Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Subtipado y Compatibilidad en Tipos de Sesión Multipartita

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Hernán Rojek Moriceau
L.U. 631/99

Director: Hernán Melgratti

Buenos Aires, 2017

SUBTIPADO Y COMPATIBILIDAD EN TIPOS DE SESIÓN MULTIPARTITA

La necesidad de explotar el cómputo paralelo, concurrente y distribuido motivó el desarrollo de modelos que capturan los conceptos de comunicación y comportamiento concurrente. La naturaleza de éstos es ortogonal a la del cómputo secuencial, y requiere abstracciones diferentes, pero aún así puede valerse de estrategias exitosas desarrolladas para los modelos secuenciales.
Los tipos de datos son una de las técnicas más utilizadas en la computación secuencial para dar garantías de correctitud y confiabilidad a los programas. Los tipos comportamentales permiten capturar aspectos dinámicos de un programa, principalmente relacionados con las capacidades de comunicación e interacción. Entre ellos se destacan los tipos de sesión.
Los primeros desarrollos de tipos de sesión operan a nivel de canal de comunicación, lo cual restringe la captura de propiedades globales como por ejemplo, la interdependencia de canales. Para superar estas limitaciones, se desarrollaron los Tipos de Sesión Multipartita.
Entre los avances recientes sobre esta teoría se destaca un método para garantizar correctitud, mediante la caracterización de las especificaciones como una clase denominada *compatibilidad multipartita generalizada* (GMC). Esta noción impone fuertes restricciones a las características de las especificaciones a verificar, y como consecuencia descarta implementaciones correctas.
En la presente Tesis de Licenciatura se explora la noción de subtipado aplicada a los Tipos de Sesión Multipartita, como mecanismo para flexibilizar la verificación de correctitud arriba mencionada.
En particular, proponemos dos nociones de subtipado: por eliminación de escrituras y por agregado de lecturas. Mostramos que tales nociones preservan la correctitud de la comunicación del sistema. Es decir, es posible reemplazar una componente por otra que sea de un subtipo del tipo de la original. Presentamos además un enfoque en el cual utilizamos las nociones de subtipado propuesto para probar que un sistema es correcto. Dado un sistema que no satisface la propiedad GMC, se buscan supertipos de las componentes dadas tal que el sistema que considera a los supertipos satisfaga la propiedad GMC. Si tal sistema se encuentra, entonces se puede concluir que el sistema original, si bien no satisface GMC, es seguro. Para una de las nociones de subtipado (subtipado por eliminación de escrituras), presentamos una implementación del procedimiento propuesto en Haskell. Se ilustra además la utilización de la herramienta para probar la correctitud de algunos sistemas.

Palabras clave: sistemas concurrentes, Tipos de Sesión Multipartita, subtipado, compatibilidad multipartita generalizada, autómatas finitos comunicantes.

Índice general

1. INTRODUCCIÓN

1.1. Contexto

Los tipos de sesión ("*Session Types*" en bibliografía anglosajona [12, 14, 20, 6]) son formalismos asociados a la familia de las Álgebras de Procesos, área de interés investigada desde la década de 1970 para estudiar propiedades de los sistemas concurrentes.

Inicialmente utilizados para describir sesiones binarias, los tipos de sesión son expresiones algebraicas que sirven para representar estructuras de comunicación entre procesos[13].

De manera similar a los sistemas de tipos de datos de los lenguajes de cómputo, la idea central es inferir que un programa satisface alguna buena propiedad asignando tipos. En el caso de los Tipos de Sesión, los tipos generalmente toman la forma de términos de algún cálculo de procesos, en general variantes de CCS o π-calculus[19]. En general, al asignar tipos a las distintas partes de un programa concurrente/distribuido, se requiere razonar también a nivel de éstos.

En los Tipos de Sesión, se requiere mostrar que los participantes que comparten un canal, lo usan de manera compatible. En los sistemas binarios, la compatibilidad se traduce a *dualidad*[1], donde ambos extremos de un canal deben leer y escribir de manera complementaria.

En la última década fue propuesta una forma más expresiva, con el objetivo de especificar formalmente los protocolos en los que interactúan un número arbitrario de participantes. Denominados *Tipos de Sesión Multipartita* ("*Multiparty Session Types*"), cuentan con una gran variedad de alternativas en su formulación[15]. En este trabajo nos concentraremos en tipos de sesión expresados como autómatas finitos comunicados a través de canales sincrónicos[2, 5].

La presente tesis toma como base un trabajo reciente[16] cuyo aporte principal es un algoritmo que analiza propiedades de seguridad en protocolos representados por tipos de sesión multipartita. Este desarrollo se vale de resultados previos en Teoría de Regiones para caracterizar una clase de sistemas, denominados GMC, que tienen buenas propiedades.

1.2. Motivación

En [16] se presenta un método de semidecisión para ofrecer garantías de seguridad determinando si un sistema pertenece a una clase caracterizada por una propiedad denominada *compatibilidad multipartita generalizada* (GMC). En el caso de no pertenecer a dicha clase, el método no puede decidir sobre la seguridad del sistema.

Para establecer condiciones suficientes que garanticen la correctitud del sistema, el método propuesto impone restricciones muy fuertes en el comportamiento de los sistemas. Por ejemplo, exige que todas las funcionalidades provistas por un servidor deban ser utilizadas por un cliente. En este contexto, el método no permite concluir que sistemas que usan patrones usuales de interacción son seguros. Como es usual en la teoría de Sistemas de Tipos, la noción de subtipado otorga cierta flexibilidad en el análisis de programas.

Estas estrategias motivaron la búsqueda de mecanismos que permitan al algoritmo [16] extender el dominio en el que puede decidir sobre seguridad.

En este sentido pretendemos estudiar nociones de subtipado para tipos comportamentales, inspirados en [10] y [3], para sistemas GMC. La idea central, es que si bien las nociones de subtipado no preservarán la propiedad de ser GMC, sí preservarán las propiedades de correctitud (imposibilidad de *mensajes huérfanos*, *recepciones no especificadas* o *deadlocks*) que son garantizadas por

los sistemas GMC.

En protocolos de sistemas distribuídos resulta natural modelar cada agente como una parte indivisible en la sesión de comunicación. Esta metodología sigue las líneas divisorias marcadas tanto por la localidad espacial como temporal de la ejecución de las implementaciones como así también por las divisiones del tipo políticas establecidas por procesos de desarrollo y puesta en producción. La utilidad documental de los protocolos radica en que cada agente especificado provee una interfaz a ser respetada por las implementaciones, ganando de esta manera independencia técnica sin perder interoperabilidad global. En estos contextos puede resultar útil la noción de subtipo para la validación de cada agente de manera independiente. Este es el caso, por ejemplo, de las arquitecturas cliente-servidor, la más simple de las arquitecturas distribuidas, donde se busca particionar la puesta en producción de un sistema entre clientes iniciadores y un servidor centralizador de lógica y estado. Por ejemplo, un cliente puede iniciar un subconjunto acotado de interacciones de entre las disponibles, y un servidor puede ampliar las opciones ofrecidas sin forzar la puesta en producción de nuevas versiones de clientes. Estas prácticas son habituales en el desarrollo de software, y encuentran en la noción de subtipado un fundamento formal donde basar una metodología sistemática.

1.3. Objetivos del presente trabajo

En este trabajo nos proponemos definir nociones de subtipado que preserven las propiedades de seguridad de los sistemas asincrónicos en general, para extender el dominio del algoritmo presentado en [16], o dicho de otro modo, extender las garantías de comportamiento seguro a un superconjunto de los sistemas que verifican *compatibilidad multipartita generalizada*.

En particular, buscamos relaciones de subtipado con las siguientes características:

Preservación de seguridad Naturalmente, y para cumplir con la intuición de substitutabilidad segura, deben preservar las propiedades de seguridad de los sistemas a subtipar.

Participantes como unidad mínima tipable Si queremos aplicar la substitutabilidad segura a nivel de participantes, las relaciones de subtipado deberán poder caracterizar los subtipos posibles únicamente con propiedades de éstos, es decir, para cualquier contexto arbitrario.

Generalidad La relación de subtipado debe ser inclusiva. A menores restricciones, mayor cantidad de casos posibles.

Adicionalmente, mostramos cómo utilizar las nociones de subtipado propuestas para determinar si un sistema que no es GMC es seguro porque es subtipo de un sistema GMC.

1.4. Organización

En el capítulo 2 se introducen los conceptos básicos necesarios de las teorías tratadas, para luego en el capítulo 3 definir, en términos de éstas, las nociones de subtipado desarrolladas como objetivo de este trabajo, acompañadas por los teoremas que dan cuenta de sus propiedades.

Dedicamos el capítulo 4 a la implementación de un método que se basa en una da las relaciones de subtipado aquí desarrollada para complementar el funcionamiento de un predicado de *compatibilidad multipartita generalizada*.

Y por último, cerramos el trabajo con el capítulo 5, de conclusiones y trabajo futuro.

2. CONCEPTOS PRELIMINARES

A continuación presentaremos los fundamentos de la teoría de Tipos de Sesión Multipartita, siguiendo la convención publicada en [16].

2.1. Formalismos utilizados

Notamos con $\mathcal{P}$ a un conjunto finito de participantes, los cuales suelen referenciarse individualmente con etiquetas del estilo "`r`" o "`s`" y con $\mathbb{A}$ a un alfabeto finito. Llamamos $C = \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ y } \mathtt{p} \neq \mathtt{q}\}$ al conjunto de *canales unidireccionales* entre cada par ordenado de participantes mientras que $Act = C \times \{!, ?\} \times \mathbb{A}$ es el conjunto de *acciones* posibles, referenciadas con variables como ℓ.

Definición 1 (AFC)**.** Un *autómata finito comunicante*[16], abreviado AFC, es una 4-tupla $M = (Q, q_0, \mathbb{A}, \delta)$ donde Q es un conjunto finito de estados, $q_0 \in Q$ es el estado inicial y $\delta \subseteq Q \times Act \times Q$ es el conjunto de transiciones.

Las transiciones de un AFC están etiquetadas con *acciones* del conjunto Act. La etiqueta $\mathtt{sr}!a$ representa el envío del mensaje a desde el participante `s` al participante `r` a través del canal `sr` y, dualmente, $\mathtt{sr}?a$ representa la recepción de a por `r` a través del mismo canal.

Un autómata sintácticamente válido tiene sus transiciones etiquetadas correctamente: como participante, `p` lee de canales que terminan con `p` y escribe en canales que comienzan con `p`.

Un AFC $M = (Q, q_0, \mathbb{A}, \delta)$ se dice *determinístico* si para todos los estados $q \in Q$ y todas las acciones $\ell \in \mathbb{A}$, si $(q, \ell, q'), (q, \ell, q'') \in \delta$, entonces $q' = q''$.

Un estado se dice *final* si no es el estado de partida de ninguna transición. La concatenación de las acciones de las transiciones tomadas desde el estado inicial hasta un estado final se denomina *cadena aceptada* o *reconocida*.

Se denomina $\mathcal{L}(M) \subseteq Act^*$ al lenguaje sobre Act aceptado por un autómata M.

Un AFC se dice *minimal* si no existe otro AFC M' con menos estados y transiciones tal que $\mathcal{L}(M) = \mathcal{L}(M')$. Para el presente trabajo sólo se tomarán en cuenta AFCs sintácticamente válidos, determinísticos y minimales.

Ejemplo 1. En la figura 2.1 se muestra un AFC que consiste de tres estados. Es sintácticamente válido porque sus escrituras y lectura tienen al participante `s` como emisor y receptor respectivamente. Es determinístico porque de cada estado parten transiciones que no repiten etiquetas, y es minimal porque el lenguaje reconocido no puede representarse con menos de tres transiciones.

Ejemplo 2. En la figura 2.2 se presenta un AFC similar al anterior, pero con una transición adicional que elimina el determinismo (desde el estado q_1 parten dos transiciones con idénticas etiquetas). Notar que el lenguaje reconocido por este AFC es el mismo que el reconocido por el ejemplo de la figura 2.1, por lo tanto no es minimal. No obstante, el AFC en cuestión preserva la validez sintáctica.

Definición 2 (Sistema de comunicación asincrónico)**.** Un *sistema de comunicación asincrónico*[16], o para el presente trabajo simplemente *sistema*, es una tupla $S = (M_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$ donde cada $M_\mathtt{p} = (Q_\mathtt{p}, q_{0_\mathtt{p}}, \mathbb{A}, \delta_\mathtt{p})$ es un AFC identificado con una etiqueta de un conjunto finito $\mathcal{P}$, y comparte con el resto el uso de un alfabeto común $\mathbb{A}$.

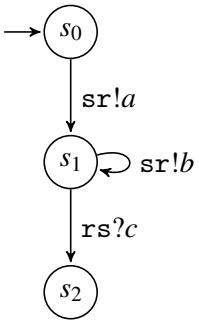

Fig. 2.1: s, un AFC sintácticamente válido, determinístico y minimal.

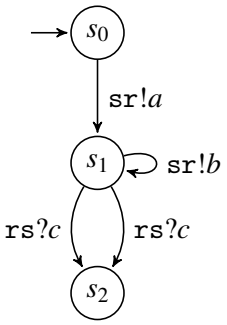

Fig. 2.2: s, un AFC sintácticamente válido, no determinístico y no minimal.

Un par $s = (\vec{q}; \vec{w})$ es una *configuración* del sistema S, donde $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ con $q_{\mathtt{p}} \in Q_{\mathtt{p}}$, los estados de cada participante en un momento dado, y $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{pq} \in C}$ con $w_{\mathtt{pq}} \in \mathbb{A}^*$, los contenidos de los canales unidireccionales.

Ejemplo 3. En la figura 2.3 se muestra un sistema de comunicación asincrónico que consiste de tres AFCs. Se muestra el contenido de los canales (colas FIFO) y se resaltan los estados correspondientes a la configuración $(\langle r_2, s_2, t_2 \rangle; \vec{w})$, donde $w_{\mathtt{sr}} = b$, $w_{\mathtt{tr}} = d$ y $w_{\mathtt{pq}} = \varepsilon$ para cualquier otro caso.

Definición 3 (Estados y configuraciones alcanzables)**.** En un sistema S una configuración $s' = (\vec{q}'; \vec{w}')$ es alcanzable[16] desde otra configuración $s = (\vec{q}; \vec{w})$ a través de una transición etiquetada con ℓ, notado $s \xrightarrow{\ell} s'$ (o $s \to s'$ si la etiqueta no es relevante), si existe $a \in \mathbb{A}$ tal que:

1. $\ell = \mathtt{sr}!a$ y $(q_{\mathtt{s}}, \ell, q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ y
   a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ para todo $\mathtt{p} \neq \mathtt{s}$, y
   b) $w'_{\mathtt{sr}} = w_{\mathtt{sr}} \cdot a$ y $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ para todo $\mathtt{pq} \neq \mathtt{sr}$; o
2. $\ell = \mathtt{sr}?a$ y $(q_{\mathtt{r}}, \ell, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ y
   a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ para todo $\mathtt{p} \neq \mathtt{r}$, y
   b) $w_{\mathtt{sr}} = a \cdot w'_{\mathtt{sr}}$ y $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ para todo $\mathtt{pq} \neq \mathtt{sr}$.

La clausura reflexiva y transitiva de $\to$ es $\to^*$. Se escribe $s_0 \xrightarrow{\ell_1 \ldots \ell_m} s_{m+1}$ cuando, para algunos $s_1, \ldots, s_m$, $s_0 \xrightarrow{\ell_1} s_1 \cdots s_m \xrightarrow{\ell_m} s_{m+1}$.

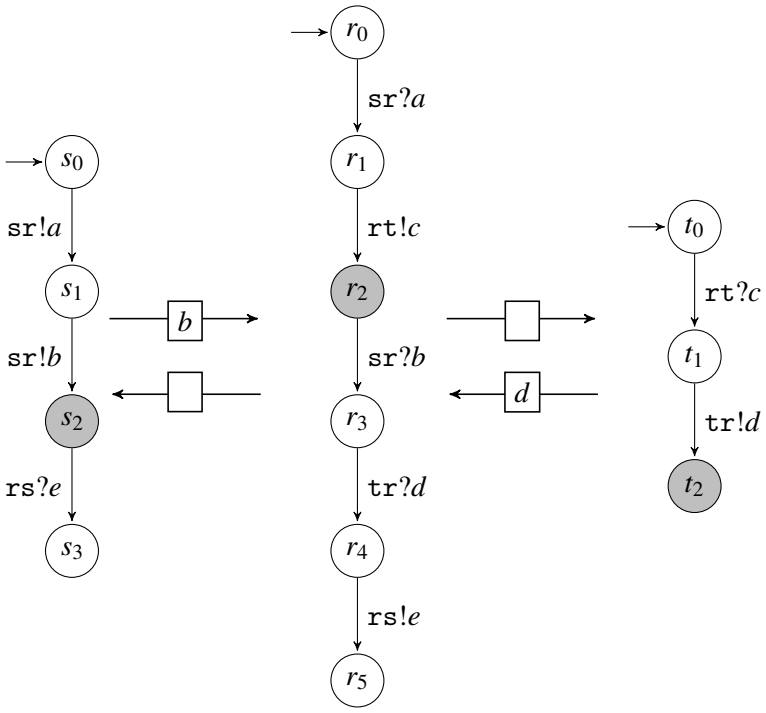

Fig. 2.3: Un ejemplo de sistema de comunicación asincrónico, con la configuración alcanzable $(\langle s_2, r_2, t_2\rangle; \langle b, \varepsilon, \varepsilon, d, \varepsilon, \varepsilon\rangle)$ representada gráficamente.

