Facultad de Matemática, Astronomía y Física

Universidad Nacional de Córdoba

Verificación en Tiempo de Ejecución con Streams

Santiago Gabriel Romero

Director: Dr. César Sánchez

Colaborador: Dr. Pedro R. D'Argenio

"Los problemas son para solucionarlos.
La libertad para comprobarla.
Y en tanto tengamos fe en nuestros sueños,
nada sucede por simple azar"

Richard Bach.

Resumen

En este trabajo presentamos NSRV, un lenguaje de especificación simple y expresivo y un algoritmo para la monitorización de sistemas síncronos. El lenguaje permite especificar propiedades no regulares como corrección con respecto a pre y post condiciones, o propiedades *locales* a un contexto que abstraen la ejecución de otros procedimientos. Si bien existen formalismos capaces de describir tales propiedades, la mayoría tiene bases en la lógica temporal lineal (LTL) y por lo tanto están restringidos a chequear valores de verdad, mientras que NSRV describe también requerimientos numéricos.

El algoritmo de evaluación construye incrementalmente los valores de salida a partir de los valores descritos por el sistema, manteniendo un almacén de expresiones parcialmente resueltas. Caracterizamos sintácticamente una clase de especificaciones cuyos requerimientos de memoria para el algoritmo pueden ser acotados en términos del máximo anidamiento de llamadas del sistema. Además, probamos que decidir si una espeficiación *mal formada* está *bien definida* es un problema indecidible y que el problema es decidible para especificaciones puramente booleanas.

Clasificación: D.2.4 Software / Program Verification

Palabras clave: verificación de sistemas, especificación, monitoreo, programa recursivo, decibilidad

Abstract

We present NSRV, a simple and expressive specification language along with an algorithm for monitoring of synchronous systems. The language can express non-regular properties such as correctness of procedures with respect to pre and post conditions, o *local* properties to a context where the execution of other modules is abstracted away. Most of the formalisms able to specify such properties have their roots on linear temporal logic (LTL) and thus are restricted to checking boolean values, whereas NSRV also describes numerical requirements.

The evaluation algorithm presented incrementally constructs the output from the input, while maintaining a store of partially resolved expressions. We syntactically characterize a class of specifications for which the algorithm's memory requirement can be bounded in terms of the maximum number of nested calls in the program. Also, we prove that deciding if a *bad formed* specification is *well defined* is an undecidable problem and that the problem is decidable for boolean specifications.

Classification: D.2.4 Software / Program Verification

Keywords: system verification, specification, monitoring, recursive program, decidability

Agradecimientos

A mi mamá y Albert, a mis hermanos, Seba y Guti, y a Flor y Naty por el apoyo incondicional a lo largo de estos años, y por darme todo lo que necesité, desde un abrazo amigo hasta un consejo implacable, en el momento que lo necesité.
Esto es también fruto de todo aquello.

A mis amigos, hermanos de la vida, por estar en los momentos para brindar, aunque sobre todo por estar en aquellos de nube negra, y por estar en los momentos intermedios también.

A César Sánchez, por su dedicación, por la ayuda, en persona y océano de por medio, y por su inagotable paciencia. Por darme la posibilidad de trabajar con el y de enriquecerme con esta experiencia más que en el orden académico.

A Pedro D'Argenio, porque estuvo para escucharme, auxiliarme y darme su opinión y consejo siempre que pudo.

Y a todos a los que a pesar de no estar entre estas letras me acompañaron en estos gratos años, de manera fugaz o constante, porque todos forman parte de un camino que disfruté.

Índice general

Capítulo 1

Introducción

"Never trust a computer you can't throw out a window."

Steve Wozniak

Los *sistemas de software* están cada vez más presentes en la vida cotidiana, además de aplicaciones sencillas como procesadores de textos u hojas de cálculo en estaciones de trabajo, los sistemas de software se encuentan hoy en día casi en cualquier lugar: teléfonos móviles, GPSs, cámaras digitales, automóviles, sistemas de control médico, aviones, plantas nucleares, etc.. En algunos casos, como el control de un tren de alta velocidad o de las vías del subterráneo de una ciudad, una falla puede resultar en víctimas fatales; en otros, como equipos en el espacio o ambientes hostiles para el ser humano, un error en el funcionamiento puede ser irreversible. Especialmente en tales dominios de aplicación, en los que una falla puede resultar crítica, es esencial garantizar que el software desarrollado funciona de manera correcta, segura y confiable.

La ingeniería del software es un campo que fue iniciado y siempre guiado por la lucha en busca de garantizar software de calidad, pero hoy en día, y sobre todo en el dominio de sistemas embebidos, las autoridades de legislación y certificación requieren pruebas de la mayor parte de las propiedades críticas

en base a la aplicación de procesos de verificación bien documentados [13]. Además, en las últimas décadas, el concepto de *software como servicio* agregó un nuevo paradigma a la arquitectura de sistemas de software: son vistos cada vez más como agentes autónomos que actúan de acuerdo a un *contrato* preestablecido. En este sentido, la verificación de programas se ha convertido en una tarea importante para controlar que cada parte actúa de acuerdo a lo convenido por el contrato.

Diferentes técnicas se han usado tradicionalmente para verificar programas: *theorem proving*, *model checking* y *testing*. Theorem proving es principalmente aplicado a mano, y permite mostrar la corrección de programas de manera similar a como se demuestra en matemáticas la corrección de un teorema. Al ser una técnica principalmente aplicada a mano, aún con la ayuda de probadores de teoremas semiautomáticos, la demostración de propiedades medianamente complejas puede convertirse en una tarea complicada y lenta. *Model checking* es una técnica de verificación automática y estática, y es principlamente aplicable a sistemas de estados finitos: dado un modelo formal que representa al sistema, comprueba que el modelo cumple con la especificación. Dada la complejidad temporal de los algoritmos, especial cuidado debe ponerse en la construcción del modelo para evitar que su tamaño sea demasiado grande y la técnica deje de ser útil. Apararece entonces muchas veces en este contexto una negociación entre la abstracción del modelo formal y el tiempo de cómputo posible. *Testing*, por otra parte, cubre un amplio campo de diversas técnicas, usualmente *ad-hoc*, e incompletas para encontrar errores en los mismos, y por ende no garantiza la ausencia de fallas.

