf{skip}]\!]^{\dagger}_{\mathtt{s}} = \mathcal{T}[\![\mathbf{skip}]\!]_{\mathtt{s}}$, el último paso se sigue inmediatamente por Lem. 23.

- **wr** x E.

$$
\begin{array}{lll}
 & [\![\mathbf{wr}\ \mathtt{x}\ E]\!]_{\mathtt{s}} & \\
= & \{\alpha(\sigma,\sigma\cdot[\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\}\{\mathtt{r}\}}_{\mathtt{s,r}})_{\{\mathtt{s}\}} \mid \sigma \in \mathbb{U}^*,\ (\alpha,\mathtt{v}) \in (\!|E|\!)_{\mathtt{s}}, \sigma \circledcirc [\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\mathtt{s}\}\{\mathtt{r}\}}_{\mathtt{s,r}}\}^{\dagger} & \text{by def.} \\
= & \{\alpha(\sigma,\sigma\cdot[\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\}\{\mathtt{r}\}}_{\mathtt{s,r}})_{\{\mathtt{s}\}} \mid \sigma \in \mathbb{U}^*,\ (\alpha,\mathtt{v}) \in \mathcal{T}[\![E]\!]_{\mathtt{s}}, \sigma \circledcirc [\mathtt{x} \leftarrowtail \mathtt{v}]^{\{\mathtt{s}\}\{\mathtt{r}\}}_{\mathtt{s,r}}\}^{\dagger} & \text{by Lem. 24} \\
= & \mathcal{T}[\![\mathbf{wr}\ \mathtt{x}\ E]\!]^{\dagger}_{\mathtt{s}} & \text{by def.} \\
= & \mathcal{T}[\![\mathbf{wr}\ \mathtt{x}\ E]\!]_{\mathtt{s}} & \text{por Lem. 23}
\end{array}
$$

El resto de los casos son inmediatos. □

La noción de *transition trace* se extiende a los programas de la siguiente manera.

$$
\begin{array}{l}
\mathcal{T}[\![P]\!] = \{(\sigma_0,\sigma'_0)\ldots(\sigma_n,\sigma'_n)_{\mathtt{s}(S)} \mid \quad \mathtt{wf}(\sigma_0) \quad \wedge \quad \forall k \in [0,n-1]. \\
\qquad\qquad P_k,\ \sigma_k \rightarrow^* P_{k+1},\ \sigma'_k \quad \wedge \quad P_0 = P \quad \wedge \quad P_n = \|_{\mathtt{s}\in\mathtt{s}(S)}\ \{\mathbf{skip}\}_{\mathtt{s}}\}
\end{array}
$$

También, Lem. 23 puede ser sencillamente extendido para programas, es decir, $\mathcal{T}[\![P]\!] = (\mathcal{T}[\![P]\!])^{\dagger}$. Para la caracterización denotacional de $\mathcal{T}[\![P]\!]$ definimos el interleaving de trazas respecto al conjunto de identificaciones de sesión S.

$$
\begin{array}{lll}
\alpha \|_{\mathtt{S}} \varepsilon = \{\alpha\} & & a\alpha \|_{\mathtt{S}} b\beta = \{a\gamma_{\mathtt{S}} \mid \gamma \in \alpha \|_{\mathtt{S}} b\beta\} \cup \{b\gamma_{\mathtt{S}} \mid \gamma \in a\alpha \|_{\mathtt{S}} \beta\} \\
\varepsilon \|_{\mathtt{S}} \alpha = \{\alpha\} & & T_1 \|_{\mathtt{S}} T_2 = \bigcup \{\alpha \|_{\mathtt{S}} \beta \mid \alpha \in T_1, \beta \in T_2\}^{\dagger}
\end{array}
$$

Luego, la denotación de los programas viene dada por las siguientes dos ecuaciones:

$$
[\![\{C\}_{\mathtt{s}}]\!] = [\![C]\!]_{\mathtt{s}} \qquad\qquad [\![P_1 \parallel P_2]\!] = [\![P_1]\!] \|_{\mathtt{s}(P_1\parallel P_2)} [\![P_2]\!]
$$

Lema 26. *Para cada programa P,* $\mathcal{T}[\![P]\!] = [\![P]\!]$.

El resto de esta sección está dedicada a mostrar la adecuación de la semántica denotacional. La semántica denotacional induce un orden en los comandos debido a la inclusión de trazas, y tal orden explica la sustitutibilidad.

Definición 7.2.2. $C_1 \sqsubseteq C_2$ *si y solo para toda sesión* s, $[\![C_1]\!]_\mathsf{s} \subseteq [\![C_2]\!]_\mathsf{s}$.

El comportamiento entrada/salida de un programa P puede ser caracterizado en términos de sus trazas ininterrumpidas, como describimos a continuación.

Definición 7.2.3 (Comportamiento entrada/salida)**.** *Para cada programa P,*

$$\mathtt{IO}[\![P]\!] = \{(\sigma,\sigma') \mid (\sigma,\sigma') \in \mathcal{T}[\![P]\!]\}.$$

El comportamiento entrada/salida induce un orden sobre los comandos de acuerdo a la capacidad de los programas de distinguirlos. Considere el siguiente contexto

$$\mathbb{C} ::= \{[\,]\}_\mathsf{s} \;\mid\; \mathbb{C} \,\|\, P \;\mid\; P \,\|\, \mathbb{C}$$

y escribimos $\mathbb{C}[C]$ para denotar el término obtenido por sustituir $[\,]$ por C en $\mathbb{C}$. Luego,

Definición 7.2.4 (Orden Operacional)**.** $C_1 \sqsubseteq_{\mathtt{IO}} C_2$ *sii* $\forall \mathbb{C}.\mathtt{IO}[\![\mathbb{C}[C_1]]\!] \subseteq \mathtt{IO}[\![\mathbb{C}[C_2]]\!]$.

El siguiente resultado, que es inmediato por la definición de $[\![\_]\!]_\mathsf{s}$, asegura que el orden inducido por las denotaciones se conserva por orden de entrada/salida.

Lema 27 (Monotonicidad Composicional)**.** *Para todo* C_1 *y* C_2 *se mantiene que* $C_1 \sqsubseteq C_2 \implies \forall \mathbb{C}.\mathbb{C}[C_1] \sqsubseteq \mathbb{C}[C_2]$

Corolario 1 (Adecuación)**.** *Para todo* C_1 *y* C_2 *se cumple que* $C_1 \sqsubseteq C_2 \implies \forall \mathbb{C}.\mathbb{C}[C_1] \sqsubseteq_{\mathtt{IO}} \mathbb{C}[C_2]$

En el framework original de Brookes [45], se muestra que la caracterización denotacional es (inequitativamente) completamente abstracta, es decir, lo contrario de Cor. 1 se cumple. Ese resultado se basa en el comando `await`, ofrecido por el lenguaje, y que puede atómicamente (i) verificar el estado de todo store y también (ii) cambiar múltiples variables. No consideramos este tipo de primitivas en nuestro lenguaje porque no son realistas en un escenario distribuido.

Ejemplo 7.2.1. *La semántica denotacional nos permite mostrar que las leyes estándar para programas se mantienen en nuestro framework (para cada modelo de consistencia). En particular,*

$$\textbf{skip};C \equiv C \equiv C;\textbf{skip} \tag{7.1}$$

$$(C_1;C_2);C_3 \equiv C_1;(C_2;C_3) \tag{7.2}$$

$$\textbf{if } E \textbf{ then } C_1 \textbf{ else } C_2;C \equiv \textbf{ if } E \textbf{ then } C_1;C \textbf{ else } C_2;C \tag{7.3}$$

$$\textbf{while } E \textbf{ do } C \equiv \textbf{ if } E \textbf{ then } C;\textbf{while } E \textbf{ do } C \textbf{ else skip} \tag{7.4}$$

Ley 7.1 establece que **skip** *es la identidad (a izquierda o derecha) para composición secuencial* $\overset{\mathtt{s}}{;}$ *y esto es a causa de la definición de composición secuencial y el hecho que las trazas son cerradas para stuttering y mumbling. Ley 7.2 dice que la composición* $\overset{\mathtt{s}}{;}$ *es asociativa, mientras que el resto de las dos reglas son obtenidas por razonamiento ecuacional.*

Además de permitir la derivación de leyes para programas como en otros entornos, podemos movernos a otros problemas específicos de modelos de consistencia en el marco denotacional. Por ejemplo, un comando C se dice *robusto* si se comporta como operando en un store secuencial incluso ejecutándose en un modelo más débil $\mathtt{C}$, que se puede formalizar de la siguiente manera: P es robusto en $\mathtt{C}$ si $\forall \mathbb{C}.\mathbb{C}[C]_{\mathtt{C}} \sqsubseteq_{\mathtt{IO}} \mathbb{C}[C]_{SC}$ (donde los subíndices $\mathtt{C}$ y SC indican el modelo de consistencia utilizado para calcular las denotaciones).

Capítulo 8

Trabajos Relacionados

Tanto los tipos de datos replicados, como los modelos de consistencia son temas relevante en varias comunidades o disciplinas de la ciencias de la computación. A continuación describiremos las conexiones más importantes con las que nos encontramos en la literatura, discriminando por áreas de investigación.

8.1. Métodos Formales

El área de métodos formales involucra, entre otros temas, la *verificación*, es decir, separar el qué (la especificación) del cómo (la implementación). Por lo tanto, especificar y verificar sistemas distribuidos y replicados es un tema recurrente del área. Concretamente, al hablar de verificación podemos centrarnos en (i) probar que los estados alcanzables o las ejecuciones satisfacen alguna propiedad; o (ii) probar que las ejecuciones son observacionalmente equivalentes a alguna ejecución ofrecida por una semántica operacional.

Diferentes líneas de trabajo han abordado el problema de especificar e implementar tipos de datos replicados [6, 8, 19, 46, 47, 48, 49], considerando incluso a los que requieren políticas para la resolución de conflictos [8, 29, 36]. El enfoque presentado en [8] se ha estudiado en detalle en el Capítulo 3, notando que todas sus especificaciones definen implícitamente una estrategia particular para resolver conflictos. Por el contrario, nuestras especificaciones funcionales proporcionan una visión más abstracta de los RDTs, es decir, no se comprometen con una estrategia particular para resolver conflictos cuando ocurren escrituras concurrentes. La correctitud de una implementación está caracterizada en [8] en términos de *replication-aware simulations* que al igual que en [50], asocian estados concretos de una implementación con su descripción abstracta, en este caso, estructuras de eventos. Las ejecuciones abstractas son decoradas con información como *time-stamps*, los cuales son definidos ad-hoc para cada tipo de dato. La correctitud sobre el comportamiento de varias réplicas es derivado de una propiedad que impone un conjunto de *proof obligations* que deben ser revisadas. Contrariamente, tal como mostramos en el Capítulo 5, nuestra caracterización funcional induce una interpretación operacional por lo que nuestras pruebas de correctitud se reducen a pruebas estándar de simulación sobre LTSs.

En [29], se ha propuesto un enfoque diferente para lidiar con sistemas de replicación optimistas. En este caso, una especificación asocia una operación y un valor de retorno al conjunto de todas las ejecuciones posibles (representadas en términos de órdenes parciales) que explican ese valor de retorno particular para esa operación. Además, permiten asociar diferentes operaciones con diferentes órdenes parciales. De esta manera, tratan con sistemas especulativos, moviéndose de el problema de verificar la consistencia eventual a un problema de *model checking* y *reachability*.

En [36] presenta un framework para verificar automáticamente la convergencia de los CRDTs(*conflict-free replicated data types*) bajo diferentes políticas de consistencia. Muestran que no todas las operaciones de un CRDT necesitan conmutar para lograr la convergencia. Finalmente desarrollan una *proof rules* parametrizada por la especificación de consistencia.

Por otro lado, hasta donde sabemos, el único marco general para dar semántica operacional a sistemas débilmente consistentes se ha propuesto en [12], pero trata con modelos transaccionales. Diferentemente, la presentación en el Capítulo 6 se centra en modelos no transaccionales. A pesar de las diferencias en los modelos abordados, los enfoques también difieren en su estilo. El marco en [12] combina un modelo operacional que sobreaproxima las trazas de ejecución de un modelo de consistencia. Estos utilizan *restricciones específicas del modelo*, para filtrar aquellas trazas que no deberían alcanzarse para un modelo particular. Por el contrario, nuestra caracterización es puramente operacional, y nuestro modelo solo genera las trazas alcanzables por cada modelo de consistencia.

En cuanto a la verificación de los tipos de datos replicados (conmutativos), se ha propuesto un framework en Isabelle/HOL en [51]. Otras líneas de trabajo [9, 28, 33] se han centrado al problema de verificar las propiedades o invariantes de las aplicaciones que usan datos replicados. A la larga, nuestro objetivo es explotar las herramientas y técnicas de la teoría de la simulación para estos fines.

Finalmente, es importante resaltar que si bien los órdenes parciales, como las estructuras de eventos [52], han sido utilizados para definir la semántica de programas concurrentes y distribuidos, nuestro trabajo de tesis considera las relaciones presentadas en Def. 2.6.1, que resultan ser más adecuadas para el propósito de este trabajo.

8.2. Sistemas Distribuidos

Términos como *replicación*, *propagación asincrónica* y *resolución de conflictos* son referenciados comúnmente en el área de sistemas distribuídos.

En 1994, Terry at al. presentan un sistema de almacenamiento llamado *Bayou* [53] cuya arquitectura posibilita el desarrollo de aplicaciones colaborativas haciendo foco en mecanismos de detección y resolución de conflictos sobre escrituras concurrentes, asegurando de esta manera, que las réplicas del sistema convergen a un mismo estado y así garantizar consistencia eventual. El desarrollo de este tipo de sistemas significó la formulación de propiedades desde el punto de vista de los clientes, llamadas *session guarantees* introducidas en [44] y estudiadas en numerosos trabajos como [53, 54, 55, 56, 57, 58].

Un hito importante del área fue la conjetura de CAP, presentada por Brewer en el año 2000 [59], y demostrada en [1] afirmando la imposibilidad de alcanzar alta disponibilidad, consistencia fuerte y tolerancia a la partición simultáneamente. Entre sus implicancias, nos encontramos con el desarrollo de bases de datos no relacionales (conocidas como *key-value store*) como *Dynamo*, *Cassandra*, *Spanner* y *MongoDB* ([2, 3, 16, 60]), entre otras, que muestran la utilidad de trabajar con sistemas escalables, altamente disponibles a expensas de ofrecer una noción más débil de consistencia. Por consiguiente, los últimos años hemos visto un desplazamiento al desarrollo de aplicaciones NoSQL[61] implementadas sobre infraestructuras *cloud*.

Por otra parte, como hemos comentado en esta tesis, en [6, 8, 19, 46, 47, 48] se desarrollaron tipos de datos utilizados para trabajar en el contexto de sistemas replicados. En particular, los CRDTs que ofrecen implementaciones escalables para tipos de datos bien-conocidos como son registros, contadores y conjuntos.

8.3. Bases de Datos

En las bases de datos, el término consistencia no refiere solo a operaciones individuales, sino a secuencias de operaciones llamadas *transacciones*. De hecho, una transacción podría pensarse como un tipo de dato *key-value store* que permite ejecutar varias operaciones de manera atómica como si estas fueran una sola operación. Si bien en el Capítulo 3 estudiamos tipos de datos como registros, contadores y conjuntos, planeamos extender la presentación a tipos de datos más complejos como los *key-value stores*.

Por otra parte, la semántica de las transacciones también ha sido estudiada en el contexto de las bases de datos. Por ejemplo, [62] propone definiciones axiomáticas para capturar modelos de consistencia débil en bases de datos relacionales. La literatura propone varios modelos de consistencia para este tipo de bases de datos: *snapshot isolation* [63] y *serializability* [64]. Sin embargo, estos no pueden ser implementados satisfaciendo los requerimientos de disponibilidad y tolerancia a fallas. Por esta razón, los sistemas transaccionales, ofrecen garantías más débiles como transacciones eventualmente consistentes[65], transacciones causalmente consistentes [57, 66], o transacciones de alta disponibilidad [67, 68]. Planeamos extender la caracterización operacional presentada en el Capítulo 6 con modelos de consistencia transaccionales con el fin de ofrecer una especificación de bajo nivel que describa el comportamiento de sistemas transaccionales que requieren consistencia fuerte y pagan el costo de la sincronización. Trabajos como [69, 70, 71, 72] estudian propiedades como *snapshot isolation*, sin embargo todos estos trabajos resultan ser especificaciones de alto nivel con respecto a la propuesta en 6.

Capítulo 9

Conclusiones

En esta tesis, hemos estudiado y abordado el problema de razonar sobre sistemas cuya información se encuentra replicada. Hemos presentado técnicas de especificación para tipos de datos replicados aprovechándonos de las ventajas que ofrecen los enfoques denotacionales y funtoriales en contrapropuesta a la caracterización operacional que ofrece la literatura hoy en día. Nuestras contribuciones están organizadas de acuerdo a los principales objetos de estudio con los que un programador se encuentra a la hora de lidiar con consistencia eventual, estos son:

- **Tipos de datos replicados** (PARTE UNO): comenzamos con la propuesta de una descripción denotacional sobre RDTs. Más precisamente, en el Capítulo 3 asociamos a cada configuración (es decir, visibilidad) un conjunto de arbitraciones admisibles. A diferencia de los enfoques tradicionales, nuestra presentación permite acomodarse de forma natural a especificaciones no determinísticas, permitiendo definiciones lo suficientemente generales como para abstraerse de las estrategias de resolución de conflictos. Adicionalmente, este tipo de formulación permite caracterizar propiedades como *coherencia* y *saturación*. Una especificación *coherente* no puede ni obligar un orden de arbitración entre eventos que no están relacionados por la visibilidad ni permitir arbitraciones adicionales sobre eventos pasados cuando se extiende una configuración (es decir, se agrega un nuevo elemento superior a la visibilidad). En cambio, una especificación *saturada* no puede imponer ninguna restricción al arbitrar eventos. Observamos que la saturación no se cumple cuando se requiere que las arbitraciones admisibles también sean órdenes topológicos de la visibilidad.

  La presentación denotacional nos permitió desarrollar una caracterización functorial para especificar RDTs. Por lo tanto, el Capítulo 4 consideramos la categoría $\mathbf{PDag}(\mathcal{L})$ para representar la visibilidad, donde los objetos son grafos dirigidos, acíclicos y etiquetados y los morfismos cumplen la propiedad de ser *pr-morfismos*. Equipamos $\mathbf{PDag}(\mathcal{L})$ con operadores para describir la evolución de los grafos de visibilidad. Para arbitraciones, presentamos $\mathbf{SPath}(\mathcal{L})$, la cual es la categoría de los conjuntos de órdenes totales, etiquetados con *ps-morfismos*. Luego, mostramos que

cada especificación coherente, induce un functor $\mathbb{M}(\mathbb{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$. Contrariamente, probamos que un functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ que preserva colímites finitos y *binary pullbacks* induce una especificación coherente $\mathbb{S}(\mathbb{F})$.

En el Capítulo 5, propusimos un enfoque para verificar que una implementación sea correcta con respecto a una especificación. Nuestra caracterización se basa en una noción de simulación y aprovecha de la propiedad de composicionalidad con respecto a los modelos de comunicación estándar. Esto nos alivia de verificar que el comportamiento de varias réplicas sea correcto. Nuestra caracterización se centra solo en implementaciones basadas en estado.

- **Modelos de consistencia** (PARTE DOS): Propusimos un marco operacional para la definición formal de sistemas débilmente consistentes. La observación principal es que se pueden obtener varios modelos de consistencia concretos como casos particulares del framework operacional instanciando tres operadores diferentes que regulan el comportamiento del sistema. Esto se encuentra en el Capítulo 6.

  Nuestra contribución también vincula la semántica de los sistemas débilmente consistentes con la de los programas que se ejecutan sobre ellas (Capítulo 7). Si bien este es una primera aproximación, esperamos que esta línea de trabajo proporcione un terreno fértil para el desarrollo de técnicas para el análisis de programas.

La formalización propuesta tanto para RDTs como para modelos de consistencia contribuye al marco formal de razonamiento que tenemos para especificar e implementar sistemas globalmente distribuidos.

9.1. Trabajo futuro

El trabajo realizado a lo largo de esta tesis, posibilita la exploración de varias líneas de trabajo a futuro. Si bien esta tesis es un paso imprescindible para proveer una definición precisa de los lenguajes, el salto real de calidad ocurre cuando tales modelos formales son utilizados para desarrollar instrumentos de análisis de programas, evaluaciones comparativas entre las primitivas de programación, y herramientas para asistir al diseño y a la implementación de los sistemas. Son estos elementos los que incrementan la productividad en el desarrollo de software, ya sea reduciendo los tiempos de producción o aumentando la calidad de los productos. Es por ello que si bien en esta tesis hemos abordamos la problemática de manera conjunta, como objetivo a largo plazo esperemos analizar aplicaciones corriendo sobre infraestructura de geo-replicación.

Dado que cada capítulo en sí mismo deja abierto varias cuestiones sin responder, resumiremos los principales puntos a mediano y corto plazo con los que seguiremos trabajando. Concretamente,

(I) creemos que dado que la relación entre la visibilidad y la arbitración tiene una interpretación distinta en esta tesis, esto siguiere que aquellos modelos de consistencia

que se definen como como relaciones entre visibilidad y arbitración (por ejemplo, *causalidad*) podrían tener caracterizaciones alternativas. Por lo tanto, planeamos explorar estas conexiones en trabajos futuros;

(II) observamos que la caracterización puramente funcional de RDTs, proporcionan un entorno ideal para el desarrollo de técnicas que permitan lidiar con la composición de RDTs. Nuestro objetivo entonces es: (i) equipar las especificaciones de RDTs con un conjunto de operadores que nos permite especificar y razonar sobre la composición RDTs complejos y (ii), proporcionar un tratamiento uniforme a los enfoques composicionales propuestos en [27, 48, 73];

(III) finalmente, aunque nuestra presentación de sistemas replicados se centra en modelos basados en estado, es decir, aquellos en los que una operación de lectura devuelve un valor previamente escrito, la definición de nuestro modelo operacional puede ampliarse para tratar con modelos cuya propagación está basada en operaciones, es decir, aquellos en los que una operación de lectura devuelve el resultado de evaluar todas las operaciones que se han aplicado a un objeto. Esto podría acomodarse en el modelo (1) representando un sistema como un orden parcial de las operaciones realizadas sobre objetos en lugar de solo valores simples; (2) tomar el valor de retorno de cada operación de lectura como resultado de aplicar la secuencia de operaciones visibles ordenadas en consecuencia a la arbitración. Si bien, hemos mostrado que nuestro modelo operacional permite instanciar varios de los modelos de consistencia bien-conocidos, dejamos como trabajo futuro, caracterizar modelos transaccionales.

Bibliografía

[1] S. Gilbert, N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, SIGACT News 33 (2) (2002) 51–59.

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, in: SOSP '07, ACM, 2007, pp. 205–220. doi:10.1145/1294261.1294281.
URL http://doi.acm.org/10.1145/1294261.1294281

[3] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, ACM SIGOPS Operating Systems Review 44 (2) (2010) 35–40.

[4] P. Bailis, A. Ghodsi, Eventual consistency today: limitations, extensions, and beyond, Commun. ACM 56 (5) (2013) 55–63.

[5] M. Shapiro, N. M. Preguiça, C. Baquero, M. Zawirski, Conflict-free replicated data types, in: X. Défago, F. Petit, V. Villain (Eds.), SSS 2011, Vol. 6976 of Lecture Notes in Computer Science, Springer, 2011, pp. 386–400.

[6] M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski, A comprehensive study of convergent and commutative replicated data types, Tech. Rep. RR-7506, Inria–Centre Paris-Rocquencourt (2011).

[7] P. Viotti, M. Vukolić, Consistency in non-transactional distributed storage systems, ACM Computing Surveys (CSUR) 49 (1) (2016) 19.

[8] S. Burckhardt, A. Gotsman, H. Yang, M. Zawirski, Replicated data types: specification, verification, optimality, in: S. Jagannathan, P. Sewell (Eds.), POPL 2014, ACM, 2014, pp. 271–284.

[9] K. C. Sivaramakrishnan, G. Kaki, S. Jagannathan, Declarative programming over eventually consistent data stores, in: D. Grove, S. Blackburn (Eds.), PLDI 2015, ACM, 2015, pp. 413–424.

[10] L. Brutschy, D. Dimitrov, P. Müller, M. Vechev, Serializability for eventual consistency: criterion, analysis, and applications, in: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 458–472.

[11] S. Burckhardt, A. Gotsman, H. Yang, Understanding eventual consistency, Tech. Rep. MSR-TR-2013-39, Microsoft Research (2013).

[12] A. Cerone, G. Bernardi, A. Gotsman, A framework for transactional consistency models with atomic visibility, in: L. Aceto, D. de Frutos-Escrig (Eds.), CONCUR 2015, Vol. 42 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 58–71.

[13] A. Cerone, A. Gotsman, H. Yang, Algebraic laws for weak consistency, in: LIPIcs-Leibniz International Proceedings in Informatics, Vol. 85, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[14] F. Gadducci, H. Melgratti, C. Roldán, On the semantics and implementation of replicated data types, Science of Computer Programming 167 (2018) 91–113.