Definición 4 (Conjunto de configuraciones k-alcanzables)**.** El *conjunto de configuraciones k-alcanzables* de un sistema S es el conjunto de configuraciones que pueden ser alcanzadas con una ejecución de longitud k, es decir: $RS_k(S) = \{s \mid s_0 \xrightarrow{\ell_1 \ldots \ell_k} s\}$.

Notar que $RS_0(S) = \{s_0\}$ y que esta definición no es exactamente la misma que la utilizada en [16].

Definición 5 (Conjunto de configuraciones alcanzables)**.** El *conjunto de configuraciones alcanzables* de un sistema S se define como:

$$RS(S) = \bigcup_{k=1}^{\infty} RS_k(S) \tag{2.1}$$

Ejemplo 4. En la figura 2.4 se muestra el conjunto de configuraciones alcanzables del sistema presentado en la figura 2.3.

2.2. Propiedades de seguridad

Definición 6 (Configuración en estado de *deadlock*)**.** [16] Una configuración $s = (\vec{q}; \vec{w}) \in RS(S)$ se encuentra en estado de *deadlock* si:

$$\vec{w} = \vec{\varepsilon} \qquad \text{(Todos los canales están vacíos,}$$
$$, \exists \mathtt{r} \in \mathcal{P} \;.\; (q_{\mathtt{r}}, \mathtt{sr?}a, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}} \qquad \text{existe al menos un autómata esperando un mensaje}$$
$$\text{y } \forall \mathtt{p} \in \mathcal{P} \;.\; (q_{\mathtt{p}}, \mathtt{pu!}c, q'_{\mathtt{p}}) \notin \delta_{\mathtt{p}}. \qquad \text{y los otros autómatas están o bien en estado final o de recepción).}$$

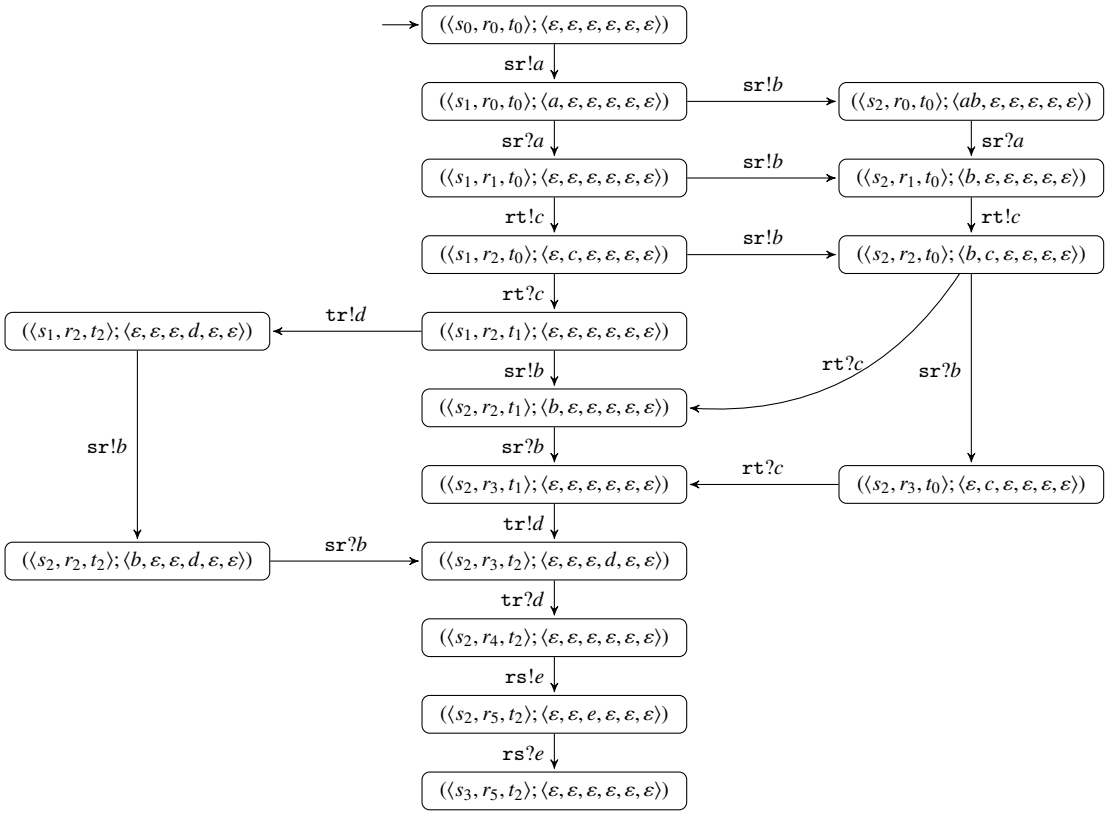

Fig. 2.4: Conjunto de configuraciones alcanzables del sistema de la figura 2.3. Las que se encuentran organizadas en la columna central, completan una *ejecución sincrónica* del sistema, ya que se leen inmediatamente los canales que se escriben.

Notar que s es un estado final en el grafo $RS(S)$.

Ejemplo 5. En la figura 2.5 se muestra un sistema que alcanzó una configuración en estado de *deadlock*. Luego de consumir el mensaje a del canal, el AFC lector espera el mensaje b, que no puede ser escrito, porque el autómata emisor alcanzó su estado final. Éste autómata no puede alcanzar un estado final. El conjunto de configuraciones alcanzables de este sistema muestra que el único camino posible es el que conduce al estado de *deadlock*.

Definición 7 (Configuración en estado de *mensaje huérfano*)**.** Una configuración $s = (\vec{q}; \vec{w}) \in RS(S)$ se encuentra en estado de *mensaje huérfano* si:

$$\forall \mathtt{p} \in \mathcal{P} \;.\; (q_{\mathtt{p}}, \ell, q'_{\mathtt{p}}) \notin \delta_{\mathtt{p}} \quad \text{(Todo autómata se encuentra en estado final}$$
$$\text{y } \vec{w} \neq \vec{\varepsilon}. \qquad \text{y hay al menos un canal no vacío).}$$

Notar que s es un estado final en el grafo $RS(S)$.

Ejemplo 6. En la figura 2.6 se puede ver un sistema que alcanzó una configuración en estado de *mensaje huérfano*. El primer AFC puede escribir dos mensajes sucesivos en el canal, el segundo de los cuales no puede ser leído por el AFC lector, indistintamente del orden global que tomen las transiciones. El sistema queda de este modo en una configuración donde ambos autómatas alcanzan estados finales, pero con un canal no vacío.

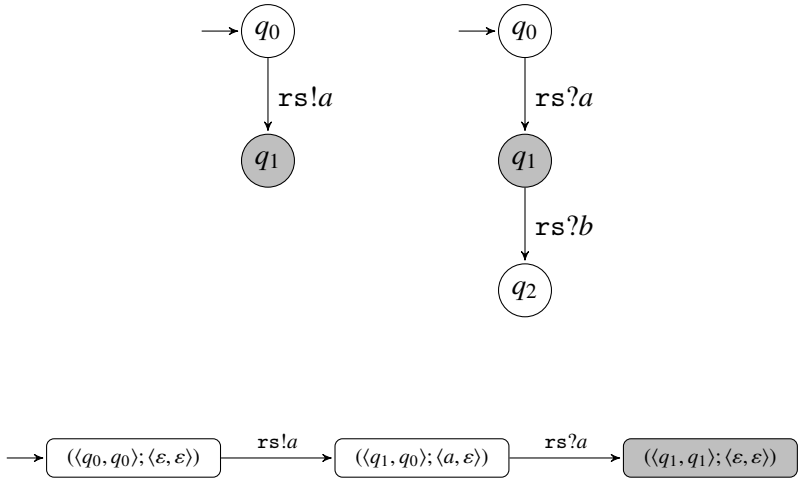

Fig. 2.5: Sistema en estado de *deadlock*, con su conjunto de configuraciones alcanzables.

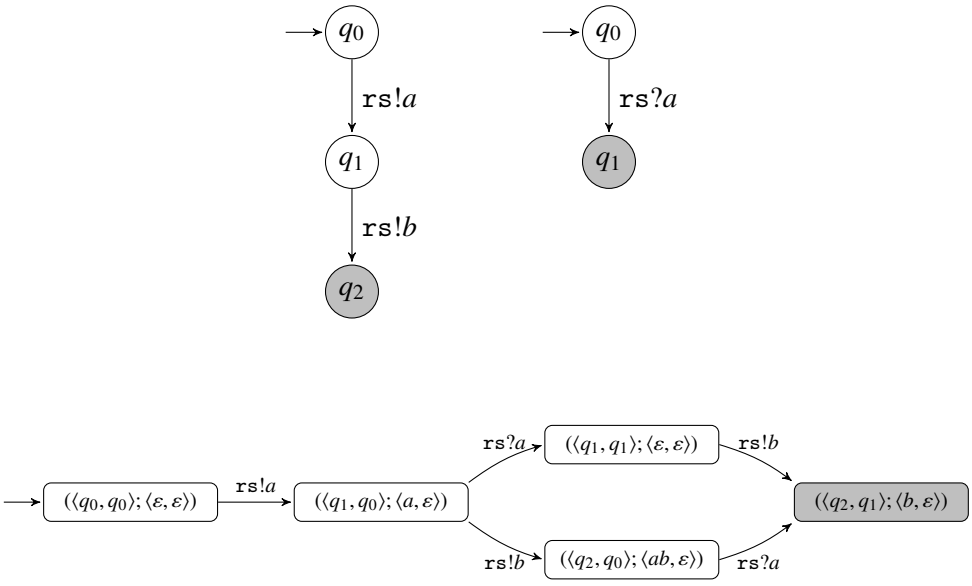

Fig. 2.6: Sistema en estado de *mensaje huérfano*, con su conjunto de configuraciones alcanzables.

Definición 8 (Configuración en estado de *recepción no especificada*)**.** [16] Una configuración $s = (\vec{q}; \vec{w}) \in RS(S)$ se encuentra en estado de *recepción no especificada* si:

$$\exists \mathtt{r} \in \mathcal{P} \,.\, \Big((\forall \mathtt{t} \in \mathcal{P} \,.\, (\forall a \in \mathbb{A} \,.\, (q_{\mathtt{r}}, \mathtt{rt}!a, q'_{\mathtt{r}}) \notin \delta_{\mathtt{r}})) \qquad \text{(Tiene algún participante sólo posibilitado para lecturas,}$$

$$\wedge (\forall b \in \mathbb{A} \,.\, (q_{\mathtt{r}}, \mathtt{sr}?b, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}} \qquad \text{de las cuales ninguna}$$

$$\implies |w_{\mathtt{sr}}| > 0 \wedge w_{\mathtt{sr}} \notin b\mathbb{A}^*)\Big). \qquad \text{se corresponde con los contenidos de su canal correspondiente.)}$$

Notar que s no necesariamente es un estado final en el grafo $RS(S)$.

Ejemplo 7. En la figura 2.7 se aprecia un sistema en el que se alcanza una configuración en estado de *recepción no especificada*. El AFC emisor escribe en el canal un mensaje que el receptor no puede leer. El autómata lector no puede alcanzar un estado final y el canal no puede ser vaciado.

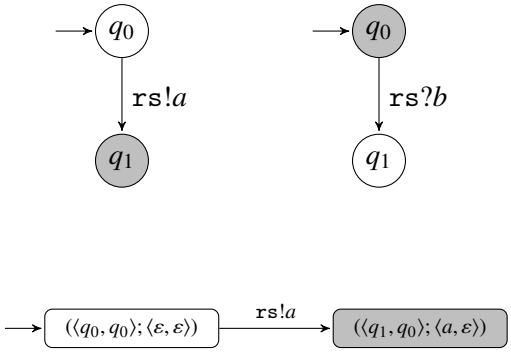

Fig. 2.7: Sistema en estado de *recepción no especificada*, con su conjunto de configuraciones alcanzables.

Ejemplo 8. En la figura 2.8 tenemos un sistema en el que también se alcanza una configuración en estado de *recepción no especificada*, pero en este caso no se trata de un estado final global. Uno de los autómatas no puede tomar ninguna de sus transiciones de lectura por no coincidir ninguna de ellas con el mensaje que encabeza el canal. No obstante, el otro participante puede seguir escribiendo hasta llegar a un estado final, también de *recepción no especificada*.

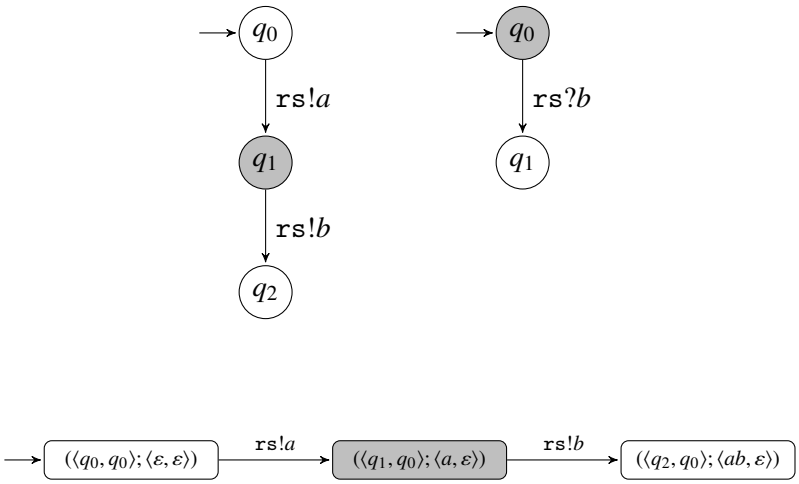

Fig. 2.8: Sistema en estado de *recepción no especificada*, con su conjunto de configuraciones alcanzables.

Definición 9 (Sistema de comunicación asincrónico seguro)**.** [16] Un *sistema de comunicación asincrónico* se dice *seguro*, si no tiene ninguna configuración alcanzable en estado de *deadlock*, de *mensaje huérfano* o de *recepción no especificada*.

2.3. Conceptos adicionales

A continuación presentamos el concepto de *sistema de transiciones sincrónico*, instrumental para la introducción de la noción de *compatibilidad multipartita generalizada*, fundamento del método de verificación analizado en este trabajo.

Intuitivamente, un sistema de transiciones sincrónico de un sistema S, o bien $TS(S)$ representa todas las posibles ejecuciones sincrónicas del sistema S, y cada transición distingue las diferentes ocurrencias de una misma comunicación, mientras que preserva el paralelismo de los AFCs participantes.

Este sistema es generado a partir de la composición de los estados locales de los AFCs del sistema S, identificada como conjunto de *nodos* N, y con un conjunto $\hat{\delta}$ de transiciones etiquetadas con eventos que representan la transmisión de mensajes a través de canales de longitud 0.

La definición formal de *TS* requiere de notación auxiliar no relevante para el presente trabajo, por lo cual prescindiremos de ella y la ilustraremos con ejemplos. La misma es presentada en [16].

Ejemplo 9. En la figura 2.9 se muestra un sistema de comunicación asincrónico que consiste de cuatro AFCs. El participante `p` elige inicialmente entre dos posibles mensajes a enviar al participante `q`. Luego cada uno de los dos envía asincrónicamente un mensaje a otro interlocutor, `r` y `s`, correspondientemente. El mensaje enviado por `p` a `r` está predeterminado por el mensaje que eligió para `q`, mientras que éste último selecciona el mensaje a enviar de manera independiente a lo que haya recibido en el estado anterior. El conjunto de configuraciones alcanzables de este sistema se presenta en la figura 2.10, mientras que en la figura 2.11 se aprecia el sistema sincrónico que representa esas interacciones. Notar que los intercambios de mensajes con los participantes `r` y `s` son representados como eventos paralelos.

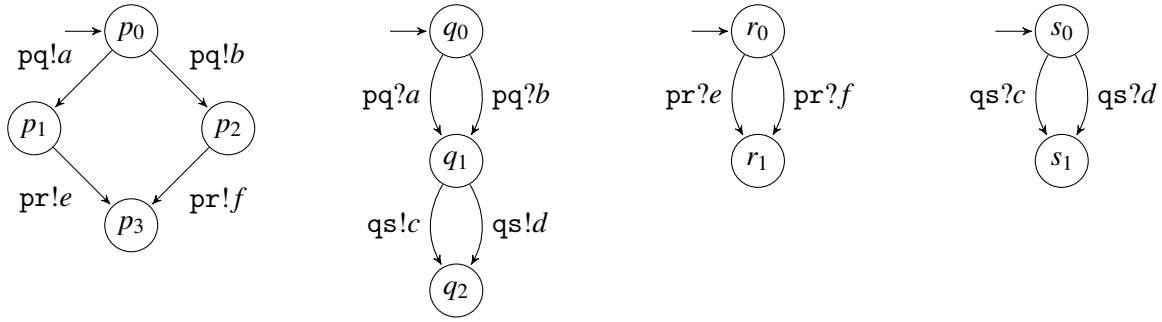

Fig. 2.9: Un ejemplo de sistema de comunicación asincrónico.