La *verificación en tiempo de ejecución* es una técnica que combina la verificación formal y la ejecución de programas, y es vista como una técnica liviana que complementa a las técnicas de *theorem proving* y *model checking*, estableciendo un nuevo punto medio que ha recibido gran atención en los últimos años. Una de las cualidades más destacables de la verificación en tiempo de ejecución está dada por su naturaleza: se desarrolla mientras el programa es ejecutado, lo cual abre la posibilidad de reaccionar ante cual-

quier comportamiento no deseado del sistema, teniendo en cuenta el estado actual del mismo. Además, muchas propiedades deseables sobre un sistema dependen de eventos que sólo están disponibles mientras el sistema está en funcionamiento, como la cantidad de mensajes retransmitidos o tiempo de respuesta de un equipo. Por otro lado, la verificación en tiempo de ejecución lidia solo con las ejecuciones observadas a medida que son generadas por el sistema, y entonces es una técnica aplicable a los *sistemas de caja negra* para los cuales no hay un modelo del sistema disponible. En estos casos *model checking* no puede ser aplicado ya que el modelo formal del sistema debe ser construído antes de ejecutar realmente el sistema, para poder controlar todas las posibles ejecuciones.

Mientras la verificación estática como *model checking* pretende mostrar que toda traza (infinita) de un sistema satisface una propiedad descripta por una especificación, el *monitoreo* en tiempo de ejecución se centra en una sola traza (finita). Chequear si una ejecución satisface una propiedad de corrección es usualmente hecho mediante un *monitor*. En esencia, un monitor decide si una ejecución satisface una propiedad de corrección dada, generando una salida *true* o *false*. Un monitor puede ser usado para chequear propiedades durante la ejecución, y bajo esta configuración se lo denomina monitorización *en línea*. En este caso, para que la verificación en tiempo de ejecución sea un proceso pasivo y no afecte la ejecución normal del sistema bajo observación, el monitoreo debe realizarse de manera *eficiente*, con algoritmos de complejidad razonable y analizando las ejecuciones *incrementalmente* para evitar almacenar la ejecucución completa, y siendo también *eficiente* en espacio. Complementariamente, un monitor puede ser usado para analizar un conjunto finito de ejecuciones previamente *almacenadas*, en cuyo caso se trata de monitorización *desconectada*, donde la eficiencia con la que se evalúan las ejecuciones no es vital y puede ser relativa al tiempo disponible para la verificación.

La verificación y recolección de estadísticas en tiempo de ejecución ha sido objeto de estudio en los último años y la lógica temporal lineal (LTL)

es una elección recurrente a la hora de especificar la corrección de requerimientos de sistemas reactivos [15] [14]. Alur et al. presentaron CaRet [1], una lógica temporal de llamadas y retornos para la especificación y verificación de programas estructurados. La lógica es similar a LTL, pero además de las modalidades temporales usuales, CaRet tiene *operadores abstractos* que permiten, por ejemplo, saltar desde una llamada hasta el retorno correspondiente. De esta manera, se puede hacer una abstracción de subprocedimientos y especificar una gran variedad de especificaciones no regulares que no son expresables en LTL puro, como la corrección de procedimientos con respecto a pre y post condiciones. Havelund y Roşu presentaron ptCaRet [10], un formalismo con bases en CaRet que incluye las variantes abstractas para los operadores temporales pasados, junto con un algoritmo para sintetizar monitores a partir de la propiedad especificada.

Finkbeiner et al. han presentado otra extensión de la LTL [9] que combina especificaciones temporales con la recolección de datos estadísticos y permite responder *pedidos* como *"cuántas veces accede a la región crítica un proceso en particular"* y describieron un algoritmo de evaluación eficiente.

El *framework* EAGLE para la definición y la monitorización de lógicas para trazas finitas fue presentado en 2004 [4]. En él, es posible definir varias lógicas, incluyendo lógica temporal lineal con operadores futuros y pasados, formas de lógicas temporales cuantificadas y lógicas temporales para tiempo real, basándose en un pequeño conjunto de poderosas primitivas que permiten la definición de ecuaciones recursivas parametrizadas. Las reglas escritas pueden ser parametrizadas no sólo con fórmulas sino también con tipos primitivos como *float*, *long*, etc., lo cual hace posible expresar propiedades como *"si se alcanza un estado donde $k > 0$, entonces eventualmente se alcanza un estado donde $k = 0$"*.

Más recientemente, en 2005, D'Angelo et al. presentan LOLA [8], un lenguaje de especificación para la verificación de sistemas síncronos, junto con los algoritmos necesarios para la monitorización tanto *en línea* como *desconectada*. El algoritmo para la monitorización de las especificaciones sigue una

estrategia de evaluación parcial: construye la salida a partir de la entrada de manera incremental, manteniendo un conjunto de expresiones parcialmente evaluadas. Además, se distingue una clase de especificaciones cuyo requerimientos de memoria para la monitorización en línea es independiente de la longitud de la traza. Por otro lado, las especificaciones no están restringidas a valores booleanos sino que también pueden expresar propiedades numéricas haciendo referencia al pasado y al futuro, y el algoritmo de evaluación ha sido implementado y aplicado a sistemas industriales probando que el lenguaje es suficientemente expresivo para manipular especificaciones de interés para la industria.

En este trabajo presentamos NSRV, una extensión del lenguaje de especificación LOLA [8]. Además de contar con los constructores originales para expresiones, agregamos constructores para el *futuro abstracto* y *pasado abstracto* de una posición de la traza. Esto permite expresar en NSRV una amplia gama de propiedades no regulares, utilizando valores booleanos y enteros con referencias futuras y pasadas, como por ejemplo *"si la cantidad de mensajes perdidos se duplica durante la ejecución de un procedimiento respecto a su llamada y retorno, el módulo de corrección debe ser invocado inmediatamente después"*. Presentamos el algoritmo para la monitorización en línea de especificaciones y definimos el conjunto de especificaciones *eficientemente monitoreables*, para las cuales los requerimientos de memoria para su monitoreo depende del máximo nivel de anidamiento de llamadas en la traza y es independiente de su longitud. Caracterizamos sintácticamente el conjunto especificaciones que están *bien formadas* y mostramos que las especificaciones de esta clase están *bien definidas* y pueden ser monitorizadas con el algoritmo dado. Para aquellas especificaciones *mal formadas* probamos que saber si está bien definida es decidible si se restringe a valores de verdad, y que el problema es indecidible en el caso general con valores enteros. Finalmente, introducimos el concepto de *inevitabilidad*, que captura la noción de *anticipación* con respecto al futuro, su problema de decisión y una solución al mismo. La inclusión de inevitabilidad a la semántica del

lenguaje permitiría reaccionar tan pronto como es posible ante escenarios de interés del sistema observado. El lenguaje presentado es, bajo nuestro conocimiento, más expresivo que los otros formalismos actuales para verificación en tiempo de ejecución y las especificaciones resultan intuitivas en contraste con aquellos de similar expresividad.