[15] S. Burckhardt, Principles of eventual consistency, Foundations and Trends in Programming Languages 1 (1-2) (2014) 1–150.

[16] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., Spanner: Google?s globally distributed database, ACM Transactions on Computer Systems (TOCS) 31 (3) (2013) 8.

[17] H. Melgratti, C. Roldán, A formal analysis of the global sequence protocol, in: International Conference on Coordination Languages and Models, Springer, 2016, pp. 175–191.

[18] A. Gotsman, S. Burckhardt, Consistency models with global operation sequencing and their composition, in: A. W. Richa (Ed.), DISC 2017, Vol. 91 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 23:1–23:16.

[19] S. Burckhardt, M. Fähndrich, D. Leijen, B. P. Wood, Cloud types for eventual consistency, in: European Conference on Object-Oriented Programming, Springer, 2012, pp. 283–307.

[20] M. Kleppmann, V. B. Gomes, D. P. Mulligan, A. R. Beresford, Opsets: Sequential specifications for replicated datatypes (extended version), arXiv preprint arXiv:1805.04263.

[21] S. Burckhardt, D. Leijen, J. Protzenko, M. Fähndrich, Global sequence protocol: A robust abstraction for replicated shared state, in: 29th European Conference on Object-Oriented Programming (ECOOP 2015), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[22] A. Bouajjani, C. Enea, M. Mukund, G. Shenoy, S. Suresh, Formalizing and checking multilevel consistency, in: International Conference on Verification, Model Checking, and Abstract Interpretation, Springer, 2020, pp. 379–400.

[23] F. Gadducci, H. C. Melgratti, C. Roldán, A denotational view of replicated data types, in: J. Jacquet, M. Massink (Eds.), COORDINATION 2017, Vol. 10319 of Lecture Notes in Computer Science, Springer, 2017, pp. 138–156.

[24] F. Gadducci, H. Melgratti, C. Roldán, M. Sammartino, A categorical account of replicated data types, in: 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[25] R. Milner, et al., A calculus of communicating systems.

[26] S. Awodey, Category theory, Oxford University Press, 2010.

[27] A. Gotsman, H. Yang, Composite replicated data types, in: J. Vitek (Ed.), ESOP 2015, Vol. 9032 of LNCS, Springer, 2015, pp. 585–609.

[28] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, M. Shapiro, 'cause I'm strong enough: reasoning about consistency choices in distributed systems, in: R. Bodík, R. Majumdar (Eds.), POPL 2016, ACM, 2016, pp. 371–384.

[29] A. Bouajjani, C. Enea, J. Hamza, Verifying eventual consistency of optimistic replication systems, in: S. Jagannathan, P. Sewell (Eds.), POPL 2014, ACM, 2014, pp. 285–296.

[30] A. Bouajjani, C. Enea, R. Guerraoui, J. Hamza, On verifying causal consistency, in: ACM SIGPLAN Notices, Vol. 52, ACM, 2017, pp. 626–638.

[31] K. Nagar, S. Jagannathan, Automated detection of serializability violations under weak consistency, arXiv preprint arXiv:1806.08416.

[32] M. Emmi, C. Enea, Monitoring weak consistency, in: International Conference on Computer Aided Verification, Springer, 2018, pp. 487–506.

[33] M. Emmi, C. Enea, Weak-consistency specification via visibility relaxation, Proceedings of the ACM on Programming Languages 3 (POPL) (2019) 60.

[34] J. He, C. Hoare, J. W. Sanders, Data refinement refined resume, in: European Symposium on Programming, Springer, 1986, pp. 187–196.

[35] Z. Luo, Program specification and data refinement in type theory, in: Colloquium on Trees in Algebra and Programming, Springer, 1991, pp. 143–168.

[36] K. Nagar, S. Jagannathan, Automated parameterized verification of crdts, arXiv preprint arXiv:1905.05684.

[37] G. Boudol, G. Petri, Relaxed memory models: an operational approach, in: ACM SIGPLAN Notices, Vol. 44, ACM, 2009, pp. 392–403.

[38] G. Bernardi, A. Gotsman, Robustness against consistency models with atomic visibility, in: LIPIcs-Leibniz International Proceedings in Informatics, Vol. 59, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[39] M. Dubois, C. Scheurich, F. Briggs, Memory access buffering in multiprocessors, ACM SIGARCH computer architecture news 14 (2) (1986) 434–442.

[40] X. Shen, L. R. Arvind, Commit-reconcile and fences (crf): a new memory model for architects and compiler writers, in: Proceedings of the 26th International Symposium on Computer Architecture (Cat. No. 99CB36367), IEEE, 1999, pp. 150–161.

[41] C. Blundell, E. C. Lewis, M. M. Martin, Subtleties of transactional memory atomicity semantics, IEEE Computer Architecture Letters 5 (2) (2006) 17–17.

[42] G. Boudol, Atomic actions.

[43] V. A. Saraswat, R. Jagadeesan, M. Michael, C. von Praun, A theory of memory models, in: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, 2007, pp. 161–172.

[44] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. Theimer, B. B. Welch, Session guarantees for weakly consistent replicated data, in: PDIS 1994, IEEE, 1994, pp. 140–149.

[45] S. Brookes, Full abstraction for a shared-variable parallel language, Information and Computation 127 (2) (1996) 145–163.

[46] M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski, Convergent and commutative replicated data types.

[47] H.-G. Roh, M. Jeon, J.-S. Kim, J. Lee, Replicated abstract data types: Building blocks for collaborative applications, Journal of Parallel and Distributed Computing 71 (3) (2011) 354–368.

[48] A. Leijnse, P. S. Almeida, C. Baquero, Higher-order patterns in replicated data types, in: Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data, ACM, 2019, p. 5.

[49] C. Wang, C. Enea, S. O. Mutluergil, G. Petri, Replication-aware linearizability, in: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019, pp. 980–993.

[50] C. A. R. Hoare, Proof of correctness of data representations, in: Programming methodology, Springer, 1978, pp. 269–281.

[51] P. Zeller, A. Bieniusa, A. Poetzsch-Heffter, Formal specification and verification of CRDTs, in: E. Ábrahám, C. Palamidessi (Eds.), FORTE 2014, Vol. 8461 of Lecture Notes in Computer Science, Springer, 2014, pp. 33–48.

[52] G. Winskel, Event structure semantics for ccs and related languages, in: International Colloquium on Automata, Languages, and Programming, Springer, 1982, pp. 561–576.

[53] D. B. Terry, M. Theimer, K. Petersen, A. J. Demers, M. Spreitzer, C. Hauser, Managing update conflicts in Bayou, a weakly connected replicated storage system, in: M. B. Jones (Ed.), SOSP 1995, ACM, 1995, pp. 172–183.

[54] A. S. Tanenbaum, M. Van Steen, Distributed systems: principles and paradigms, Prentice-Hall, 2007.

[55] Y. Saito, M. Shapiro, Optimistic replication, ACM Computing Surveys (CSUR) 37 (1) (2005) 42–81.

[56] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, A. J. Demers, Flexible update propagation for weakly consistent replication, in: Proceedings of the sixteenth ACM symposium on Operating systems principles, 1997, pp. 288–301.

[57] W. Lloyd, M. J. Freedman, M. Kaminsky, D. G. Andersen, Don't settle for eventual: scalable causal consistency for wide-area storage with cops, in: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM, 2011, pp. 401–416.

[58] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, B. Welch, The bayou architecture: Support for data sharing among mobile users, in: 1994 First Workshop on Mobile Computing Systems and Applications, IEEE, 1994, pp. 2–7.

[59] E. A. Brewer, Towards robust distributed systems, in: PODC, Vol. 7, 2000.

[60] K. Banker, MongoDB in action, Manning Publications Co., 2011.

[61] J. Han, E. Haihong, G. Le, J. Du, Survey on nosql database, in: 2011 6th international conference on pervasive computing and applications, IEEE, 2011, pp. 363–366.

[62] A. Adya, Weak consistency: a generalized theory and optimistic implementations for distributed transactions.

[63] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil, A critique of ansi sql isolation levels, arXiv preprint cs/0701157.

[64] C. Papadimitriou, The theory of database concurrency control.

[65] S. Burckhardt, D. Leijen, M. Fähndrich, M. Sagiv, Eventually consistent transactions, in: European Symposium on Programming, Springer, 2012, pp. 67–86.

[66] W. Lloyd, M. J. Freedman, M. Kaminsky, D. G. Andersen, Stronger semantics for low-latency geo-replicated storage, in: Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13), 2013, pp. 313–328.

[67] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, I. Stoica, Hat, not cap: towards highly available transactions, in: Proceedings of the 14th USENIX conference on Hot Topics in Operating Systems, USENIX Association, 2013, pp. 24–24.

[68] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, I. Stoica, Highly available transactions: Virtues and limitations, Proceedings of the VLDB Endowment 7 (3) (2013) 181–192.

[69] M. S. Ardekani, P. Sutra, M. Shapiro, N. Preguiça, On the scalability of snapshot isolation, in: European Conference on Parallel Processing, Springer, 2013, pp. 369–381.

[70] T. Riegel, C. Fetzer, P. Felber, Snapshot isolation for software transactional memory, in: First ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing (TRANSACT'06), Association for Computing Machinery (ACM), 2006, pp. 1–10.

[71] K. Antoniadis, D. Didona, R. Guerraoui, W. Zwaenepoel, The impossibility of fast transactions.

[72] H. Attiya, F. Ellen, A. Morrison, Limitations of highly-available eventually-consistent data stores, IEEE Transactions on Parallel and Distributed Systems 28 (1) (2016) 141–155.

[73] C. Baquero, P. S. Almeida, A. Cunha, C. Ferreira, et al., Composition in state-based replicated data types, Bulletin of EATCS 3 (123).

Apéndice A

Demostraciones para Parte I

A.1. Demostraciones sobre los resultados del Capítulo 5

DEMOSTRACIÓN DE EJ. 5.1.2 La prueba se sigue por análisis de casos en la derivación de $\sigma \xrightarrow{\alpha} \sigma'$. Considerar $\sigma = \langle \mathsf{r}_j, \mathsf{v} \rangle$ y $(\sigma, \langle \mathtt{G}, \mathtt{P} \rangle) \in \mathcal{I}$. Consecuentemente, existe $f : \mathtt{E}_{\mathtt{G}} \to \mathcal{R}$ tal que

$$\forall \mathsf{r} \in \mathcal{R}.\mathsf{v}(\mathsf{r}) = |\{\mathsf{e} \mid f(\mathsf{e}) = \mathsf{r} \text{ y } \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok} \rangle\}| \tag{A.1}$$

- (READ) Por lo tanto, $\sigma = \langle \mathsf{r}_j, \mathsf{v} \rangle \xrightarrow{\mathtt{rd},\mathtt{k}} \langle \mathsf{r}_j, \mathsf{v} \rangle = \sigma'$ y $\mathtt{k} = \sum_{\mathsf{r} \in \mathcal{R}} \mathsf{v}(\mathsf{r})$. De (A.1), $\mathtt{k} = \sum_{\mathsf{r} \in \mathcal{R}} \mathsf{v}(\mathsf{r}) = \#\{\mathsf{e} \mid \mathsf{e} \in \mathtt{E}_{\mathtt{G}} \wedge \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok} \rangle\}$. Definir $\mathtt{G}' = \mathtt{G}^{\langle \mathtt{rd},\mathtt{k} \rangle}$ y $f' : \mathtt{E}_{\mathtt{G}'} \to \mathcal{R}$ de la siguiente manera

  $$f'(\mathsf{e}) = \begin{cases} f(\mathsf{e}) & \text{si } \mathsf{e} \in \mathtt{E}_{\mathtt{G}} \\ \mathsf{r}_j & \text{si } \mathsf{e} = \top \end{cases}$$

  Es inmediato revisar que

  $$\forall \mathsf{r} \in \mathcal{R}.\mathsf{v}(\mathsf{r}) = |\{\mathsf{e} \mid f'(\mathsf{e}) = \mathsf{r} \text{ y } \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok} \rangle\}|$$

  Por lo tanto,

  $$\mathtt{k} = \sum_{\mathsf{r} \in \mathcal{R}} \mathsf{v}(\mathsf{r}) = |\{\mathsf{e} \mid \mathsf{e} \in \mathtt{E}_{\mathtt{G}'} \wedge \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok} \rangle\}|$$

  Consecuentemente, $\mathtt{sat}(\mathtt{P}, \langle \mathtt{rd}, \mathtt{k} \rangle) \subseteq \mathbb{S}_{Ctr}(\mathtt{G}')$. Entonces, para cada $\mathtt{P}' \in \mathtt{sat}(\mathtt{P}, \langle \mathtt{rd}, \mathtt{k} \rangle)$, tenemos $(\langle \mathsf{r}_j, \mathsf{v} \rangle, \langle \mathtt{G}^{\langle \mathtt{rd},\mathtt{k} \rangle}, \mathtt{P}' \rangle) \in \mathcal{I}$.

- (INC) Entonces, $\sigma = \langle \mathsf{r}_j, \mathsf{v} \rangle \xrightarrow{\mathtt{inc},\mathtt{ok}} \langle \mathsf{r}_j, \mathsf{v}[\mathsf{r}_j \mapsto \mathsf{v}(\mathsf{r}_j) + 1] \rangle = \sigma'$. Tomar $\mathtt{G}' = \mathtt{G}^{\langle \mathtt{inc},\mathtt{ok} \rangle}$. Es inmediato revisar que $\mathtt{sat}(\mathtt{P}, \langle \mathtt{inc}, \mathtt{ok} \rangle) \subseteq \mathbb{S}_{Ctr}(\mathtt{G}')$. Entonces, definir $f' : \mathtt{E}_{\mathtt{G}'} \to \mathcal{R}$ como

  $$f'(\mathsf{e}) = \begin{cases} f(\mathsf{e}) & \text{si } \mathsf{e} \in \mathtt{E}_{\mathtt{G}} \\ \mathsf{r}_j & \text{si } \mathsf{e} = \top \end{cases}$$

En consecuencia, $|\{\mathsf{e} \mid f'(\mathsf{e}) = \mathsf{r} \text{ y } \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok}\rangle\}| = \mathsf{v}(\mathsf{r})$ para todo $\mathsf{r} \neq \mathsf{r}_j$, y

$$|\{\mathsf{e} \mid f'(\mathsf{e}) = \mathsf{r}_j \text{ y } \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok}\rangle\}| = \mathsf{v}(\mathsf{r}_j) + 1$$

Consecuentemente, para todo $\mathtt{P}' \in \mathtt{sat}(\mathtt{P}, \langle \mathtt{inc}, \mathtt{ok}\rangle)$ lo siguiente se mantiene

$$(\langle \mathsf{r}_j, \mathsf{v}[\mathsf{r}_j \mapsto \mathsf{v}(\mathsf{r}_j) + 1]\rangle, \langle \mathtt{G}^{\langle \mathtt{inc}, ok\rangle}, \mathtt{P}'\rangle) \in \mathcal{I}$$

- (RECEIVE) Luego, $\sigma = \langle \mathsf{r}, \mathsf{v}\rangle \xrightarrow{\mathtt{rcv}, \langle \mathsf{r}', \mathsf{v}''\rangle} \langle \mathsf{r}, \mathsf{v}'\rangle = \sigma'$ tal que $\mathsf{v}' = \max\{\mathsf{v}, \mathsf{v}''\}$. Definir $\mathtt{G}' = \langle \mathcal{F}, \prec', \lambda'\rangle$ y $f' : \mathtt{E}_{\mathtt{G}'} \to \mathcal{R}$ tal que $\forall \mathsf{r} \in \mathcal{R}$
  - $\mathsf{v}''(\mathsf{r}) = \#\{\mathsf{e} \mid f'(\mathsf{e}) = \mathsf{r} \text{ y } \lambda'(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok}\rangle\}$
  - $min\{\mathsf{v}(\mathsf{r}), \mathsf{v}''(\mathsf{r})\} = \#\{\mathsf{e} \mid f(\mathsf{e}) = \mathsf{r}\} \cap \#\{\mathsf{e} \mid f'(\mathsf{e}) = \mathsf{r}\}$
  - $\prec_{\mathtt{G}'}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}} = \prec_{\mathtt{G}}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}}$,
  - $\lambda_{\mathtt{G}'}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}} = \lambda_{\mathtt{G}}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}}$.

  Es inmediato notar que existe $\mathtt{P}'$ tal que $(\langle \mathsf{r}', \mathsf{v}''\rangle, \langle \mathtt{G}', \mathtt{P}'\rangle) \in \mathcal{I}$. Más aún, $\mathtt{G} \sqcup \mathtt{G}'$ es definida bajo las condiciones de arriba; para cualquier $\mathtt{P}' \in \mathbb{S}_{Ctr}(\mathtt{G}')$ tenemos $\mathtt{P} \otimes \mathtt{P}' \subseteq \mathbb{S}_{Ctr}(\mathtt{G} \sqcup \mathtt{G}')$. Por lo tanto, es suficiente tomar $\mathtt{P}'' \in \mathtt{P} \otimes \mathtt{P}'$.

  Definir $f'' : \mathtt{E}_{\mathtt{G} \sqcup \mathtt{G}'} \to \mathcal{R}$ como

$$f''(\mathsf{e}) = \begin{cases} f(\mathsf{e}) & \text{si } \mathsf{e} \in \mathtt{E}_{\mathtt{G}} \\ f'(\mathsf{e}) = \mathsf{r}_j & \text{caso contrario} \end{cases}$$

  De la definición de $\mathtt{G}'$ y f'' se cumple que para todo $\mathsf{r} \in \mathcal{R}$

$$|\{\mathsf{e} \mid \mathsf{e} \in \mathtt{E}_{\mathtt{G} \sqcup \mathtt{G}'} \text{ y } \lambda(\mathsf{e}) = \langle \mathtt{inc}, \mathtt{ok}\rangle\}| = max\{\mathsf{v}(\mathsf{r}), \mathsf{v}''(\mathsf{r})\} = \mathsf{v}'(\mathsf{r})$$

  Consecuentemente, $(\langle \mathsf{r}, \mathsf{v}'\rangle, \langle \mathtt{G} \sqcup \mathtt{G}', \mathtt{P}''\rangle) \in \mathcal{I}$.

- (SEND) Entonces, $\sigma = \langle \mathsf{r}, \mathsf{v}\rangle \xrightarrow{\mathtt{send}, \langle \mathsf{r}, \mathsf{v}\rangle} \langle \mathsf{r}, \mathsf{v}\rangle = \sigma'$. es inmediato ya que $(\langle \mathsf{r}, \mathsf{v}\rangle, \langle \mathtt{G}, \mathtt{P}\rangle) \in \mathcal{I}$ se mantiene por hipótesis. □

DEMOSTRACIÓN DE EJ. 5.1.3 La prueba será por análisis de casos sobre la regla aplicada por la derivación de $\sigma \xrightarrow{\alpha} \sigma'$. Considerar $\sigma = \langle \mathsf{r}_j, \mathtt{V}, \mathtt{W}\rangle$ y $(\sigma, \langle \mathtt{G}, \mathtt{P}\rangle) \in \mathcal{I}$. En consecuencia, existe $f : \mathtt{E}_{\mathtt{G}} \to \mathcal{R}$ y

$$\forall \mathtt{k}.(\exists \mathsf{r}.\mathtt{W}(\mathtt{k}, \mathsf{r}) > 0 \iff \exists\, \mathtt{S}.\ \mathbb{S}_{Or\text{-}Set}(\mathtt{G}^{\langle \mathtt{lookup}, \mathtt{S}\rangle}) \neq \emptyset \ \wedge\ \mathtt{k} \in \mathtt{S}) \tag{A.2}$$

- (LOOKUP) Por lo tanto, $\sigma \xrightarrow{\texttt{lookup,S}} \sigma' = \sigma$. Define $\texttt{G}' = \texttt{G}^{\langle \texttt{lookup,S} \rangle}$ y tomar $f' : \texttt{E}_{\texttt{G}'} \to \mathcal{R}$ definida como

$$f'(\texttt{e}) = \begin{cases} f(\texttt{e}) & \text{si } \texttt{e} \in \texttt{E}_{\texttt{G}} \\ \texttt{r}_j & \text{si } \texttt{e} = \top \end{cases}$$

  Notar que

$$\forall \texttt{k}.(\exists \texttt{r}.\texttt{W}(\texttt{k},\texttt{r}) > 0 \iff \exists\, \texttt{S}.\, \mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \rangle}) \neq \emptyset \wedge \texttt{k} \in \texttt{S})$$

  por definición de $\mathbb{S}_{\textit{Or-Set}}$ y (A.2). Más aún, concluimos que $\texttt{sat}(\texttt{P}, \langle \texttt{lookup}, \texttt{S} \rangle) \subseteq \mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \rangle})$. En consecuencia, $(\sigma', \langle \texttt{G}', \texttt{P}' \rangle) \in \mathcal{I}$ para cada $\texttt{P}' \in \texttt{sat}(\texttt{P}, \langle \texttt{lookup}, \texttt{S} \rangle)$.

- (ADD) Entonces, $\sigma \xrightarrow{\texttt{add(k')},\texttt{ok}} \langle \texttt{r}_j, \texttt{V}[\texttt{r}_j \mapsto \texttt{V}(\texttt{r}_j) + 1], \texttt{W}[(\texttt{k}', \texttt{r}_j) \mapsto \texttt{V}(\texttt{r}_j) + 1] \rangle = \sigma'$. Definir $\texttt{G}' = \texttt{G}^{\langle \texttt{add(k')},\texttt{ok} \rangle}$ y tomar $f' : \texttt{E}_{\texttt{G}'} \to \mathcal{R}$ definida como

$$f'(\texttt{e}) = \begin{cases} f(\texttt{e}) & \text{si } \texttt{e} \in \texttt{E}_{\texttt{G}} \\ \texttt{r}_j & \text{si } \texttt{e} = \top \end{cases}$$

  Ahora, mostramos que se mantiene la siguiente proposición

$$\forall \texttt{k}(\exists \texttt{r}.\texttt{W}[(\texttt{k}', \texttt{r}_j) \mapsto \texttt{V}(\texttt{r}_j) + 1](\texttt{k},\texttt{r}) > 0 \iff \exists\, \texttt{S}.\, \mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \rangle}) \neq \emptyset \wedge \texttt{k} \in \texttt{S}) \tag{A.3}$$

  Para todo $\texttt{k} \neq \texttt{k}'$, la condición se mantiene por (A.2)

$$\texttt{W}[(\texttt{k}', \texttt{r}_j) \mapsto \texttt{V}(\texttt{r}_j) + 1](\texttt{k},\texttt{r}) > 0 \iff \exists \texttt{S}.\mathbb{S}_{\textit{Or-Set}}(\texttt{G}^{\langle \texttt{lookup,S}' \rangle}) \neq \emptyset \wedge \texttt{k} \in \texttt{S}$$

  Luego, por $\texttt{k} = \texttt{k}'$ es suficiente tomar $\texttt{r} = \texttt{r}_j$ para concluir que la doble implicación se mantiene.

  Por definición de $\mathbb{S}_{\textit{Or-Set}}$, tenemos que $\mathbb{S}_{\textit{Or-Set}}(\texttt{G}^{\langle \texttt{lookup,S}' \rangle}) \neq \emptyset$ implica $\mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \cup \{\texttt{k}'\} \rangle}) \neq \emptyset$. Adicionalmente, $\texttt{sat}(\texttt{P}, \langle \texttt{add}(\texttt{k}), \texttt{ok} \rangle) \subseteq \mathbb{S}_{\textit{Or-Set}}(\texttt{G}')$ ya que $\texttt{P} \in \mathbb{S}_{\textit{Or-Set}}(\texttt{G})$. Entonces, para cada $\texttt{P}' \in \texttt{sat}(\texttt{P}, \langle \texttt{add}(\texttt{k}), \texttt{ok} \rangle)$, tenemos $(\sigma', \langle \texttt{G}', \texttt{P}' \rangle) \in \mathcal{I}$.

- (REM) Entonces, $\sigma \xrightarrow{\texttt{rem(k')},\texttt{ok}} \langle \texttt{r}_j, \texttt{V}, \texttt{W}' \rangle = \sigma'$, y $\forall \texttt{s}.\texttt{W}'(\texttt{k}', \texttt{s}) = 0$, y $\forall \texttt{s}, \texttt{k}'' \neq \texttt{k}'.\texttt{W}'(\texttt{k}'', \texttt{s}) = \texttt{W}(\texttt{k}'', \texttt{s})$. Definir $\texttt{G}' = \texttt{G}^{\langle \texttt{rem(k')},\texttt{ok} \rangle}$ y tomar $f' : \texttt{E}_{\texttt{G}'} \to \mathcal{R}$ definida como

$$f'(\texttt{e}) = \begin{cases} f(\texttt{e}) & \text{si } \texttt{e} \in \texttt{E}_{\texttt{G}} \\ \texttt{r}_j & \text{si } \texttt{e} = \top \end{cases}$$

  Como en el caso anterior, concluimos que se cumple para todo $\texttt{k} \neq \texttt{k}'$

$$\exists \texttt{r}.\texttt{W}'(\texttt{k},\texttt{r}) > 0 \iff \exists \texttt{S}.\mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \rangle}) \neq \emptyset \wedge \texttt{k} \in \texttt{S}$$

  por (A.2) y el hecho que $\texttt{W}'(\texttt{k}'', \texttt{r}) = \texttt{W}(\texttt{k}'', \texttt{r})$ se mantiene para todo $\texttt{k}'' \neq \texttt{k}'$ y $\texttt{r}$. Por $\mathbb{S}_{\textit{Or-Set}}$, $\mathbb{S}_{\textit{Or-Set}}(\texttt{G}'^{\langle \texttt{lookup,S} \rangle}) \neq \emptyset$ implica $\{\texttt{k}'\} \notin \texttt{S}$. Entonces, la prueba se completa notando que $\forall \texttt{r}.\texttt{W}'(\texttt{k}', \texttt{r}) = 0$.