Ejemplo 10. Por otro lado, no siempre un sistema tiene ejecuciones sincrónicas. Por ejemplo, el sistema que se muestra en la figura 2.12 es un caso sencillo que exige que las escrituras y lecturas queden separadas por otras transiciones, obteniendo configuraciones con canales de longitud 1. No puede generarse un *TS* que lo describa.

Se conoce como *compatibilidad multipartita generalizada* (*Generalised multiparty compatibility* en la bibliografía) a una condición que puede cumplir un sistema S y que lo garantiza como seguro. Esta condición requiere que el sistema tenga alguna ejecución sincrónica (se puede construir $TS(S)$), y es válida cuando se verifican simultáneamente dos condiciones conocidas como *propiedad de representabilidad* y *propiedad de branching*. La primera condición se computa sobre $TS(S)$ y valida que éste contiene la suficiente información para decidir las propiedades de seguridad de cualquier ejecución asincrónica de S, las cuales no están explícitamente representadas. La *propiedad de branching* también se computa sobre $TS(S)$ y garantiza que si una elección es representada en $TS(S)$, entonces ésta está "bien formada".

Del mismo modo que el concepto anterior, la definición formal de *compatibilidad multipartita generalizada* se puede encontrar en [16].

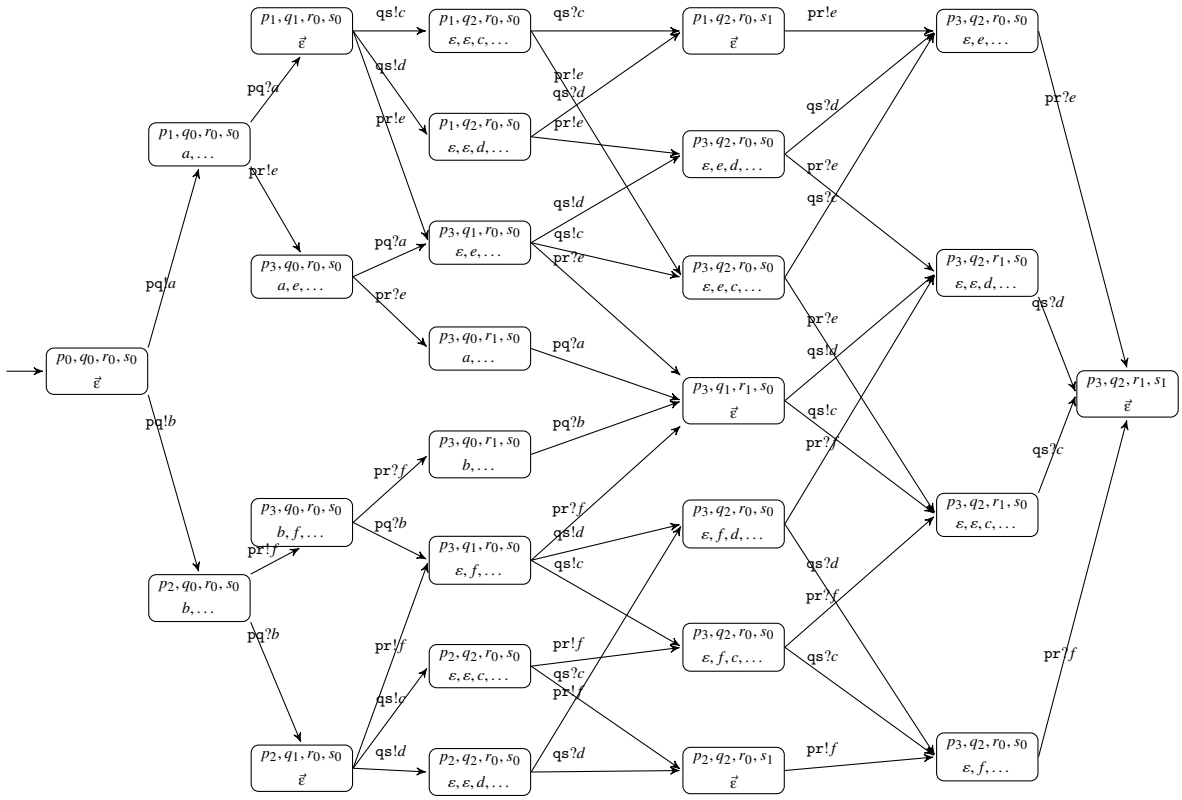

Fig. 2.10: El conjunto de configuraciones alcanzables de la figura 2.9.

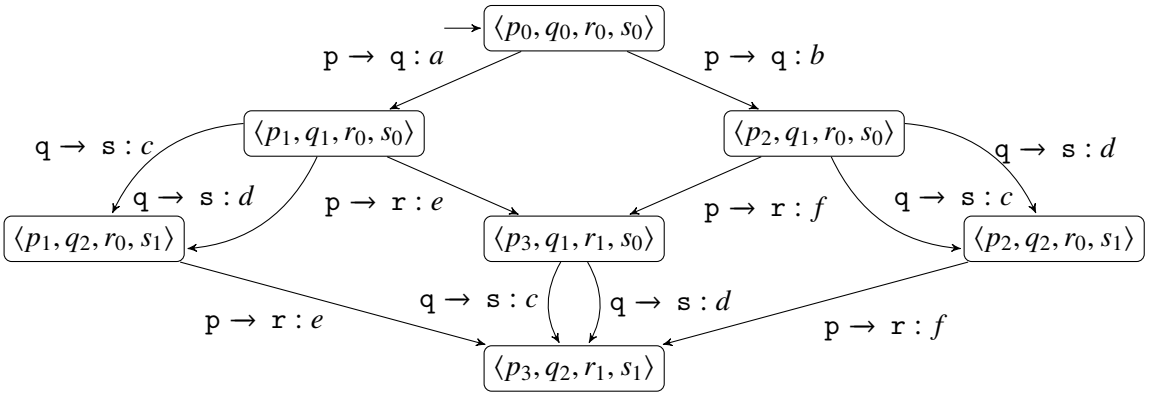

Fig. 2.11: El sistema de transiciones sincrónico del sistema de la figura 2.9.

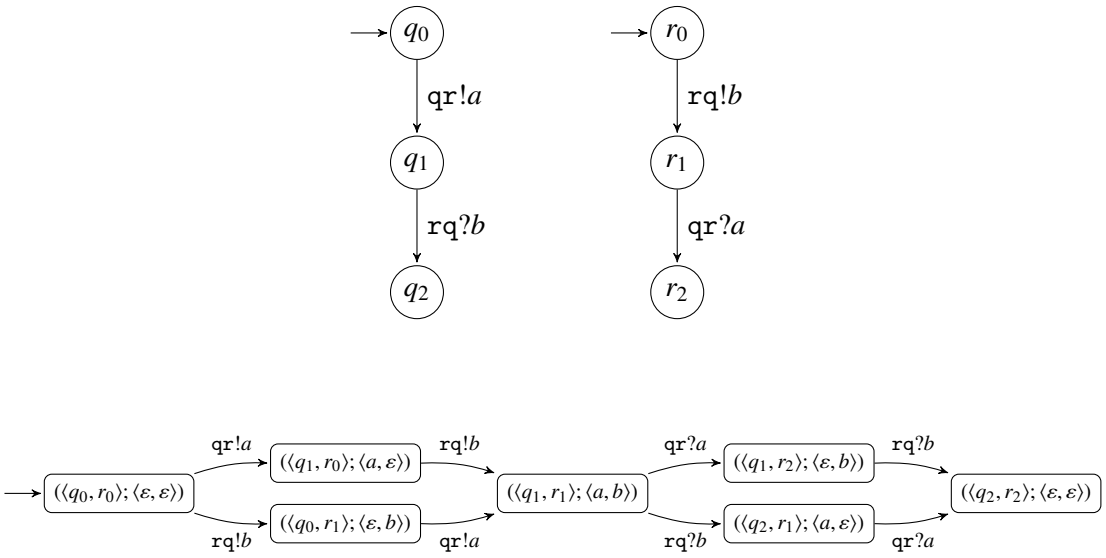

Fig. 2.12: Sistema que requiere canales de longitud 1 (no sincrónico).

3. NOCIONES DE SUBTIPADO

3.1. Definición de relaciones de subtipado

En *Teoría de Tipos*, la noción intuitiva de *subtipado* sugiere que si en un contexto arbitrario, una entidad tipada se puede sustituir por otra sin violar las garantías de seguridad del sistema de tipos, entonces el tipo de la segunda se considera un subtipo del tipo de la primera. Esta intuición se denomina *substitutabilidad segura* y se la considera fundamental para cualquier lenguaje[18].

En la familia de los tipos de sesión binaria se desarrollaron dos corrientes en las estrategias de subtipado, diferenciadas por el objeto a sustituir: o bien los canales ([9, 10]) o los procesos ([4]). La conciliación de estas dos perspectivas fue propuesta en [8].

La teoría sobre la que se basa el presente trabajo, los Tipos de Sesión Multipartita, modelan los tipos que toman las posibles sesiones entre dos o más participantes. Los elementos en esta teoría son solamente los tipos, o sea, no hay en ella nada que represente entidades a tipar. Deducir un tipo de sesión a partir de un programa o protocolo es considerado un problema ortogonal [13]. De esta manera, la teoría desarrollada hasta el momento resulta útil para cualquier especificación o implementación de la que se pueda deducir un tipo de sesión.

Los tipos de sesión multipartita se componen de especificaciones de participantes, cada uno representado con un AFC [5], los cuales a su vez son la unidad mínima a tomar en cuenta para subtipar. Luego de definir el subtipado entre participantes (entre autómatas), lo definiremos entre tipos de sesión componiendo los conceptos.

Decimos que una relación binaria $\leqslant:$ entre AFCs, escrito $M' \leqslant: M$, es una relación de *subtipado* si para todo posible sistema el cambio de un autómata comunicante por un subtipo, el sistema de comunicación resultante preserva alguna propiedad de interés presente en el original.

En el presente trabajo consideraremos relaciones de subtipado que preservan propiedades de seguridad, analizando distintas posibilidades. Para diferenciarlas, se notarán con superíndices.

3.1.1. Entre tipos de sesión multipartita

Es posible extender una relación de subtipado entre autómatas a otra entre tipos de sesión multipartita, como definimos a continuación.

Definición 10 (Relación de subtipado entre tipos de sesión multipartita)**.** Sean $S = (M_{\mathtt{p}})$ y $S' = (M'_{\mathtt{p}})$ con $\mathtt{p} \in \mathcal{P}$.

$$S' \overset{\alpha}{\leqslant:} S$$

si y sólo si

$$\forall \mathtt{p} \in \mathcal{P} \;.\; M'_{\mathtt{p}} \overset{\alpha}{\leqslant:} M_{\mathtt{p}}.$$

Al definir las relaciones de subtipado entre tipos de sesión multipartita, no contamos con un contexto arbitrario a nivel sistema contra el cual validar la preservación de propiedades. Tomando todo el sistema, el criterio de sustituibilidad tiene que validarse participante a participante, esto es, un sistema es subtipo de otro si cada uno de sus participantes puede reemplazar a su correspondiente en el sistema original. Esto permite establecer un límite entre las modificaciones seguras,

basadas en propiedades locales de cada participante y aquellas para las cuales todo un sistema debe ser analizado.

3.2. Estrategias de subtipado

Como mencionamos más arriba, el objetivo del presente trabajo es definir relaciones de subtipado entre tipos de sesión multipartita para extender las garantías de los sistemas que verifican *compatibilidad multipartita generalizada*, la cual garantiza que las propiedades de seguridad son decidibles según el algoritmo presentado en [16]. Para ello, necesitamos analizar estrategias de subtipado que vinculen sistemas manteniendo propiedades de seguridad.

Aún dentro de estos requerimientos, tuvimos en cuenta que la relación podría limitarse a garantizar propiedades de seguridad en subtipos de sistemas GMC solamente. Esta opción no fue descartada sino que preferimos mantener la relación de subtipado definida entre sistemas en general.

La primera noción de *subtipo seguro* la obtuvimos al considerar que si se limita un sistema a un subconjunto de sus ejecuciones completas, éste sistema limitado representa un subconjunto de comportamientos del original con las mismas garantías de seguridad. Aquí se entiende por *ejecución completa*, aquella ejecución perteneciente al sistema original y que no es una ejecución prefijo de otra. Si el sistema original es seguro, todas sus posibles ejecuciones lo son. Esta noción es infalible según el concepto de substitutabilidad segura, pero no resulta muy práctica a la hora de caracterizar los subtipos ya que requiere operar sobre el conjunto de configuraciones alcanzables para seleccionar de él un subgrafo que comparta raíz y algún estado final, teniendo cuidado de no crear nuevos. Esta necesidad constituye una barrera prohibitiva en sistemas grandes y asemeja el costo de obtención de un subtipo al costo de analizar pormenorizadamente el comportamiento del sistema.

A esta primera noción siguió otra más pragmática, en la que buscamos determinar el impacto de modificaciones limitadas y locales, que pudieran luego ser compuestas en la forma de modificaciones más complejas.

Si se busca acotar las ejecuciones, una manera muy simple es a través de la eliminación de transiciones. Para el caso de las escrituras, resultó evidente que se pueden forzar situaciones en las que un participante no pueda progresar sin que otro escriba alguno de los mensajes esperados, y para el caso de las lecturas, se puede forzar una recepción no especificada si además no se eliminan las escrituras correspondientes.

Para salvar estos casos, buscamos estrategias de eliminación de transiciones que se puedan determinar de acuerdo a propiedades locales de los estados involucrados o del AFC en la que tienen lugar.

Como primera distinción, diferenciamos la eliminación de transiciones por tipo: escrituras o lecturas, ya que diferentes tipos de transiciones representan distintos tipos de decisiones no-determinísticas.

La eliminación segura de lecturas se ve muy restringida: para evitar una recepción no especificada, sólo pueden eliminarse aquellas que no posean una escritura como contraparte. En consecuencia, se trata de transiciones que no son tomadas en ninguna ejecución, y para identificarlas se requiere analizar propiedades que no son locales de los autómatas o del sistema sino más bien de sus modelos semánticos, como el conjunto de configuraciones alcanzables.

Por otro lado, las escrituras son transiciones que se toman de manera independiente al contexto, por lo cual su eliminación es más tratable, y puede ser refinada partiendo de la eliminación arbitraria y condicionando los casos según propiedades locales de los AFCs afectados.

En la siguiente sección presentamos una relación de subtipado basada en eliminación de escrituras con preservación de propiedades de seguridad.

3.3. Eliminación de escrituras como subtipado

Un AFC modela dos formas de no-determinismo: uno interno como iniciador de una comunicación y otro externo determinado por las comunicaciones iniciadas por otros participantes. Eliminar escrituras equivale a acotar los mensajes posibles para iniciar comunicaciones, afectando solamente el no-determinismo interno. Al acotar las opciones de escrituras se obtiene un tipo de sesión que representa un subconjunto de los comportamientos posibles originales.

Definición 11. Sean $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta_{\mathtt{p}})$ y $M'_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta'_{\mathtt{p}})$ dos AFCs. $M'_{\mathtt{p}}$ *es subtipo de* $M_{\mathtt{p}}$ *mediante eliminación de escrituras*, escrito $M'_{\mathtt{p}} \overset{-!}{\leqslant:} M_{\mathtt{p}}$, si y sólo si

$$
\begin{array}{ll}
\Big(\delta'_{\mathtt{p}} \subseteq \delta_{\mathtt{p}} & \text{(Se eliminan transiciones,} \\
\wedge \forall (q, \mathtt{tp}?a, q') \in \delta_{\mathtt{p}} \;.\; \exists (q, \mathtt{tp}?a, q') \in \delta'_{\mathtt{p}} & \text{conservándose las lecturas} \\
\wedge (q, \mathtt{pu}!b, q') \in \delta_{\mathtt{p}} \setminus \delta'_{\mathtt{p}} \implies \exists (q, \mathtt{pr}!c, q'') \in \delta'_{\mathtt{p}} \;.\; b \neq c\Big) & \text{y el conjunto de estados que escriben)}
\end{array}
$$

Notar que para obtener un subtipo solamente se pueden eliminar escrituras que parten de estados que tienen escrituras alternativas, esto es, *sin crear estados finales en cuanto a escrituras*.

Por definición 10, se escribe $S' \overset{-!}{\leqslant:} S$ al subtipado entre sistemas por eliminación de escrituras.