Este trabajo está organizado de la siguiente manera: el capítulo 2 presenta la extensión del lenguaje. Primero introduce las nociones básicas que componen la extensión y luego su sintaxis, semántica y las diferentes categorizaciones de especificaciones. Finalmente, se describe el algoritmo de evaluación y los requerimientos de memoria para especificaciones eficientemente monitoreables. El capítulo 3 encara el problema de decidir si una especificación que no garantiza de manera sintáctica que tiene una única solución, puede aún tenerla. Para las diferentes especificaciones se analiza el problema de decisión y se presentan resultados concluyentes. En el capítulo 4 se presentan ejemplos con aplicaciones reales donde se aprecia la expresividad y potencia de la extensión presentada, incluyendo propiedades clásicas de la literatura. El capítulo 5 introduce el concepto de *inevitabilidad*, define el problema a resolver y prueba que es decidible. El capítulo 6 repasa los principales aportes de este trabajo y las posibles líneas de trabajo futuro, tanto en el campo de investigación como en la parte práctica.

Capítulo 2

Lenguaje de especificación NSRV

En este capítulo se presenta y describe el lenguaje de *verificación en tiempo de ejecución con streams anidados*, NSRV, por su definición en inglés *Nested Stream Runtime Verification*. Primero, se explican las bases sobre las cuales se define el lenguaje: *streams*, trazas estructuradas y caminos abstractos y concretos de las mismas. Luego se describe la sintaxis del lenguaje, su semántica y diversos conceptos y resultados sobre especificaciones del lenguaje.

2.1. Streams

NSRV modela la verificación en tiempo de ejecución como el cálculo de valores de *streams*.

Definición 2.1 (Stream)**.** *Un stream σ de tipo T y longitud $j+1$ es una secuencia finita de valores $\sigma = v_0...v_j$ tal que $v_i \in T$, $\forall i \leq j$.*

En la versión simple de NSRV que presentamos aquí, un stream puede ser únicamente entero o booleano, es decir, de tipo Int o $Bool$. Si σ es un stream de longitud $j+1$, entonces $Tipo(\sigma)$ denota el tipo de σ y $\sigma[i]$ denota el valor del stream σ la posición $i \leq j$. Existen dos clases de streams en NSRV:

- *Streams de entrada*: cuyos valores son independientes de la especificación y los proporciona el sistema monitorizado durante la ejecución.

- *Streams de salida*: cuyos valores son descritos por la especificación, y pueden depender tanto de los streams de entrada como de los streams de salida.

Una especificación NSRV se evalúa a partir de una *traza* que define los valores de los streams de entrada y utilizando la descripción de los streams salida.

2.2. Trazas

Las trazas definidas en base a streams, aunque finitas, son semejantes a los *cómputos estructurados* de Alur et al. [1], donde un *cómputo estructurado* es una secuencia infinita de estados, cada uno etiquetado con un conjunto de proposiciones atómicas, y posiblemente enriquecido con un símbolo de *llamada* o de *retorno*. Una llamada denota la invocación de un módulo, y el correspondiente retorno denota la salida de tal módulo, donde un módulo puede corresponderse con un procedimiento o una función de algún lenguaje imperativo estructurado como C, o métodos de lenguajes orientados a objetos como Java, Python o Ruby, o invocaciones remotas de componentes en sistemas distribuídos. Sin embargo, tal como proponen Havelund y Roşu [10], distinguimos los estados de llamada y retorno de aquellos de entrada y salida a un procedimiento o función. La distinción, como se verá más adelante, permite una mayor flexibilidad al expresar propiedades pero requiere condiciones extras sobre la traza: una llamada siempre precede a una entrada, y una salida siempre precede a un retorno.

Así como las llamadas y retornos en la ejecución de programas estructurados definen un lenguaje de paréntesis equilibrados, esta propiedad es capturada para las trazas en NSRV. Para ello, definimos *modelos de ejecución*, que son *nested words* y representan la jerarquía de una traza. Las *nested*

words fueron propuestas por Alur y Madhusudan [3] para capturar modelos en los cuales existe una secuencia lineal natural de las posiciones y una jerarquía anidada de posiciones que se corresponden (dada por las llamadas y retornos).

Definición 2.2 (Modelo de ejecución)**.** *Un modelo de ejecución de ancho* $k \geq 0$ *es una relación binaria* η *sobre* $\{0, ..., k-1\}$ *tal que:*

- *Si* $\eta(i, j)$*, entonces* $i < j$*.*
- *Si* $\eta(i, j)$ *y* $\eta(i, j')$*, entonces* $j = j'$*; y si* $\eta(i, j)$ *y* $\eta(i', j)$*, entonces* $i = i'$*.*
- *Si* $\eta(i, j)$*,* $\eta(i', j')$ *y* $i < i'$*, entonces* $j < i'$ *o* $j' < j$*.*

Si η es un modelo de ejecución, $i \rightsquigarrow_\eta j$ denotará $\eta(i, j)$, o simplemente $i \rightsquigarrow j$ si el modelo de ejecución es obvio en el contexto. La figura 2.1 muestra un modelo de ejecución ejemplo.

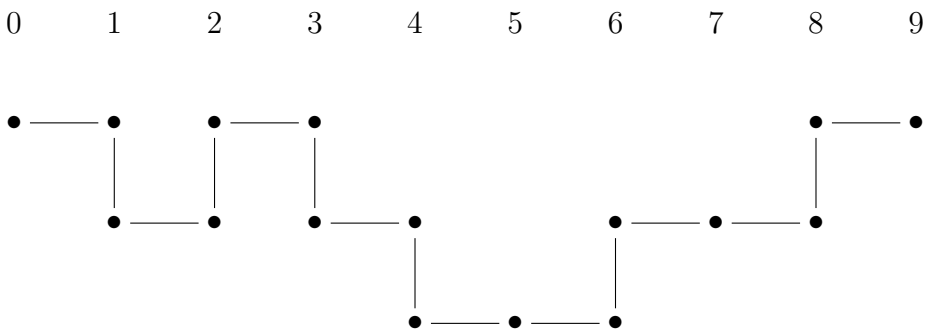

Figura 2.1: Representación del modelo de ejecución $\eta = \{(1, 2), (3, 8), (4, 6)\}$.