- (SEND) Entonces, $\sigma \xrightarrow{\mathtt{send},\sigma} \sigma$. se mantiene ya que $(\sigma, \langle \mathtt{G}, \mathtt{P} \rangle) \in \mathcal{I}$ por hipótesis.
- (RECEIVE) Entonces, $\sigma \xrightarrow{\mathtt{receive}\langle \mathtt{r}_k, \mathtt{V}', \mathtt{W}' \rangle} \langle \mathtt{r}_j, \max\{\mathtt{V}, \mathtt{V}'\}, (\mathtt{V}, \mathtt{W}) \oplus (\mathtt{V}', \mathtt{W}') \rangle = \sigma'$. Definir $\mathtt{G}' = \langle \mathtt{E}_{\mathtt{G}'}, \prec_{\mathtt{G}'}, \lambda_{\mathtt{G}'} \rangle$ donde
  - $\mathtt{E}_{\mathtt{G}'} = \biguplus_{r \in \mathcal{R}, \mathtt{k} \in \mathbb{N}} \mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ (donde $\uplus$ representa la unión disjunta) y

$$\mathcal{F}_{\mathtt{r}}^{\mathtt{k}} = \begin{cases} \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} & \text{si } \mathtt{W}'(\mathtt{k},\mathtt{r}) \neq 0 \ \wedge \ \mathtt{W}'(\mathtt{k},\mathtt{r}) \leq \mathtt{V}(\mathtt{r}) \quad \text{(i)} \\ \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \uplus \{\mathtt{a}_{\mathtt{r}}^{\mathtt{k}}\} & \text{si } \mathtt{W}'(\mathtt{k},\mathtt{r}) \neq 0 \ \wedge \ \mathtt{W}'(\mathtt{k},\mathtt{r}) > \mathtt{V}(\mathtt{r}) \quad \text{(ii)} \\ \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \uplus \{\mathtt{r}_{\mathtt{r}}^{\mathtt{k}}\} & \text{si } \mathtt{W}'(\mathtt{k},\mathtt{r}) = 0 \ \wedge \ \mathtt{W}(\mathtt{k},\mathtt{r}) \leq \mathtt{V}'(\mathtt{r}) \quad \text{(iii)} \\ \emptyset & \text{si } \mathtt{W}'(\mathtt{k},\mathtt{r}) = 0 \ \wedge \ \mathtt{W}(\mathtt{k},\mathtt{r}) > \mathtt{V}'(\mathtt{r}) \quad \text{(iv)} \end{cases}$$

$$\text{donde } \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \ = \ \{\mathtt{e} \mid \mathtt{e} \in \mathtt{E}_{\mathtt{G}} \ \wedge \ f(\mathtt{e}) = \mathtt{r} \ \wedge \ \lambda_{\mathtt{G}}(\mathtt{e}) = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok} \rangle\}$$

  - $\lambda_{\mathtt{G}'}$ es definida tal que

$$\lambda_{\mathtt{G}'}(\mathtt{e}) = \begin{cases} \langle \mathtt{add}(\mathtt{k}), \mathtt{ok} \rangle & \text{si } \mathtt{e} \in \mathcal{F}_{\mathtt{r}}^{\mathtt{k}} \wedge \mathtt{e} \neq \mathtt{r}_{\mathtt{r}}^{\mathtt{k}} \\ \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok} \rangle & \text{si } \mathtt{e} = \mathtt{r}_{\mathtt{r}}^{\mathtt{k}} \end{cases}$$

  - $\prec_{\mathtt{G}'}$ definida tal que

$$\prec_{\mathtt{G}'}\big|_{\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}} = \begin{cases} \prec\big|_{\mathtt{E}_{\mathtt{r}}^{\mathtt{k}}} & \text{si } \mathcal{F}_{\mathtt{r}}^{\mathtt{k}} = \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \\ \prec\big|_{\mathtt{E}_{\mathtt{r}}^{\mathtt{k}}} & \text{si } \mathcal{F}_{\mathtt{r}}^{\mathtt{k}} = \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \uplus \{\mathtt{a}_{\mathtt{r}}^{\mathtt{k}}\} \\ \prec\big|_{\mathtt{E}_{\mathtt{r}}^{\mathtt{k}}} \cup (\mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \times \{\mathtt{r}^{\mathtt{k}}\}) & \text{si } \mathcal{F}_{\mathtt{r}}^{\mathtt{k}} = \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \uplus \{\mathtt{r}_{\mathtt{r}}^{\mathtt{k}}\} \\ \emptyset & \text{si } \mathcal{F}_{\mathtt{r}}^{\mathtt{k}} = \emptyset \end{cases}$$

Ahora revisamos que $(\langle \mathtt{r}_k, \mathtt{V}', \mathtt{W}' \rangle, \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}$. Por definición de $\mathbb{S}_{\textit{Or-Set}}$ en Ej. 3.3.2, podemos concluir que $\mathbb{S}_{\textit{Or-Set}}(\mathtt{G}'^{\langle \mathtt{lookup}, \mathtt{S}' \rangle}) \neq \emptyset$ siempre que

$$\mathtt{S}' = \{\mathtt{k} \mid \exists \mathtt{e}' \in \mathtt{E}_{\mathtt{G}'}. \exists \mathtt{k} \in \mathbb{N}. \lambda_{\mathtt{G}'}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok} \rangle \ \text{ y } \\ \forall \mathtt{e}''. \mathtt{e}' \prec_{\mathtt{G}'} \mathtt{e}'' \text{ implca } \lambda_{\mathtt{G}'}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok} \rangle\}$$

De la definición de $\mathtt{G}'$, es inmediato que $\mathtt{k} \in \mathtt{S}'$ si y sólo si $\exists \mathtt{r}. \mathtt{W}'(\mathtt{k},\mathtt{r}) > 0$ (los casos (i) y (ii) sobre la definición de $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$). Entonces, para cada $\mathtt{P}' \in \mathbb{S}_{\textit{Or-Set}}(\mathtt{G}')$, tenemos $(\langle \mathtt{r}_k, \mathtt{V}', \mathtt{W}' \rangle, \langle \mathtt{G}', \mathtt{P}' \rangle) \in \mathcal{I}$ tomando $f'' : \mathtt{E}_{\mathtt{G}'} \to \mathcal{R}$ definida tal que $f''(\mathtt{e}) = \mathtt{r}$ si existe $\mathtt{k} \in \mathbb{N}$ tal que $\mathtt{e} \in \mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$.

Notar que $\mathtt{G} \sqcup \mathtt{G}'$ es bien-definido porque $\mathtt{G}'$ es definido tal que $\lambda_{\mathtt{G}'}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}} = \lambda_{\mathtt{G}}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}}$ y $\prec_{\mathtt{G}'}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}} = \prec_{\mathtt{G}}|_{\mathtt{E}_{\mathtt{G}} \cap \mathtt{E}_{\mathtt{G}'}}$. Más aún, $\mathtt{P} \otimes \mathtt{P}' \neq \emptyset$ por definición de $\mathtt{G}'$. Resta probar que $(\sigma', \langle \mathtt{G} \sqcup \mathtt{G}', \mathtt{P}'' \rangle) \in \mathcal{I}$ para cualquier $\mathtt{P}'' \in \mathtt{P} \otimes \mathtt{P}'$.

Ahora definimos $f' : \mathtt{E}_{\mathtt{G} \sqcup \mathtt{G}'} \to \mathcal{R}$ tal que

$$f'(\mathtt{e}) = \begin{cases} f(\mathtt{e}) & \text{si } \mathtt{e} \in \mathtt{E}_{\mathtt{G}} \\ f''(\mathtt{e}) & \text{si } \mathtt{e} \in \mathtt{E}_{\mathtt{G}'} \end{cases}$$

Notar que f' es bien-definida porque $\{\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}\}_{r\in\mathcal{R},k\in\mathbb{N}}$ es una partición de $\mathtt{E}_{\mathtt{G}'}$, y la definición de $\mathtt{G}'$ asegura que $\mathtt{E}_{\mathtt{r}}^{\mathtt{k}}\cap\mathcal{F}_{\mathtt{s}}^{\mathtt{k}'}=\emptyset$ para todo $\mathtt{k},\mathtt{k}'$ y $\mathtt{r}\neq\mathtt{s}$. Finalmente, mostramos que para todo $\mathtt{k}$, se mantiene

$$\exists \mathtt{r}.(\mathtt{V},\mathtt{W})\oplus(\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r})>0 \iff \exists\,\mathtt{S}.\,\mathbb{S}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle})\neq\emptyset\wedge\mathtt{k}\in\mathtt{S}$$

Para $\Rightarrow$), asumir que $(\mathtt{V},\mathtt{W})\oplus(\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r})>0$ para alguna $\mathtt{r}$. Por la definición de $\oplus$, $\neg(\textsc{IsREM}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{r})\ \vee \textsc{IsREM}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r}))$. Consecuentemente,

$$\neg\ (\ (\mathtt{W}(\mathtt{k},\mathtt{r})=0\ \wedge\ \mathtt{W}'(\mathtt{k},\mathtt{r})\leq\mathtt{V}(\mathtt{r}))\ \ \vee\ \ (\mathtt{W}'(\mathtt{k},\mathtt{r})=0\ \wedge\ \mathtt{W}(\mathtt{k},\mathtt{r})\leq\mathtt{V}'(\mathtt{r}))\ )$$

Reorganizando términos,

$$\begin{array}{l}(\mathtt{W}(\mathtt{k},\mathtt{r})\neq 0\wedge\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0)\ \ \vee\ \ (\mathtt{W}(\mathtt{k},\mathtt{r})\neq 0\wedge\mathtt{W}(\mathtt{k},\mathtt{r})>\mathtt{V}'(\mathtt{r}))\ \ \vee\\ (\mathtt{W}'(\mathtt{k},\mathtt{r})>\mathtt{V}(\mathtt{r})\wedge\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0)\ \ \vee\ \ (\mathtt{W}'(\mathtt{k},\mathtt{r})>\mathtt{V}(\mathtt{r})\wedge\mathtt{W}(\mathtt{k},\mathtt{r})>\mathtt{V}'(\mathtt{r}))\end{array}$$

Procedemos por análisis de casos:

a) $\mathtt{W}(\mathtt{k},\mathtt{r})\neq 0\wedge\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0$. Dado que $(\langle\mathtt{r},\mathtt{V},\mathtt{W}\rangle,\langle\mathtt{G},\mathtt{P}\rangle)\in\mathcal{I}$, $\mathtt{W}(\mathtt{k},\mathtt{r})>0$ implica $\exists\,\mathtt{S}.\ \mathbb{S}_{\textit{Or-Set}}(\mathtt{G}^{\langle \mathtt{lookup},\mathtt{S}\rangle})\neq\emptyset\ \wedge\ \mathtt{k}\in\mathtt{S}$. Por la definición de $\mathbb{S}_{\textit{Or-Set}}$, $\exists\mathtt{e}'\in\mathtt{E}_{\mathtt{G}}$ tal que $\lambda_{\mathtt{G}}(\mathtt{e}')=\langle\mathtt{add}(\mathtt{k}),\mathtt{ok}\rangle$ y $\forall\mathtt{e}''.\mathtt{e}'\prec_{\mathtt{G}}\mathtt{e}''$ implica $\lambda_{\mathtt{G}}(\mathtt{e}'')\neq\langle\mathtt{rem}(\mathtt{k}),\mathtt{ok}\ \rangle$. Más aún, dado que $\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0$, la definición de $\mathtt{G}'$ es tal que solo los casos (i) y (ii) en $\mathcal{F}^{\mathtt{k}_{\mathtt{r}}}$ aplican. Consecuentemente, $\forall\mathtt{e}''.\mathtt{e}'\prec_{\mathtt{G}\sqcup\mathtt{G}'}\mathtt{e}''$ implica $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'')\neq\langle\mathtt{rem}(\mathtt{k}),\mathtt{ok}\ \rangle$. En consecuencia, $\mathbb{S}_{\textit{Or-Set}}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle})\neq\emptyset$ implica $\mathtt{k}\in\mathtt{S}$.

b) $\mathtt{W}(\mathtt{k},\mathtt{r})\neq 0\wedge\mathtt{W}(\mathtt{k},\mathtt{r})>\mathtt{V}'(\mathtt{r})$. Si $\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0$, entonces la prueba es análoga al caso anterior. La otra posibilidad ($\mathtt{W}'(\mathtt{k},\mathtt{r})=0$), el único caso posible en la definición de $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ es (iv) y podemos razonar también análogamente al caso anterior para concluir que $\mathbb{S}_{\textit{Or-Set}}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle})\neq\emptyset$ implica $\mathtt{k}\in\mathtt{S}$.

c) $\mathtt{W}'(\mathtt{k},\mathtt{r})>\mathtt{V}(\mathtt{r})\wedge\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0$. En consecuencia, solo el caso (ii) en la definición de $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ aplica. Consecuentemente, para $\mathtt{a}_{\mathtt{r}}^{\mathtt{k}}$ se mantiene que $\forall\mathtt{e}''.\mathtt{a}_{\mathtt{r}}^{\mathtt{k}}\prec_{\mathtt{G}\sqcup\mathtt{G}'}\mathtt{e}''$ implica $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'')\neq\langle\mathtt{rem}(\mathtt{k}),\mathtt{ok}\ \rangle$. En consecuencia, $\mathbb{S}_{\textit{Or-Set}}(\mathtt{G}^{\langle \mathtt{lookup},\mathtt{S}\rangle})\neq\emptyset$ implica $\mathtt{k}\in\mathtt{S}$.

d) $\mathtt{W}'(\mathtt{k},\mathtt{r})>\mathtt{V}(\mathtt{r})\wedge\mathtt{W}(\mathtt{k},\mathtt{r})>\mathtt{V}'(\mathtt{r})$. Ahora, notar que $\mathtt{W}'(\mathtt{k},\mathtt{r})>\mathtt{V}(\mathtt{r})$ implica $\mathtt{W}'(\mathtt{k},\mathtt{r})\neq 0$ y $\mathtt{W}(\mathtt{k},\mathtt{r})>\mathtt{V}'(\mathtt{r})$ implica $\mathtt{W}(\mathtt{k},\mathtt{r})\neq 0$. Entonces, la prueba se sigue como en el caso a)).

Ahora, considerar el caso en el cual $(\mathtt{V},\mathtt{W})\oplus(\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r})=0$ para todo $\mathtt{r}$. Al igual que antes, concluimos que para todo $\mathtt{r}$, se mantiene:

$$(\mathtt{W}(\mathtt{k},\mathtt{r})=0\ \wedge\ \mathtt{W}'(\mathtt{k},\mathtt{r})\leq\mathtt{V}(\mathtt{r}))\ \ \vee\ \ (\mathtt{W}'(\mathtt{k},\mathtt{r})=0\ \wedge\ \mathtt{W}(\mathtt{k},\mathtt{r})\leq\mathtt{V}'(\mathtt{r}))$$

Procedemos por análisis de casos para todo

a) $\mathtt{W}(\mathtt{k},\mathtt{r}) = 0 \wedge \mathtt{W}'(\mathtt{k},\mathtt{r}) \leq \mathtt{V}(\mathtt{r})$. Por $\mathtt{W}(\mathtt{k},\mathtt{r}) = 0$ y $(\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle, \langle \mathtt{G},\mathtt{P}\rangle) \in \mathcal{I}$ deducimos que no hay $\mathtt{e}' \in \mathtt{E}_{\mathtt{G}}$ tal que $\lambda(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ y $\forall \mathtt{e}''.\mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}''$ implica $\lambda(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$. Más aún, el caso aplicable en la definición de $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ son (i), (iii), y (iv). Consecuentemente, no hay $\mathtt{e}' \in \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \cup \mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ tal que $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ y para todo $\mathtt{e}''$ si $\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{e}''$ entonces $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$.

b) $\mathtt{W}'(\mathtt{k},\mathtt{r}) = 0 \wedge \mathtt{W}(\mathtt{k},\mathtt{r}) \leq \mathtt{V}'(\mathtt{r})$. Este caso se corresponde a (iii) en la definición de $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$. Por lo tanto, para todo $\mathtt{e}' \in \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \cup \mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ tal que $\lambda(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ tenemos que $\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{r}^{\mathtt{k}}$. Consecuentemente, no hay $\mathtt{e}' \in \mathtt{E}_{\mathtt{r}}^{\mathtt{k}} \cup \mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ tal que $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ y para todo $\mathtt{e}''$ si $\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{e}''$ entonces $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$.

Dado que los dos casos de arriba se mantienen para todo k and r, concluimos que

$$\forall \mathtt{r}.(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) = 0 \Rightarrow \forall\, \mathtt{S}.\ \mathbb{S}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle}) = \emptyset \vee \mathtt{k} \notin \mathtt{S}$$

Para $\Leftarrow$), asumir que existe S y k tal que $\mathbb{S}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle}) \neq \emptyset \wedge \mathtt{k} \in \mathtt{S}$. Entonces, existe $\mathtt{e}'$ tal que $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ y $\forall \mathtt{e}''.\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{e}''$ implica $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$. Hay dos casos:

1. $\mathtt{e}' \in \mathtt{E}_{\mathtt{G}}$. Por definición de $\mathtt{G}\sqcup\mathtt{G}'$, $\lambda_{\mathtt{G}}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ y $\forall \mathtt{e}''.\mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}''$ implica $\lambda_{\mathtt{G}}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$. Dado que $(\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle, \langle \mathtt{G},\mathtt{P}\rangle) \in \mathcal{I}$, entonces existe r tal que $\mathtt{W}(\mathtt{k},\mathtt{r}) > 0$. Consecuentemente, $\neg\textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{r})$ se mantiene. Dado que $\forall \mathtt{e}''.\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{e}''$ implica $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'') \neq \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\,\rangle$, concluimos que $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ es tal que o (i), (ii) ó (iv) se mantienen. Para (i) o (ii), notar que $\mathtt{W}'(\mathtt{k},\mathtt{s}) \neq 0$ y, por lo tanto, $\neg\textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r})$ se mantiene. Para (iii), $\mathtt{W}(\mathtt{k},\mathtt{r}) \leq \mathtt{V}'(\mathtt{r})$ implica $\neg\textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r})$. Consecuentemente, $\neg\textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{r})$ y $\neg\textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r})$. En consecuencia,

$$(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) = \mathtt{max}\{\mathtt{W}(\mathtt{k},\mathtt{r}), \mathtt{W}'(\mathtt{k},\mathtt{r})\}$$

   Dado que $\mathtt{W}(\mathtt{k},\mathtt{r}) > 0$, $(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) > 0$ se mantiene.

2. $\mathtt{e}' \notin \mathtt{E}_{\mathtt{G}}$. Then, $\mathtt{e}' \notin \mathtt{E}_{\mathtt{G}'}$. La única posibilidad es $\mathtt{e}' = \mathtt{a}_{\mathtt{r}}^{\mathtt{k}}$ para algún $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ (caso (ii)). En consecuencia, $\mathtt{W}'(\mathtt{k},\mathtt{r}) \neq 0$ y $\mathtt{W}'(\mathtt{k},\mathtt{r}) > \mathtt{V}(\mathtt{r})$. El siguiente caso vale al notar que $\mathtt{W}'(\mathtt{k},\mathtt{r}) \neq 0$ implica $\neg\textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r})$ y $\mathtt{W}'(\mathtt{k},\mathtt{r}) > \mathtt{V}(\mathtt{r})$ implica $\neg\textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{r})$.

Ahora, considere que existe k tal que para todo S, si $\mathbb{S}((\mathtt{G}\sqcup\mathtt{G}')^{\langle \mathtt{lookup},\mathtt{S}\rangle}) \neq \emptyset$ then $\mathtt{k} \notin \mathtt{S}$. Consecuentemente, para todo $\mathtt{e}' \in \mathtt{E}_{\mathtt{G}\sqcup\mathtt{G}'}$ if $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}') = \langle \mathtt{add}(\mathtt{k}), \mathtt{ok}\rangle$ entonces existe $\mathtt{e}''$ tal que $\mathtt{e}' \prec_{\mathtt{G}\sqcup\mathtt{G}'} \mathtt{e}''$ y $\lambda_{\mathtt{G}\sqcup\mathtt{G}'}(\mathtt{e}'') = \langle \mathtt{rem}(\mathtt{k}), \mathtt{ok}\rangle$. Entonces, $\mathcal{F}_{\mathtt{r}}^{\mathtt{k}}$ fue obtenida por usar o bien (i), (iii) ó (iv). Caso (iii) es usada, entonces $\mathtt{W}'(\mathtt{k},\mathtt{r}) = 0$ y $\mathtt{W}(\mathtt{k},\mathtt{r}) \leq \mathtt{V}'(\mathtt{r})$. En consecuencia, $\textsc{IsRem}(\mathtt{W}',\mathtt{W},\mathtt{V}',\mathtt{k},\mathtt{r})$ se mantiene y $(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) = 0$. Para casos (i) y (iv), primero notamos que $\mathtt{e}'' \in \mathtt{E}_{\mathtt{G}}$. Por lo tanto, para todo S, si $\mathbb{S}(\mathtt{G}^{\langle \mathtt{lookup},\mathtt{S}\rangle}) \neq \emptyset$ entonces $\mathtt{k} \notin \mathtt{S}$. Dado que $(\langle \mathtt{r},\mathtt{V},\mathtt{W}\rangle, \langle \mathtt{G},\mathtt{P}\rangle) \in \mathcal{I}$, tenemos $\mathtt{W}(\mathtt{k},\mathtt{r}) = 0$ para todo r. Adicionalmente, (i) implica $\mathtt{W}'(\mathtt{k},\mathtt{r}) \leq \mathtt{V}(\mathtt{r})$ y en consecuencia, $\textsc{IsRem}(\mathtt{W},\mathtt{W}',\mathtt{V},\mathtt{k},\mathtt{r})$ se mantiene. Consecuentemente, $(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) = 0$.

Caso (iv) también implica $\mathtt{W}'(\mathtt{k},\mathtt{r}) = 0$. Por lo tanto, $\mathtt{max}\{\mathtt{W}(\mathtt{k},\mathtt{r}),\mathtt{W}'(\mathtt{k},\mathtt{r})\} = 0$ y $(\mathtt{V},\mathtt{W}) \oplus (\mathtt{V}',\mathtt{W}')(\mathtt{k},\mathtt{r}) = 0$.

En consecuencia, $(\langle \mathtt{r}, \mathtt{max}\{\mathtt{V},\mathtt{V}'\}, (\mathtt{W},\mathtt{V}) \oplus (\mathtt{W}',\mathtt{V}')\rangle, \langle \mathtt{G} \sqcup \mathtt{G}', \mathtt{P}''\rangle) \in \mathcal{I}$ se mantiene. □

Apéndice B

Demostraciones para Parte II

B.1. Demostraciones sobre los resultados del Capítulo 6

DEMOSTRACIÓN DE LEM. 18.

Demostración.

Item 1. La prueba se realiza por análisis de casos sobre la regla aplicada.

- rule (A-UPDATE). Entonces, $\sigma \xrightarrow{e \rhd [x \leftarrow\!\shortmid v]_{s,r},} \sigma'$ y $\sigma' = \sigma \cdot e \rhd [x \leftarrow\!\shortmid v]_{s,r}^{\{\}\{r\}}$ por regla (UPDATE). Por Def. 6.2.2, σ' es bien-formado. Además, sabemos que $\mathcal{A}' = \mathcal{A} \overset{e \rhd [x \leftarrow\!\shortmid v]_s}{\oplus} \mathtt{E}_r$ es una ejecución abstracta por Lem. 17.