Preservación de las propiedades de seguridad

Queremos demostrar que si S es un sistema seguro, y $S' \overset{-!}{\leqslant:} S$ entonces S' es un sistema seguro.

A continuación presentamos un resultado que establece que la relación de subtipado $\overset{-!}{\leqslant:}$ no introduce configuraciones alcanzables nuevas:

Lema 1 ($\overset{-!}{\leqslant:}$ configuraciones alcanzables)**.**

$$\text{Si } S' \overset{-!}{\leqslant:} S \implies RS(S') \subseteq RS(S)$$

Demostración. Por inducción sobre n, probamos que

$$RS_n(S') \subseteq RS_n(S)$$

Caso base: $(n = 0)$

Se muestra que

$$RS_0(S') = RS_0(S)$$

La igualdad se justifica en cada paso por definición 4.

Paso inductivo: Asumimos que $RS_k(S') \subseteq RS_k(S)$ y mostramos que

$$RS_{k+1}(S') \subseteq RS_{k+1}(S)$$

$RS_{k+1}(S')$

$= RS_k(S') \cup \{s \mid r \xrightarrow{\ell} \ s \wedge r \in RS_k(S')\}$ (por definición 4, de RS_k)

$\subseteq RS_k(S) \cup \{s \mid r \xrightarrow{\ell} \ s \wedge r \in RS_k(S')\}$ (por Hipótesis inductiva)

$\subseteq RS_k(S) \cup \{s \mid r \xrightarrow{\ell} \ s \wedge r \in RS_k(S)\}$ (por Hipótesis inductiva y $\forall \mathtt{p} \in \mathcal{P} \ . \ \delta'_{\mathtt{p}} \subseteq \delta_{\mathtt{p}}$)

$= RS_{k+1}(S)$ (por definición 4, de RS_k)

□

Este resultado es instrumental a las demostraciones sobre propiedades de seguridad.

Lema 2 ($\overset{-!}{\leqslant}$ no introduce configuraciones alcanzables en estado de *deadlock*)**.**

Demostración. Por reducción al absurdo.

Siendo S un sistema libre de configuraciones en estado de *deadlock*, suponemos que $RS(S')$ contiene alguna configuración en estado de *deadlock*:

$$\begin{aligned} S' \overset{-!}{\leqslant} \ & S, \exists s = (\vec{q}'; \vec{\varepsilon}) \in RS(S') \ . \\ & , \exists \mathtt{r} \in \mathcal{P} \ . \ (q_{\mathtt{r}}, \mathtt{sr?}a, q'_{\mathtt{r}}) \in \delta'_{\mathtt{r}} \\ & \text{y } \forall \mathtt{p} \in \mathcal{P} \ . \ (q_{\mathtt{p}}, \mathtt{pu!}c, q'_{\mathtt{p}}) \notin \delta'_{\mathtt{p}}. \quad (\forall \mathtt{i} \in \mathcal{P} \ . \ \delta'_{\mathtt{i}} \subseteq \delta_{\mathtt{i}}, \text{definición 11}) \end{aligned}$$

Dado que $RS(S)$ no contiene configuraciones alcanzables en estado de *deadlock*,

entonces

$$s = (\vec{q}'; \vec{\varepsilon}) \in RS(S') \subseteq RS(S) \qquad \text{(Lema 1)}$$

pero

$$\exists \mathtt{p} \in \mathcal{P} \ . \ (q_{\mathtt{p}}, \mathtt{pu!}c, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} \qquad (\textbf{Absurdo}: \overset{-!}{\leqslant} \text{ fue aplicada creando un estado final})$$

o bien

$$\neg \exists \mathtt{r} \in \mathcal{P} \ . \ (q_{\mathtt{r}}, \mathtt{sr?}a, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}. \qquad (\textbf{Absurdo}: \delta'_{\mathtt{r}} \subseteq \delta_{\mathtt{r}})$$

□

Lema 3 ($\overset{-!}{\leqslant}$ no introduce configuraciones alcanzables en estado de *mensaje huérfano*)**.**

Demostración. Por reducción al absurdo.

Siendo S un sistema libre de configuraciones en estado de *mensaje huérfano*, suponemos que existe $S' \overset{-!}{\leqslant} \ S$ tal que $RS(S')$ contiene alguna configuración en estado de *mensaje huérfano*:

$$\exists s = (\vec{q}'; \vec{w}) \in RS(S') \,.\, \Big((\forall \mathtt{p} \in \mathcal{P} \,.\, (q_\mathtt{p}, \ell, q'_\mathtt{p}) \notin \delta'_\mathtt{p}) \wedge \vec{w} \neq \vec{\varepsilon}\Big).$$

Como sabemos que

$$s \in RS(S') \subseteq RS(S) \qquad \text{(Lema 1)}$$

y asumimos que $RS(S)$ no contiene configuraciones alcanzables en estado de *mensaje huérfano*, podemos afirmar que

$$\begin{aligned}
&(\exists \mathtt{p} \in \mathcal{P} \,.\, (p_\mathtt{p}, \ell, q'_\mathtt{p}) \in \delta_\mathtt{p}) \vee \vec{w} = \vec{\varepsilon} && \text{(Negación de definición 7)}\\
\Longrightarrow &\Big(\exists \mathtt{p} \in \mathcal{P} \,.\, (\,(p_\mathtt{p}, \mathtt{rp?}a, q'_\mathtt{p}) \in \delta_\mathtt{p}) && \text{(\textbf{Absurdo}, } (p_\mathtt{p}, \mathtt{rp?}a, q'_\mathtt{p}) \in \delta_\mathtt{p} \setminus \delta'_\mathtt{p} \text{ viola definición 11)}\\
&\quad \vee ((q_\mathtt{p}, \mathtt{ps!}a, q'_\mathtt{p}) \in \delta_\mathtt{p})\Big) && \text{(\textbf{Absurdo}, } (q_\mathtt{p}, \mathtt{ps!}a, q'_\mathtt{p}) \in \delta_\mathtt{p} \setminus \delta'_\mathtt{p} \text{ viola definición 11)}\\
&\quad \vee \vec{w} = \vec{\varepsilon} && \text{(\textbf{Absurdo}, } \vec{w} = \vec{\varepsilon} \wedge \vec{w} \neq \vec{\varepsilon})
\end{aligned}$$

□

Lema 4 ($\overset{-!}{\leqslant}$ no introduce configuraciones alcanzables en estado de *recepción no especificada*)**.**

Demostración. Por reducción al absurdo.

Siendo S un sistema libre de configuraciones en estado de *recepción no especificada*, suponemos que existe $S' \overset{-!}{\leqslant} S$ tal que $RS(S')$ contiene alguna configuración en estado de *recepción no especificada*:

$$\begin{aligned}
\exists s = (\vec{q}'; \vec{w}) \in RS(S') \,.\, \exists \mathtt{r} \in \mathcal{P} \,.\, \Big(&(\forall \mathtt{t} \in \mathcal{P} \,.\, (\forall a \in \mathbb{A} \,.\, (q_\mathtt{r}, \mathtt{rt!}a, q'_\mathtt{r}) \notin \delta'_\mathtt{r}))\\
&\wedge (\forall b \in \mathbb{A} \,.\, (q_\mathtt{r}, \mathtt{sr?}b, q'_\mathtt{r}) \in \delta'_\mathtt{r}\\
&\quad \Longrightarrow |w_\mathtt{sr}| > 0 \wedge w_\mathtt{sr} \notin b\mathbb{A}^*)\Big).
\end{aligned} \tag{3.1}$$

Como sabemos que

$$s \in RS(S') \subseteq RS(S) \qquad \text{(Lema 1)}$$

y asumimos que $RS(S)$ no contiene configuraciones alcanzables en estado de *recepción no especificada*, podemos afirmar que

$$\begin{aligned}
\forall \mathtt{u} \in \mathcal{P} \,.\, &\Big(\exists \mathtt{t} \in \mathcal{P} \,.\, \Big(\exists a \in \mathbb{A} \,.\, (q_\mathtt{u}, \mathtt{ut!}a, q'_\mathtt{u}) \in \delta_\mathtt{u}\Big)\\
&\vee \exists \mathtt{s} \in \mathcal{P} \,.\, \Big(\exists b \in \mathbb{A} \,.\, (q_\mathtt{u}, \mathtt{su?}b, q'_\mathtt{u}) \in \delta_\mathtt{u} \wedge (|w_\mathtt{su}| = 0 \vee w_\mathtt{su} \in b\mathbb{A}^*)\Big)\Big) && \text{(Negación de definición 8)}
\end{aligned}$$

Tomando a $\mathtt{r} \in \mathcal{P}$ y $b \in \mathbb{A}$ existentes en 3.1,

$$\begin{aligned}
\Longrightarrow &\exists \mathtt{t} \in \mathcal{P} \,.\, \Big(\exists a \in \mathbb{A} \,.\, (q_\mathtt{r}, \mathtt{rt!}a, q'_\mathtt{r}) \in \delta_\mathtt{r} \setminus \delta'_\mathtt{r}\Big) && \text{(\textbf{Absurdo}: } \overset{-!}{\leqslant} \text{ creó un estado final)}\\
&\vee \exists \mathtt{s} \in \mathcal{P} \,.\, \Big((q_\mathtt{r}, \mathtt{sr?}b, q'_\mathtt{r}) \in \delta_\mathtt{r} \wedge (|w_\mathtt{sr}| = 0 \vee w_\mathtt{sr} \in b\mathbb{A}^*)\Big) && \text{(\textbf{Absurdo}: Niega la suposición)}
\end{aligned}$$

□

Teorema 1 (Preservación seguridad $\overset{-!}{\leqslant\!:}$)**.**

$$\text{Si } S \text{ es seguro y } S' \overset{-!}{\leqslant\!:} S \implies S' \text{ es seguro.}$$

Demostración. $S' \overset{-!}{\leqslant\!:} S$ y S es seguro, entonces $RS(S)$ no posee estados alcanzables en *deadlock*, *mensaje huérfano* ni *recepción no especificada*. Por lema 2, lema 3 y lema 4, $RS(S')$ no posee estados alcanzables en *deadlock*, *mensaje huérfano* ni *recepción no especificada*, por lo tanto, S' es seguro □

En el caso opuesto, entendemos que agregar lecturas es una forma de subtipar según la Teoría de Tipos, ya que entidades que entienden más mensajes son casos especializados de entidades que entienden menos.

A continuación presentamos una relación de subtipado basada en la incorporación de nuevas lecturas que fue refinada a partir de un criterio irrestricto de incorporación. Encontramos que las propiedades de seguridad son preservadas si imponemos una restricción basada en analizar las transiciones que parten del estado de origen del cual partiría la nueva lectura.

3.4. Incorporación de lecturas como subtipado

Un servidor que ejecuta servicios solicitados por un cliente puede incorporar un nuevo servicio que solamente sería ejecutado en caso de que el cliente también cuente con una nueva especificación. Previo a ello, es necesario que la nueva especificación de servidor sea compatible con la anterior, además de ser segura. Estas actualizaciones parciales de servicios pueden modelarse en los tipos de sesión como nuevas transiciones de lectura de un canal a partir de estados que ya leen de éste. Como se espera que las versiones del servidor sean intercambiables sin comprometer la seguridad del sistema, podemos formalizar la incorporación de lecturas como una relación de subtipado [10].

Definición 12. Sean $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta_{\mathtt{p}})$ y $M'_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta'_{\mathtt{p}})$ dos AFCs. Decimos que $M'_{\mathtt{p}}$ *es subtipo de* $M_{\mathtt{p}}$ *mediante incorporación de lecturas*, escrito $M'_{\mathtt{p}} \overset{+?}{\leqslant\!:} M_{\mathtt{p}}$, si y sólo si

$$\begin{array}{ll}
\Big(\delta_{\mathtt{p}} \subseteq \delta'_{\mathtt{p}} & \text{(Sólo se incorporan transiciones,} \\
\wedge \forall \mathtt{t} \in \mathcal{P} \;.\; (q, \mathtt{tp!}a, q') \in \delta'_{\mathtt{p}} \implies (q, \mathtt{tp!}a, q') \in \delta_{\mathtt{p}} & \text{no de escritura} \\
\wedge \forall q \in Q_{\mathtt{p}} \;.\; (q, \mathtt{pu?}b, q') \in \delta'_{\mathtt{p}} \setminus \delta_{\mathtt{p}} & \\
\quad \implies \forall (q, \ell, r) \in \delta_{\mathtt{p}} \;.\; \exists c \in \mathbb{A} \;.\; c \neq b \wedge \ell = \mathtt{pu?}c\Big) & \text{y desde estados que sólo leen del mismo canal).}
\end{array}$$

Por definición 10, se escribe $S' \overset{+?}{\leqslant\!:} S$ al subtipado entre sistemas por incorporación de lecturas.

Preservación de las propiedades de seguridad

$$\text{Si } S \text{ es un sistema seguro, y } S' \overset{+?}{\leqslant\!:} S \text{ entonces } S' \text{ es un sistema seguro.}$$

A continuación mostramos que toda incorporación de transiciones resulta en un sistema que puede o bien introducir configuraciones alcanzables nuevas o vincular dos configuraciones originales con nuevas transiciones, cualquiera sea el caso, las configuraciones originales son preservadas.

Lema 5 ($\overset{+?}{\leqslant\!:}$ configuraciones alcanzables)**.**

$$\text{Si } S' \overset{+?}{\leqslant\!:}\ S \implies RS(S') \supseteq RS(S)$$

Demostración. Por inducción sobre n, probamos que

$$RS_n(S') \supseteq RS_n(S)$$

Caso base: $(n = 0)$
Se muestra que

$$RS_0(S') \supseteq RS_0(S)$$

La igualdad se justifica en cada paso por definición 4.

Paso inductivo: Se asume que $RS_k(S') \supseteq RS_k(S)$ y se muestra que

$$RS_{k+1}(S') \supseteq RS_{k+1}(S)$$

$$\begin{aligned}
&RS_{k+1}(S') \\
&\quad = RS_k(S') \cup \{s \mid r \xrightarrow{\ell}\ s \wedge r \in RS_k(S')\} && \text{(por definición 4, de } RS_k\text{)} \\
&\quad \supseteq RS_k(S) \cup \{s \mid r \xrightarrow{\ell}\ s \wedge r \in RS_k(S')\} && \text{(por Hipótesis inductiva)} \\
&\quad \supseteq RS_k(S) \cup \{s \mid r \xrightarrow{\ell}\ s \wedge r \in RS_k(S)\} && \text{(por Hipótesis inductiva y } \forall \mathtt{p} \in \mathcal{P}\ .\ \delta'_{\mathtt{p}} \supseteq \delta_{\mathtt{p}}\text{)} \\
&\quad = RS_{n+1}(S) && \text{(por definición 4, de } RS_k\text{)}
\end{aligned}$$

□

Por motivos técnicos utilizaremos una relación de subtipado auxiliar, definida a continuación:

Definición 13. Una relación binaria $\lesssim$, escrita $M'_{\mathtt{p}} \lesssim M_{\mathtt{p}}$ es una relación de subtipado *mediante incorporación de una lectura* entre AFCs si y sólo si $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta_{\mathtt{p}})$, $M'_{\mathtt{p}} = (Q_{\mathtt{p}}, q_0, \mathbb{A}, \delta_{\mathtt{p}} \cup \{(q, \mathtt{tp}?a, q')\})$ y $M'_{\mathtt{p}} \overset{+?}{\leqslant\!:}\ M_{\mathtt{p}}$.

Extendemos la relación entre AFCs a otra entre tipos de sesión multipartita, donde solamente se afecta a un participante:

$$S' \lesssim S$$

si y sólo si

$$\exists \mathtt{p} \in \mathcal{P}\ .\ M'_{\mathtt{p}} \lesssim M_{\mathtt{p}} \wedge (\forall \mathtt{u} \in \mathcal{P}\ .\ \mathtt{u} \neq \mathtt{p} \implies M'_{\mathtt{u}} = M_{\mathtt{u}}).$$

Notar que la clausura transitiva de $\lesssim$ equivale a $\overset{+?}{\leqslant\!:}$, lo que equivale a decir que cualquier subtipo vía $\overset{+?}{\leqslant\!:}$ es también subtipo a través de varias aplicaciones de $\lesssim$.

Lema 6 (Las nuevas configuraciones sólo son alcanzables a través de lecturas incorporadas)**.** Sean S y S' dos tipos de sesión multipartita tal que $S' \lesssim S$, entonces:

Si $s' \in RS(S') \setminus RS(S)$ y $s_0 \xrightarrow{\ell_0 \dots \ell_k} s'$ entonces $\exists i \,.\, s_i \xrightarrow{\ell_i} s_{i+1} \wedge (q_{i_\mathtt{p}}, \ell_i, q'_{i_\mathtt{p}}) = (q, \mathtt{tp}?a, q')$.

Demostración. Por reducción al absurdo.