Una *traza estructurada* es un conjunto de streams de entrada que satisfacen ciertos requerimientos sobre sus valores. Para distinguir los estados de llamada, entrada, salida y retorno mencionados anteriormente, y de manera similar a los predicados atómicos utilizados por Havelund y Roşu [10], pedimos que cada traza estructurada siempre defina cuatro streams de entrada particulares: τ_{call} para llamadas, τ_{en} para entradas, τ_{ex} para salidas y τ_{ret} para retornos de un módulo. Llamaremos a éstos *streams de control*.

Definición 2.3 (Trazas)**.** *Sea τ una tupla de streams de entrada $\tau = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m \rangle$, todos de longitud N, y η un modelo de ejecución de ancho N. La tupla τ es una* traza estructurada*:*

a. Los streams τ_{call}, τ_{en}, τ_{ex} y τ_{call} son booleanos y a lo sumo uno de ellos puede ser ***true*** *en cualquier posición $i < N$.*

b. Para toda posición $i < N$:

- *Si $\tau_{call}(i) =$* ***true*** *entonces $i + 1 < N$ y $\tau_{en}(i + 1) =$* ***true****. Análogamente, si $\tau_{en}(i + 1) =$* ***true*** *entonces $\tau_{call}(i) =$* ***true****.*
- *Si $\tau_{ret}(i) =$* ***true*** *entonces $i > 0$ y $\tau_{ex}(i-1) =$* ***true****. Análogamente, si $\tau_{ex}(i - 1) =$* ***true*** *entonces $\tau_{ret}(i) =$* ***true****.*

c. Si $\tau_{call}(i) =$ ***true****, entonces $i \rightsquigarrow j$ para algún j y $\tau_{ret}(j) =$* ***true****.*

d. Si $\tau_{ret}(j) =$ ***true****, entonces $i \rightsquigarrow j$ para algún i, y $\tau_{call}(i) =$* ***true****.*

Si τ sólo satisface a, b y d, diremos que es una traza estructurada parcial inicial*, y si sólo satisface a, b y c entonces τ es una* traza estructurada parcial final*. Además, diremos que τ es de ancho m y que η es el modelo de ejecución inducido por τ.*

Note que bajo esta definición una traza estructurada parcial inicial o final *puede* ser una traza estructurada también.

Para toda traza τ, una posición en la cual τ_{call} es **true** será definida como una *llamada*, si τ_{en} es **true** será llamada *entrada*, será llamada *salida* si τ_{ex} es **true** y *retorno* si τ_{ret} lo es. La restricción (a) es natural. Las restricciones (b), (c) y (d) no sólo imponen la fuerte asunción de que no existen posiciones entre llamadas y entradas, o entre salidas y retornos de procedimientos, sino además exige que cada llamada tenga su correspondiente retorno, lo cual es razonable para cualquier ejecución finita. Aunque los métodos y algoritmos presentados pueden ser modificados y adaptados para considerar posiciones intermedias entre llamadas y comienzos o entre salidas y retornos, asumimos

que tales posiciones no existen ni en el contexto de la llamada ni en el de la entrada para facilitar la comprensión de la semántica del lenguaje. Las llamadas y retornos suceden en el contexto de la función o procedimiento que realiza la invocación, y las entradas y salidas suceden en el contexto del procedimiento llamado.

Informalmente, una traza estructurada es aquella que tiene para cada llamada su correspondiente retorno, y no existen retornos sin una llamada que le corresponda. Una traza estructurada parcial inicial identifica aquellos prefijos de trazas estructuradas donde algunos retornos a llamadas pueden haber no sucedido aún. Y las trazas estructuradas parciales finales denotan los sufijos de trazas estructuradas, donde pueden ocurrir más retornos que llamadas.

Si $\tau = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m \rangle$ es una traza de longitud N y ancho m, y $\tau' = \langle \tau'_{call}, \tau'_{en}, \tau'_{ex}, \tau'_{ret}, \tau'_1, ..., \tau'_m \rangle$ es una traza de longitud N' y ancho m, tales que $Tipo(\tau_i) = Tipo(\tau'_i)$ para todo $i \leq m$, denotaremos con $\tau\tau'$ a la traza de longitud $N + N'$ y ancho m, $\tau\tau' = \langle \tau''_{call}, \tau''_{en}, \tau''_{ex}, \tau''_{ret}, \tau''_1, ..., \tau''_m \rangle$, donde:

- $\tau''_j(i) = \begin{cases} \tau_j(i) & \text{si } 0 \leq i < N \\ \tau''_j(i+N) & \text{si } 0 \leq i < N' \end{cases}$

  para todo $j \in \{call, en, ex, ret, 1, ..., m\}$.

2.2.1. Camino concreto y camino abstracto

En toda traza, para cada posición dentro de ella, existen dos formas de moverse hacia la posición predecesora o sucesora: sobre el *camino concreto* o sobre el *camino abstracto*. Si η es un modelo de ejecución y $i \rightsquigarrow j$, j será llamada el *sucesor abstracto* de la posición i, y la posición i será el *predecesor abstracto* de la posición j. Para definir *caminatas* hacia adelante y hacia atrás sobre el camino abstracto de una traza τ definimos dos funciones, suc_η y $pred_\eta$, donde η es el modelo de ejecución inducido por τ.