  Entonces, revisamos que las condiciones en Def. 6.2.3 se cumplen:

  1.

$$\begin{array}{rcll}
|\sigma'| & = & |\sigma \cdot e \rhd [x \leftarrow\!\shortmid v]_{s,r}^{\{\}\{r\}}| & \text{por Def.} \\
& = & |\sigma| + |[e \rhd [x \leftarrow\!\shortmid v]_{s,r}^{\{\}\{r\}}]| & \text{por Prop. de } |_| \\
& = & |\{e' \in \mathcal{E} \mid \mathtt{OP}(e') \in \mathbb{U}\}| + |[e \rhd [x \leftarrow\!\shortmid v]_{s,r}^{\{\}\{r\}}]| & \text{por Def. 6.2.3.1} \\
& = & |\{e' \in \mathcal{E} \mid \mathtt{OP}(e') \in \mathbb{U}\}| + |\{e \mid \mathtt{OP}(e) \in \mathbb{U}\}| & \text{por Def. 6.2.3.2} \\
& = & |\{e' \in \mathcal{E} \mid \mathtt{OP}(e') \in \mathbb{U}\} \cup \{e \mid \mathtt{OP}(e) \in \mathbb{U}\}| & \text{por Prop. de } |_| \\
& = & |\{e' \in \mathcal{E} \cup \{e\} \mid \mathtt{OP}'(e') \in \mathbb{U}\}| & \text{por Def. de } \oplus \\
& = & |\{e' \in \mathcal{E}' \mathtt{OP}'(e') \in \mathbb{U}\}| &
\end{array}$$

  2. $e' \rhd u \in \sigma'$ si y sólo si $e' \in \mathcal{E}' \wedge \mathtt{OP}'(e') = u$. Esta vale ya que $e' \rhd u \in \sigma'$ sii $e' \rhd u \in \sigma$ ó $e' = e$ y $u = e \rhd [x \leftarrow\!\shortmid v]_{s,r}^{\{\}\{r\}}$. El caso $e' \rhd u \in \sigma$ se sigue por $\langle \mathcal{A}, \sigma \rangle$. El segundo caso se cumple por la definición de $\_ \overset{\rhd}{\oplus} \_$.
  3. Dado que $\mathtt{AR}' = \mathtt{AR} \cup (\mathtt{E}_r \times \{e\})$, la propiedad se mantiene para $(e', e) \in \mathtt{AR}$. Resta revisar que sise mantiene también para $(e', e) \in (\mathtt{E}_r \times \{e\})$.

$$\forall e' \in \mathtt{E}_r . (e', e) \in \mathtt{AR}' \text{ sii } \forall \sigma' \equiv \sigma'', \sigma'' = \sigma_0 \cdot (e' \rhd u') \cdot \sigma_1 \cdot (e \rhd u) \cdot \sigma_2$$

Entonces,

$$\begin{array}{lll} e' \in E_r & \text{sii} \quad e' \rhd u' \in \sigma & \text{por Def. 6.2.3.2} \\ (e', e) \in (E_r \times \{e\}) & \text{sii} \quad \{r\} \cap u'.R \neq \emptyset & \text{por Def. 6.1.1} \end{array}$$

Por lo tanto, e y e' no pueden ser intercambiados en σ', y la propiedad se mantiene.

4. Mostramos que:

$$\forall j.\ AR'\big|_{E_j} \text{ con } E_j = \{e \mid e \rhd u \in \sigma' \wedge\ j \in u.R\} \text{ es un orden total}$$

Para $j \neq r$, la propiedad se mantiene $AR'|_{E_j} = AR|_{E_j}$ Para $j = r$, notar que $AR_{E_i \setminus \{e\}}$ es total para todos los eventos de $E_i \setminus \{e\}$. Más aún, $AR' = AR \cup (E_i \times \{e\})$ implica que $AR'_{E_j} = AR_{E_i \setminus \{e\}} \cup (E_i \times \{e\})$, el cual es total.

- rule (A-READ). Esta se sigue inmediatamente porque todas las propiedades en Def. 6.2.3 refieren a escrituras en σ'.

- rule (A-ARBITRATION). No aplica.

Item 2. La única regla aplicable es (ARBITRATION). Sea $\sigma = \sigma_0 \cdot e \rhd u \cdot \sigma_1 \cdot \sigma_2$. por regla (PROPAGATION), $\sigma \xrightarrow{\tau[\tilde{\sigma_0} \cdot e \cdot \tilde{\sigma_1}],} \sigma'$ con $\sigma' = \sigma_0 \cdot e \rhd u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1 \cdot \sigma_2$ y:

(i) $\mathtt{arb}(\sigma_0 \cdot e \rhd u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1)$;

(ii) $\sigma_1 \cdot \sigma_2 \equiv \sigma_2 \cdot \sigma_1$;

(iii) $\forall u' \in \sigma_2.r \notin u'.R$.

Entonces, Conditiones de Def. 6.2.3.1-2 se mantienen fácilmente porque $OP' = OP$ y $\mathcal{E}' = \mathcal{E}$ y los eventos en σ' son aquellos en σ. Análizamos las condiciones restantes a continuación:

3. Dado que $AR' = AR \cup [\tilde{\sigma_0} \cdot e \cdot \tilde{\sigma_1}]$, la propiedad se mantiene para $(e', e) \in AR$. Resta revisar que:

$$\begin{array}{l} \forall e' \in \tilde{\sigma_0}.(e', e) \in AR' \text{ sii } \forall \sigma'' \equiv \sigma', \sigma'' = \sigma'_0 \cdot (e' \rhd u') \cdot \sigma'_1 \cdot (e \rhd u) \cdot \sigma'_2 \text{ y} \\ \forall e'' \in \tilde{\sigma_1}.(e, e'') \in AR' \text{ sii } \forall \sigma''' \equiv \sigma', \sigma''' = \sigma''_0 \cdot (e \rhd u) \cdot \sigma''_1 \cdot (e'' \rhd u'') \cdot \sigma''_2 \end{array}$$

La parte del *si* de la doble implicación vale inmediatamente en ambos casos porque (i) asegura que $\sigma_0 \cdot e \rhd u[R \mapsto u.R \cup \{r\}] \cdot \sigma_1$ es completamente arbitrado y en consecuencia los eventos no pueden ser intercambiados. La parte del *sólo-si* vale por (iii) que asegura que la propagación no introduce ninguna arbitración nueva entre e y los eventos en σ_2.

4. Mostramos que:

$$\forall \mathsf{j}.\ \mathtt{AR}|_{\mathsf{E_j}} \text{ con } \mathsf{E_j} = \{\mathsf{e} \mid \mathsf{e} \rhd \mathsf{u} \in \sigma' \wedge\ \mathsf{j} \in \mathsf{u.R}\} \text{ es un orden total}$$

Cuando $\mathsf{j} \neq \mathsf{r}$, la propiedad se mantiene porque $\mathtt{AR}'|_{\mathsf{E_j}} = \mathtt{AR}|_{\mathsf{E_j}}$. Para $\mathsf{j} = \mathsf{r}$, tenemos que $\mathsf{E}'_\mathsf{r} = \mathsf{E_r} \cup \{\mathsf{e}\}$ ya que $\mathsf{e} \rhd \mathsf{u} \in \sigma'$ por Def. 6.2.3.2 y $\mathsf{r} \in \mathsf{u.R}$. Además,

$$\forall \mathsf{e}' \in \mathsf{E_r}.\mathsf{e}' \in [\tilde{\sigma_0} \cdot \mathsf{e} \cdot \tilde{\sigma_1}]$$

por (i) y (iii). La prueba se completa notando que $\mathtt{AR}'|_{\mathsf{E_i}}$ es total porque $[\tilde{\sigma_0} \cdot \mathsf{e} \cdot \tilde{\sigma_1}]$ es total.

□

DEMOSTRACIÓN DE RYW (LEM. 19).

Demostración. La prueba se hace por inducción en la longitud de la derivación $\Rightarrow_{RYW}$.

- **n=0**. Este caso se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{RYW} \langle \sigma', \mathcal{A}' \rangle \rightarrow_{RYW} \langle \sigma, \mathcal{A} \rangle$$

Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{RYW} \langle \sigma', \mathcal{A}' \rangle$, tenemos

$$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle \text{ y } \mathtt{SO}'\big|_{\mathcal{U} \times \mathcal{R}} \subseteq \mathtt{VIS}'$$

Procedemos por análisis de casos sobre la última regla aplicada:

- rule (A-UPDATE). Entonces, $\mu = \mathsf{e} \rhd [\mathsf{x} \leftarrowtail \mathsf{v}]_\mathsf{s}$, $\mathsf{E_r} = \{\ \mathsf{e}' \mid \mathsf{e}' \rhd \mathsf{u}' \in \sigma' \ \wedge\ \mathsf{r} \in \mathsf{u}'.\mathsf{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathsf{e} \rhd [\mathsf{x} \leftarrowtail \mathsf{v}]_\mathsf{s}}{\oplus} \mathsf{E_r}$ y $\sigma' \xrightarrow{\mathsf{e} \rhd [\mathsf{x} \leftarrowtail \mathsf{v}]_{\mathsf{s},\mathsf{r}},} \sigma$. Por definición de la operación $\oplus$ sobre ejecuciones abstractas:
  (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\})$.
  (ii) $\mathtt{VIS} = \mathtt{VIS}'$.
  Entonces,

$$\begin{array}{rcll} \mathtt{SO}|_{\mathcal{U} \times \mathcal{R}} & = & \mathtt{SO}'|_{\mathcal{U} \times \mathcal{R}} & \text{por (i)} \\ & \subseteq & \mathtt{VIS}' & \text{por IH} \\ & = & \mathtt{VIS} & \text{por (ii)} \end{array}$$

- regla (A-READ). Entonces, $\mu = \mathsf{e} \rhd (\mathsf{v} \leftarrowtail \mathsf{x})_\mathsf{s}$, $\mathsf{E_r} = \{\ \mathsf{e}' \mid \mathsf{e}' \rhd \mathsf{u}' \in \sigma \wedge \mathsf{r} \in \mathsf{u}'.\mathsf{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathsf{e} \rhd (\mathsf{v} \leftarrowtail \mathsf{x})_\mathsf{s}}{\oplus} \mathsf{E_r}$, y $\sigma' \xrightarrow{\mathsf{e} \rhd (\mathsf{v} \leftarrowtail \mathsf{x})_{\mathsf{s},\mathsf{r}},} \sigma$. Por definición de extensión de una ejecución abstracta:
  (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\})$.
  (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup (\mathsf{E_r} \times \{\mathsf{e}\})$.

Por lo tanto, probaremos que $(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\}))|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

$$\begin{array}{rcll} \mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} & \subseteq & \mathtt{VIS}' & \text{por IH} \\ & \subseteq & \mathtt{VIS} & \text{por (ii)} \end{array}$$

Resta probar que

$$(\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\})\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$$

Por lo tanto,

$$\begin{array}{rcll} & \mathtt{SS}'(\mathsf{s})|_{\mathcal{U}} & & \\ = & \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \delta \wedge \delta.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & & \text{por def of } \mathtt{SS}'(\mathsf{s}) \\ = & \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{u}.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & & \text{por } |_{\mathcal{U}\times\mathcal{R}} \end{array}$$

Ahora, hay dos posibilidades para $\sigma' \xrightarrow{\mathsf{e}\rhd(\mathsf{v}\leftarrow\!\shortmid\mathsf{x})_{\mathsf{s},\mathsf{r}},} \sigma$:

- regla (READ), en consecuencia $\sigma' \circledcirc \rhd(\mathsf{v} \leftarrow\!\shortmid \mathsf{x})_{\mathsf{s},\mathsf{r}}$; por definición de *read-consistency* para RYW(6.2) tenemos:

  $$\forall(\mathsf{e}'' \rhd \mathsf{u}'') \in \sigma'.\ \mathsf{s} = \mathsf{u}''.\mathsf{s} \implies \mathsf{r} \in \mathsf{u}''.\mathtt{R}$$

  En consecuencia,

  $$\begin{array}{rcll} & \mathtt{SS}'(\mathsf{s})|_{\mathcal{U}} & & \\ = & \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{u}.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & & \\ \subseteq & \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{r} \in \mathsf{u}''.\mathtt{R}\}|_{\mathcal{U}} & & \text{por } \circledcirc_{RYW} \\ = & \mathtt{E}_{\mathsf{r}} & & \text{por Def. 6.2.3.2} \end{array}$$

  Por lo tanto, $(\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\})|_{\mathcal{U}\times\mathcal{R}} \subseteq (\mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\}) \subseteq \mathtt{VIS}$.
- regla (READ-EMPTY). Esta es análoga al caso anterior notando que la condición $\sigma' \circledcirc \rhd(\mathsf{v} \leftarrow\!\shortmid \mathsf{x})_{\mathsf{s},\mathsf{r}}$ se mantiene.

- regla (A-ARBITRATION) se cumple inmediatamente por hipótesis inductiva luego de notar que $\mathtt{VIS}' = \mathtt{VIS}$ y $\mathtt{SO}' = \mathtt{SO}$.

□

DEMOSTRACIÓN DE COMPLETENESS OF RYW (LEM. 20).

Demostración. Esta se realiza por inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. Se mantiene trivialmente tomando $\sigma = \varepsilon$ porque $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$.
- **n=k**. Sea e un elemento maximal en AR, es decir,, $\nexists \mathsf{e}'.(\mathsf{e}, \mathsf{e}') \in \mathtt{AR}$. De aquí en adelante, $\mathcal{E}' = \mathcal{E} \setminus \{\mathsf{e}\}$ y $\mathcal{A}|_{\mathcal{E}'} = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'} \rangle$. Es sencillo chequear

que $\mathcal{A}|_{\mathcal{E}'}$ es un ejecución abstracta tal que $\mathtt{SO}|_{\mathcal{E}'} \subseteq \mathtt{VIS}|_{\mathcal{E}'}$. Dado e es maximal, AR puede ser escrito como:

$$\mathtt{AR}|_{\mathcal{E}'} \cup (\mathcal{E}^l \times \{\mathsf{e}\}) \text{ para algún } \mathcal{E}^l \subseteq \mathcal{E}'$$

Por IH sobre $\mathcal{A}|_{\mathcal{E}'}$, existe $\sigma', \mathtt{VIS}'', \mathtt{AR}''$ tal que

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{RYW} \langle \sigma', \mathcal{A}'' \rangle$$

y $\mathcal{A}'' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}'', \mathtt{AR}'' \rangle$, $\mathtt{VIS}'' = (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}$, $(\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}''$ y $(\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}''$.
La prueba se realiza mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ tal que la siguiente computación es posible $\langle \sigma', \mathcal{A}'' \rangle \Rightarrow_{RYW} \langle \sigma, \mathcal{A}' \rangle$ y

$$\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} \text{ y } \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$$

Hay dos casos:

- $\mathtt{OP}(\mathsf{e}) = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s}} = \delta$. Sea $\alpha = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s},\mathsf{r}}$ tal que r es una réplica fresca, es decir,, $\nexists \mathsf{x}', \mathsf{v}', \mathsf{s}'$ tal que $\sigma' = \sigma_0 \cdot \mathsf{e} \rhd [\mathsf{x}' \hookleftarrow \mathsf{v}']_{\mathsf{s}',\mathsf{r}} \cdot \sigma_1$. La estrategia para construir la computación es (i) realizar la escritura δ sobre la réplica fresca r y luego, (ii) propagar cada escritura relacionada a e en AR a la réplica fresca r.
  Ahora chequeamos que existe σ'' y $\mathcal{A}'''$ tal que $\langle \sigma', \mathcal{A}'' \rangle \to \langle \sigma'', \mathcal{A}''' \rangle$ aplicando (A-UPDATE). Notar que:
  - e es fresca: porque $\mathsf{e} \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2 e no aparece en σ';
  - $\sigma' \xrightarrow{\mathsf{e}\rhd\alpha,} \sigma''$ puede ser derivado por la definición de *write-consistency* para RYW (6.3), la cual establece que la escritura es posible sobre cualquier sistema, es decir,, $\_ \circledcirc \_$ sii **true** $\forall \sigma', \mathsf{e} \rhd \alpha$;

  Consecuentemente, (A-UPDATE) puede ser aplicado. Por definición de extensión de una ejecución abstracta:

$$\mathcal{A}''' = \mathcal{A}'' \overset{\mathsf{e}\rhd\delta}{\oplus} \mathtt{E}_{\mathsf{r}} \quad = \quad \langle \begin{array}{l} \mathcal{E}' \cup \{\mathsf{e}\}, \\ \mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta], \\ \mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{\mathsf{e}\}) \\ \mathtt{VIS}|_{\mathcal{E}'}, \\ \mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\}) \rangle \end{array}$$

  Notar que

$$\mathtt{E}_{\mathsf{r}} = \{\, \mathsf{e}' \mid \mathsf{e}' \rhd \alpha \in \sigma' \wedge \mathsf{r} \in \alpha.\mathtt{R} \,\} = \emptyset$$

  ya que r es fresca. Por lo tanto, $\mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\}) = \mathtt{AR}|_{\mathcal{E}'}$. Entonces, tenemos:
  - $\mathcal{E}' \cup \{\mathsf{e}\} = \mathcal{E}$

- $\mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta] = \mathtt{OP}$
- $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{\mathsf{e}\}) = \mathtt{SO}$
- Dado que $\mathtt{OP}(\mathsf{e}) \in \mathbb{U}$, por definición $\mathtt{VIS}|_{\mathcal{E}'} = \mathtt{VIS}|_{\mathcal{E}}$. En consecuencia, $(\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} = (\mathtt{VIS}|_{\mathcal{E}})|_{\mathcal{U}\times\mathcal{R}} = \mathtt{VIS}''$
- Sin embargo, $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}}$ no se mantiene en general, porque el evento fresco e no ha sido arbitrado. Mostramos que la computación puede ser completada con varias aplicaciones de la regla propagación para obtener una ejecución abstracta que coincide con $\mathcal{A}'''$ en todas las componentes but $\mathtt{AR}|_{\mathcal{E}'}$.

Considere el siguiente mecanismo para propagar escrituras a una réplica fresca r. Sea $\sigma^l = [\mathsf{e}_1^l \rhd \mathsf{u}_1^l, \cdots, \mathsf{e}_m^l \rhd \mathsf{u}_m^l]$ la subsecuencia de σ'' tal que $\mathsf{e}_j^l \in \mathcal{E}^l$,es decir,, σ^l representa las escrituras en σ relacionada a e por AR. Entonces, usamos la regla (A-ARBITRATION) m-veces para propagar cada u_j^l a una réplica fresca r la cual $\mathsf{e} \rhd \alpha$ ha sido realizada.

En la primera propagación, tenemos:

$$\begin{array}{lll} \sigma_0 & = & \varepsilon, \\ \sigma_1 & = & \mathsf{e} \rhd \alpha \text{ y} \\ \sigma_2 & \text{s.t.} & \sigma'' = \sigma_0 \cdot (\mathsf{e}_1^l \rhd \mathsf{u}_1^l[\mathtt{R} \mapsto \mathsf{u}_1^l.\mathtt{R} \cup \{\mathsf{r}\}]) \cdot \sigma_1 \cdot \sigma_2 \end{array}$$

Entonces, $\sigma'' \xrightarrow{\tau[\tilde{\sigma_0} \cdot \mathsf{e}_1^l \cdot \tilde{\sigma_1}]} \sigma_1''$, con $\mathtt{AR}_1'' = \mathtt{AR}'' \cup (\mathsf{e}_1^l \times \mathsf{e})$. En general, para la propagación j^{th}, tomaremos:

$$\begin{array}{lll} \sigma_{0_j} & = & [\mathsf{e}_1^l \rhd \mathsf{u}_1^l, \ldots, \mathsf{e}_{j-1}^l \rhd \mathsf{u}_{j-1}^l], \\ \sigma_{1_j} & = & \mathsf{e} \rhd \alpha \text{ y} \\ \sigma_{2_j} & \text{s.t.} & \sigma_j'' = \sigma_{0_j} \cdot (\mathsf{e}_j^l \rhd \mathsf{u}_j^l[\mathtt{R} \mapsto \mathsf{u}_j^l.\mathtt{R} \cup \{\mathsf{r}\}]) \cdot \sigma_{1_j} \cdot \sigma_{2_j} \end{array}$$

Y $\sigma_{j-1}'' \xrightarrow{\tau[\tilde{\sigma_{0_j}} \cdot \mathsf{e}_j^l \cdot \tilde{\sigma_{1_j}}]} \sigma_j''$ con $\mathtt{AR}_j'' = \mathtt{AR}_{j-1}'' \cup [\tilde{\sigma_{0_j}} \cdot \mathsf{e} \cdot \tilde{\sigma_{1_j}}]$.
Consecuentemente,

$$\langle \sigma'', \mathcal{A}''' \rangle \stackrel{\cdot}{\rightarrow}, \langle \sigma_1'', \mathcal{A}_1''' \rangle \stackrel{\cdot}{\rightarrow} \cdots \stackrel{\cdot}{\rightarrow}, \langle \sigma_m'', \mathcal{A}_m''' \rangle = \langle \sigma, \mathcal{A}' \rangle$$

y notar que

$$\begin{array}{lll} \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} & = & (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \cup (\mathcal{E}^l \times \{\mathsf{e}\}) \\ & \subseteq & (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \cup (\mathcal{E}^l \times \{\mathsf{e}\}) \quad \text{por IH} \\ & \subseteq & (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \cup [\tilde{\sigma_{0_m}} \cdot \mathsf{e} \cdot \tilde{\sigma_{1_m}}] \\ & = & \mathtt{AR}' \end{array}$$

- $\mathtt{OP}(\mathsf{e}) = (\mathsf{v} \leftarrow \mathsf{x})_{\mathsf{s}} = \delta$. Sea $\alpha = (\mathsf{v} \leftarrow \mathsf{x})_{\mathsf{s},\mathsf{r}}$ tal que r es una réplica fresca. Diferentemente del caso anterior, la estrategia aquí es (i) propagar cada escritura

relacionada a e en AR a la réplica fresca r, y, luego, (ii) realizar la acción de lectura.

La propagación es análoga al caso de la escritura. En consecuencia chequeamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma''_m, \mathcal{A}'''_m \rangle \rightarrow \langle \sigma, \mathcal{A}' \rangle$ usando (A-READ). Notar que:

- e es fresca: porque $e \notin \mathcal{E}'$ y en consecuencia por Def. 6.2.3.2 e no aparece en σ';
- $\sigma''_m \xrightarrow{e \triangleright \alpha,}, \sigma$: ya que acción (READ) se satisface $\sigma''_m \circledcirc \alpha$, es decir,:

$$\forall (e'' \triangleright u'') \in \sigma''_m.\ \alpha.s = u''.s \implies r \in u''.R$$

Tomando los eventos que aseguran las condiciones anteriores, tenemos:

$$\begin{array}{l} \{e'' \mid e'' \triangleright u'' \in \sigma'' \ \wedge\ (\ \alpha.s = u''.s \implies r \in u''.R)\} \\ \subseteq \{\ e'' \mid e'' \triangleright u'' \in \sigma'' \ \wedge\ r \in u''.R\ \} = E_r \end{array}$$

Finalmente, por definición de extensión de una ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A} = \mathcal{A}''_m \overset{e \triangleright \delta}{\oplus} E_r & = & \langle & \mathcal{E}' \cup \{e\}, \\ & & & OP|_{\mathcal{E}'}[e \mapsto \delta], \\ & & & SO|_{\mathcal{E}'} \cup (\ SS(s)|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{e\}) \\ & & & VIS|_{\mathcal{E}'} \cup (E_r \times \{e\}) \\ & & & AR''_m \cup (E_r \times \{e\})\} \rangle \end{array}$$

Entonces

- $\mathcal{E}' \cup \{e\} = \mathcal{E}$
- $OP|_{\mathcal{E}'}[e \mapsto \delta] = OP$
- $SO|_{\mathcal{E}'} \cup (\ SS(s)|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{e\}) = SO$
- $VIS|_{\mathcal{E}'} \cup (E_r \times \{e\}) = VIS$
- Observar que $(E_r \times \{e\}) \subseteq AR''_m$ ya que hemos propagado los eventos asociados con e en AR. En consecuencia

$$\begin{array}{rcll} AR|_{\mathcal{U} \times \mathcal{U}} & = & (AR|_{\mathcal{E}'})|_{\mathcal{U} \times \mathcal{U}} \cup (\mathcal{E}^l \times \{e\}) & \\ & \subseteq & AR'' \cup (\mathcal{E}^l \times \{e\}) & \text{por IH} \\ & \subseteq & AR''_1 \cup (\mathcal{E}^l \times \{e\}) & \text{por (PROPAGATION)} \\ & \vdots & & \\ & \subseteq & AR''_m \cup (\mathcal{E}^l \times \{e\}) & \\ & \subseteq & AR''_m = AR' & \end{array}$$

□ □

DEMOSTRACIÓN DE SOUNDNESS Y COMPLETENESS PARA EL RESTO DE LOS MODELOS DE CONSISTENCIA

Lema 28 (Soundness MR)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MR} \langle \sigma, \mathcal{A} \rangle$ *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $(\mathtt{VIS};\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

Demostración. La prueba se hace por inducción en la longitud de la derivación $\Rightarrow_{MR}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces
$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{MR} \langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$$
Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MR} \langle \sigma', \mathcal{A}' \rangle$, tenemos $\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle$ y $(\mathtt{VIS}';\mathtt{SO}')|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$. Procedemos por análisis de casos sobre la última regla aplicada:
  - regla (A-UPDATE). Entonces, $\mu = e \rhd [x \leftarrow\!\shortmid\, v]_s$, $E_r = \{\, e' \mid e' \rhd u' \in \sigma' \ \wedge\ r \in u'.\mathtt{R} \,\}$, $\mathcal{A} = \mathcal{A}' \overset{e\rhd[x\leftarrow\!\shortmid v]_s}{\oplus} E_r$ y $\sigma' \xrightarrow{e\rhd[x\leftarrow\!\shortmid v]_{s,r},} \sigma$. Por definición de la operación $\oplus$ de ejecución abstracta:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\})$.
    (ii) $\mathtt{VIS} = \mathtt{VIS}'$.