Siendo S' un sistema según se describe en el enunciado del lema, y s' una configuración alcanzable tal que $s' \in RS(S') \setminus RS(S)$, suponemos que existe una traza $\ell_0 \dots \ell_k$ tal que:

$$
\begin{aligned}
& \forall i \,.\, s_i \xrightarrow{\ell_i} s_{i+1} \wedge (q_{i_\mathtt{p}}, \ell_i, q'_{i_\mathtt{p}}) \neq (q, \mathtt{tp}?a, q') && \text{(La traza no pasa por la transición incorporada)} \\
\Longrightarrow\ & \forall i \,.\, s_i \xrightarrow{\ell_i} s_{i+1} \wedge (q_i, \ell_i, q'_i) \in \delta_p && \text{(toda transición de la traza es original)} \\
\Longrightarrow\ & s' \in RS_k(S) && \text{(por definición 4, de } RS_k\text{)} \\
\Longrightarrow\ & s' \in RS(S) && \text{(por definición 5, de } RS\text{: } \textbf{Absurdo}\text{)}
\end{aligned}
$$

□

Observación 1. Al existir una longitud de traza i, necesariamente existe una longitud mínima de traza i' que toma la nueva transición. Esto permite asegurar que la configuración alcanzable $s_{i'-1}$ pertenece $RS(S) \subseteq RS(S')$.

A continuación mostraremos que si la lectura introducida en un sistema subtipado por $\lesssim$, es tomada en una ejecución de longitud k, entonces el sistema original tiene configuraciones alcanzables en estado de *recepción no especificada*:

Proposición 1 (Subtipo vía $\lesssim$ con nuevo comportamiento)**.** Sean $M_\mathtt{u} = (Q_\mathtt{u}, q_0, \mathbb{A}, \delta_\mathtt{u})$ y $M'_\mathtt{u} = (Q_\mathtt{u}, q_0, \mathbb{A}, \delta_\mathtt{u} \cup \{(q, \mathtt{pu}?b, q')\})$, de manera tal que $M'_\mathtt{u} \lesssim M_\mathtt{u}$, y sean $S = \{M_\mathtt{t}\}_{\mathtt{t} \in \mathcal{P}}$ y $S' = \{M'_\mathtt{u}\} \cup \{M_\mathtt{t}\}_{\mathtt{t} \in \mathcal{P} \setminus \{\mathtt{u}\}}$, de manera tal que $S' \lesssim S$, entonces:

Si existe una ejecución de longitud k en $RS(S')$ que tome la transición $(q, \mathtt{pu}?b, q')$, entonces $s_0 \to^* s_{k-1} \xrightarrow{\mathtt{pu}?b} s_k$. Luego $s_{k-1} \in RS(S)$ es una configuración alcanzable en estado de *recepción no especificada*.

Demostración. Por reducción al absurdo.

Siendo S' un sistema según se describe en el enunciado del lema, y k la mínima longitud de traza tal que $s_{k-1} \xrightarrow{\mathtt{pu}?b} s_k$, asumimos que S no posee una configuración alcanzable en estado de *recepción no especificada*, entonces:

$$
\begin{aligned}
&\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S')\ . \\
&\qquad s_{k-1} \xrightarrow{\mathtt{pu?}b} s_k \\
&\qquad \wedge (q_{k-1_\mathtt{u}}, \mathtt{pu?}b, q') \in \delta'_\mathtt{u} \setminus \delta_\mathtt{u} \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S')\ . \\
&\qquad s_{k-1} \xrightarrow{\mathtt{pu?}b} s_k \\
&\qquad \wedge (q_{k-1_\mathtt{u}}, \mathtt{pu?}b, q') \in \delta'_\mathtt{u} \setminus \delta_\mathtt{u} \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X && \text{(por definición 4, de } RS_k\text{)} \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S')\ . \\
&\qquad s_{k-1} \xrightarrow{\mathtt{pu?}b} s_k \\
&\qquad \wedge \Big(\forall (q_{k-1_\mathtt{u}}, \ell, r) \in \delta_\mathtt{u}\ .\ \exists c \in \mathbb{A}\ .\ c \neq b \wedge \ell = \mathtt{pu?}c\Big) \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X && \text{(por definición 12, de } \overset{+?}{\leqslant}\text{)} \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall (q_{k-1_\mathtt{u}}, \ell, r) \in \delta_\mathtt{u}\ .\ \exists c \in \mathbb{A}\ .\ c \neq b \wedge \ell = \mathtt{pu?}c\Big) \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X && \text{(por lema 6, observación 1)} \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall \mathtt{t} \in \mathcal{P}\ .\ \forall a \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{ut!}a, v) \notin \delta_\mathtt{u}\Big) \\
&\qquad \wedge \Big(\forall (q_{k-1_\mathtt{u}}, \ell, r) \in \delta_\mathtt{u}\ .\ \exists c \in \mathbb{A}\ .\ c \neq b \wedge \ell = \mathtt{pu?}c\Big) \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X && (q_{k-1_\mathtt{u}} \text{ es sólo lectura}) \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall \mathtt{t} \in \mathcal{P}\ .\ \forall a \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{ut!}a, v) \notin \delta_\mathtt{u}\Big) \\
&\qquad \wedge \Big(\forall d \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{pu?}d, r) \in \delta_\mathtt{u} \wedge d \neq b\Big) \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X && (d \in \mathbb{A} \setminus \{b\}) \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall \mathtt{t} \in \mathcal{P}\ .\ \forall a \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{ut!}a, v) \notin \delta_\mathtt{u}\Big) \\
&\qquad \wedge \Big(\forall d \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{pu?}d, r) \in \delta_\mathtt{u} \wedge d \neq b \\
&\qquad \wedge w_{k-1\mathtt{pu}} = b \cdot X\Big) && \text{(independiente a cuantificador)} \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall \mathtt{t} \in \mathcal{P}\ .\ \forall a \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{ut!}a, v) \notin \delta_\mathtt{u}\Big) \\
&\qquad \wedge \Big(\forall d \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{pu?}d, r) \in \delta_\mathtt{u} \\
&\qquad \wedge (|w_{k-1\mathtt{pu}}| > 0 \wedge w_{k-1\mathtt{pu}} \notin d \cdot \mathbb{A}^*)\Big) && (d \neq b) \\
\Longrightarrow\ &\exists s_{k-1} = (\vec{q}_{k-1}; \vec{w}_{k-1}) \quad \in RS_k(S)\ . \\
&\qquad \Big(\forall \mathtt{t} \in \mathcal{P}\ .\ \forall a \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{ut!}a, v) \notin \delta_\mathtt{u}\Big) \\
&\qquad \wedge \Big(\forall d \in \mathbb{A}\ .\ (q_{k-1_\mathtt{u}}, \mathtt{pu?}d, r) \in \delta_\mathtt{u} \\
&\qquad\quad \Longrightarrow\ (|w_{k-1\mathtt{pu}}| > 0 \wedge w_{k-1\mathtt{pu}} \notin d \cdot \mathbb{A}^*)\Big) && ((p \wedge q) \implies (p \implies q))
\end{aligned}
$$

$s_{k-1} \in RS_k(S)$ se encuentra en estado de *recepción no especificada* (**Absurdo**).

□

El estado alcanzable s_k no necesariamente pertenece a $RS(S') \setminus RS(S)$, sino que también puede tratarse de una configuración también alcanzable por el sistema S a través de otras lecturas. En este caso, la tercera implicación de la demostración de arriba es directa, no solamente a través del lema 6.

A continuación mostraremos que si la lectura introducida en un sistema subtipado por $\lesssim$, nunca es tomada en una ejecución de longitud k, entonces este sistema y el original tienen comportamientos equivalentes:

Lema 7 (Subtipo vía $\lesssim$ sin nuevo comportamiento)**.** Sean $M_{\mathtt{u}} = (Q_{\mathtt{u}}, q_0, \mathbb{A}, \delta_{\mathtt{u}})$ y $M'_{\mathtt{u}} = (Q_{\mathtt{u}}, q_0, \mathbb{A}, \delta_{\mathtt{u}} \cup \{(q, \mathrm{pu?}b, q')\})$, de manera tal que $M'_{\mathtt{u}} \lesssim M_{\mathtt{u}}$, y sean $S = \{M_{\mathtt{t}}\}_{\mathtt{t} \in \mathcal{P}}$ y $S' = \{M'_{\mathtt{u}}\} \cup \{M_{\mathtt{t}}\}_{\mathtt{t} \in \mathcal{P} \setminus \{\mathtt{u}\}}$, tal que $S' \lesssim S$, entonces:

Si para toda ejecución de longitud k en $RS(S')$ no se toma la transición $(q, \mathrm{pu?}b, q')$, entonces ambos sistemas tienen comportamientos equivalentes, esto es, $RS(S) = RS(S')$.

Demostración. Siendo S' y S sistemas según se describen en el enunciado del lema, mostramos que:

Caso $RS(S') \subseteq RS(S)$:

Si $S' \lesssim S$ y la transición $(q, \mathrm{pu?}b, q')$ no es tomada en ninguna ejecución de longitud k en $RS(S')$, entonces por lema 6, $RS(S') \setminus RS(S) = \varnothing$ luego $RS(S') \subseteq RS(S)$.

Caso $RS(S') \supseteq RS(S)$:

Por lema 5. □

Teorema 2 (Preservación seguridad $\overset{+?}{\lesssim}$)**.**

$$\text{Si } S \text{ es seguro y } S' \overset{+?}{\lesssim} S \implies S' \text{ es seguro.}$$

Demostración. Sean $M_{\mathtt{u}} = (Q_{\mathtt{u}}, q_0, \mathbb{A}, \delta_{\mathtt{u}})$ y $M'_{\mathtt{u}} = (Q_{\mathtt{u}}, q_0, \mathbb{A}, \delta_{\mathtt{u}} \cup \{(q, \mathrm{pu?}b, q')\})$, de manera tal que $M'_{\mathtt{u}} \lesssim M_{\mathtt{u}}$, y sean $S = \{M_{\mathtt{t}}\}_{\mathtt{t} \in \mathcal{P}}$ y $S'' = \{M'_{\mathtt{u}}\} \cup \{M_{\mathtt{t}}\}_{\mathtt{t} \in \mathcal{P} \setminus \{\mathtt{u}\}}$, de manera tal que $S'' \lesssim S$, entonces:

Si S es seguro, entonces $RS(S)$ no posee ninguna configuración alcanzable en estado de *recepción no especificada*. Luego por proposición 1, la lectura $(q, \mathrm{pu?}b, q')$ no es tomada por S'' y por lema 7, $RS(S) = RS(S'')$. Como $RS(S)$ no posee configuraciones alcanzables inseguras, $RS(S'')$ tampoco, entonces S'' es un sistema seguro.

Como $\overset{+?}{\lesssim}$ es la clausura transitiva de $\lesssim$, $\overset{+?}{\lesssim}$ preserva seguridad.

□

4. IMPLEMENTACIÓN

En la siguiente sección presentamos un método que muestra la utilidad práctica de la relación de subtipado por eliminación de escrituras ($\overset{-!}{\leqslant:}$).

El algoritmo presentado en [16] garantiza la pertenencia de un sistema a la clase GMC. Esta clase está propiamente contenida en la clase de sistemas seguros.

Si un sistema no se verifica como GMC, aún puede tratarse de un subtipo de un sistema GMC. Si se puede encontrar un supertipo que verifique como GMC, entonces por teorema 1, el sistema analizado también es seguro.

En este capítulo abordamos el problema de encontrar, a partir de un sistema que no es GMC, supertipos que sean GMC. En particular, nos centraremos en generar supertipos según la relación $\overset{-!}{\leqslant:}$.

4.1. Estrategia de operación

Para encontrar un supertipo GMC según la relación $\overset{-!}{\leqslant:}$ de un sistema S_0 se deben agregar transiciones tal que una hipotética eliminación respete la definición 11. Por ello, el criterio al subtipar de *sólo eliminar escrituras sin crear estados finales en cuanto a escrituras* se invierte al supertipar como *incorporar a los supertipos candidatos las escrituras necesarias sólo en estados que no sean finales en cuanto a escrituras*.

El predicado es fácil de determinar, pero la relación de supertipado que se obtiene como resultado no tiene garantías de preservación de seguridad como sí tiene la relación de subtipado. Respetar además las condiciones necesarias para verificar GMC, partiendo de un sistema que en principio no lo es, requiere explorar exhaustivamente las posibilidades.

La estrategia elegida tiene forma de algoritmo de generación de supertipos y prueba de propiedad de GMC, esto es, genera sistemas a partir del sistema original, hasta encontrar una solución satisfactoria. La presente implementación agrega incrementalmente transiciones faltantes según la inversa de la definición 11, para ser analizados uno a uno por la herramienta de chequeo de GMC. El sistema inicial se garantiza seguro al encontrar el primer sistema generado que se verifique como GMC.

El algoritmo de generación fue implementado en el lenguaje funcional Haskell. Una de las principales características de este lenguaje es su *runtime* de evaluación *lazy*, que permite definir expresiones que reducen a secuencias potencialmente infinitas y generarlas a demanda. El código fuente puede encontrarse en [11].

La herramienta provista en [7], *chosyn* verifica si un sistema pertenece a la clase de *compatibilidad multipartita generalizada* mediante la generación de un grafo global y la verificación de las propiedades de representabilidad y *branching*. Esta herramienta fue implementada priorizando la interacción con el usuario y no el uso modular. Por este motivo, nos concentramos en la generación irrestricta de supertipos y apartamos la adaptación modular de *chosyn* como trabajo futuro.

4.2. Consideraciones generales

Antes de presentar detalladamente nuestra implementación, encontramos necesario describir algunos conceptos:

AFC interlocutor

Dada una transición del AFC `p` con mensaje `qp`?a o bien `pr`!b, los AFCs interlocutores son, respectivamente, los AFCs que modelan los participantes `q` y `r`.

Sistema sincrónico

Se trata de la estructura de datos que representa el *sistema de transiciones sincrónico* de un sistema, concepto introducido en la subsección 2.3, y generado por la función `buildTSb` provista por la herramienta *gmc-synthesis-v0.2*[7].

Coestado

Dada una transición, puede determinarse su estado de partida. Los coestados de este estado de partida son aquellos estados del AFC interlocutor que comparten estado sincrónico con éste en $TS(S)$.

El algoritmo toma como entrada un sistema S y se vale de su sistema de transiciones sincrónico para evaluar la satisfacción de las transiciones de los participantes.

La generación de cada supertipo se realiza en dos etapas bien diferenciadas: la primera consiste en el agregado de escrituras faltantes y la segunda en el agregado de subgrafos completos (que se corresponden a casos en que el subtipo elimina una transición de escritura que cancela un subgrafo completo del supertipo).

Fig. 4.1: Estrategia de operación de la generación de supertipos, donde $S_0 \overset{-!}{\leqslant:} S_k$.

El esquema general de uso puede apreciarse en la figura 4.1.

El algoritmo de generación de supertipos, representado como una caja de líneas discontinuas, toma el sistema inicial S_0 y alimenta la primera etapa.

El resultado es una secuencia de supertipos $S_0, \ldots, S_i, \ldots, S_n$ donde cada sistema tiene una escritura adicional como máximo. Estos nuevos sistemas garantizan cumplir con el predicado de supertipado, o sea, $S_0 \overset{-!}{\leqslant:} S_i$. Luego estos supertipos alimentan la segunda etapa, donde se detecta comportamiento faltante posterior a cada nueva escritura y se genera un nuevo sistema por cada transición a agregar.

El comienzo de la secuencia resultado del algoritmo de generación incluye el sistema original S_0 (supertipo trivial), los sistemas que resultan de agregar solamente una escritura faltante (como si hubiera sido eliminada con $\overset{-!}{\leqslant:}$) y refinamientos de los anteriores donde se completa una sola transición de aquellas necesarias luego de ejecutar las nuevas escrituras.

Cada sistema de esta secuencia es chequeado por GMC, y si la verificación no es satisfactoria, vuelve a ser alimentado al generador de supertipos, donde posiblemente se generen nuevos supertipos que agregan nuevas lecturas y/o comportamiento posterior.

En caso de que un sistema S_k verifique la propiedad GMC, la ejecución termina y se garantiza que, como S_k es GMC, entonces es seguro y por teorema 1, S_0 es seguro. También es posible que se generen todos los posibles supertipos sin que ninguno sea GMC. En estos casos no se puede concluir sobre la seguridad de S_0.

Este esquema de realimentación permite explorar el espacio de búsqueda a lo ancho, chequeando los supertipos que agregan menos comportamiento antes que los que agregan más.

A continuación presentamos una descripción en estilo procedural de los pasos que conforman cada una de estas dos etapas. Comenzamos con un algoritmo auxiliar que permite identificar las transiciones de lectura en cada AFC que no son satisfechas en la composición sincrónica.