Definición 2.4. *Sea τ una traza de longitud N con modelo de ejecución inducido η, las funciones $pred_\eta, suc_\eta : \mathbb{N}_\perp \to (\{0, ..., N-1\} \cup \{\perp\})$ se definen como:*

$$pred_\eta(i) = \begin{cases} j & si\ j \rightsquigarrow i \\ i-1 & si\ 0 < i\ y\ no\ j \rightsquigarrow i \\ \perp & caso\ contrario. \end{cases}$$

$$suc_\eta(i) = \begin{cases} j & si\ i \rightsquigarrow j \\ i+1 & si\ i < N\ y\ no\ i \rightsquigarrow j \\ \top & caso\ contrario. \end{cases}$$

Luego, $suc^k_\eta(i)$ y $pred^k_\eta(i)$ son la composición de k aplicaciones de las funciones definidas:

$$suc^k_\eta(i) = suc_\eta(suc_\eta(...(suc_\eta(i))...))$$
$$pred^k_\eta(i) = pred_\eta(pred_\eta(...(pred_\eta(i))...))$$

Dada una posición i, $0 \leq i \leq N$, el resultado de $suc^k_\eta(i)$ y $pred^k_\eta(i)$ son las posiciones sucesora y predecesora *k-distantes* de i sobre el camino abstracto de la traza, respectivamente. Para simplificar la presentación del resto de la extensión del lenguaje definimos el operador

$$\hat{+}_\eta : \mathbb{N} \times \mathbb{N} \to (\{0, ..., N-1\} \cup \{\perp\})$$

de la siguiente manera:

$$i \hat{+}_\eta k = \begin{cases} pred^k_\eta(i) & \text{si } k < 0 \\ i & \text{si k} = 0 \\ succ^k_\eta(i) & \text{si } k > 0 \end{cases}$$

La Figura 2.2 muestra una traza estructurada donde cada posición de la traza, salvo la última cuyo sucesor es $\perp$, tiene indicado sus sucesores sobre el camino concreto y sobre el camino abstracto: $i \overset{\frown}{\rightarrow} j$ indica que j es el sucesor de i sobre el camino concreto y $i \overset{\cdots\rightarrow}{} j$ indica que j es el sucesor

de i sobre el camino abstracto. Además, se distingue cual de los streams de control (τ_{call}, τ_{en}, τ_{ex} o τ_{ret}) es **true** en cada posición.

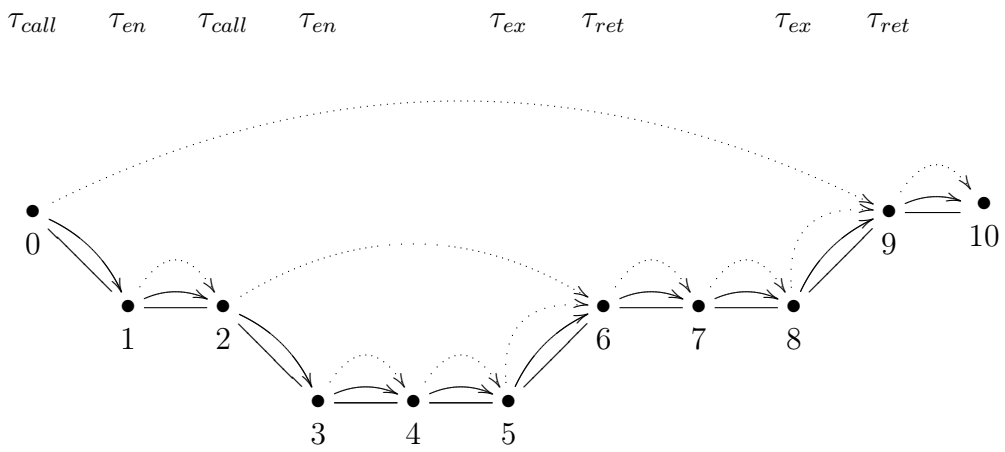

Figura 2.2: Traza estructurada de longitud 11 con sus sucesores sobre el camino concreto y abstracto.

2.3. Sintaxis

Una especificación NSRV describe cómo se computan los valores de los streams de salida a partir de una traza mediante la definición de ecuaciones.

Definición 2.5 (Especificación NSRV)**.** *Una especificación NSRV es un conjunto de ecuaciones sobre variables de streams tipadas de la forma:*

$$
\begin{aligned}
s_1[n] &= e_1(n,\, t_{call},\, t_{en},\, t_{ex},\, t_{ret},\, t_1, ..., t_m,\, s_1, ..., s_r) \\
&\;\;\vdots \qquad\quad \vdots \\
s_r[n] &= e_r(n,\, t_{call},\, t_{en},\, t_{ex},\, t_{ret},\, t_1, ..., t_m,\, s_1, ..., s_r)
\end{aligned}
$$

donde n es la variable de posición, $t_{call}, t_{en}, t_{ex}, t_{ret}, t_1, ..., t_m$ son variables

independientes de streams, $s_1, ..., s_r$ *son variables dependientes de streams y* $e_1, ..., e_r$ *son expresiones de streams sobre* n, t_{call}, t_{en}, t_{ex}, t_{ret}, $t_1, ..., t_m$, $s_1, ..., s_r$. *Las variables independientes hacen referencia a streams de entrada, incluyendo los streams de control, las variables dependientes hacen referencia a streams de salida y la variable de posición hace referencia a la posición actual en la traza. Una especificación también puede declarar expresiones booleanas como disparadores. Los disparadores generan notificaciones en el instante de tiempo en el cual el valor de la expresión se resuelve y su valor es* ***true****. Los disparadores en NSRV se definen como:*

$$\textbf{\textit{trigger}}(e)$$

donde e *es una expresión de la forma* $v[n]$ *o* $\neg v[n]$, *para alguna variable booleana* v.

Una expresión de streams se construye de la siguiente manera:

- *Si* c *es una constante de tipo* T, *entonces* c *es una expresión de stream de tipo* T.
- *Si* v *es una variable de tipo* T, *entonces* $v[n]$ *es una expresión de stream de tipo* T.
- *Sea* $f : T_1 \times T_2 \times ... \times T_k \rightarrow T$ *un operador de aridad* k. *Si para* $0 \leq i \leq k$, e_i *es una expresión sobre streams de tipo* T_i, *entonces* $f(e_0, ..., e_k)$ *es una expresión de stream de tipo* T.
- *Si* b *es una expresión sobre streams de tipo Bool y* e_1, e_2 *son expresiones sobre streams de tipo* T, *entonces* ***if*** b ***then*** e_1 ***else*** e_2 *es una expresión de stream de tipo* T.
- *Si* v *es una variable de tipo* T, c *es una constante de tipo* T *y* k *es un entero no nulo, entonces* $v[C(n) + k|c]$ *y* $v[A(n) + k|c]$ *son expresiones de desplazamiento sobre streams de tipo* T. *Informalmente, estas expresiones se refieren al desplazamiento de* k *posiciones de tiempo con respecto a la posición actual sobre el camino concreto y abstracto de la*

traza, respectivamente. El camino sobre el cual se realiza el desplazamiento está dado por el modificador *que se aplique a la posición actual: C o A. El valor c denota el* valor por defecto *que se usará en caso de que el desplazamienzo de k posiciones esté fuera de los límites de la traza (es decir, si es menor que 0 o mayor que N). Si el desplazamiento es sobre el camino concreto diremos que la expresión es un* desplazamiento concreto*, y si es sobre el camino asbtracto diremos que es un* desplazamiento abtracto*.*