    Por lo tanto,
$$\begin{array}{rcll} \mathtt{VIS};\mathtt{SO} & = & \mathtt{VIS};(\mathtt{SO}' \cup (\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\}) & \text{por (i)} \\ & = & \mathtt{VIS}';(\mathtt{SO}' \cup (\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\}) & \text{por (ii)} \\ & = & \mathtt{VIS}';\mathtt{SO}' \cup \mathtt{VIS};(\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\}) & \end{array}$$
    Por hipótesis inductiva $(\mathtt{VIS}';\mathtt{SO}')|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{VIS}' = \mathtt{VIS}$. Por lo tanto, resta probar que $(\mathtt{VIS}';(\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\}))|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$. Sin embargo, dado que e está asociada a una escritura, tenemos
$$(\mathtt{VIS}';(\mathtt{SS}'(s) \times \{e\}))\big|_{\mathcal{U}\times\mathcal{R}} = \emptyset \subseteq \mathtt{VIS}' = \mathtt{VIS}$$
  - regla (A-READ). Luego, $\mu = e \rhd (v \leftarrow\!\shortmid x)_s$, $E_r = \{\, e' \mid e' \rhd u' \in \sigma \wedge r \in u'.\mathtt{R} \,\}$, $\mathcal{A} = \mathcal{A}' \overset{e\rhd(v\leftarrow\!\shortmid x)_s}{\oplus} E_r$, y $\sigma' \xrightarrow{e\rhd(v\leftarrow\!\shortmid x)_{s,r},} \sigma$. Por definición de extensión de ejecución abstracta:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(s)|_{\mathcal{E}'} \times \{e\})$.
    (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup (E_r \times \{e\})$.

    En consecuencia,
$$\begin{array}{rcll} \mathtt{VIS};\mathtt{SO} & = & (\mathtt{VIS}' \cup (E_r \times \{e\});\mathtt{SO} & \text{por (ii)} \\ & = & (\mathtt{VIS}';\mathtt{SO}) \cup ((E_r \times \{e\});\mathtt{SO}) & \end{array}$$

Dado que e es un evento fresco, no hay e' tal que $(\mathsf{e},\mathsf{e}') \in \mathtt{SO}'$. En consecuencia $((\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\});\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} = \emptyset$. Resta probar que $(\mathtt{VIS}';\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

$$\begin{array}{rcll} \mathtt{VIS}';\mathtt{SO} & = & \mathtt{VIS}';(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})|_{\mathcal{E}'} \times \{\mathsf{e}\})) & \text{por (i)} \\ & = & (\mathtt{VIS}';\mathtt{SO}') \cup (\mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})|_{\mathcal{E}'} \times \{\mathsf{e}\})) & \text{distribuyendo} \end{array}$$

Por hipótesis inductiva, $(\mathtt{VIS}';\mathtt{SO}')|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$. Por definición, $\mathtt{VIS}' \subseteq \mathtt{VIS}$. Por lo tanto, $\mathtt{VIS}';\mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$ se mantiene. Resta probar que

$$(\mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\big|_{\mathcal{E}'} \times \{\mathsf{e}\})\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$$

Tomar $(\mathsf{e}',\mathsf{e}'') \in \mathtt{VIS}'$ tal que $\mathsf{e}'' \in \mathtt{SS}'(\mathsf{s})|_{\mathcal{E}'}$. Por $|_{\mathcal{U}\times\mathcal{R}}$, $\mathtt{OP}(\mathsf{e}') = \mathsf{u}'$, $\mathtt{OP}(\mathsf{e}'') = \mathsf{rd}''$. Más aún, por Def. 6.2.3.2 sabemos que $\forall(\mathsf{e}' \rhd \mathsf{u}') \in \sigma'$ sii $\mathsf{e}' \in \mathcal{E}'$.
Dado que $(\mathsf{e}',\mathsf{e}'') \in \mathtt{VIS}$, entonces $\mathsf{rd}''.\mathsf{r} \in \mathsf{u}'.\mathtt{R}$ y en consecuencia

$$\begin{array}{rl} \mathsf{rd}''.\mathsf{s} \in \mathsf{u}'.\mathtt{S} & \text{por (READ)} \\ \mathsf{s} \in \mathsf{u}'.\mathtt{S} & \text{por } \mathtt{SS}'(\mathsf{s})|_{\mathcal{E}'} \\ \mathsf{r} \in \mathsf{u}'.\mathtt{R} & \text{por } \circledcirc_{MR} \end{array}$$

Por lo tanto, $\mathsf{e}' \in \mathtt{E}_\mathsf{r}$, que significa $(\mathsf{e}',\mathsf{e}) \in \mathtt{VIS}$.

- (A-ARBITRATION) vale inmediatamente por hipótesis inductiva luego de notar que $\mathtt{VIS}' = \mathtt{VIS}$ y $\mathtt{SO}' = \mathtt{SO}$.

□

□

Lema 29 (Completeness MR)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}\rangle$ *una ejecución abstracta tal que* $(\mathtt{VIS};\mathtt{SO}) \subseteq \mathtt{VIS}$. *Entonces,*

$$\begin{array}{l} \exists\sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}'\rangle \textit{ tal que } \langle \varepsilon, \emptyset\rangle \Rightarrow_{MR} \langle \sigma, \mathcal{A}'\rangle, \\ \qquad \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, (\mathtt{VIS};\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} \textit{ y } \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}' \end{array}$$

Demostración. Probamos el resultado usando inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. Se mantiene trivialmente porque $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$, entonces, la propiedad se mantiene tomando $\sigma = \varepsilon$.

- **n=k**. Sea e un elemento maximal en AR, es decir,, $\nexists\mathsf{e}'.(\mathsf{e},\mathsf{e}') \in \mathtt{AR}$. De aquí en adelante, $\mathcal{E}' = \mathcal{E} \setminus \{\mathsf{e}\}$ y $\mathcal{A}|_{\mathcal{E}'} = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'}\rangle$. Es sencillo revisar que $\mathcal{A}|_{\mathcal{E}'}$ es una ejecución abstracta tal que $(\mathtt{VIS}|_{\mathcal{E}'};\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} \subseteq (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}$.
  Entonces, por hipótesis inductiva, $\exists\sigma', \mathtt{VIS}', \mathtt{AR}'$ tal que $\langle \varepsilon, \emptyset\rangle \Rightarrow_{MR} \langle \sigma', \mathcal{A}''\rangle$ con $\mathcal{A}'' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}'', \mathtt{AR}''\rangle$, $\mathtt{VIS}'' = \mathtt{VIS}|_{\mathcal{E}'}$, $(\mathtt{VIS}|_{\mathcal{E}'};\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}''$, y $\mathtt{AR}'' = \mathtt{AR}|_{\mathcal{E}'}$.

La prueba se sigue mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ tal que la siguiente computación es posible:

$$\langle \sigma', \mathcal{A}'' \rangle \Rightarrow_{MR} \langle \sigma, \mathcal{A}' \rangle, \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, (\mathtt{VIS};\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} \text{ y } \mathtt{AR}' = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}$$

Hay dos casos:

- $\mathtt{OP}(\mathsf{e}) = [\mathsf{x} \hookleftarrow \mathsf{v}]_\mathsf{s} = \delta$. Sea $\alpha = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s},\mathsf{r}}$ tal que r es una réplica fresca, es decir, $\nexists \mathsf{x}', \mathsf{v}', \mathsf{s}'$ tal que $\sigma' = \sigma_0 \cdot \mathsf{e} \rhd [\mathsf{x}' \hookleftarrow \mathsf{v}']_{\mathsf{s}',\mathsf{r}} \cdot\ \sigma_1$. La estrategia será (i) realizar la escritura α y luego, (ii) propagar cada escritura relacionad a e en $\mathtt{AR}$ a una réplica fresca r. Entonces, este caso se sigue análogamente al caso de la regla (UPDATE) en Demostración de Lem. 20.
- $\mathtt{OP}(\mathsf{e}) = (\mathsf{v} \hookleftarrow \mathsf{x})_\mathsf{s} = \delta$. Sea $\alpha = (\mathsf{v} \hookleftarrow \mathsf{x})_{\mathsf{s},\mathsf{r}}$. Diferentemente al caso anterior, la estrategia será elegir una réplica existente r. Usando la definición de *write-consistency* para MW (6.6), sabemos que:

  $$\exists \mathsf{r} \text{ tal que } (\forall \mathsf{u} \in \sigma'.\ \mathsf{s} \in \mathsf{u}.\mathtt{S} \implies \mathsf{r} \in \mathsf{u}.\mathtt{R})$$

  En consecuencia, revisamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma', \mathcal{A}'' \rangle \to \langle \sigma, \mathcal{A}' \rangle$ usando (A-READ). Notar que:

  - e es fresca: ya que $\mathsf{e} \notin \mathcal{E}'$ y en consecuencia por Def. 6.2.3.2 e no aparece en σ';
  - $\sigma' \xrightarrow{\mathsf{e}\rhd\alpha,}, \sigma$: Aplicando regla (READ), tomando la definición de *read-consistency* para MR (6.5), es decir,,

    $$\forall (\mathsf{e}' \rhd \mathsf{u}') \in \sigma'.\ \alpha.\mathsf{s} \in \mathsf{u}'.\mathtt{S} \implies \mathsf{r} \in \mathsf{u}'.\mathtt{R}$$

    Entonces, por definición de extensión de una ejecución abstracta:

    $$\begin{array}{rcll} \mathcal{A}' \overset{\mathsf{e}\rhd\delta}{\oplus} \mathtt{E}_\mathsf{r} & = & \langle & \mathcal{E}' \cup \{\mathsf{e}\}, \\ & & & \mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta], \\ & & & \mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{\mathsf{e}\}) \\ & & & \mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) \\ & & & \mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\})\rangle \end{array}$$

    Por lo tanto,
    - $\mathcal{E}' \cup \{\mathsf{e}\} = \mathcal{E}$
    - $\mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta] = \mathtt{OP}$
    - $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'}|_{\mathcal{E}'} \times \{\mathsf{e}\}) = \mathtt{SO}$
    - $\mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) = \mathtt{VIS}$
    - $\mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) = \mathtt{AR}$

Entonces, tomar $\mathtt{VIS}' = \mathtt{VIS}$, $\mathtt{AR}' = \mathtt{AR}$. Consecuentemente, $(\mathtt{VIS};\mathtt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$ se sigue análogamente a la prueba de soundness para MR.

□

Lema 30 (Soundness MW)**.** *Si* $\langle \varepsilon, \emptyset\rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A}\rangle$ *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}\rangle$ *y* $\mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow_{MW}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{AR} = \emptyset$.
- **n=k+1**. Entonces
$$\langle \varepsilon, \emptyset\rangle \Rightarrow_{MW} \langle \sigma', \mathcal{A}'\rangle \rightarrow \langle \sigma, \mathcal{A}\rangle$$
Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset\rangle \Rightarrow_{MW} \langle \sigma', \mathcal{A}'\rangle$, tenemos
$$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}'\rangle \text{ y } \mathtt{SO}'\Big|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$$
Procedemos por análisis de casos sobre la última regla aplicada:
  - regla (A-UPDATE). Entonces, $\mu = \mathsf{e} \rhd [\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathtt{s}}$, $\mathtt{E}_{\mathtt{r}} = \{\ \mathsf{e}' \mid \mathsf{e}' \rhd \mathtt{u}' \in \sigma \ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathsf{e}\rhd[\mathtt{x}\leftarrowtail\mathtt{v}]_{\mathtt{s}}}{\oplus} \mathtt{E}_{\mathtt{r}}$ y $\sigma' \xrightarrow{\mathsf{e}\rhd[\mathtt{x}\leftarrowtail\mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$. Más aún, por definición de la operación $\oplus$ de ejecuciones abstractas:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathsf{e}\})$.
    (ii) $\mathtt{AR} = \mathtt{AR}' \cup (\mathtt{E}_{\mathtt{r}} \times \{\mathsf{e}\})$.

    Entonces,
$$\mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \ = \ \mathtt{SO}'|_{\mathcal{U}\times\mathcal{U}} \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathsf{e}\})|_{\mathcal{U}\times\mathcal{U}} \quad \text{por (i) y } |_{\mathcal{U}\times\mathcal{U}}$$
    Dado que $\mathtt{SO}'|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$ vale por hipótesis inductiva resta probar que
$$(\mathtt{SS}'(\mathtt{s}) \times \{\mathsf{e}\})\Big|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{E}_{\mathtt{r}} \times \{\mathsf{e}\}$$
    En consecuencia,
$$\begin{array}{ll} & \mathtt{SS}'(\mathtt{s})|_{\mathcal{U}} \\ = & \{\mathsf{e}' \mid \mathsf{e}' \in \mathtt{E}' \ \wedge\ \mathtt{OP}(\mathsf{e}') = \delta \ \wedge\ \delta.\mathtt{s} = \mathtt{s}\}|_{\mathcal{U}} \qquad \text{por def de } \mathtt{SS}'(\mathtt{s}) \\ = & \{\mathsf{e}' \mid \mathsf{e}' \in \mathcal{E}' \ \wedge\ \mathtt{OP}(\mathsf{e}') = \mathtt{u} \ \wedge\ \mathtt{u} \in \mathbb{U} \ \wedge\ \mathtt{u}.\mathtt{s} = \mathtt{s}\}|_{\mathcal{U}} \quad \text{por } |_{\mathcal{U}} \end{array}$$
    Por regla (UPDATE), $\sigma' \xrightarrow{\mathsf{e}\rhd[\mathtt{x}\leftarrowtail\mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$, en consecuencia sabemos que $\sigma' \Subset [\mathtt{x} \leftarrowtail \mathtt{v}]_{\mathtt{s},\mathtt{r}}$, que por definición de *write-consistency* para MW(6.9) significa:
$$\forall(\mathsf{e}'' \rhd \mathtt{u}'') \in \sigma'.\ \mathtt{s} = \mathtt{u}''.\mathtt{s} \implies \{\mathtt{r}\} \subseteq \mathtt{u}''.\mathtt{R}$$

Por lo tanto,

$$\begin{array}{lll} & \mathtt{SS}'(\mathsf{s})|_{\mathcal{U}} & \\ = & \{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E}' \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{u}.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & \\ \subseteq & \{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E}' \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \{\mathsf{r}\} \subseteq \mathsf{u}''.\mathtt{R}\}|_{\mathcal{U}} & \text{por } \circledcirc_{MW} \\ = & \mathtt{E}_{\mathsf{r}} & \text{por Def. 6.2.3.2} \end{array}$$

En consecuencia, se mantiene.

- regla (A-READ) vale inmediatamente por hipótesis inductiva luego de notar que $\mathtt{SO}'|_{\mathcal{U}\times\mathcal{U}} = \mathtt{SO}|_{\mathcal{U}\times\mathcal{U}}$.
- regla (A-ARB) vale inmediatamente como en el caso anterior.

□ □

Lema 31 (Completeness MW)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}\rangle$ *una ejecución abstracta tal que* $\mathtt{SO} \subseteq \mathtt{AR}$*. Entonces,*

$$\begin{array}{l} \exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}'\rangle \textit{ tal que} \\ \qquad \langle \varepsilon, \emptyset\rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A}'\rangle, \mathtt{SO} \subseteq \mathtt{AR}\ y\ \ \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}' \end{array}$$

Demostración. Probamos el resultado usando inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. La propiedad se mantiene tomando $\sigma = \varepsilon$ ya que $\mathtt{SO} = \mathtt{AR} = \emptyset$.
- **n=k**. Sea e un elemento maximal en AR, es decir,, $\nexists \mathsf{e}'.(\mathsf{e}, \mathsf{e}') \in \mathtt{AR}$. De aquí en adelante, $\mathcal{E}' = \mathcal{E} \setminus \{\mathsf{e}\}$ y $\mathcal{A}|_{\mathcal{E}'} = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'}\rangle$. Es sencillo revisar que $\mathcal{A}|_{\mathcal{E}'}$ es una ejecución abstracta tal que $\mathtt{SO}|_{\mathcal{E}'} \subseteq \mathtt{AR}|_{\mathcal{E}'}$.

  Dado que e es maximal, AR puede ser escrita como:

  $$\mathtt{AR}|_{\mathcal{E}'} \cup (\mathcal{E}^{l} \times \{\mathsf{e}\}) \text{ para algún } \mathcal{E}^{l} \subseteq \mathcal{E}'$$

  Entonces, por hipótesis inductiva, $\exists \sigma', \mathtt{AR}''$ tal que $\langle \varepsilon, \emptyset\rangle \Rightarrow_{MW} \langle \sigma', \mathcal{A}''\rangle$ con $\mathcal{A}'' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}''\rangle$, $\mathtt{SO}'' \subseteq \mathtt{AR}|_{\mathcal{E}'}$ y $(\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}''$.

  La prueba se realiza mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}'\rangle$ tal que la siguiente computación es posible:

  $$\langle \sigma', \mathcal{A}''\rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A}'\rangle, \mathtt{SO}' = \mathtt{SO}|_{\mathcal{U}\times\mathcal{U}}, \mathtt{SO}' \subseteq \mathtt{AR}'\ y\ \ \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$$

  Entonces, hay dos casos:

  - $\mathtt{OP}(\mathsf{e}) = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s}} = \delta$. Sea $\alpha = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s},\mathsf{r}}$. Dado que $\mathtt{SO}_{\mathsf{s}}$ es total por Def. 6.2.1.4, entonces $(\mathtt{SO}_{\mathsf{s}})|_{\mathcal{E}'} \subseteq \mathtt{SO}_{\mathsf{s}}$ también es total. Entonces, sea e' un elemento maximal en $((\mathtt{SO}_{\mathsf{s}})|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}}$. Por Def. 6.2.3.2, sabemos que existe $\mathsf{e}' \rhd \mathsf{u}' \in \sigma'$. Por lo tanto, tomamos $\mathsf{r} = \mathsf{u}'.\mathsf{r}$. En este caso, la estrategia es realizar una computación con el fin de obtener $\langle \sigma, \mathcal{A}'\rangle$, es decir, sabemos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma', \mathcal{A}''\rangle \to \langle \sigma, \mathcal{A}'\rangle$ aplicando (A-UPDATE). Notar que:

- e es fresca: porque $e \notin \mathcal{E}'$ y en consecuencia por Def. 6.2.3.2, e no aparece en σ';
- $\sigma' \xrightarrow{e \triangleright \alpha,} , \sigma$: aplicando (UPDATE) ya que la definición de *write-consistency* para MW (6.9) se mantiene, es decir, $\sigma' \circledcirc [x \leftarrow\!\shortmid\, v]^{\{\}\{r\}}_{s,r}$ significa

$$\forall u' \in \sigma'.\ u'.s = s \implies \{r\} \subseteq u'.R$$

Además

$$E_r = \{\ e' \mid e' \triangleright \alpha' \in \sigma' \ \wedge\ r \in \alpha'.R\ \}$$

Entonces, por definición de extensión de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' \overset{e \triangleright \delta}{\oplus} E_r & = & \langle & \mathcal{E}' \cup \{e\}, \\ & & & OP|_{\mathcal{E}'}[e \mapsto \delta], \\ & & & SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) \\ & & & VIS|_{\mathcal{E}'}, \\ & & & AR|_{\mathcal{E}'} \cup (E_r \times \{e\})\rangle \end{array}$$

En consecuencia

- $\mathcal{E}' \cup \{e\} = \mathcal{E}$
- $OP|_{\mathcal{E}'}[e \mapsto \delta] = OP$
- $SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) = SO$
- $VIS|_{\mathcal{E}'} = VIS$
- Dado que e es maximal, entonces $E_r = \mathcal{E}^l$, por lo tanto, $AR|_{\mathcal{E}'} \cup (E_r \times \{e\}) = AR$.

Análogamente a la regla (A-UPDATE) para la prueba de soundness, $SS(s)|_{\mathcal{E}'} \subseteq E_r$. Por lo tanto, $SO \subseteq AR$

- $OP(e) = (v \leftarrow\!\shortmid\, x)_s = \delta$. Diferentemente a los casos anteriores, la estrategia aquí es (i) propagar cada escritura que fue relacionada a e en AR a una réplica fresca r, y (ii) realizar una acción de lectura.

  La propagación es análoga al caso de la escritura para la prueba de completeness de RYW. En consecuencia, chequeamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma''_m, \mathcal{A}'''_m \rangle \rightarrow \langle \sigma, \mathcal{A}' \rangle$ por (A-READ). Notar que:
  - e es fresca: ya que $e \notin \mathcal{E}'$ y en consecuencia por Def. 6.2.3.2 e no aparece en σ';
  - $\sigma''_m \xrightarrow{e \triangleright \alpha,} , \sigma$: esta puede ser derivada por la definición de *read-consistency* para MW (6.8), es decir, $\_ \circledcirc \_$ sii **true** $\forall \sigma', e \triangleright \alpha$;

  En consecuencia, por definición de extension de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' = \mathcal{A}''_m \overset{e \triangleright \delta}{\oplus} E_r & = & \langle & \mathcal{E}' \cup \{e\}, \\ & & & OP|_{\mathcal{E}'}[e \mapsto \delta], \\ & & & SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) \\ & & & VIS|_{\mathcal{E}'} \cup (E_r \times \{e\}) \\ & & & AR''_m \cup (E_r \times \{e\})\}\rangle \end{array}$$

Entonces

- $\mathcal{E}' \cup \{\mathtt{e}\} = \mathcal{E}$
- $\mathtt{OP}|_{\mathcal{E}'}[\mathtt{e} \mapsto \delta] = \mathtt{OP}$
- $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathtt{s})|_{\mathcal{E}'} \times \{\mathtt{e}\}) = \mathtt{SO}$
- $\mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E_r} \times \{\mathtt{e}\}) = \mathtt{VIS}$
- Observar que $(\mathtt{E_r} \times \{\mathtt{e}\}) \subseteq \mathtt{AR}''_m$ ya que hemos propagado los eventos asociados con e en AR. En consecuencia

$$\begin{array}{rcll} \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} & = & (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \\ & \subseteq & \mathtt{AR}'' \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \text{por IH} \\ & \subseteq & \mathtt{AR}''_1 \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \text{por (PROPAGATION)} \\ & \vdots & & \\ & \subseteq & \mathtt{AR}''_m \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \\ & \subseteq & \mathtt{AR}''_m = \mathtt{AR}' & \end{array}$$

Finalmente, $\mathtt{SO} \subseteq (\mathtt{E_r} \times \{\mathtt{e}\})$, se sigue de forma análoga al prueba de soundness para MW, por lo tanto $\mathtt{SO} \subseteq (\mathtt{E_r} \times \{\mathtt{e}\}) \subseteq \mathtt{AR}''_m \subseteq \mathtt{AR}$.

□

Lema 32 (Soundness de WFR)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma, \mathcal{A} \rangle$ *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $\mathtt{VIS}; \mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\overset{s}{\Rightarrow}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$.
- **n=k+1**. Entonces

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$$

Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \overset{s'}{\Rightarrow} \langle \sigma', \mathcal{A}' \rangle$, tenemos $\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle$ y $\mathtt{VIS}'; \mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{AR}'$. Procedemos por análisis de casos sobre la última regla aplicada:

- regla (A-UPDATE). Entonces, $\mu = \mathtt{e} \rhd [\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_\mathtt{s}$, $\mathtt{E_r} = \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma \ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_\mathtt{s}}{\oplus} \mathtt{E_r}$ y $\sigma' \xrightarrow{\mathtt{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:
  - (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.
  - (ii) $\mathtt{VIS} = \mathtt{VIS}'$.
  - (iii) $\mathtt{AR} = \mathtt{AR}' \cup (\mathtt{E_r} \times \{\mathtt{e}\})$.

Por lo tanto,

$$\begin{array}{rcll} \mathtt{VIS};\mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} & = & \mathtt{VIS};(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}))|_{\mathcal{R}\times\mathcal{U}} & \text{por (i)} \\ & = & \mathtt{VIS}';(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}))|_{\mathcal{R}\times\mathcal{U}} & \text{por (ii)} \\ & = & \mathtt{VIS}';\mathtt{SO}'|_{\mathcal{R}\times\mathcal{U}} \cup \mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})|_{\mathcal{R}\times\mathcal{U}} & \end{array}$$

Por hipótesis inductiva $\mathtt{VIS}';\mathtt{SO}'|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}' = \mathtt{AR}$. En consecuencia, resta probar que

$$\mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})\Big|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}$$

Tomar $(\mathsf{e}',\mathsf{e}'') \in \mathtt{VIS}'$ tal que $\mathsf{e}'' \in \mathtt{SS}'(\mathsf{s})$. Por $|_{\mathcal{R}\times\mathcal{U}}$, $\mathtt{OP}(\mathsf{e}'') = \mathsf{rd}''$. Más aún, por Def. 6.2.3.2 sabemos que $\forall(\mathsf{e}' \rhd \mathsf{u}') \in \sigma'$ sii $\mathsf{e}' \in \mathcal{E}'$.
Dado que $(\mathsf{e}',\mathsf{e}'') \in \mathtt{VIS}'$, entonces

$$\begin{array}{rl} \mathsf{rd}''.\mathsf{s} \in \mathsf{u}'.\mathtt{S} & \text{por (READ)} \\ \mathsf{s} \in \mathsf{u}'.\mathtt{S} & \text{por } \mathtt{SS}'(\mathsf{s}) \\ \{\mathsf{u}.\mathsf{r}\} \subseteq \mathsf{u}'.\mathtt{R} & \text{por } \lessdot_{WFR} \end{array}$$

En consecuencia, $\mathsf{e}' \in \mathtt{E}_\mathsf{r}$, que significa $(\mathsf{e}',\mathsf{e}) \in \mathtt{AR}$.