Transiciones no satisfechas El siguiente algoritmo toma un sistema S y retorna el conjunto de transiciones de todos los AFCs del sistema que no se corresponden con ninguna transición del sistema sincrónico $TS(S)$. Intuitivamente, el algoritmo identifica todos los estados alcanzables en $TS(S)$ de cada uno de los AFC. Para cada uno de estos estados, computa las transiciones que parten de ese estado en el AFC correspondiente, pero que no aparecen en $TS(S)$. Cada transición se acompaña del coestado donde se detectó su no-satisfacción, con lo cual, una transición puede estar incluida más de una vez, cada una con un coestado diferente. Notar que este algoritmo detecta la falta de satisfacciones sincrónicas de transiciones, lo que no quiere decir que la transición no pueda ser satisfecha de manera asincrónica. Esas eventualidades son cubiertas en la primera etapa.

A partir de un sistema S, y su correspondiente $TS(S)$:
Inicializar $\mathsf{TP} := \emptyset$, el conjunto de transiciones posibles, $\mathsf{TP} \subseteq \delta \times Q$
Inicializar $\mathsf{TE} := \emptyset$, el conjunto de transiciones efectivas, $\mathsf{TE} \subseteq \delta \times Q$
Para cada nodo $\vec{q} = \langle q_0, \ldots, q_n \rangle \in N$, el conjunto de nodos de $TS(S)$:
  Inicializar $\mathsf{TP}_{\vec{q}} := \emptyset$, $\mathsf{TP}_{\vec{q}} \subseteq \delta \times Q$
  Inicializar $\mathsf{TE}_{\vec{q}} := \emptyset$, $\mathsf{TE}_{\vec{q}} \subseteq \delta \times Q$
  Para cada estado q_i con $0 \leq i \leq n$:
    Computar las transiciones posibles que parten de q_i
    $\mathsf{TP}_{q_i} := \left\{ \left(t, q_j\right) \mid t = (q_i, \texttt{ji}\{? \mid !\}a, q_i') \in \delta_i \right\}$
    Acumular $\mathsf{TP}_{\vec{q}} := \mathsf{TP}_{\vec{q}} \cup \mathsf{TP}_{q_i}$
  Computar las transiciones efectivas que parten de $\vec{q}$
  $\mathsf{TE}_{\vec{q}} := \left\{ \left((q_j, \texttt{ij}?a, q_j'), q_i \right) \mid \langle q_0, \ldots, q_n \rangle \xrightarrow{\texttt{i} \rightarrow \texttt{j}:a} \langle q_0', \ldots, q_n' \rangle \in \hat{\delta} \right\}$
  $\mathsf{TE}_{\vec{q}} := \left\{ \left((q_i, \texttt{ij}!a, q_i'), q_j \right) \mid \langle q_0, \ldots, q_n \rangle \xrightarrow{\texttt{i} \rightarrow \texttt{j}:a} \langle q_0', \ldots, q_n' \rangle \in \hat{\delta} \right\} \cup \mathsf{TE}_{\vec{q}}$
  Acumular $\mathsf{TP} := \mathsf{TP} \cup \mathsf{TP}_{\vec{q}}$
  Acumular $\mathsf{TE} := \mathsf{TE} \cup \mathsf{TE}_{\vec{q}}$
Calcular $\mathsf{TI} = \mathsf{TP} \setminus \mathsf{TE}$
Retornar TI.

Ejemplo 11. En la figura 4.2 Se muestra un sistema simple con su correspondiente *TS*. El mensaje a es el único que participa de una interacción sincrónica, con lo cual puede afirmarse que la recepción $\texttt{qr}?b$ no es satisfacible. Esta transición parte del estado r_0, el cual comparte nodo en $TS(S)$ con q_0, o sea, su coestado. Para este caso, el resultado del algoritmo de detección es el conjunto $\left\{ \left(r_0 \xrightarrow{\texttt{qr}?b} r_2, q_0 \right) \right\}$.

A continuación se presenta el algoritmo que describe el comportamiento de la primera etapa. Intuitivamente el algoritmo comienza identificando lecturas que no son satisfechas de manera sincrónica.

Para cada una de esas lecturas, el algoritmo genera un posible sistema candidato agregando una operación de escritura en el AFC interlocutor.

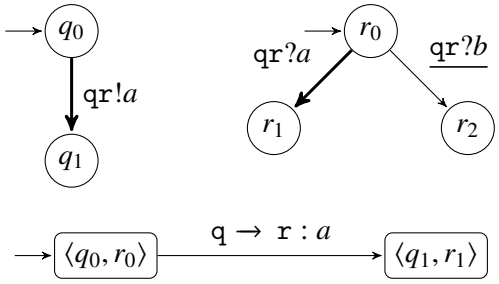

Fig. 4.2: Detección de una lectura no satisfecha.

4.3. Primera etapa

A partir del sistema S_0:
Inicializar $\mathsf{AExp} := \{S_0\}$, el conjunto de sistemas a explorar
Inicializar $\mathsf{Sol} := \emptyset$, el conjunto de soluciones
Mientras $\mathsf{AExp} \neq \emptyset$, tomar $S \in \mathsf{AExp}$

$\mathsf{AExp} := \mathsf{AExp} \setminus \{S\}$

$\mathsf{Sol} := \mathsf{Sol} \cup \{S\}$

Construir $TS(S)$

Calcular el conjunto de transiciones no satisfechas TI a partir de S y $TS(S)$, según se describe en 4.2.

$\mathsf{LI} := \{(l, q_k) \mid l = (r_i, \mathtt{qr?}a, r_i') \wedge (l, q_k) \in \mathsf{TI}\}$ (sólo las lecturas de TI)

Por cada par $(l, q_k) \in \mathsf{LI}$ tal que $l = r_i \xrightarrow{\mathtt{qr?}a} r_j$

Si $q_k \xrightarrow{\mathtt{qr!}a} q_k'' \notin \delta_{\mathtt{q}} \wedge q_k \xrightarrow{\mathtt{qr!}b} q_k' \in \delta_{\mathtt{q}} \wedge b \neq a$

Generar un nuevo sistema S' agregando un estado fresco etiquetado f y una transición $q_k \xrightarrow{\mathtt{qr!}a} f$

$\mathsf{AExp} := \mathsf{AExp} \cup \{S'\}$

Retornar Sol.

Notar que $S \overset{-!}{\leqslant}: S' \in \mathsf{Sol}$. La demostración es trivial dado que las transiciones agregadas en todos los casos satisfacen la condición de la definición 11.

Ejemplo 12. En el ejemplo 11 mostramos la detección de una lectura no satisfecha en el sistema sincrónico. En el contexto de la primera etapa del algoritmo, se verifica que desde el coestado que acompaña a la lectura, q_0, parte una escritura alternativa con mensaje distinto de b, con lo cual puede satisfacer la lectura no satisfecha. El sistema derivado con la escritura agregada se muestra en la figura 4.3.

Ejemplo 13. En la figura 4.4 se muestra un sistema un poco más complejo y la generación de sus supertipos durante la primera etapa. En este caso se ilustra la detección de las lecturas alcanzables no satisfechas (subrayadas) y la satisfacción de cada una de ellas en la forma de nuevos sistemas a analizar.

Las lecturas qr?b y qr?d son inicialmente detectadas como alcanzables no satisfechas y tratadas independientemente. En derivaciones posteriores vuelven a ser detectadas y satisfechas, lo

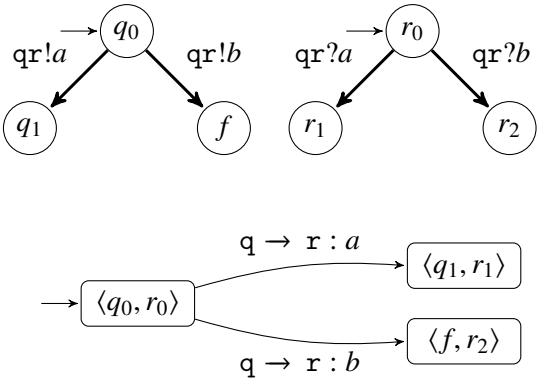

Fig. 4.3: Satisfacción de una lectura detectada.

que hace que el algoritmo de esta etapa genere sistemas isomorfos.

Puede verse además que qr?e es detectada luego de satisfacer qr?d, cuando el estado $\mathtt{r}_4$ ya es alcanzable. Esta lectura no es satisfecha en esta etapa porque ningún coestado del estado $\mathtt{r}_4$ (f_0 ó f_1, según el caso) tiene una escritura alternativa, y en consecuencia no puede agregarse una transición que satisfaga las condiciones de la definición 11.

Ejemplo 14. En la figura 4.5 se muestra la derivación del supertipo de un sistema en la primera etapa. El sistema original no es GMC por no verificar la propiedad de representabilidad, esto es, la información contenida en el *TS* no es suficiente para reconstruir el lenguaje reconocido por un participante, en este caso, r.

Este caso es interesante porque muestra la detección de una lectura no satisfecha mientras que la misma transición participa en otra ejecución sincrónica.

De esta manera, el procedimiento de detección de transiciones no satisfechas debe distinguir ambas instancias de la recepción. No puede basarse solamente en transiciones en general, porque no es lo mismo la transición $(r_1, \mathtt{qr?}c, r_2)$ mientras el interlocutor está en estado q_1 que en estado q_2.

4.4. Segunda etapa

La segunda etapa es similar a la primera, pero se detectan escrituras no leídas además de lecturas no satisfechas, se identifican sólo coestados nuevos, y se satisfacen las transiciones detectadas agregando transiciones complementarias.

La segunda etapa se aplica a cada sistema incluido en el conjunto Sol retornado por la primera etapa, aquí llamado SolE1.

A partir del conjunto de sistemas SolE1 generado por la primera etapa:
Inicializar AExp := SolE1, el conjunto de sistemas a explorar
Inicializar Sol := ∅, el conjunto de soluciones
Mientras AExp ≠ ∅, tomar $S \in$ AExp
AExp := AExp $\setminus \{S\}$
Sol := Sol $\cup \{S\}$
Construir $TS(S)$
Calcular el conjunto de transiciones no satisfechas TI a partir de S y $TS(S)$, según se describe en 4.2.
Por cada par $(t, q_k) \in$ TI tal que $t = r_i \xrightarrow{\mathtt{q}rXa} r_j$
Inicializar Y :=!
Si X =!, entonces Y :=?
Si $q_k \xrightarrow{\mathtt{q}rYa} q'_k \notin \delta_{\mathtt{q}} \wedge q_k$ es fresco
Generar un nuevo sistema S' agregando un estado fresco etiquetado f y una transición $q_k \xrightarrow{\mathtt{q}rYa} f$
AExp := AExp $\cup \{S'\}$
Retornar Sol.

Notar que el agregado de transiciones que se realiza en esta etapa no afecta la condición de los sistemas de ser supertipos del original. Estas transiciones se agregan solamente desde estados que no existen en el sistema inicial, y con ellas se busca satisfacer cualquier comunicación sincrónica requerida por el contexto del participante para que el sistema no quede en una configuración insegura.

Ejemplo 15. La segunda etapa del algoritmo es necesaria para que los participantes que tienen nuevas escrituras agregadas en la primera etapa puedan cumplir con las nuevas interacciones que se habilitan al satisfacer lecturas antes no transitadas.

Considere el sistema generado por la primera etapa que se muestra en la figura 4.6. Notar que el agregado de la transición $(q_0, \mathtt{qr}!a, f_0)$ hace que ahora el estado r_1 sea alcanzable y por lo tanto la transición $\mathtt{qr}?c$ sea insatisfecha. Notar sin embargo que la primera etapa no puede generar una transición a partir del estado nuevo f_0, dado que violaría las condiciones de subtipado de la definición 11. La segunda etapa considera estos casos, en los que el comportamiento a agregar necesariamente parte de un estado fresco que fue generado por la primera etapa.

Ejemplo 16. La satisfacción de transiciones posteriores es necesaria tanto para satisfacer lecturas como escrituras. En esta etapa no se requieren transiciones alternativas.

En el sistema ilustrado en la figura 4.7 se muestra un ejemplo en el que para satisfacer transiciones correspondientes a estados que se vuelven alcanzables en $TS(S)$, gracias a las nuevas transiciones que se van agregando a través de las etapas de generación, puede ser necesario también satisfacer escrituras. En este caso, para satisfacer la transición de escritura que parte del estado r_1, es necesario agregar en el AFC interlocutor una transición de lectura a partir del estado f_0.

Ejemplo 17. En este ejemplo mostramos que la generación puede requerir que se agreguen ramas completas en los AFCs interlocutores. En la figura 4.8 mostramos una derivación de supertipos en la que luego de aportar opciones nuevas en la primer etapa, el algoritmo busca conciliar los comportamientos replicando las interacciones de manera complementaria.

Ejemplo 18. El caso presentado en la figura 4.9 resulta útil para mostrar la utilidad de derivar un supertipo por cada transición a satisfacer. Ambas etapas trabajan de este modo, porque la incorpo-

ración de una transición puede habilitar comportamiento previamente no alcanzable. El comportamiento agregado se termina deduciendo en función de lo que el contexto espera del participante incompleto.

Ejemplo 19. Este ejemplo muestra la secuencia de casos que se generan hasta encontrar el supertipo que valida GMC del sistema que se muestra en la figura 4.10. Este sistema representa una variante defectuosa de un protocolo de selección. Un participante "selector" envía a dos interlocutores isomorfos un mensaje de inicialización i, tras el cual cada uno espera recibir un mensaje que lo declare ganador (g). El participante ganador comunica al otro que perdió (p). El participante `q` solamente puede dar por ganador al participante `s`. La lectura $(r_1, \texttt{qr}?g, r_2)$ no es tomada en ninguna ejecución posible, por lo cual el sistema no cumple con la propiedad de representabilidad de GMC.

Este caso es interesante porque muestra las características de los criterios que utiliza la primera etapa del algoritmo para detectar lecturas alcanzables no satisfechas.

La lectura $(r_1, \texttt{qr}?g, r_2)$ no es satisfecha cuando su interlocutor se encuentra en los estados q_1, q_2 y q_4. De éstos, sólo los dos primeros tienen escrituras alternativas como para servir de estado de partida de una lectura agregada; seguramente se advierta que la opción intuitivamente correcta, según la semántica descripta más arriba, consistiría en agregar la nueva escritura desde q_2. Sin embargo, el algoritmo no busca aplicar sentido común ni es su objetivo reconstruir especificaciones dañadas sino más bien encontrar algún supertipo que verifique GMC, cualquiera sea su semántica.

Adicionalmente se detectan $(s_1, \texttt{rs}?p, s_3)$ con interlocutor en r_1, $(r_1, \texttt{sr}?p, r_3)$ en s_0 y s_1, y además $(s_0, \texttt{qs}?i, s_1)$ en q_0. Las lecturas que reciben p no tienen coestados de los que partan escrituras, con lo cual no sirven para derivar un supertipo.

No se puede afirmar lo mismo de la lectura de inicialización del participante `s`, que es detectada solamente porque el selector lo inicializa en segundo lugar: la escritura que inicializa a su competidor puede interpretarse como escritura alternativa de una escritura faltante. Es en casos como estos en los que se hace evidente que el algoritmo tiene como opción válida no necesariamente satisfacer todas las lecturas que detecta.

En la figura 4.11 se muestran los distintos supertipos del AFC que modela al participante `q`. Mostramos solamente al selector porque es el único del sistema que recibe transiciones nuevas. Ilustrar el árbol de derivaciones completo de este ejemplo resultaría impráctico, pero es fácil deducir si uno deriva de otro considerando que los estados f_i se agregan en orden ascendente.

Asimismo, todas las transiciones ahí mostradas son tomadas, dado que son todas escrituras, y ninguna escritura puede ser detectada como no satisfecha en la primera etapa. Esta observación pone en evidencia que es posible encontrar una solución antes de que la segunda etapa necesite agregar nuevo comportamiento.