Las expresiones de stream de la forma $v[n]$, $v[C(n)+k|c]$ y $v[A(n)+k|c]$ serán llamadas expresiones elementales*, y las variables independientes t_{call}, t_{en}, t_{ex}, t_{ret} que denotan los streams de control serán llamadas* variables de control*.*

Notación: De aquí en adelante se omitirán los modificadores de desplazamientos concretos, excepto donde se considere adecuado hacerlos explícitos, y por convención $v[n+k \mid c]$ representará la expresión $v[C(n)+k \mid c]$.

Ejemplo 2.6. *Sean t_1 y t_2 dos variables booleanas independientes. La siguiente es un ejemplo de una especificación NSRV:*

$$s_1[n] = false$$

$$s_2[n] = t_2[n]$$

$$s_3[n] = s_1[A(n)+1|true]$$

$$s_4[n] = s_4[n-1 \mid -1] + 1$$

$$s_5[n] = 2^{s_4[n]}$$

$$s_6[n] = \textbf{\textit{if}}\ s_5[n] \leq 1\ \textbf{\textit{then}}\ 1\ \textbf{\textit{else}}\ (s_6[n-2 \mid 0] + s_6[n-1 \mid 0])$$

$$s_7[n] = t_2[n] \vee (t_1[n] \wedge s_7[n+1 \mid true])$$

La variable s_1 denota un stream que es siempre ***false*** *y s_2 denota un stream cuyo valor en cada posición es exactamente el de t_2. El valor de los streams descriptos por $s_3, s_4, ..., s_7$ se obtiene evaluando las expresiones de sus ecuaciones posición a posición. Por ejemplo, la variable independiente s_3 describe un stream que es* ***false*** *en todas las posiciones excepto en la última posición donde el valor por defecto,* ***true****, es usado. El stream correspondiente a s_4 en cada posición se obtiene tomando el valor del stream en la posición anterior (el valor por defecto -1 se usa en la primer posición) y sumándole* 1*. De esta manera, s_4 describe un stream que se comporta como un contador: en la posición i su valor es exactamente i. La variable s_5 denota un stream cuyo valor en una posición i es 2^i, mientras que s_6 describe un stream que contiene la sucesión de Fibonacci. Finalmente, el stream que corresponde a s_7 tiene en cada posición el valor de la fórmula temporal τ_1* ***Until*** *τ_2, con la asunción de que todas las eventualidades se consideran resueltas al finalizar la traza.*

A modo de simplificar la presentación de los algoritmos definimos la *forma canónica* de una especificación y las especificaciones *unitarias*.

Definición 2.7 (Forma canónica y expresiones llanas)**.** *Una especificación NSRV φ está en su* forma canónica *si cada ecuación es de la forma $s_i[n] = e_i$ donde e_i está restringida a alguna de las siguientes expresiones:*

- *c constante*
- *$v_j[n]$, con v_j una variable*
- *$f(v_1[n], ..., v_k[n])$, la aplicación de un operador sobre valores actuales*
- ***if*** *$v_0[n]$* ***then*** *$v_1[n]$* ***else*** *$v_2[n]$, un condicional sobre valores actuales*
- *$v_j[n+k|c]$ o $v_j[A(n)+k|c]$, un desplazamiento sobre el camino concreto o abstracto*

Una expresión e_i de esta forma será llamada expresión llana*.*

Definición 2.8 (Especificación unitaria)**.** *Una especificación NSRV φ es unitaria si toda expresión de desplazamiento en φ se construye con $k = 1$ o $k = -1$.*

Toda especificación puede ser llevada a forma canóninca y transformada en unitaria preservando la semántica. La demostración formal junto con la forma de realizar la transformación será detallada en el la sección 2.5. En el resto del capítulo asumiremos que todas las especificaciones NSRV están en su forma canónica.

2.4. Semántica

Para dar la semántica de una especificación NSRV definimos los *modelos de evaluación*, que representan la forma de satisfacer las ecuaciones que componen una especificación y describen la relación entre los valores de streams de entrada y los valores de los streams de salida.

Definición 2.9 (Modelo de evaluación)**.** *Sea φ una especificación NSRV con variables independientes $t_1, ..., t_m$ de tipos $T_{t_1}, ..., T_{t_m}$ además de las variables de control, y con variables dependientes $s_1, ..., s_r$ de tipos $T_{s_1}, ..., T_{s_r}$. Sea τ una traza estructurada de longitud $N+1$ y ancho m tal que τ_i es de tipo T_{t_i}, y sea η el modelo de ejecución de ancho $N+1$ inducido por τ. La tupla $\langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ de streams de longitud $N+1$ de los tipos apropiados es un modelo de evaluación de φ si para cada ecuación*

$$s_i[n] = e_i(n, t_{call}, t_{en}, t_{ex}, t_{ret}, t_1, ..., t_m, s_1, ..., s_r)$$

$\langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ satisface las siguientes ecuaciones asociadas:

$$\sigma_i(j) = [\![e_i]\!](j), \text{ para } 0 \leq j \leq N$$

donde $[\![e_i]\!] : \{0..N\} \to T_{s_i}$, y se define como sigue.

- *Los casos base:*

$$[\![c]\!](j) = c$$

$$[\![t_i[n]]\!](j) = \tau_i[j]$$

$$[\![s_i[n]]\!](j) = \sigma_i[j]$$

- *Los casos inductivos:*
  - $[\![f(e_1, ..., e_k)]\!](j) = f([\![e_1]\!](j), ..., [\![e_k]\!](j))$
  - $[\![\textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ e_1\ \textbf{\textit{else}}\ e_2]\!](j) = \begin{cases} [\![e_1]\!](j) & si\ [\![b]\!](j) \\ [\![e_2]\!](j) & caso\ contrario. \end{cases}$
  - $[\![v[C(n) + k|c]]\!](j) = \begin{cases} [\![v[n]]\!](j + k) & si\ 0 \leq j + k \leq N \\ c & caso\ contrario \end{cases}$
  - $[\![v[A(n) + k|c]]\!](j) = \begin{cases} [\![v[n]]\!](j \hat{+}_\eta k) & si\ j \hat{+}_\eta k \neq \perp \\ c & caso\ contrario \end{cases}$

Cuando sea necesario distinguir la función de evaluación $[\![\cdot]\!]$ sobre diferentes modelos de evaluación σ usaremos $[\![\cdot]\!]_\sigma$.