- regla (A-READ). Entonces, $\mu = \mathsf{e} \rhd (\mathsf{v} \leftarrowtail \mathsf{x})_\mathsf{s}$, $\mathtt{E}_\mathsf{r} = \{\, \mathsf{e}' \mid \mathsf{e}' \rhd \mathsf{u}' \in \sigma \wedge \mathsf{r} \in \mathsf{u}'.\mathtt{R} \,\}$, $\mathcal{A} = \mathcal{A}' \overset{\mathsf{e}\rhd(\mathsf{v}\leftarrowtail\mathsf{x})_\mathsf{s}}{\oplus} \mathtt{E}_\mathsf{r}$, y $\sigma' \xrightarrow{\mathsf{e}\rhd(\mathsf{v}\leftarrowtail\mathsf{x})_{\mathsf{s},\mathsf{r}},} \sigma$. Por definición de extensión de una ejecución abstracta:

  (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})$.

  (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup (\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\})$.

  (iii) $\mathtt{AR} = \mathtt{AR}' \cup (\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\})$.

  En consecuencia,

$$\begin{array}{rcll} \mathtt{VIS};\mathtt{SO} & = & (\mathtt{VIS}' \cup (\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\});\mathtt{SO} & \text{por (ii)} \\ & = & (\mathtt{VIS}';\mathtt{SO}) \cup ((\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\});\mathtt{SO}) & \text{distribuyendo} \end{array}$$

  Entonces, usando (i), tenemos

$$\begin{array}{l} (\mathtt{VIS}';(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}))) \cup ((\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\});(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}))) \\ = \mathtt{VIS}';\mathtt{SO}' \cup \mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}) \cup ((\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\});(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}))) \end{array}$$

  Por hipótesis inductiva $\mathtt{VIS}';\mathtt{SO}'|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}' = \mathtt{AR}$. Además, dado que e es la última acción, entonces $(\mathtt{E}_\mathsf{r}\times\{\mathsf{e}\});(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})) = \emptyset$. Por lo tanto, resta probar que

$$\mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})\Big|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}$$

  Sin embargo, por $|_{\mathcal{R}\times\mathcal{U}}$, $(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\}|_{\mathcal{R}\times\mathcal{U}} = \emptyset$. En consecuencia,

$$\mathtt{VIS}';(\mathtt{SS}'(\mathsf{s})\times\{\mathsf{e}\})\Big|_{\mathcal{R}\times\mathcal{U}} = \emptyset \subseteq \mathtt{AR}$$

- (A-ARBITRATION) vale inmediatamente por hipótesis inductiva luego de notar que $\mathtt{VIS}' = \mathtt{VIS}$ y $\mathtt{SO}' = \mathtt{SO}$.

□

Lema 33 (Completeness WFR)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *una ejecución abstracta tal que* $\mathtt{VIS}; \mathtt{SO} \subseteq \mathtt{AR}$*. Entonces,*

$$\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle \textit{ tal que } \langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma, \mathcal{A}' \rangle,$$
$$\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{VIS}'; \mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}' \textit{ y } \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$$

Demostración. Probamos el resultado usando inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. Se mantiene trivialmente porque $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$, entonces, la propiedad se mantiene tomando $\sigma = \varepsilon$.
- **n=k**. Sea e un elemento maximal en AR, es decir,, $\nexists e'.(e, e') \in \mathtt{AR}$. De aquí en adelante, $\mathcal{E}' = \mathcal{E} \setminus \{e\}$ y $\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'} \rangle$. Es sencillo revisar que $\mathcal{A}'$ es una ejecución abstracta tal que $\mathtt{VIS}|_{\mathcal{E}'}; (\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{E}'}$.

  Dado que e es maximal, AR puede ser escrito como:

  $$\mathtt{AR}|_{\mathcal{E}'} \cup (\mathcal{E}^l \times \{e\}) \text{ para algún } \mathcal{E}^l \subseteq \mathcal{E}'$$

  Entonces, por hipótesis inductiva, $\exists \sigma', \mathtt{SO}'', \mathtt{VIS}'', \mathtt{AR}''$ tal que $\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma', \mathcal{A}'' \rangle$ con $\mathcal{A}'' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}'', \mathtt{AR}'' \rangle$, $\mathtt{VIS}'' = (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}, \mathtt{VIS}''; (\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}''$ y $(\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}''$.

  La prueba sigue mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ tal que $\langle \sigma', \mathcal{A}'' \rangle \Rightarrow_{WFR} \langle \sigma, \mathcal{A}' \rangle$ es posible con:

  $$\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, \quad \mathtt{VIS}'; \mathtt{SO}' \subseteq \mathtt{AR}' \quad \text{y } \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}'$$

  Entonces hay dos casos:

  - $\mathtt{OP}(e) = [x \hookleftarrow v]_s = \delta$. Sea $\alpha = [x \hookleftarrow v]_{s,r}$ tal que r es una réplica fresca, es decir,, $\nexists x', v', s'$ tal que $\sigma' = \sigma_0 \cdot e \rhd [x' \hookleftarrow v']_{s',r} \cdot \sigma_1$. La estrategia será (i) propagar cada escritura que fue relacionada a e en AR para una réplica fresca r y entonce, (ii) realizar una acción de escritura.

    La propagación es análoga al caso de escritura de la prueba de completeness de RYW.

    $$\langle \sigma', \mathcal{A}'' \rangle \xrightarrow{\cdot}, \langle \sigma'_1, \mathcal{A}''_1 \rangle \xrightarrow{\cdot} \cdots \xrightarrow{\cdot}, \langle \sigma'_m, \mathcal{A}''_m \rangle = \langle \sigma'', \mathcal{A}''' \rangle$$

    En consecuencia, revisamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma'', \mathcal{A}''' \rangle \to \langle \sigma, \mathcal{A}' \rangle$ aplicando (A-UPDATE). Notar que:

- e es fresca: ya que $e \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2, e no aparece en σ';
- $\sigma' \xrightarrow{e \triangleright \alpha,} , \sigma''$: por aplicar (UPDATE) ya que $\sigma' \circledcirc [x \hookleftarrow v]^{\{\}\{r\}}_{s,r}$ se mantiene, es decir,

$$\forall u' \in \sigma'.\ s \in u'.S \implies \{r\} \subseteq u'.R$$

Luego, por la definición de extensión de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' \overset{e \triangleright \delta}{\oplus} E_r & = & \langle & \mathcal{E}' \cup \{e\}, \\ & & & OP|_{\mathcal{E}'}[e \mapsto \delta], \\ & & & SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) \\ & & & VIS|_{\mathcal{E}'}, \\ & & & AR''_m \cup (E_r \times \{e\})\rangle \end{array}$$

Por lo tanto

- $\mathcal{E}' \cup \{e\} = \mathcal{E}$
- $OP|_{\mathcal{E}'}[e \mapsto \delta] = OP$
- $SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) = SO$
- Dado que $OP(e) \in \mathbb{U}$, por definición $VIS|_{\mathcal{E}'} = VIS|_{\mathcal{E}}$. En consecuencia, $(VIS|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} = (VIS|_{\mathcal{E}})|_{\mathcal{U}\times\mathcal{R}} = VIS'$
- Dado que $E_r = \mathcal{E}^l$ entonces $AR|_{\mathcal{U}\times\mathcal{U}} \subseteq AR|_{\mathcal{E}'} \cup (E_r \times \{e\}) = AR'$.

Análogamente a la regla (A-UPDATE) para la prueba de soundness, $SS(s)|_{\mathcal{E}'} \subseteq E_r$. Por lo tanto, $VIS'; SO|_{\mathcal{R}\times\mathcal{U}} \subseteq AR'$.

- $OP(e) = (v \hookleftarrow x)_s = \delta$. Sea $\alpha = (v \hookleftarrow x)_{s,r}$ tal que r es una réplica fresca. Análogamente al caso anterior, propagamos cada escritura que fue relacionada a e en AR a la réplica fresca r, y entonces, realizaremos la acción de lectura. Por lo tanto, revisamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma'', \mathcal{A}''' \rangle \to \langle \sigma, \mathcal{A}' \rangle$ aplicando (A-READ). Notar que:
  - e es fresca: porque $e \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2, e no aparece en σ';
  - $\sigma' \xrightarrow{e \triangleright \alpha,} , \sigma''$: esta puede ser derivada porque la definición de *read-consistency* para WFR (6.12), es decir, $\_ \circledcirc \_$ sii **true** $\forall \sigma', e \triangleright \alpha$;

Entonces, por la definición de extensión de una ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' \overset{e \triangleright \delta}{\oplus} E_r & = & \langle & \mathcal{E}' \cup \{e\}, \\ & & & OP|_{\mathcal{E}'}[e \mapsto \delta], \\ & & & SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) \\ & & & VIS|_{\mathcal{E}'} \cup (E_r \times \{e\}), \\ & & & AR''_m \cup (E_r \times \{e\})\rangle \end{array}$$

En consecuencia

- $\mathcal{E}' \cup \{\mathtt{e}\} = \mathcal{E}$
- $\mathtt{OP}|_{\mathcal{E}'}[\mathtt{e} \mapsto \delta] = \mathtt{OP}$
- $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathtt{s})|_{\mathcal{E}'} \times \{\mathtt{e}\}) = \mathtt{SO}$
- Dado que $\mathtt{OP}(\mathtt{e}) \in \mathbb{U}$, por definición $\mathtt{VIS}|_{\mathcal{E}'} = \mathtt{VIS}|_{\mathcal{E}}$. Por lo tanto, $(\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} = (\mathtt{VIS}|_{\mathcal{E}})|_{\mathcal{U}\times\mathcal{R}} = \mathtt{VIS}'$
- Dado que $\mathtt{E_r} = \mathcal{E}^l$ entonces $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E_r} \times \{\mathtt{e}\}) = \mathtt{AR}'$.

Análogamente a la regla (A-UPDATE) para la prueba de soundness, $\mathtt{SS}(\mathtt{s})|_{\mathcal{E}'} \subseteq \mathtt{E_r}$. Por lo tanto, $\mathtt{VIS}'; \mathtt{SO}|_{\mathcal{R}\times\mathcal{U}} \subseteq \mathtt{AR}'$.

□

Comenzamos probando algunos resultados auxiliares que serán de utilidad para probar visibilidad causal.

Lema 34. *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A} \rangle$, *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $(\mathtt{SO}; \mathtt{VIS})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow_{CV}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces,

  $$\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$$

  Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma', \mathcal{A}' \rangle$, tenemos

  $$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle \text{ y } (\mathtt{SO}'; \mathtt{VIS}')\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$$

  Procedemos por análisis de casos sobre la última regla aplicada:

  - regla (A-UPDATE). Entonces, $\mu = \mathtt{e} \rhd [\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s}}$, $\mathtt{E_r} = \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma \ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_{\mathtt{s}}}{\oplus} \mathtt{E_r}$ y $\sigma' \xrightarrow{\mathtt{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_{\mathtt{s,r}},} \sigma$. Por definición de la operación $\oplus$ de una ejecución abstracta:

    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.

    (ii) $\mathtt{VIS} = \mathtt{VIS}'$.

    Entonces,

    $$\begin{array}{lcll}
    (\mathtt{SO};\mathtt{VIS})|_{\mathcal{U}\times\mathcal{R}} & = & (\mathtt{SO};\mathtt{VIS}')|_{\mathcal{U}\times\mathcal{R}} & \text{por (ii)} \\
    & = & ((\mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\}));\mathtt{VIS}')|_{\mathcal{U}\times\mathcal{R}} & \text{por (i)} \\
    & = & (\mathtt{SO}';\mathtt{VIS}')|_{\mathcal{U}\times\mathcal{R}} \cup ((\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\}));\mathtt{VIS}')|_{\mathcal{U}\times\mathcal{R}} & \text{distribuyendo} \\
    & \subseteq & (\mathtt{SO}';\mathtt{VIS}')|_{\mathcal{U}\times\mathcal{R}} & |_{\mathcal{U}\times\mathcal{R}} \\
    & = & \mathtt{VIS}' \cup \emptyset & \text{IH} \\
    & = & \mathtt{VIS} & \text{por (ii)}
    \end{array}$$

- regla (A-READ). Entonces, $\mu = \mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s}}$, $\mathtt{E_r} = \{\, \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma \wedge \mathtt{r} \in \mathtt{u}'.\mathtt{R} \,\}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s}}}{\oplus} \mathtt{E_r}$, y $\sigma' \xrightarrow{\mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s,r}};} \sigma$. Por definición de extensión de una ejecución abstracta:

  (i) $\mathtt{SO} = \mathtt{SO}' \cup \mathtt{SO_{new}}$ con $\mathtt{SO_{new}} = (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.

  (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup \mathtt{VIS_{new}}$ con $\mathtt{VIS_{new}} = (\mathtt{E_r} \times \{\mathtt{e}\})$.

  Por lo tanto, probamos que

  $$((\mathtt{SO}' \cup \mathtt{SO_{new}}); (\mathtt{VIS}' \cup \mathtt{VIS_{new}}))\Big|_{\mathcal{U} \times \mathcal{R}} \subseteq \mathtt{VIS}$$

  Notar que $\mathtt{SO_{new}}; \mathtt{VIS}' = (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\}); \mathtt{VIS}' = \emptyset$ porque e es fresca y en consecuencia no aparece en $\mathtt{VIS}'$. Entonces, usando IH:

  $$(\mathtt{SO}'; \mathtt{VIS}')\Big|_{\mathcal{U} \times \mathcal{R}} \subseteq \mathtt{VIS}' = \mathtt{VIS}$$

  Resta probar que

  $$((\mathtt{SO}' \cup \mathtt{SO_{new}}); \mathtt{VIS_{new}})\Big|_{\mathcal{U} \times \mathcal{R}} \subseteq \mathtt{VIS}$$

  Análogamente al caso anterior $\mathtt{SO_{new}}; \mathtt{VIS_{new}}; = \emptyset$ porque e es fresca.

  Por lo tanto, queremos probar que

  $$\forall (\mathtt{e}'', \mathtt{e}) \in (\mathtt{SO}'; \mathtt{VIS_{new}})|_{\mathcal{U} \times \mathcal{R}} . \exists \mathtt{e}' \text{ tal que } (\mathtt{e}'', \mathtt{e}') \in \mathtt{SO} \text{ y } (\mathtt{e}', \mathtt{e}) \in \mathtt{VIS_{new}} \text{ sii } (\mathtt{e}'', \mathtt{e}) \in \mathtt{VIS}$$

  Analizamos dos casos:

  - $\mathtt{e}' \in \mathtt{SS}(\mathtt{s})$, entonces sabemos que $\sigma' \circledcirc \rhd (\mathtt{v} \leftarrow\!\shortmid\, \mathtt{x})_{\mathtt{s,r}}$, por definición de *read-consistency* para CV(6.17):

    $$\forall \mathtt{u} \in \sigma . (\mathtt{s} = \mathtt{u.s} \vee \mathtt{s} \in \mathtt{u.S}) \implies \mathtt{r} \in \mathtt{u.R}$$

    Esto significa que:

    $$\begin{array}{lll} & \mathtt{SS}'(\mathtt{s})|_{\mathcal{U}} & \\ = & \{\mathtt{e}'' \mid \mathtt{e}'' \in \mathcal{E} \wedge \mathtt{OP}(\mathtt{e}'') = \delta \wedge \delta.\mathtt{s} = \mathtt{s}\}|_{\mathcal{U}} & \text{por def } \mathtt{SS}'(\mathtt{s}) \\ = & \{\mathtt{e}'' \mid \mathtt{e}'' \in \mathcal{E} \wedge \mathtt{OP}(\mathtt{e}'') = \mathtt{u} \wedge \mathtt{u} \in \mathbb{U} \wedge \mathtt{u.s} = \mathtt{s}\}|_{\mathcal{U}} & \text{por } |_{\mathcal{U}} \\ \subseteq & \{\mathtt{e}'' \mid \mathtt{e}'' \in \mathcal{E} \wedge \mathtt{OP}(\mathtt{e}'') = \mathtt{u} \wedge \mathtt{u} \in \mathbb{U} \wedge \mathtt{r} \in \mathtt{u.R}\}|_{\mathcal{U}} & \text{por } \circledcirc_{CV} \\ = & \mathtt{E_r} & \end{array}$$

    Por lo tanto, $\mathtt{e}'' \in \mathtt{E_r}$ que implica que

    $$(\mathtt{e}'', \mathtt{e}) \in (\mathtt{E_r} \times \{\mathtt{e}\}) = \mathtt{VIS_{new}} \subseteq \mathtt{VIS}$$

- $e' \notin \mathtt{SS}(s)$, dado que $e' \in \mathtt{E_r}$, por Def. 6.2.3.2 existe $e' \rhd u' \in \sigma'$ tal que $r \in u.\mathtt{R}$. Además, $\mathtt{OP}(e'') = u''$ y $e'' \rhd u'' \in \sigma'$.
  Por IH, sabemos que ambas escrituras fueron arbitradas y la definición de *write-consistency* para CV(6.18) se mantiene, es decir,

$$\forall u'' \in \sigma.\ (s = u'.s \ \vee\ s \in u'.\mathtt{S} \implies u'.\mathtt{R} \subseteq u''.\mathtt{R})$$

  Por lo tanto, $e' \in \mathtt{E_r}$ y en consecuencia $r \in u'.\mathtt{R} \subseteq u''.\mathtt{R}$ que significa que

$$(e'', e) \in (\mathtt{E_r} \times \{e\}) = \mathtt{VIS_{new}} \subseteq \mathtt{VIS}$$

□

Lema 35. *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A} \rangle$*, entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $((\mathtt{VIS};\mathtt{SO})|_{\mathcal{U}\times\mathcal{U}};\mathtt{VIS})\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow_{CV}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,, $\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces,

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma', \mathcal{A}' \rangle \rightarrow\ \langle \sigma, \mathcal{A} \rangle$$

  Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma', \mathcal{A}' \rangle$, tenemos

$$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle \text{ y } (\mathtt{VIS}';\mathtt{SO}';\mathtt{VIS}')\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$$

  Procedemos por análisis de casos sobre la última regla aplicada:

  - regla (A-UPDATE). Entonces, $\mu = e \rhd [x \leftarrow\!\shortmid\ v]_s$, $\mathtt{E_r} = \{\ e' \mid e' \rhd u' \in \sigma\ \wedge\ r \in u'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{e\rhd[x\leftarrow\!\shortmid v]_s}{\oplus} \mathtt{E_r}$ y $\sigma' \xrightarrow{e\rhd[x\leftarrow\!\shortmid v]_{s,r},} \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(s) \times \{e\})$.
    (ii) $\mathtt{VIS} = \mathtt{VIS}'$.

    Dado VIS no cambia, esta vale inmediatamente.
  - regla (A-READ). Entonces, $\mu = e \rhd (v \leftarrow\!\shortmid x)_s$, $\mathtt{E_r} = \{\ e' \mid e' \rhd u' \in \sigma\ \wedge\ r \in u'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{e\rhd(v\leftarrow\!\shortmid x)_s}{\oplus} \mathtt{E_r}$, y $\sigma' \xrightarrow{e\rhd(v\leftarrow\!\shortmid x)_{s,r},} \sigma$. Por definición de extensión de ejecución abstracta:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup \mathtt{SO_{new}}$ con $\mathtt{SO_{new}} = (\mathtt{SS}'(s) \times \{e\})$.
    (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup \mathtt{VIS_{new}}$ con $\mathtt{VIS_{new}} = (\mathtt{E_r} \times \{e\})$.

Por regla (READ), $\sigma' \xrightarrow{e \rhd (v \leftarrow\!\shortmid x)_{s,r},} \sigma$ y sabemos que por IH that la regla de escritura ($e' \rhd u'$) cumple *write-consistency* para CV(6.18), es decir, :

$$\forall u'' \in \sigma''.(u'.s \in u''.S \vee u'.s = u''.s) \implies u'.R \subseteq u''.R$$

Por lo tanto, $r \in u'.R \subseteq u''.R$.
Consecuentemente, $e'' \in E_r$ que implica

$$(e'', e) \in (E_r \times \{e\}) = \mathtt{VIS}_{\mathsf{new}} \subseteq \mathtt{VIS}$$

- $e' \notin \mathtt{SS}(s)$, dado que $e' \in E_r$, por Def. 6.2.3.2 existe $e' \rhd u' \in \sigma'$ tal que $r \in u.R$. Además, $\mathtt{OP}(e'') = u''$ y $e'' \rhd u'' \in \sigma'$.
  Por IH, sabemos que ambas escrituras fueron arbitradas y la definición de *write-consistency* para CV(6.18) se mantiene, es decir,

  $$\forall u'' \in \sigma.\ (s = u'.s \ \vee\ s \in u'.S \implies u'.R \subseteq u''.R)$$

  En consecuencia, $r \in u'.R \subseteq u''.R$ que significa que si $(e''', e'') \in \mathtt{VIS}'$, $(e'', e') \in \mathtt{SO}'$ y $(e', e) \in \mathtt{VIS}$ entonces $e''', e' \in E_r$ y por lo tanto

  $$(e''', e) \in (E_r \times \{e\}) = \mathtt{VIS}_{\mathsf{new}} \subseteq \mathtt{VIS}$$

□

Lema 36 (Monotonic Write en Visibilidad)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MW} \langle \sigma, \mathcal{A} \rangle$, *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $\mathtt{SO}; \mathtt{VIS} \subseteq \mathtt{VIS}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow_{MW}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,$\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces,

  $$\langle \varepsilon, \emptyset \rangle \Rightarrow_{MW} \langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$$

  Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{MW} \langle \sigma', \mathcal{A}' \rangle$, tenemos

  $$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle \text{ y } \mathtt{SO}'; \mathtt{VIS}' \subseteq \mathtt{VIS}'$$

  Procedemos por análisis de casos sobre la última regla aplicada:

  - regla (A-UPDATE). Entonces, $\mu = e \rhd [x \leftarrow\!\shortmid v]_s$, $E_r = \{\ e' \mid e' \rhd u' \in \sigma \ \wedge\ r \in u'.R\ \}$, $\mathcal{A} = \mathcal{A}' \overset{e \rhd [x \leftarrow\!\shortmid v]_s}{\oplus} E_r$ y $\sigma' \xrightarrow{e \rhd [x \leftarrow\!\shortmid v]_{s,r},} \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:
    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(s) \times \{e\})$.
    (ii) $\mathtt{VIS} = \mathtt{VIS}'$.

Como VIS no cambia, esta vale inmediatamente.

- regla (A-READ). Entonces, $\mu = \mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\ \mathtt{x})_{\mathtt{s}}$, $\mathtt{E_r} = \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma\ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\ \mathtt{x})_{\mathtt{s}}}{\oplus} \mathtt{E_r}$, y $\sigma' \xrightarrow{\mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\ \mathtt{x})_{\mathtt{s},\mathtt{r}},} \sigma$. Por definición de extensión de ejecución abstracta:
  (i) $\mathtt{SO} = \mathtt{SO}' \cup \mathtt{SO_{new}}$ con $\mathtt{SO_{new}} = (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.
  (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup \mathtt{VIS_{new}}$ con $\mathtt{VIS_{new}} = (\mathtt{E_r} \times \{\mathtt{e}\})$.

  Entonces, tenemos que probar que:

$$\mathtt{SO}' \cup \mathtt{SO_{new}}; \mathtt{VIS}' \cup \mathtt{VIS_{new}} \subseteq \mathtt{VIS}$$

  Distribuyendo, tenemos dos casos:

  - $\mathtt{SO_{new}}; \mathtt{VIS}' \cup \mathtt{VIS_{new}} = \emptyset$ porque el evento fresco es un maximal, y por lo tanto $\emptyset \subseteq \mathtt{VIS}$.
  - $\mathtt{SO}'; \mathtt{VIS}' \cup \mathtt{VIS_{new}} = \mathtt{SO}'; \mathtt{VIS}' \cup \mathtt{SO}'; \mathtt{VIS_{new}}$. Luego,
    - $\mathtt{SO}'; \mathtt{VIS}' \subseteq \mathtt{VIS}'$ por IH, y $\mathtt{VIS}' \subseteq \mathtt{VIS}$.
    - $\mathtt{SO}'; \mathtt{VIS_{new}} \subseteq \mathtt{VIS}$.