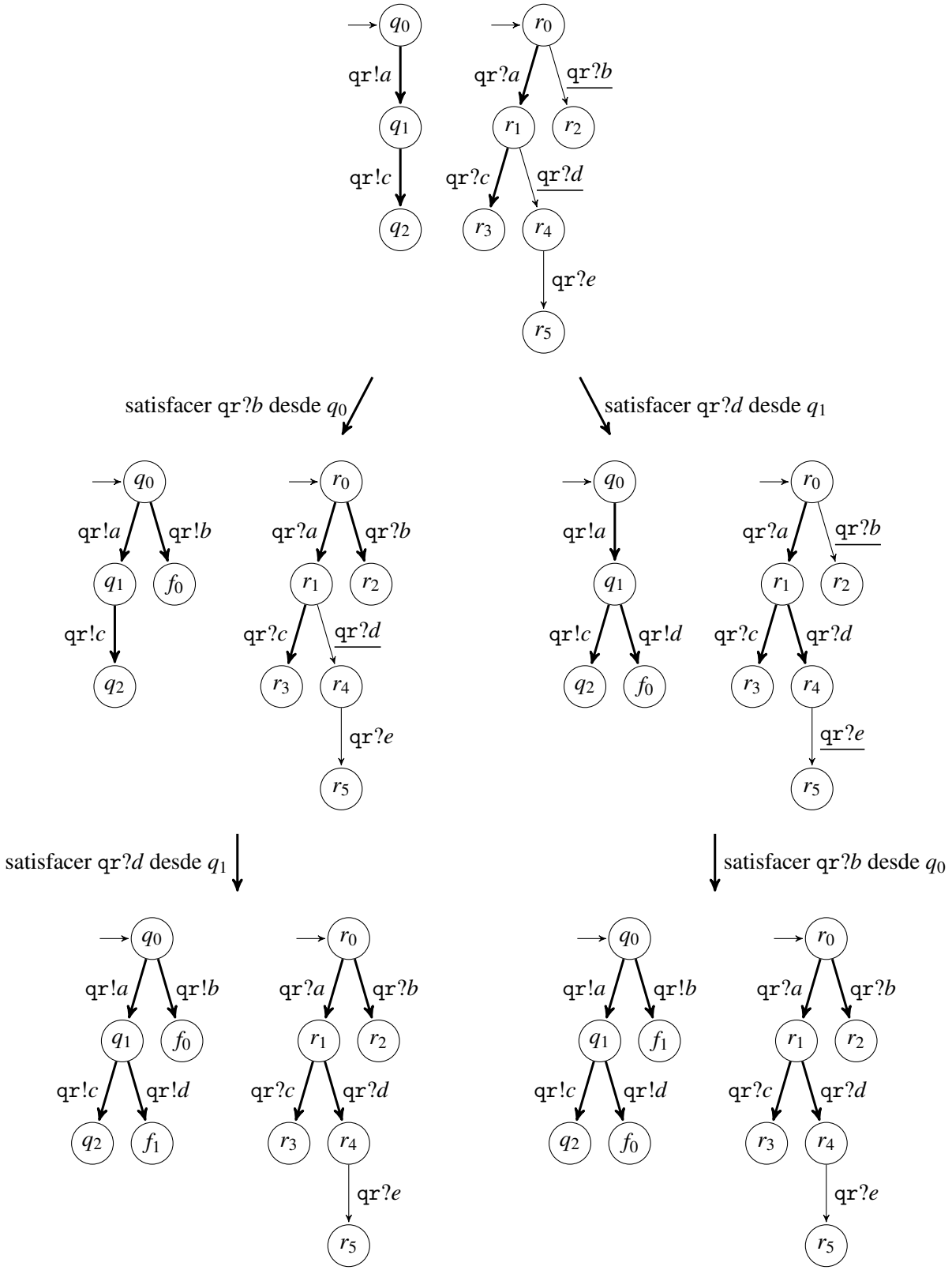

Fig. 4.4: Ejecución de la primera etapa a partir de un sistema inicial.

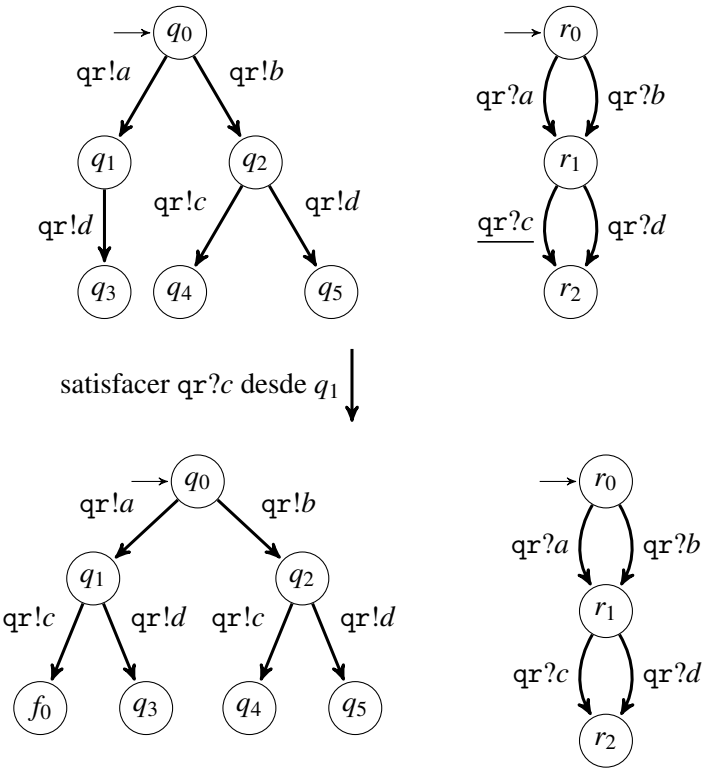

Fig. 4.5: Satisfacción de una lectura a veces satisfecha.

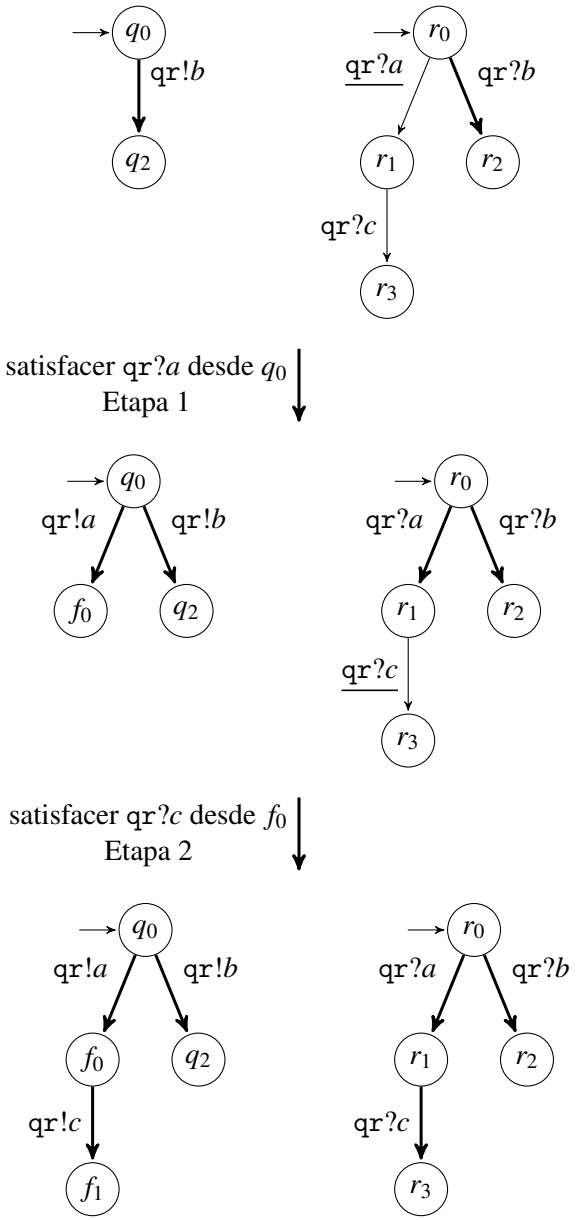

Fig. 4.6: Generación de dos supertipos a través de las dos etapas del algoritmo. La segunda etapa satisface una lectura previamente no alcanzable agregando una escritura a partir del estado f_0.

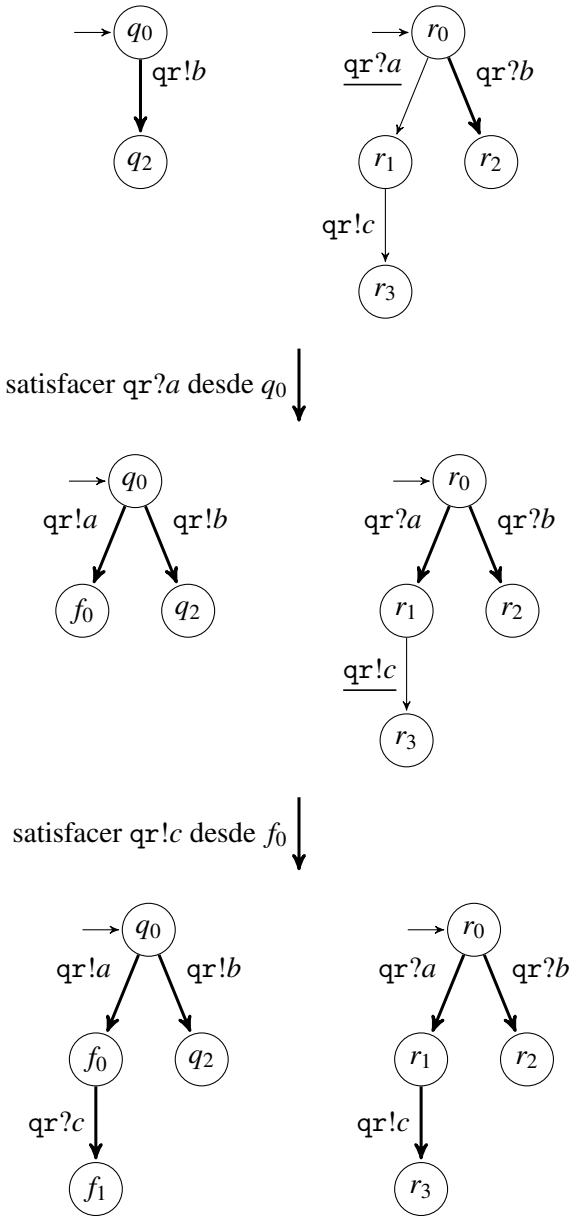

Fig. 4.7: En la segunda etapa también se satisfacen las escrituras no leídas.

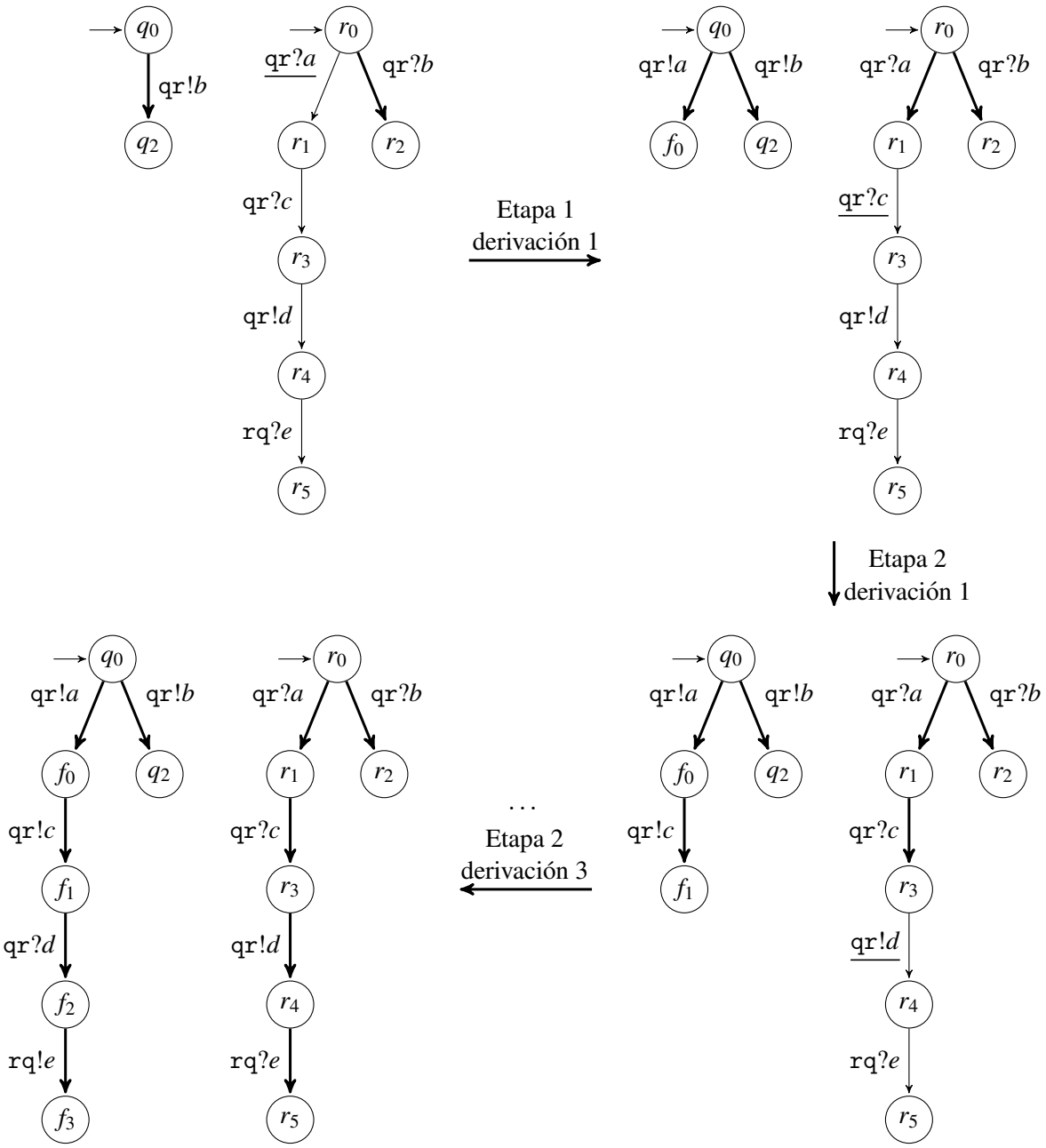

Fig. 4.8: Este caso muestra el funcionamiento de la segunda etapa en un sistema que habilita varias transiciones de comportamiento en profundidad luego de satisfacer una lectura en la primera etapa.

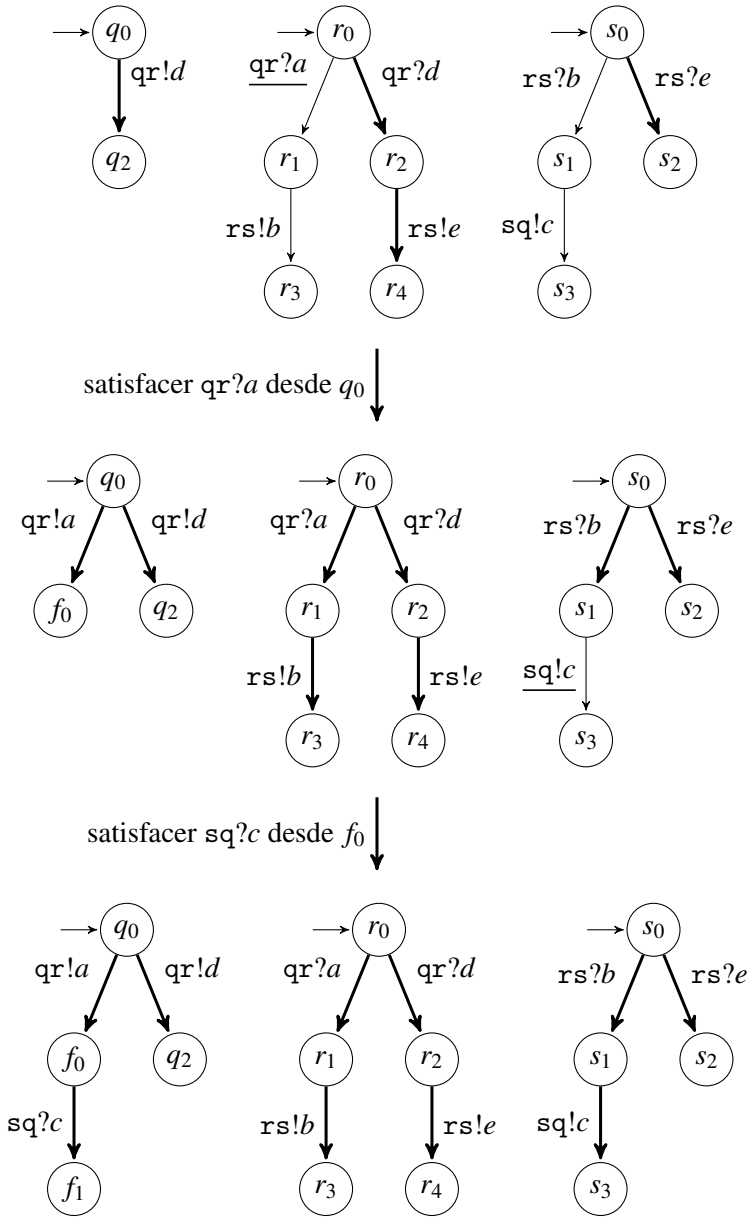

Fig. 4.9: Derivación de un sistema que representa una comunicación transitiva.

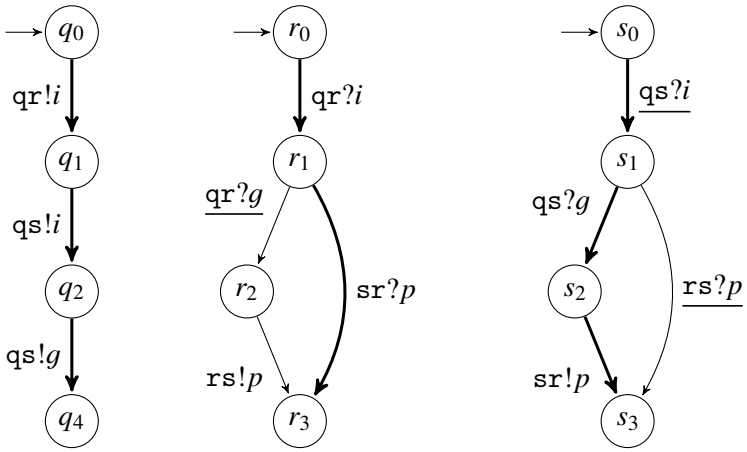

Fig. 4.10: Sistema donde el participante "selector" (q) sólo puede elegir a s como ganador.