Es decir, un modelo de evaluación describe el valor de todos los streams de salida en todas las posiciones de la traza. Todos los elementos de las expresiones son variables de streams de entrada o variables de streams de salida, junto a constructores de los tipos adecuados con semántica bien definida. En consecuencia, dado un modelo de evaluación $\langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$, todas las expresiones que definen una ecuación de φ corresponden con un único valor.

Nótese que una especificación no necesariamente tiene un único modelo de evaluación para una traza estructurada dada, puede tener muchos o incluso ningún modelo de evaluación para ella.

Definición 2.10 (Traza compatible)**.** *Si φ es una especificación NSRV, $\tau[N]$ es una traza compatible con φ si τ es una traza estructurada de longitud N y tiene los tipos apropiados para φ.*

Definición 2.11 (Especificación bien definida)**.** *Una especificación NSRV φ está bien definida si para toda traza compatible $\tau[N]$, φ tiene exactamente un modelo de evaluación.*

Además, si existe una traza compatible tal que la especificación tiene más de un modelo de evaluación diremos que está *subdefinida*, y diremos que está *sobredefinida* si existe una traza compatible tal que no hay un modelo de evaluación de φ para tal traza.

Ejemplo 2.12. *Considérense las siguientes especificaciones:*

$$\varphi_1 : s[n] = s[n]$$

$$\varphi_2 : s[n] = s[n-1|0] + 1$$

$$\varphi_3 : s[n] = \neg s[n]$$

Para cualquier traza compatible $\tau[N]$, la especificacion φ_1 puede tener varios modelos de evaluación, por ejemplo, asumiendo que s define un stream booleano y que $N = 3$, $\langle \textbf{\textit{true}}, \textbf{\textit{false}}, \textbf{\textit{true}} \rangle$ y $\langle \textbf{\textit{true}}, \textbf{\textit{true}}, \textbf{\textit{true}} \rangle$ son posibles modelos de evaluación y por lo tanto está subdefinida. Por otra parte, la especificación φ_2 está bien definida, tiene exactamente un modelo de evaluación: para cada traza compatible $\tau[N]$ su modelo de evaluación es $\langle 1, 2, ..., N \rangle$. Finalmente, la especificación φ_3 está sobredefinida, no existe modelo de evaluación posible puesto que no existe una solución a la ecuación $\sigma(i) = \neg\sigma(i)$.

Para identificar aquellas especificaciones que están bien definidas imponemos una restricción sintáctica sobre especificaciones NSRV que garantiza que existe exactamente un modelo de evaluación para la especificación. Para ello, definimos el *grafo de dependencia* de una especificación, que captura la dependencia estática entre valores de variables en distintas posiciones relativas, y luego describimos las condiciones que debe satisfacer dicho grafo. Sin embargo, como se verá mas adelante, el grafo de dependencia puede contener dependencias espurias dependiendo de la especificación y la traza.

El conjunto (infinito) de todas las expresiones de streams, sobre las variables $\{t_i\}_{i\in\mathbb{N}}$ y $\{s_i\}_{i\in\mathbb{N}}$, será denotado con $\mathcal{E}$, y el subconjunto que contiene todas las expresiones elementales será denotado con $\mathcal{E}_{[\cdot]}$. Definimos la función $SubExpr : \mathcal{E} \to \mathcal{P}(\mathcal{E})$ que dada una expresión sobre steams retorna el conjunto de subexpresiones en ella, recursivamente como:

- Casos base:
  - $SubExpr(c) = \{\ c\ \}$
  - $SubExpr(v[n]) = \{\ v[n]\ \}$
  - $SubExpr(v[n+k|c]) = \{\ v[n+k|c]\ \}$
  - $SubExpr(v[A(n)+k|c]) = \{\ v[A(n)+k|c]\ \}$
- Casos inductivos:
  - $SubExpr(f(e_0, ..., e_\ell)) = \bigcup_{j\leq\ell} SubExpr(e_j)$
  - $SubExpr(\textbf{if } e_0 \textbf{ then } e_1 \textbf{ else } e_2) = SubExpr(e_0) \cup SubExpr(e_1) \cup SubExpr(e_2)$

Definición 2.13 (Grafo de dependencia)**.** *Sea φ una especificación NSRV. El grafo de dependencia de φ es un multi-grafo dirigido $\mathcal{G}_\varphi = (V, E_c \cup E_a)$, con $V = \{t_1, ..., t_m, s_1, ..., s_r\}$ y tal que el arco $\langle s_i, v_j, k\rangle$ con peso k está en E_c si y sólo si $v_j[n+k|c] \in SubExpr(e_j)$ o si $k = 0$ y $v_j[n] \in SubExpr(e_j)$, y el arco $\langle s_i, v_j, k\rangle$ con peso k está en E_a si y sólo si $v_j[A(n)+k|c] \in SubExpr(e_j)$. Los arcos en E_c serán llamados arcos concretos y los arcos en E_a serán llamados arcos abstractos.*

Luego, si φ es una especificación y $s_j[n] = e_j$ es una ecuación en φ, los siguientes arcos corresponden al grafo de dependencia:

- $\{s_j \xrightarrow{0} v \mid v[n] \in SubExpr(e_j) \cap \mathcal{E}_{[\cdot]}\} \subseteq E_c$
- $\{s_j \xrightarrow{k} v \mid v[n+k|\ c] \in SubExpr(e_j) \cap \mathcal{E}_{[\cdot]}\} \subseteq E_c$
- $\{s_j \xrightarrow{k} v \mid v[A(n)+k|\ c] \in SubExpr(e_j) \cap \mathcal{E}_{[\cdot]}\} \subseteq E_a$

Un *camino* en un grafo es una secuencia $v_1, ..., v_{k+1}$ de vértices, para $k \geq 1$, y arcos $e_1, ..., e_k$ tales que $e_i : \langle v_i, v_{i+1}, w_i \rangle$. El camino es *cerrado* si y sólo si $v_1 = v_{k+1}$. El peso total de un camino es la suma de todos los pesos de sus arcos. El *peso concreto* de un camino es la suma de los pesos de sus arcos concretos y el *peso abstracto* es la suma de los pesos de sus arcos abstractos. Además, si todos los arcos de un camino son concretos diremos que es un *camino concreto*, y diremos que es un *camino abstracto* si tiene al menos un arco abstracto. Es interesante notar que no existen arcos abstractos de peso 0 en el grafo de dependencia de un especificación.