      Por regla (READ), $\sigma' \xrightarrow{\mathtt{e} \rhd (\mathtt{v} \leftarrow\!\shortmid\ \mathtt{x})_{\mathtt{s},\mathtt{r}},} \sigma$ y sabemos que por IH that la regla de escritura ($\mathtt{e}' \rhd \mathtt{u}'$) cumple *write-consistency* para MW(6.9), es decir, :

$$\forall \mathtt{u}'' \in \sigma''.\mathtt{u}'.\mathtt{s} = \mathtt{u}''.\mathtt{s} \implies \mathtt{u}'.\mathtt{R} \subseteq \mathtt{u}''.\mathtt{R}$$

      Por lo tanto, $\mathtt{r} \in \mathtt{u}'.\mathtt{R} \subseteq \mathtt{u}''.\mathtt{R}$.
      Consecuentemente, $\mathtt{e}'' \in \mathtt{E_r}$ que implica

$$(\mathtt{e}'', \mathtt{e}) \in (\mathtt{E_r} \times \{\mathtt{e}\}) = \mathtt{VIS_{new}} \subseteq \mathtt{VIS}$$

□

Lema 37 (Writes Follow Read en Visibilidad)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma, \mathcal{A} \rangle$*, entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $\mathtt{VIS}; \mathtt{SO}; \mathtt{VIS} \subseteq \mathtt{VIS}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow_{WFR}$.

- **n=0**. Se mantiene trivialmente porque $\mathcal{A} = \emptyset$, es decir,$\mathtt{SO} = \mathtt{VIS} = \emptyset$.
- **n=k+1**. Entonces,

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma', \mathcal{A}' \rangle \rightarrow\ \langle \sigma, \mathcal{A} \rangle$$

  Por hipótesis inductiva sobre $\langle \varepsilon, \emptyset \rangle \Rightarrow_{WFR} \langle \sigma', \mathcal{A}' \rangle$, tenemos

$$\mathcal{A}' = \langle \mathcal{E}', \mathtt{OP}', \mathtt{SO}', \mathtt{VIS}', \mathtt{AR}' \rangle \text{ y } \mathtt{VIS}'; \mathtt{SO}'; \mathtt{VIS}' \subseteq \mathtt{VIS}'$$

  Procedemos por análisis de casos sobre la última regla aplicada:

- regla (A-UPDATE). Entonces, $\mu = e \rhd [x \leftarrow\!\shortmid\, v]_s$, $E_r = \{\ e' \mid e' \rhd u' \in \sigma \ \wedge\ r \in u'.R\ \}$, $\mathcal{A} = \mathcal{A}' \overset{e \rhd [x \leftarrow\!\shortmid\, v]_s}{\oplus} E_r$ y $\sigma' \xrightarrow{e \rhd [x \leftarrow\!\shortmid\, v]_{s,r},} \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:
  (i) $SO = SO' \cup (SS'(s) \times \{e\})$.
  (ii) $VIS = VIS'$.

  Como VIS no cambia, esta vale inmediatamente.
- regla (A-READ). Entonces, $\mu = e \rhd (v \leftarrow\!\shortmid\, x)_s$, $E_r = \{\ e' \mid e' \rhd u' \in \sigma \ \wedge\ r \in u'.R\ \}$, $\mathcal{A} = \mathcal{A}' \overset{e \rhd (v \leftarrow\!\shortmid\, x)_s}{\oplus} E_r$, y $\sigma' \xrightarrow{e \rhd (v \leftarrow\!\shortmid\, x)_{s,r},} \sigma$. Por definición de extensión de ejecución abstracta:
  (i) $SO = SO' \cup SO_{new}$ con $SO_{new} = (SS'(s) \times \{e\})$.
  (ii) $VIS = VIS' \cup VIS_{new}$ con $VIS_{new} = (E_r \times \{e\})$.

  Entonces, tenemos que probar que:

  $$(VIS' \cup VIS_{new}); (SO' \cup SO_{new}); (VIS' \cup VIS_{new}) \subseteq VIS$$

  Es importante notar que al distribuir aparecen casos donde el evento maximal aparece del lado de izquierdo de las relaciones, por lo que las composiciones son vacías. Por lo tanto, los casos interesantes son:
  - $VIS'; SO'; VIS'$ el cual es inmediato por IH.
  - $VIS'; SO'; VIS_{new}$ Por regla (READ), $\sigma' \xrightarrow{e \rhd (v \leftarrow\!\shortmid\, x)_{s,r},} \sigma$ y sabemos que por IH that la regla de escritura ($e' \rhd u'$) cumple *write-consistency* para WFR(6.12), es decir, :

    $$\forall u'' \in \sigma''.u'.s \in u''.S \implies u'.R \subseteq u''.R$$

    Por lo tanto, $r \in u'.R \subseteq u''.R$.
    Consecuentemente, $e'' \in E_r$ que implica

    $$(e'', e) \in (E_r \times \{e\}) = VIS_{new} \subseteq VIS$$

□

Lema 38. *Sea $\mathcal{A} = \langle \mathcal{E}, OP, SO, VIS, AR \rangle$ una ejecución abstracta que cumple RYW, MR, MW y WFR, entonces $(SO \cup VIS)^n \subseteq (VIS; SO) \cup SO \cup VIS$*

Demostración. La prueba se muestra por inducción en *n*:

- **n=1**. Trivial, ya que $SO \subseteq SO$ y $VIS \subseteq VIS$.
- **n=k+1**. Por definición de composición tenemos $(SO \cup VIS)^n = (SO \cup VIS)^k; (SO \cup VIS)$ Entonces,

  $$(SO \cup VIS)^k; (SO \cup VIS) = ((SO \cup VIS)^k; SO) \cup ((SO \cup VIS)^k; VIS)$$

  y por IH, sabemos que $((SO \cup VIS)^k; SO) \cup ((SO \cup VIS)^k; VIS) \subseteq$

- $((\texttt{VIS};\texttt{SO}) \cup \texttt{SO} \cup \texttt{VIS});\texttt{SO}$ y esto es:
  - $\texttt{VIS};\texttt{SO};\texttt{SO} = \texttt{VIS};\texttt{SO} \subseteq \texttt{VIS}$ por Monotonic Read..
  - $\texttt{SO};\texttt{SO} \subseteq \texttt{SO}$
  - $\texttt{VIS};\texttt{SO} \subseteq \texttt{VIS}$ por Monotonic Read.
- $((\texttt{VIS};\texttt{SO}) \cup \texttt{SO} \cup \texttt{VIS});\texttt{VIS}$ y esto es:
  - $\texttt{VIS};\texttt{SO};\texttt{VIS} \subseteq \texttt{VIS}$ por Writes Follow Reads en Visibilidad.
  - $\texttt{SO};\texttt{VIS} \subseteq \texttt{VIS} = \texttt{VIS}$ por Monotonic Writes en Visibilidad. Lem. 36
  - $\texttt{VIS};\texttt{VIS} = \texttt{VIS}$ ya que Writes Follow Reads en Visibilidad implica que visibilidad es transitiva.

□

Lema 39 (Soundness CV)**.** *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A} \rangle$*, entonces* $\mathcal{A} = \langle \mathcal{E}, \texttt{OP}, \texttt{SO}, \texttt{VIS}, \texttt{AR} \rangle$ *y* $(\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$.

Demostración. Primero mostramos por inducción en n que

$$(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{SO} \cup \texttt{VIS}$$

- **n=1**. Trivial, porque $(\texttt{SO} \cup \texttt{VIS}) \subseteq (\texttt{SO} \cup \texttt{VIS})$.
- **n=k+1**. Por definición de composición, $(\texttt{SO} \cup \texttt{VIS})^n = (\texttt{SO} \cup \texttt{VIS})^k;(\texttt{SO} \cup \texttt{VIS})$. Más aún, $(\texttt{SO} \cup \texttt{VIS})^k$ puede ser particionado en dos conjuntos:

  $$(\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{U}\times\_} \text{ y } (\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{R}\times\_}$$

  Por lo tanto,

  $$(\texttt{SO} \cup \texttt{VIS})^k;(\texttt{SO} \cup \texttt{VIS}) = ((\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{U}\times\_} \cup (\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{R}\times\_});(\texttt{SO} \cup \texttt{VIS})$$

  Notar que $((\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{R}\times\_};(\texttt{SO} \cup \texttt{VIS}))\Big|_{\mathcal{U}\times\mathcal{R}} = \emptyset$. Consecuentemente,

  $$(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} = ((\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{U}\times\_};(\texttt{SO} \cup \texttt{VIS}))\Big|_{\mathcal{U}\times\mathcal{R}}$$

  Hay dos casos:

  - $(\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{U}\times\mathcal{R}}$, es decir, cuando la segunda componente de la relación es una lectura. Entonces, por IH, $(\texttt{SO} \cup \texttt{VIS})^k\big|_{\mathcal{U}\times\mathcal{R}} \subseteq (\texttt{SO} \cup \texttt{VIS})$ Por lo tanto,

    $$(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} \subseteq ((\texttt{SO} \cup \texttt{VIS});(\texttt{SO} \cup \texttt{VIS}))|_{\mathcal{U}\times\mathcal{R}}$$

Usando operación distributiva sobre conjuntos,

$$(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} \subseteq (\texttt{SO};\texttt{SO})|_{\mathcal{U}\times\mathcal{R}} \cup (\texttt{SO};\texttt{VIS})|_{\mathcal{U}\times\mathcal{R}} \cup (\texttt{VIS};\texttt{SO})|_{\mathcal{U}\times\mathcal{R}} \cup (\texttt{VIS};\texttt{VIS})|_{\mathcal{U}\times\mathcal{R}}$$

Notar que $\texttt{VIS};\texttt{VIS} = \emptyset$ ya que $(\mathsf{e},\mathsf{e}') \in \texttt{VIS}$ implica $\texttt{OP}(\mathsf{e}) \in \mathbb{U}$ y $\texttt{OP}(\mathsf{e}') \in \mathbb{R}$. Más aún $\texttt{SO};\texttt{SO} = \texttt{SO}$ ya que $\texttt{SO}_\mathsf{s}$ es un orden total, por lo tanto, es transitivo para cada sesión. Entonces,

$$(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \cup (\texttt{SO};\texttt{VIS})|_{\mathcal{U}\times\mathcal{R}} \cup (\texttt{VIS};\texttt{SO})|_{\mathcal{U}\times\mathcal{R}}$$

La prueba se sigue notando que:

- $\texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$ por *Lem.* 19.
- $(\texttt{SO};\texttt{VIS})|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$ por *Lem.* 34.
- $(\texttt{VIS};\texttt{SO})|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$ por *Lem.* 28.

- $(\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}}$, es decir, cuando la segunda componente de la relación es una escritura. Entonces,

$$\begin{aligned}(\texttt{SO} \cup \texttt{VIS})^n &= (\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}};(\texttt{SO} \cup \texttt{VIS}) \\ &= ((\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}};\texttt{SO}) \cup ((\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS})\end{aligned}$$

Más aún, por Lem. 38

$$(\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS} \subseteq ((\texttt{VIS};\texttt{SO}) \cup \texttt{SO} \cup \texttt{VIS})|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS}$$

Entonces, distribuyendo tenemos:

$$((\texttt{VIS};\texttt{SO})|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS}) \cup (\texttt{SO}|_{\mathcal{U}\times\mathcal{U}};\texttt{SO}) \cup (\texttt{VIS}|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS})$$

La prueba vale inmediatamente notando que:

- $((\texttt{VIS};\texttt{SO})|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS})\Big|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$ por *Lem.* 35.
- $(\texttt{SO}|_{\mathcal{U}\times\mathcal{U}};\texttt{SO})\Big|_{\mathcal{U}\times\mathcal{R}} = \texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$ ya que $\texttt{SO}_\mathsf{s}$ es un orden total y por *Lem.* 19.
- $(\texttt{VIS}|_{\mathcal{U}\times\mathcal{U}};\texttt{VIS})\Big|_{\mathcal{U}\times\mathcal{R}} = \emptyset$.

Y por lo tanto, $\texttt{VIS} \subseteq (\texttt{SO} \cup \texttt{VIS})$. El caso $((\texttt{SO} \cup \texttt{VIS})^k\Big|_{\mathcal{U}\times\mathcal{U}};\texttt{SO})\Big|_{\mathcal{U}\times\mathcal{R}}$ vale inmediatamente luego de notar que $\texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}$.

Notar que $(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{SO} \cup \texttt{VIS}$ para todo n implica $(\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{SO} \cup \texttt{VIS}$. Más aún,

$$\begin{array}{lcll}(\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{R}}\Big|_{\mathcal{U}\times\mathcal{R}} & \subseteq & (\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} & \text{por } |_{\mathcal{U}\times\mathcal{R}} \text{ en ambos lados} \\ & = & \texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \cup \texttt{VIS}|_{\mathcal{U}\times\mathcal{R}} & \text{distribuyendo} \\ & \subseteq & \texttt{VIS} \cup \texttt{VIS}|_{\mathcal{U}\times\mathcal{R}} & \text{por } \textit{Lem.}\ 19 \\ & = & \texttt{VIS} & \end{array}$$

□ □

Lema 40 (Completeness CV)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *una ejecución abstracta tal que* $(\mathtt{SO} \cup \mathtt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$*. Entonces,*

$$\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle \textit{ tal que } \langle \varepsilon, \emptyset \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A}' \rangle$$
$$\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}},\ (\mathtt{SO} \cup \mathtt{VIS}')^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}' \textit{ y } \mathtt{AR} \subseteq \mathtt{AR}'$$

Demostración. Probamos el resultado usando inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. Se mantiene trivialmente tomando $\sigma = \varepsilon$ ya que $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$.
- **n=k**. Sea e un elemento maximal en $\mathtt{AR}$, es decir,, $\nexists \mathsf{e}'.(\mathsf{e}, \mathsf{e}') \in \mathtt{AR}$. Por lo tanto, $\mathcal{E}' = \mathcal{E} \setminus \{\mathsf{e}\}$ y $\mathcal{A}|_{\mathcal{E}'} = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'} \rangle$. Es sencillo mostrar que $\mathcal{A}'$ es una ejecución abstracta tal que $((\mathtt{SO}|_{\mathcal{E}'}) \cup (\mathtt{VIS}|_{\mathcal{E}'}))^+|_{\mathcal{U}\times\mathcal{R}} \subseteq (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}$. Dado que e es maximal, $\mathtt{AR}$ puede ser escrito como:

  $$\mathtt{AR}|_{\mathcal{E}'} \cup (\mathcal{E}^l \times \{\mathsf{e}\}) \text{ para algún } \mathcal{E}^l \subseteq \mathcal{E}'$$

  Por IH sobre $\mathcal{A}|_{\mathcal{E}'}$, existe $\sigma', \mathtt{VIS}'', \mathtt{AR}''$ tal que $\langle \varepsilon, \emptyset \rangle \Rightarrow_{RYW} \langle \sigma', \mathcal{A}'' \rangle$ y

  $$\mathcal{A}'' = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}'', \mathtt{AR}'' \rangle$$

  , y además

  $$\mathtt{VIS}'' = (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}, (\mathtt{SO}|_{\mathcal{E}'} \cup \mathtt{VIS}'')\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'' \text{ y } (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}''$$

  La prueba se realiza mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}' \rangle$ tal que la siguiente computación es posible:

  $$\langle \sigma', \mathcal{A}'' \rangle \Rightarrow_{CV} \langle \sigma, \mathcal{A}' \rangle,\ \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}},\ (\mathtt{SO} \cup \mathtt{VIS}')^+\big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}' \text{ y } \mathtt{AR} \subseteq \mathtt{AR}'$$

  Entonces, hay dos casos:

  - $\mathtt{OP}(\mathsf{e}) = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s}} = \delta$. Sea $\alpha = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s},\mathsf{r}}$ tal que r es una réplica fresca, es decir, $\nexists \mathsf{x}', \mathsf{v}', \mathsf{s}'$ tal que $\sigma' = \sigma_0 \cdot \mathsf{e} \rhd [\mathsf{x}' \hookleftarrow \mathsf{v}']_{\mathsf{s}',\mathsf{r}} \cdot\ \sigma_1$. La estrategia es construyendo la computación que (i) propaga cada escritura relacionada a e en $\mathtt{AR}$ a una réplica fresca r y luego, (ii) realiza una escritura δ sobre una réplica fresca r.

    La propagación es análoga a la propagación en el caso de la escritura de Lem. 20. Por lo tanto, revisamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma''_m, \mathcal{A}'''_m \rangle \to \langle \sigma, \mathcal{A}' \rangle$. usando (A-UPDATE). Notar que:

    - e es fresca: porque $\mathsf{e} \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2 e no aparece en σ';

- $\sigma' \xrightarrow{e \rhd \alpha,} , \sigma''$ puede ser derivado por la definición de *write-consistency* para CV (6.18), la cual establece que

  $$\sigma''_m \circledcirc \alpha \text{ iff } \forall e' \rhd u' \in \sigma''_m . (\alpha.s \in u'.S \vee \alpha.s = u'.s) \implies \alpha.R \subseteq u'.R$$

  Esto significa que aquellos elementos que mantienen la condición de que los antecedentes son arbitrados. Tomando los eventos que aseguran la condición anterior, tenemos:

  $$\{e'' \mid e'' \rhd u'' \in \sigma'' \ \wedge \ (\alpha.s \in u''.S \vee \alpha.s = u''.s)\}$$
  $$\subseteq \{\ e'' \mid e'' \rhd u'' \in \sigma'' \ \wedge \ r \in u''.R\ \} = E_r$$

Finalmente, por definición de extensión de ejecución abstracta:

$$\mathcal{A} = \mathcal{A}'''_m \overset{e \rhd \delta}{\oplus} E_r \ = \ \langle \ \mathcal{E}' \cup \{e\},$$
$$OP|_{\mathcal{E}'}[e \mapsto \delta],$$
$$SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}),$$
$$VIS|_{\mathcal{E}'},$$
$$AR''_m \cup (E_r \times \{e\})\}\rangle$$

Luego, tenemos:

- $\mathcal{E}' \cup \{e\} = \mathcal{E}$
- $OP|_{\mathcal{E}'}[e \mapsto \delta] = OP$
- $SO|_{\mathcal{E}'} \cup (SS(s)|_{\mathcal{E}'} \times \{e\}) = SO$
- Dado que $OP(e) \in \mathbb{U}$, por definición $VIS|_{\mathcal{E}'} = VIS|_{\mathcal{E}}$.
  Por lo tanto, $(VIS|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} = (VIS|_{\mathcal{E}})|_{\mathcal{U}\times\mathcal{R}} = VIS''$
- Observar que $(E_r \times \{e\}) \subseteq AR''_m$ porque hemos propagado los eventos asociados con e en AR. En conecuencia $AR|_{\mathcal{U}\times\mathcal{U}} \subseteq AR''_m = AR'$.

Notar que $(SO \cup VIS')^+|_{\mathcal{U}\times\mathcal{R}} \subseteq VIS'$ sigue inmediatamente por IH, luego de notar que $VIS' = VIS''$.

- $OP(e) = (v \leftarrow\!\shortmid\ x)_s = \delta$. Sea $\alpha = (v \leftarrow\!\shortmid\ x)_{s,r}$ tal que r es una réplica fresca. Análogamente al caso anterior la estrategia será (i) propagar cada escritura relacionada a e en AR a una réplica fresca r, y (ii) realizar una acción de lectura.

  La propagación es análoga al caso de escritura. Por lo tanto, chequeamos que existe σ y $\mathcal{A}'$ tal que $\langle \sigma''_m, \mathcal{A}'''_m \rangle \rightarrow \langle \sigma, \mathcal{A}' \rangle$ usando (A-READ). Notar que:

  - e es fresca: porque $e \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2 e no aparece en σ';
  - $\sigma''_m \xrightarrow{e \rhd \alpha,} , \sigma$: ya que la regla (READ) satisface $\sigma''_m \circledcirc \alpha$, es decir,:

    $$\forall (e'' \rhd u'') \in \sigma''_m . \ (\alpha.s = u''.s \vee \alpha.s \in u''.S) \implies \alpha.r \in u''.R$$

    Tomando los eventos que aseguran la condición anterior, tenemos:

    $$\{e'' \mid e'' \rhd u'' \in \sigma'' \ \wedge \ (\ \alpha.s = u''.s \implies r \ \in u''.R)\}$$
    $$\subseteq \{\ e'' \mid e'' \rhd u'' \in \sigma'' \ \wedge \ r \in u''.R\ \} = E_r$$

Finalmente por definición de extensión de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A} = \mathcal{A}'''_m \overset{e \rhd \delta}{\oplus} \mathtt{E_r} & = & \langle & \mathcal{E}' \cup \{\mathtt{e}\}, \\ & & & \mathtt{OP}|_{\mathcal{E}'}[\mathtt{e} \mapsto \delta], \\ & & & \mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS(s)}|_{\mathcal{E}'} \times \{\mathtt{e}\}) \\ & & & \mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E_r} \times \{\mathtt{e}\}) \\ & & & \mathtt{AR}''_m \cup (\mathtt{E_r} \times \{\mathtt{e}\})\} \rangle \end{array}$$

Entonces

- $\mathcal{E}' \cup \{\mathtt{e}\} = \mathcal{E}$
- $\mathtt{OP}|_{\mathcal{E}'}[\mathtt{e} \mapsto \delta] = \mathtt{OP}$
- $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS(s)}|_{\mathcal{E}'} \times \{\mathtt{e}\}) = \mathtt{SO}$
- $\mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E_r} \times \{\mathtt{e}\}) = \mathtt{VIS}$
- Observar que $(\mathtt{E_r} \times \{\mathtt{e}\}) \subseteq \mathtt{AR}''_m$ ya que hemos propagado los eventos con e en AR. Por lo tanto,

$$\begin{array}{rcll} \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} & = & (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}} \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \\ & \subseteq & \mathtt{AR}'' \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \text{por IH} \\ & \subseteq & \mathtt{AR}''_1 \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \text{por (PROPAGATION)} \\ & \vdots & & \\ & \subseteq & \mathtt{AR}''_m \cup (\mathcal{E}^l \times \{\mathtt{e}\}) & \\ & \subseteq & \mathtt{AR}''_m = \mathtt{AR}' & \end{array}$$

□

Lema 41. *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CA} \langle \sigma, \mathcal{A} \rangle$, *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $(\mathtt{AR};\mathtt{SO})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\Rightarrow$.

- **n=0**. Se mantiene trivialmente porque $\mathtt{SO} = \mathtt{AR} = \emptyset$ y $\sigma = \varepsilon$, entonces la propiedad es alcanzada.
- **n=k+1**. Luego,

$$\langle \varepsilon, \emptyset \rangle \Rightarrow_{CA} \langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$$

  Por iH, $(\mathtt{AR};\mathtt{SO})|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$. Procedemos por análisis de caso sobre la última regla aplicada:

  - rule (A-UPDATE). Entonces, $\mu = \mathtt{e} \rhd [\mathtt{x} \hookleftarrow \mathtt{v}]_\mathtt{s}$, $\mathtt{E_r} = \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma \ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e} \rhd [\mathtt{x} \hookleftarrow \mathtt{v}]_\mathtt{s}}{\oplus} \mathtt{E_r}$, y $\sigma' \xrightarrow{\mathtt{e} \rhd [\mathtt{x} \hookleftarrow \mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$. Por definición de extensión de ejecución abstracta:

    (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.

(ii) $\mathtt{AR} = \mathtt{AR}' \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\})$.

Por lo tanto, probaremos que $(\mathtt{AR};(\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\}))|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}$. Entonces, para todo $(\mathsf{e}'',\mathsf{e}) \in (\mathtt{AR};\mathtt{SO})|_{\mathcal{U}\times\mathcal{U}}$, $(\mathsf{e}'',\mathsf{e}) \in \mathtt{AR}$ sii existe e' tal que $(\mathsf{e}'',\mathsf{e}') \in \mathtt{AR}$ y $(\mathsf{e}',\mathsf{e}) \in \mathtt{SO}$. Analizamos dos casos:

- $\mathsf{e}''.\mathsf{s} = \mathsf{e}'.\mathsf{s}$, entonces por definición de *write-consistency* para CA(6.15):

$$\forall \mathsf{u}' \in \sigma'.(\mathsf{s} \in \mathsf{u}'.\mathtt{S} \vee \mathsf{s} = \mathsf{u}'.\mathsf{s}) \implies \{\mathsf{r}\} \subseteq \mathsf{u}'.\mathtt{R}$$

  Entonces, por Def. 6.2.3.2, sabemos que existe $\mathsf{e}'' \rhd \mathsf{u}' \in \sigma'$. En particular, $\mathsf{u}'.\mathsf{s} = \mathsf{s}$, y en consecuencia, $\mathsf{e}'' \in \mathtt{E}_\mathsf{r}$. Consequentemente, por B.1,

$$(\mathsf{e}'',\mathsf{e}) \in (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) \subseteq \mathtt{AR}$$

- $\mathsf{e}''.\mathsf{s} \neq \mathsf{e}'.\mathsf{s}$, entonces hay dos casos:
  - $\mathtt{OP}(\mathsf{e}') \in \mathbb{U}$. Entonces, por Def. 6.2.3.2, sabemos que existe $\mathsf{e}'' \rhd \mathsf{u}', \mathsf{e}' \rhd \mathsf{u} \in \sigma'$. Dado que $(\mathsf{e}'',\mathsf{e}') \in \mathtt{AR}$, existe r tal que $\mathsf{r} \in \mathsf{u}'.\mathtt{R}$ y $\mathsf{r} \in \mathsf{u}''.\mathtt{R}$. En consecuencia, $\mathsf{e}'' \in \mathtt{E}_\mathsf{r}$ y consecuentemente $(\mathsf{e}'',\mathsf{e}) \in \mathtt{AR}$.
  - $\mathtt{OP}(\mathsf{e}') \in \mathbb{R}$. Dado que $(\mathsf{e}'',\mathsf{e}) \in \mathtt{AR}'$ significa por definición de extensión de ejecución abstracta con una lectura que $(\mathsf{e}'',\mathsf{e}) \in \mathtt{VIS}'$. Además, por Def. 6.2.3.2, sabemos que existe $\mathsf{e}'' \rhd \mathsf{u} \in \sigma'$. Entonces, por (READ), sabemos que $\mathtt{OP}(\mathsf{e}').\mathsf{s} \in \mathsf{u}.\mathtt{S}$. En consecuencia, $\{\mathsf{r}\} \subseteq \mathsf{u}.\mathtt{R}$ que significa $\mathsf{e}'' \in \mathtt{E}_\mathsf{r}$ y consecuentemente $(\mathsf{e}'',\mathsf{e}) \in \mathtt{AR}$.