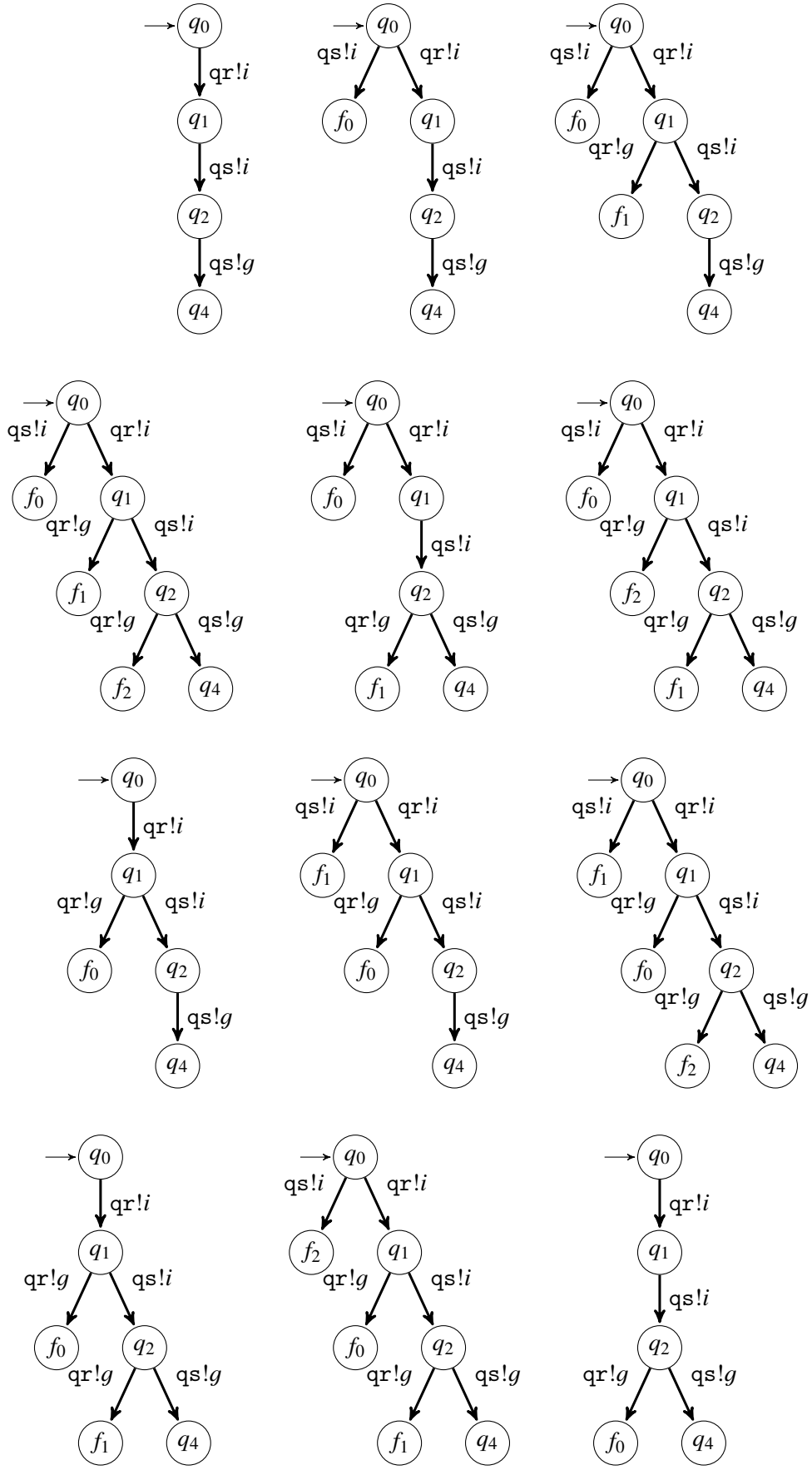

Fig. 4.11: Sucesivas versiones de q generadas. La doceava y última logra que el sistema verifique GMC.

5. CONCLUSIONES Y TRABAJO FUTURO

Se evitarán las conjeturas y especulaciones que resultaron menos conducentes, si bien resultó sumamente enriquecedor superarlas.

A continuación recapitulamos el desarrollo de la presente Tesis de Licenciatura, partiendo de la propuesta y culminando en sus resultados.

El desarrollo de los tipos de sesión es reciente, y busca replicar el éxito que tiene el concepto de tipo de datos en los formalismos de cómputo. Esta referencia a emular pone a disposición de los investigadores muchos conceptos preexistentes para los cuales sus paralelos en el territorio de la concurrencia no presentan una forma obvia. Los formalismos para concurrencia son más heterogéneos que sus contrapartes computacionales, las cuales gozan de modelos canónicos y establecidos.

Actualmente estas teorías están en pleno desarrollo, con lo cual un concepto intuitivamente arraigado como el de subtipado debe ser ejercitado en una variedad de propuestas diferentes, a veces complementarias y otras veces disruptivas entre sí. Se hizo necesario entonces indagar los antecedentes del formalismo elegido, y cómo y hasta qué punto serían aplicables otros intentos de flexibilización de verificación de propiedades.

Consideramos tomar los conceptos existentes de subtipos basados en la reducción de escrituras o incremento de las posibilidades de lectura de Tipos de Sesión Binaria, y adaptarlos a los Tipos de Sesión Multipartita, resultando en las relaciones de subtipado por eliminación de escrituras y por incorporación de lecturas del presente trabajo.

Además, se tomó en cuenta la posibilidad de explorar la permutación de mensajes como procedimiento de subtipado. Esta variante, denominada *subtipado asincrónico*, es propuesto por [17] sobre Tipos de Sesión Binaria. Intuitivamente, un participante puede ser reemplazado por otro que en determinados tramos de la comunicación puede adelantar escrituras y retrasar lecturas manteniendo órdenes relativos. Para priorizar las dos primeras relaciones, suspendimos el desarrollo de esta última.

Las dos primeras relaciones sí fueron abordadas y desarrollamos las primeras definiciones a partir de lo que en ese momento consideramos como premisas más bien generales, puestas a prueba mediante la elaboración de la demostración y la búsqueda de contraejemplos.

La definición de subtipado por eliminación de escrituras soportó la demostración de preservación de propiedades de seguridad con la sola adición de la condición de escritura alternativa de la definición 11. La otra definición, en cambio, precisó una mayor restricción a la hora de permitir la incorporación de lecturas.

Las dos demostraciones requirieron diferentes estrategias, lo que descartó la intuición inicial de que se trataría de relaciones muy similares.

La eliminación segura de escrituras permite un razonamiento más claro a la hora de considerar qué sucede con el comportamiento resultante. Pudo probarse, con demostraciones independientes entre sí, que la definición de subtipado por eliminación de escrituras no atenta contra ninguna de las tres propiedades de seguridad. El lema 1 muestra que el impacto causado por la eliminación no altera el comportamiento que sigue en pie, permitiendo deducir que cualquier configuración alcanzable insegura es imposible o preexistente.

Al incorporar lecturas, el impacto en el conjunto de configuraciones alcanzables es similar al anterior: la naturaleza del comportamiento preexistente no cambia. Este resultado es reflejado por el lema 5, aunque en este caso no facilita el razonamiento sobre garantías de seguridad: la

seguridad se preserva porque el comportamiento no cambia. Sólo puede agregarse nuevo comportamiento subtipando un sistema inseguro, en particular alguno con un estado de recepción no especificada. La conclusión es drástica pero no resultó trivial, requirió más esfuerzo que el caso anterior, en la forma de una relación auxiliar, una proposición y un lema, transversales a la demostración de las tres propiedades de seguridad.

Tanto en esta relación como en la anterior, la modificación del sistema se realiza tomando en cuenta exclusivamente propiedades sintácticas del participante involucrado. Si este comportamiento se ejecuta en función de una elección de otro participante, como es el caso de cualquier lectura, la posibilidad de ejecución pasa a depender de factores externos a la condición necesaria para subtipar.

Resultó claro que había que modificar la definición de la relación condicionándola en función de criterios globales o reduciendo los casos posibles solamente a aquellos que permitan la preservación de seguridad cualquiera sea el contexto del participante. Se eligió esta segunda opción, porque la primera convertiría a la relación en un subtipado semántico, donde no se podría aplicar substitutabilidad segura de participantes con contexto arbitrario.

La utilidad de esta última relación parece limitada. Supondría un aporte inocuo sin cambio de comportamiento. No obstante, la incorporación de una lectura garantizada como inerte puede ser necesaria en algunos escenarios. Puede representar nueva funcionalidad accesible para clientes futuros en un sistema distribuido cambiante. O puede funcionar como parte de otra relación, como primer paso en un procedimiento seguro de agregado incremental de comportamiento, no necesariamente un subtipado.

La diferencia conceptual que mostraron ambas relaciones resultó llamativa e hizo del trabajo teórico un desafío más heterogéneo.

La estrategia a seguir, entonces, estructuró la tesis como un desarrollo teórico con aplicación práctica, aunque no en la típica forma dual de especificación-implementación.

Para avanzar con la segunda parte de la propuesta debimos decidir cual de las dos relaciones de subtipado se mostraba más interesante para aplicar. Consideramos que el escenario de aplicación consistiría en salvar casos en los que el predicado GMC no verifica positivamente. De esta manera, un caso que no verifica pero es subtipo de uno que sí, es garantizado seguro. Por lo tanto, la implementación deseada debe buscar supertipos y verificar alguno como GMC.

Como mencionamos más arriba, la relación de subtipado por incorporación de lecturas agrega lecturas inertes en sistemas seguros. Para estos casos, el predicado GMC obtendría el mismo resultado que el sistema original, excepto por la condición de representabilidad, que detectaría como no reconocida la parte del lenguaje que aporta la nueva lectura. En [16] se muestra en la proposición 3.3 que si un sistema cumple la propiedad de branching pero no la de representabilidad, entonces el sistema reconstruido a partir de las proyecciones de sus participantes es GMC. Entonces, este tipo de situaciones estaría cubierto por la herramienta a través del mecanismo de proyección, la parte del procedimiento de verificación de GMC que evalúa la condición de representabilidad. Estas proyecciones son resultados intermedios que consisten en un nuevo conjunto de participantes equivalentes al original a partir de la especificación global que se pueda construir. No se trataría de un mecanismo completo, pero la posibilidad de generar un sistema equivalente que verifica GMC excepto por una parte puede servir para simplemente detectar una lectura inerte y reconstruir el tipo original que verifica GMC. Esta conjetura es puramente especulativa y preferimos proponer esta posibilidad como trabajo futuro.

Para el caso de un subtipo resultado de eliminar escrituras según $\overset{-!}{\leqslant}$, la verificación de GMC puede ser exitosa o no. Lo que es fácil de advertir es que un subtipo de un sistema que verifica GMC, no verificará GMC pero será seguro. En este caso, la verificación de representabilidad del

predicado de GMC detecta que hay lecturas que no se comunican con la escritura contraparte que fue eliminada. El método para buscar un supertipo GMC, entonces, debe recrear comportamiento desconocido.

Puede resultar intuitivo detectar los estados desde donde deberían partir las escrituras a recrear, pero hay que decidir qué estados le servirían de llegada, porque es información que se pierde. Como nuestro objetivo no es reconstruir tipos de sesión dañados, podemos tranquilamente conformarnos con cualquier supertipo que verifique GMC.

Pusimos en marcha el desarrollo de nuestra herramienta, que reutiliza el modelo de la herramienta GMC provista por E. Tuosto desde [7], y obtuvimos una primera versión que era semejante a la actual primera etapa de la implementación, definida en la sección 4.3, aunque con un mecanismo de detección de lecturas no satisfechas menos preciso que el definitivo y con una estrategia errónea de generación de supertipos.

Esta primera versión derivaba, a partir de la información de un solo *TS* generado al comienzo, una colección de resultados a partir de la combinatoria de las posibles escrituras a agregar. Muy pronto esta estrategia mostró muchos casos duplicados y otros donde se agregaban escrituras de más, generando estados de *mensaje huérfano*.

Estos casos evidenciaban que la satisfacción de una lectura cambia la semántica subyacente al punto de que una nueva reevaluación completa puede ser necesaria. Esta estrategia se cambió a la actual, basada en el agregado de escrituras de a una en la forma de árbol de derivación de supertipos, donde en cada nodo se construye un *TS* y se detectan lecturas insatisfechas.

La estrategia de crear los subgrafos completos que la interacción requiera, a partir del nuevo comportamiento agregado, fue implementada a continuación (en la forma de la segunda etapa, definida en la sección 4.4), y el procedimiento de detección de transiciones no satisfechas que requería para su operación evidenció que su equivalente de la primera etapa no detectaba correctamente algunos casos. El procedimiento correcto fue generalizado y sirvió para ambas etapas.

Como resultado de la implementación podemos destacar que el algoritmo tiene un comportamiento aceptable en sistemas subtipo de GMC, donde es esperable que la eliminación de una escritura invalide antes que nada la condición de representabilidad.

El algoritmo de generación de supertipos es exhaustivo, pero se puede mejorar. Se pueden priorizar las derivaciones que menos transiciones satisfacen y las transiciones a satisfacer por cantidad de coestados posibles. Reducir la cantidad de transiciones a satisfacer y la cantidad de coestados de cada una puede bajar dramáticamente la multiplicidad de candidatos.

La implementación de la herramienta no considera sistemas con ciclos. Una lectura no satisfecha dentro de un ciclo genera una derivación en profundidad en el participante interlocutor.

Buscamos asegurar la resolución de los casos acíclicos como una etapa inicial, y creemos que su tratamiento involucra un método de detección de ciclos en el *TS* junto con la adaptación del concepto de *coestado* pero para estados de llegada.

Nuevamente, estas prospecciones son especulativas y se reservan para desarrollos futuros.

Otra inquietud a investigar radica en que el procedimiento de detección puede detectar como insatisfechas transiciones que solamente son satisfechas en ejecuciones asincrónicas del sistema. Esto sucede porque el *TS* da cuenta solamente de las ejecuciones sincrónicas. Para no agregar estos falsos positivos, el generador de supertipos tiene una guarda que evita agregar transiciones repetidas. La otra manifestación de este inconveniente es que se intente satisfacer desde coestados que permiten longitud de canal mayor a cero, pero en estos casos el predicado GMC tiene la suficiente precisión para determinar si se introdujo un estado inseguro. Esta problemática no fue abordada en el presente trabajo por considerarse un asunto de mayor profundidad teórica.

La etapa preliminar de investigación sirvió de introducción a la teoría estudiada, y dio cuenta

de la gran variedad de inquietudes que motivan la investigación en esta disciplina. En los trabajos consultados son recurrentes las demostraciones de otras propiedades más allá de la de seguridad. Sería de especial interés profundizar el conocimiento de las propiedades investigadas y las necesidades e inquietudes que las motivan, como así también utilizar las nociones desarrolladas en este trabajo como un instrumento familiar para analizarlas.

La totalidad del trabajo muestra la articulación entre los desarrollos teóricos previos, sus interrelaciones, y los problemas abiertos.

Bibliografía

[1] G. Bernardi, O. Dardha, S. J. Gay, and D. Kouzapas. On duality relations for session types. In *TGC*, volume 8902 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2014.

[2] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[3] T. Chen, M. Dezani-Ciancaglini, A. Scalas, and N. Yoshida. On the preciseness of subtyping in session types. *CoRR*, abs/1610.00328, 2016.

[4] R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2011.

[5] P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[6] M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and session types: An overview. In *WS-FM*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009.

[7] Emilio Tuosto. gmc-synthesis-v0.2. `https://bitbucket.org/emlio_tuosto/gmc-synthesis-v0.2/`.

[8] S. J. Gay. Subtyping supports safe session substitution. In *A List of Successes That Can Change the World*, volume 9600 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2016.

[9] S. J. Gay and M. Hole. Types and subtypes for client-server interactions. In *ESOP*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999.

[10] S. J. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

[11] Hernán Rojek Moriceau. generador-supertipos. `https://bitbucket.org/hrojek/generador-supertipos/`.

[12] K. Honda. Types for dyadic interaction. In *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[13] K. Honda, R. Hu, R. Neykova, T. Chen, R. Demangeon, P. Deniélou, and N. Yoshida. Structuring communication with session types. In *Concurrent Objects and Beyond*, volume 8665 of *Lecture Notes in Computer Science*, pages 105–127. Springer, 2014.

[14] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[15] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

[16] J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 221–232. ACM, 2015.

[17] J. Lange and N. Yoshida. On the undecidability of asynchronous session subtyping. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 441–457, 2017.

[18] B. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.

[19] R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[20] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.