- (A-READ) vale inmediatamente luego de notar la restricción $|_{\mathcal{U}\times\mathcal{U}}$.
- (A-ARB) Dado que $\mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$ significa que $(\mathsf{e},\mathsf{e}'') \in \mathtt{SO}|_{\mathcal{U}\times\mathcal{U}}$ sii $(\mathsf{e},\mathsf{e}'') \in \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$. Más aún, $(\mathsf{e}',\mathsf{e}) \in \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$, es fácil chequear que $(\mathsf{e}',\mathsf{e}'') \in \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$

□

Lema 42 (Soundness CA). *Si* $\langle \varepsilon, \emptyset \rangle \Rightarrow_{CA} \langle \sigma, \mathcal{A} \rangle$, *entonces* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR} \rangle$ *y* $(\mathtt{SO} \cup \mathtt{VIS})^+|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}$.

Demostración. Probamos por inducción sobre *n* que $(\mathtt{SO} \cup \mathtt{VIS})^n|_{\mathcal{U}\times\mathcal{U}} \subseteq \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$

- **n=1**. Entonces,

$$\begin{array}{rcll} (\mathtt{SO} \cup \mathtt{VIS})|_{\mathcal{U}\times\mathcal{U}} & = & \mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \cup \mathtt{VIS}|_{\mathcal{U}\times\mathcal{U}} & \text{distribuyendo SO} \\ & = & \mathtt{SO}|_{\mathcal{U}\times\mathcal{U}} \cup \emptyset & \text{por Def.} \\ & \subseteq & \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} & \text{por Lem. 30} \end{array}$$

- **n=k+1**. Entonces,

$$((\mathtt{SO} \cup \mathtt{VIS})^k;(\mathtt{SO} \cup \mathtt{VIS}))\big|_{\mathcal{U}\times\mathcal{U}} = ((\mathtt{SO} \cup \mathtt{VIS})^k;\mathtt{SO})\big|_{\mathcal{U}\times\mathcal{U}}$$

ya que $((\texttt{SO} \cup \texttt{VIS})^k;\texttt{VIS})\big|_{\mathcal{U}\times\mathcal{U}} = \emptyset$. Usando IH,

$$\begin{array}{lcll} ((\texttt{SO} \cup \texttt{VIS})^k;\texttt{SO})\big|_{\mathcal{U}\times\mathcal{U}} & \subseteq & ((\texttt{AR} \cup \texttt{SO});\texttt{SO})|_{\mathcal{U}\times\mathcal{U}} & \text{IH} \\ & = & (\texttt{AR};\texttt{SO})|_{\mathcal{U}\times\mathcal{U}} \cup\ (\texttt{SO};\texttt{SO})|_{\mathcal{U}\times\mathcal{U}} & \text{distribuyendo} \\ & = & (\texttt{AR};\texttt{SO})|_{\mathcal{U}\times\mathcal{U}} \cup\ (\texttt{SO})|_{\mathcal{U}\times\mathcal{U}} & \text{por Lem. 41 and Lem. 30} \\ & \subseteq & \texttt{AR}|_{\mathcal{U}\times\mathcal{U}} & \end{array}$$

Dado que $(\texttt{SO} \cup \texttt{VIS})^n|_{\mathcal{U}\times\mathcal{U}} \subseteq \texttt{AR}|_{\mathcal{U}\times\mathcal{U}}$ se mantiene para cualquier n, tenemos $(\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{U}} \subseteq \texttt{AR}|_{\mathcal{U}\times\mathcal{U}}$.

□

Lema 43 (Completeness CA)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \texttt{OP}, \texttt{SO}, \texttt{VIS}, \texttt{AR}\rangle$ *una ejecución abstracta tal que* $(\texttt{SO} \cup \texttt{VIS})^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{AR}$*. Entonces,*

$$\begin{array}{l} \exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \texttt{OP}, \texttt{SO}, \texttt{VIS}', \texttt{AR}'\rangle \textit{ such that } \langle \varepsilon, \emptyset\rangle \Rightarrow_{CA} \langle \sigma, \mathcal{A}'\rangle \\ \qquad \texttt{VIS}' = \texttt{VIS}|_{\mathcal{U}\times\mathcal{R}},\ (\texttt{SO} \cup \texttt{VIS}')^+|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{AR}' \textit{ y } \texttt{AR} \subseteq \texttt{AR}' \end{array}$$

Demostración. La prueba es sencilla haciendo análisis de casos sobre la última regla aplicada. La prueba es igual a la demostración de completeness de CV. La diferencia está en la regla (A-READ), en la cual es inmediata por definición de *read-consistency* para CA(6.14). □

Lema 44 (Soundness SC)**.** *Si* $\langle \varepsilon, \emptyset\rangle \Rightarrow_{SC} \langle \sigma, \mathcal{A}\rangle$*, entonces* $\mathcal{A} = \langle \mathcal{E}, \texttt{OP}, \texttt{SO}, \texttt{VIS}, \texttt{AR}\rangle$*,* $\texttt{VIS}|_{\mathcal{U}\times\mathcal{R}} = \texttt{AR}|_{\mathcal{U}\times\mathcal{R}}$ *y* $\texttt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}|_{\mathcal{U}\times\mathcal{R}}$.

Demostración. La prueba se realiza por inducción en la longitud de la derivación $\overset{s}{\Rightarrow}$.

- **n=0**. Se cumple trivialmente porque $\mathcal{A} = \emptyset$, es decir,$\texttt{SO} = \texttt{VIS} = \texttt{AR} = \emptyset$.
- **n=k+1**. Entonces

  $$\langle \varepsilon, \emptyset\rangle \Rightarrow_{SC} \langle \sigma', \mathcal{A}'\rangle \rightarrow \langle \sigma, \mathcal{A}\rangle$$

  Por IH sobre $\langle \varepsilon, \emptyset\rangle \Rightarrow_{SC} \langle \sigma', \mathcal{A}'\rangle$, tenemos $\mathcal{A}' = \langle \mathcal{E}', \texttt{OP}', \texttt{SO}', \texttt{VIS}', \texttt{AR}'\rangle$ $\texttt{VIS}'|_{\mathcal{U}\times\mathcal{R}} = \texttt{AR}'|_{\mathcal{U}\times\mathcal{R}}$, $\texttt{SO}'|_{\mathcal{U}\times\mathcal{R}} \subseteq \texttt{VIS}'$ y $\texttt{AR}'|_{\mathcal{U}\times\mathcal{U}}$ es un orden total. Procedemos por análisis de casos sobre la última regla aplicada:

  - rule (A-UPDATE). Entonces, $\mu = \mathsf{e} \rhd [\mathtt{x} \leftarrow\!\shortmid\, \mathtt{v}]_{\mathtt{s}}$, $\mathtt{E}_{\mathtt{r}} = \{\ \mathsf{e}' \mid \mathsf{e}' \rhd \mathtt{u}' \in \sigma\ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}'\ \overset{\mathsf{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_{\mathtt{s}}}{\oplus}\ \mathtt{E}_{\mathtt{r}}$ y $\sigma'\ \xrightarrow{\mathsf{e}\rhd[\mathtt{x}\leftarrow\!\shortmid\mathtt{v}]_{\mathtt{s},\mathtt{r}},}\ \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:
    (i) $\texttt{SO} = \texttt{SO}' \cup (\texttt{SS}'(\mathtt{s}) \times \{\mathsf{e}\})$.
    (ii) $\texttt{VIS} = \texttt{VIS}'$.
    (iii) $\texttt{AR} = \texttt{AR}' \cup (\mathtt{E}_{\mathtt{r}} \times \{\mathsf{e}\})$.

Por lo tanto, tenemos que $\mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}$:

$$\begin{array}{lcll}
\mathtt{AR}|_{\mathcal{U}\times\mathcal{R}} & = & (\mathtt{AR}' \cup (\mathtt{E_r}\times\{\mathtt{e}\}))|_{\mathcal{U}\times\mathcal{R}} & \text{por (iii)} \\
& = & \mathtt{AR}'|_{\mathcal{U}\times\mathcal{R}} \cup (\mathtt{E_r}\times\{\mathtt{e}\})|_{\mathcal{U}\times\mathcal{R}} & \text{distribuyendo} \\
& = & \mathtt{AR}'|_{\mathcal{U}\times\mathcal{R}} & |_{\mathcal{U}\times\mathcal{R}} \\
& = & \mathtt{VIS}'|_{\mathcal{U}\times\mathcal{R}} & \text{IH} \\
& = & \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} & \text{por (ii)}
\end{array}$$

Además, mostramos que $\mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$

$$\begin{array}{lcll}
\mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} & = & \mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} & \text{por (i)} \\
& \subseteq & \mathtt{VIS}' & \text{por IH} \\
& = & \mathtt{VIS} & \text{por (ii)}
\end{array}$$

Finally,

$$\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \text{ es total sii } \mathtt{AR}' \cup (\mathtt{E_r}\times\{\mathtt{e}\})\big|_{\mathcal{U}\times\mathcal{U}} \text{ es total}$$

Por HI $\mathtt{AR}'|_{\mathcal{U}\times\mathcal{U}}$ es total, entonces resta probar que

$$(\mathtt{E_r}\times\{\mathtt{e}\}) \text{ es total}$$

Entonces, por la regla (UPDATE) $\sigma' \xrightarrow{\mathtt{e}\rhd[\mathtt{x}\leftarrow\mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$ sii

1. e es *fresca* y
2. $\sigma \Subset [\mathtt{x} \leftarrow\!\!\shortmid\ \mathtt{v}]^{\{\}\{\mathtt{r}\}}_{\mathtt{s},\mathtt{r}}$

Condición (2) significa

$$\begin{array}{lcl}
\forall \mathtt{e}' \rhd \mathtt{u} \in \sigma'.\{\mathtt{r}\} \subseteq \mathtt{u.R} & \text{sii} & \forall \mathtt{e}' \rhd \mathtt{u} \in \sigma'.\ \mathtt{r} \in \mathtt{u.R} \\
& \equiv & \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u} \in \sigma' \ \wedge\ \mathtt{r} \in \mathtt{u.R}\ \} \\
& = & \mathtt{E_r}
\end{array}$$

Por lo tanto, $|\mathtt{E_r}| = |\sigma'|$ y por Def. 6.2.3.1 $|\sigma'| = |\mathcal{E}'|$, luego $|\mathtt{E_r}| = |\mathcal{E}'|$. Entonces por Def. 6.2.3.2 $\mathtt{E_r} = \mathcal{E}'$. Más aún, por Def. 6.2.3.4, sabemos que $\mathtt{AR}|_{\mathtt{E_r}}$ es un orden total. En consecuencia $(\mathtt{E_r}\times\{\mathtt{e}\})$ es total.

- regla (A-READ). Luego, $\mu = \mathtt{e} \rhd [\mathtt{x} \leftarrow\!\!\shortmid\ \mathtt{v}]_{\mathtt{s}}$, $\mathtt{E_r} = \{\ \mathtt{e}' \mid \mathtt{e}' \rhd \mathtt{u}' \in \sigma \ \wedge\ \mathtt{r} \in \mathtt{u}'.\mathtt{R}\ \}$, $\mathcal{A} = \mathcal{A}' \overset{\mathtt{e}\rhd[\mathtt{x}\leftarrow\mathtt{v}]_{\mathtt{s}}}{\oplus} \mathtt{E_r}$ y $\sigma' \xrightarrow{\mathtt{e}\rhd[\mathtt{x}\leftarrow\mathtt{v}]_{\mathtt{s},\mathtt{r}},} \sigma$. Por definición de la operación $\oplus$ de ejecuciones abstractas:

  (i) $\mathtt{SO} = \mathtt{SO}' \cup (\mathtt{SS}'(\mathtt{s}) \times \{\mathtt{e}\})$.

  (ii) $\mathtt{VIS} = \mathtt{VIS}' \cup (\mathtt{E_r} \times \{\mathtt{e}\})$.

  (iii) $\mathtt{AR} = \mathtt{AR}' \cup (\mathtt{E_r} \times \{\mathtt{e}\})$.

Entonces, $\mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}} = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}$ sigue inmediatamente por IH. Por lo tanto, probaremos que $(\mathtt{SO}' \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\}))|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$.

$$\begin{array}{lll} \mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} & \subseteq & \mathtt{VIS}' \quad \text{por IH} \\ & \subseteq & \mathtt{VIS} \quad \text{por (ii)} \end{array}$$

Resta probar que

$$(\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\})\Big|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}$$

En consecuencia,

$$\begin{array}{lll} & \mathtt{SS}'(\mathsf{s})|_{\mathcal{U}} & \\ = & \{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \delta \wedge \delta.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & \text{por def } \mathtt{SS}'(\mathsf{s}) \\ = & \{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{u}.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & \text{por } |_{\mathcal{U}\times\mathcal{R}} \end{array}$$

Hay dos posibles reglas:

- Por regla (READ), $\sigma' \xrightarrow{\mathsf{e}\rhd(\mathsf{v}\leftarrow\mathsf{x})_{\mathsf{s},\mathsf{r}},} \sigma$, por lo tanto sabemos que $\sigma' \circledcirc \rhd(\mathsf{v} \leftarrow\!\shortmid\, \mathsf{x})_{\mathsf{s},\mathsf{r}}$, que por definición de *read-consistency* por SC significa:

  $$\forall(\mathsf{e}' \rhd \mathsf{u}') \in \sigma'.\ \mathsf{r} \in \mathsf{u}'.\mathtt{R}$$

  En consecuencia,

  $$\begin{array}{lll} & \mathtt{SS}'(\mathsf{s})|_{\mathcal{U}} & \\ = & \{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{u}.\mathsf{s} = \mathsf{s}\}|_{\mathcal{U}} & \\ \subseteq & (\{\, \mathsf{e}' \mid \mathsf{e}' \in \mathcal{E} \wedge \mathtt{OP}(\mathsf{e}') = \mathsf{u} \wedge \mathsf{u} \in \mathbb{U} \wedge \mathsf{r} \in \mathsf{u}.\mathtt{R}\}|_{\mathcal{U}} & \text{por } \circledcirc_{SC} \\ = & \mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\} & \text{por Def. 6.2.3.2} \\ \subseteq & \mathtt{VIS} & \end{array}$$

- Por regla (READ-EMPTY) podemos seguir de forma análoga al caso anterior, notando que se mantiene la condición $\sigma' \circledcirc \rhd(\mathsf{v} \leftarrow\!\shortmid\, \mathsf{x})_{\mathsf{s},\mathsf{r}}$.

Finalmente, mostramos que

$$\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \text{ es total sii } \mathtt{AR}' \cup (\mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\})\Big|_{\mathcal{U}\times\mathcal{U}} \text{ es total}$$

Esto vale inmediatamente por IH luego de notar que $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} = \mathtt{AR}'|_{\mathcal{U}\times\mathcal{U}}$.

- (A-ARBITRATION) Vale inmediatamente por IH luego de notar que $\mathtt{SO}' = \mathtt{SO}$, $\mathtt{VIS}' = \mathtt{VIS}$ y $\mathtt{AR}' = \mathtt{AR}$.

□

Lema 45 (Completeness SC)**.** *Sea* $\mathcal{A} = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}, \mathtt{AR}\rangle$ *una ejecución abstracta tal que* $\mathtt{VIS} = \mathtt{AR}$, $\mathtt{SO} \subseteq \mathtt{VIS}$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$ *es total. Entonces,*

$\exists \sigma, \mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR}\rangle$ *tal que* $\langle \varepsilon, \emptyset\rangle \Rightarrow_{SC} \langle \sigma, \mathcal{A}'\rangle$,
$\mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}$, $\quad \mathtt{VIS}' = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}$, $\quad \mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'$ $\quad$ *y* $\mathtt{AR}|_{\mathcal{U}\times\mathcal{U}}$ *es total.*

Demostración. Probamos el resultado usando inducción en $|\mathcal{E}| = n$, es decir,, el número de eventos.

- **n=0**. Se mantiene trivialmente tomando $\sigma = \varepsilon$ ya que $\mathtt{SO} = \mathtt{VIS} = \mathtt{AR} = \emptyset$.
- **n=k**. Sea e un elemento maximal en AR, es decir,, $\nexists \mathsf{e}'.(\mathsf{e},\mathsf{e}') \in \mathtt{AR}$. De aquí en adelante, $\mathcal{E}' = \mathcal{E} \setminus \{\mathsf{e}\}$ y $\mathcal{A}|_{\mathcal{E}'} = \langle \mathcal{E}', \mathtt{OP}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{AR}|_{\mathcal{E}'} \rangle$. Es sencillo chequear que $\mathcal{A}|_{\mathcal{E}'}$ es una ejecución abstracta tal que se mantienen las hipótesis.

  Dado que e es maximal, AR puede ser escrito como:

$$\mathtt{AR} = \mathtt{AR}|_{\mathcal{E}'} \cup (\mathcal{E}^l \times \{\mathsf{e}\}) \text{ para algún } \mathcal{E}^l \subseteq \mathcal{E}'$$

  Por IH sobre $\mathcal{A}|_{\mathcal{E}'}$, existe $\sigma', \mathtt{VIS}''$ tal que $\langle \varepsilon, \emptyset \rangle \Rightarrow_{SC} \langle \sigma', \mathcal{A}'' \rangle$ y $\mathcal{A}'' = \langle \mathcal{E}', \mathtt{VIS}|_{\mathcal{E}'}, \mathtt{SO}|_{\mathcal{E}'}, \mathtt{VIS}'', \mathtt{AR}|_{\mathcal{E}'}$ con $\mathtt{VIS}'' = (\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}}$, $(\mathtt{SO}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}''$, $\mathtt{VIS}'' = (\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}}$ y $(\mathtt{AR}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{U}}$ es un orden total.

  La prueba sigue mostrando que existe $\mathcal{A}' = \langle \mathcal{E}, \mathtt{OP}, \mathtt{SO}, \mathtt{VIS}', \mathtt{AR} \rangle$ tal que la siguiente computación es posible:

$$\langle \sigma', \mathcal{A}'' \rangle \to \langle \sigma, \mathcal{A}' \rangle, \text{ with } \mathtt{VIS}' = \mathtt{VIS}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{VIS}' = \mathtt{AR}|_{\mathcal{U}\times\mathcal{R}}, \mathtt{SO}|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}', \text{ and } \mathtt{AR}|_{\mathcal{U}\times\mathcal{U}} \text{ es un orden total}$$

  Entonces hay dos casos:

  - $\mathtt{OP}(\mathsf{e}) = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s}} = \delta$. Sea $\alpha = [\mathsf{x} \hookleftarrow \mathsf{v}]_{\mathsf{s},\mathsf{r}}$, $\sigma' = [\mathsf{e}_0 \rhd \mathsf{u}_0, \ldots, \mathsf{e}_n \rhd \mathsf{u}_n]$ y r una réplica tal que $\mathsf{r} \in \mathsf{u}_n.\mathtt{R}$. Más aún, sabemos por IH que $[\mathsf{u}_0, \ldots, \mathsf{u}_{n-1}] \circledcirc_{SC} \mathsf{u}_n$ se cumple (6.21), es decir, :

$$\forall j \in [0..n].\mathsf{u}_n.\mathtt{R} \subseteq \mathsf{u}_j.\mathtt{R}$$

    Ahora revisamos que $\langle \sigma', \mathcal{A}' \rangle \to \langle \sigma, \mathcal{A} \rangle$ aplicando (A-UPDATE). Notar que:

    - e es fresca: porque $\mathsf{e} \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2 e no aparece en σ';
    - $\sigma' \xrightarrow{\mathsf{e}\rhd\alpha,}, \sigma$: este puede ser derivado por la definición de *write-consistency* por SC (6.21), porque $\{\mathsf{r}\} \subseteq \mathsf{u}_n.\mathtt{R}$ y en consecuencia $\{\mathsf{r}\} \in \mathsf{u}_j.\mathtt{R}$.

    Ahora, usando la definición de extensión de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' \overset{\mathsf{e}\rhd\delta}{\oplus} \mathtt{E}_{\mathsf{r}} & = & \langle & \mathcal{E}' \cup \{\mathsf{e}\}, \\ & & & \mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta], \\ & & & \mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\}) \\ & & & \mathtt{VIS}|_{\mathcal{E}'}, \\ & & & \mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_{\mathsf{r}} \times \{\mathsf{e}\}) \rangle \end{array}$$

    En consecuencia

- $\mathcal{E}' \cup \{\mathsf{e}\} = \mathcal{E}$
- $\mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta] = \mathtt{OP}$
- $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}'(\mathsf{s}) \times \{\mathsf{e}\}) = \mathtt{SO}$
- Dado que $\mathtt{OP}(\mathsf{e}) \in \mathbb{U}$, por definición $\mathtt{VIS}|_{\mathcal{E}'} = \mathtt{VIS}|_{\mathcal{E}}$. En consecuencia, $(\mathtt{VIS}|_{\mathcal{E}'})|_{\mathcal{U}\times\mathcal{R}} = (\mathtt{VIS}|_{\mathcal{E}})|_{\mathcal{U}\times\mathcal{R}} = \mathtt{VIS}''$
- $\mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) = \mathtt{AR}$. Análogamente a la regla (A-UPDATE) para la prueba de soundness de SC, notando que $\mathtt{E}_\mathsf{r} = \mathcal{E}'$, por lo tanto, podemos chequear que $\mathtt{VIS}'|_{\mathcal{U}\times\mathcal{R}} = \mathtt{AR}'|_{\mathcal{U}\times\mathcal{R}}$, $\mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'|_{\mathcal{U}\times\mathcal{R}}$ y finalmente $\mathtt{AR}'|_{\mathcal{U}\times\mathcal{U}}$ es total.

- $\mathtt{OP}(\mathsf{e}) = (\mathsf{v} \hookleftarrow \mathsf{x})_\mathsf{s} = \delta$. Entonces, sea $\alpha = (\mathsf{v} \hookleftarrow \mathsf{x})_{\mathsf{s},\mathsf{r}}$ tal que r es elegido como el caso anterior. Ahora chequeamos que $\langle \sigma', \mathcal{A}' \rangle \rightarrow \langle \sigma, \mathcal{A} \rangle$ aplicando (A-READ). Notar que:
  - e es fresca: porque $\mathsf{e} \notin \mathcal{E}'$ y por lo tanto por Def. 6.2.3.2 e no aparece en σ';
  - $\sigma' \xrightarrow{\mathsf{e} \rhd \alpha,} , \sigma$: puede ser derivado ya que la definición de *read-consistency* para SC (6.20), porque $\{\mathsf{r}\} \subseteq \mathsf{u}_n.\mathtt{R}$ y en consecuencia $\mathsf{r} \in \mathsf{u}_j.\mathtt{R}$.

  Ahora, usando la definición de extensión de ejecución abstracta:

$$\begin{array}{rcll} \mathcal{A}' \overset{\mathsf{e} \rhd \delta}{\oplus} \mathtt{E}_\mathsf{r} & = & \langle & \mathcal{E}' \cup \{\mathsf{e}\}, \\ & & & \mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta], \\ & & & \mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'} \times \{\mathsf{e}\}) \\ & & & \mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}), \\ & & & \mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) \rangle \end{array}$$

  Por lo tanto
  - $\mathcal{E}' \cup \{\mathsf{e}\} = \mathcal{E}$
  - $\mathtt{OP}|_{\mathcal{E}'}[\mathsf{e} \mapsto \delta] = \mathtt{OP}$
  - $\mathtt{SO}|_{\mathcal{E}'} \cup (\mathtt{SS}(\mathsf{s})|_{\mathcal{E}'} \times \{\mathsf{e}\}) = \mathtt{SO}$
  - $\mathtt{VIS}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) = \mathtt{VIS}$.
  - $\mathtt{AR}|_{\mathcal{E}'} \cup (\mathtt{E}_\mathsf{r} \times \{\mathsf{e}\}) = \mathtt{AR}$.

  Análogamente a la regla (A-READ) para la prueba de soundness de SC, notando que $\mathtt{E}_\mathsf{r} = \mathcal{E}'$ podemos chequear que $\mathtt{VIS}'|_{\mathcal{U}\times\mathcal{R}} = \mathtt{AR}'|_{\mathcal{U}\times\mathcal{R}}$, $\mathtt{SO}'|_{\mathcal{U}\times\mathcal{R}} \subseteq \mathtt{VIS}'|_{\mathcal{U}\times\mathcal{R}}$ y $\mathtt{AR}'|_{\mathcal{U}\times\mathcal{U}}$ es un orden total .

□