Definición 2.14 (Especificación bien formada)**.** *Una especificación φ está bien definida si su grafo de dependencia $\mathcal{G}_\varphi$:*

- *No tiene caminos concretos cerrados de peso total 0.*
- *No tiene un camino abstracto cerrado conteniendo arcos abstractos $e_1 : \langle v_1, v_1', k_1 \rangle$ y $e_2 : \langle v_2, v_2', k_2 \rangle$ tales que $k_1 \cdot k_2 < 0$.*
- *Si tiene un camino cerrado abstracto con peso concreto K y peso abstracto K' tal que $K \cdot K' < 0$, entonces $|K| < |K'|$.*

Aquí $k \cdot k' < 0$ captura exactamente si los valores k y k' tienen signos contrarios.

Un desplazamiento sobre el camino abstracto, por pequeño que sea, puede significar un desplazamiento mucho mayor, en posiciones, sobre el camino concreto dependiendo del modelo de ejecución. La tercera condición garantiza que el valor de una expresión que depende de valores en posiciones futuras (pasadas) sobre el camino abstracto y de valores en posiciones pasadas (futuras) sobre el camino concreto, nunca depende de sí mismo.

Ejemplo 2.15. *Considere la siguiente expecificación NSRV sobre las variables independientes t_1 y t_2:*

$$s_1[n] = \textbf{\textit{if}}\ t_{call}[n] \wedge t_1[n]\ \textbf{\textit{then}}\ s_3[A(n)+1|false]\ \textbf{\textit{else}}\ true$$

$$s_2[n] = t_2[n+1|true] \vee s_1[n-1|true]$$

$$s_3[n] = s_2[n-2|false] \wedge s_3[n+1|true]$$

La variable s_1 define que si el stream de entrada τ_1 es ***true*** *en cualquier llamada a un procedimiento, entonces el stream descripto por s_3 debe ser* ***true*** *cuando ese procedimiento retorne. Es decir, puede verse como la correción en cuanto a la precondición $\tau_1 =$* ***true*** *y la postcondición especificada por s_3. Sin embargo, esta especificación no está bien definida ya que su grafo de dependencia contiene un camino cerrado abstracto de peso concreto K y peso abstracto K' tal que $K \cdot K' < 0$ y $|K| \geq |K'|$. El camino cerrado está dado por $s_1 \overset{+1}{\cdots\!\!>} s_3 \xrightarrow{-2} s_2 \xrightarrow{-1} s_1$, como puede verse en la Figura 2.3 que muestra el grafo de dependencia de la especificación.*

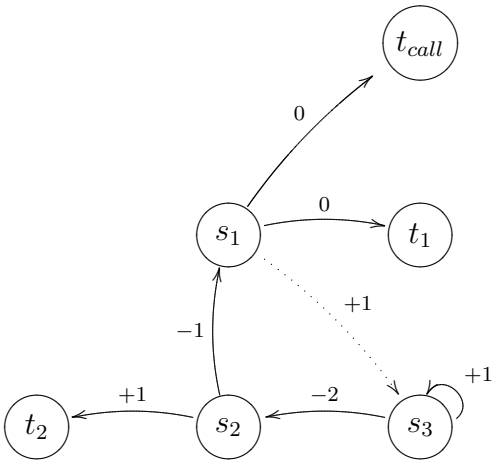

Figura 2.3: Grafo de dependencia del Ejemplo 2.15

Para probar que si una especificación está bien formada entonces está bien definida construimos un grafo de dependencia explícito que tiene un vértices por cada stream y cada posición de una traza estracturada dada. El grafo describe la dependencia explícita (y no estática, como el grafo de dependencia) que hay entre los valores de los streams en diferentes posiciones

de la traza, y por lo tanto su tamaño depende también de la longitud de la traza. Luego, probaremos que la ausencia de ciclos en el grafo de dependencia garantiza la existencia y unicidad del modelo de evaluación para la traza, y que en caso de tener un ciclo, la especificación no está bien formada.

Definición 2.16 (Grafo de dependencia explícito)**.** *Sea φ una especificación NSRV, $\tau[N+1]$ una traza compatible con φ y η el modelo de ejecución de ancho $N+1$ inducido por τ. El grafo de dependencia explícito para φ, $N+1$ y η es un grafo dirigido $\mathcal{G}_{(\varphi,N+1,\eta)}$: $(V, E_c \cup E_a)$, donde V contiene un vértice para cada posición $i \leq N+1$ y variable dependiente en φ y:*

- *Un arco $e : \langle (s_j, i), (v, i) \rangle$ pertenece a E_c si y sólo si $v[n] \in SubExpr(e_j)$.*
- *Un arco $e : \langle (s_j, i), (v, i+k) \rangle$ pertenece a E_c si y sólo si $v[n+k|c] \in SubExpr(e_j)$ y $0 \leq i+k \leq N$.*
- *Un arco $e : \langle (s_j, i), (v, i \hat{+}_\eta k) \rangle$ pertenece a E_a si y sólo si $v[A(n)+k|c] \in SubExpr(e_j)$ y $i \hat{+}_\eta k \in \{0..N\}$.*

Los arcos en E_c son arcos concretos y aquellos en E_a son arcos abstractos.

Si $\mathcal{G}_{(\varphi,N+1,\eta)}$ es un grafo de dependencia explícito, diremos que un ciclo en $\mathcal{G}_{(\varphi,N+1,\eta)}$ es un *ciclo concreto* si todos los arcos que lo componen son concretos, y diremos que es un *ciclo abstracto* si contiene al menos un arco abstracto.

Ejemplo 2.17. *Considere la siguiente especificación bien formada sobre la variable independiente t_1:*

$$s_1[n] = \textbf{\textit{if}}\ t_{ret}[n]\ \textbf{\textit{then}}\ t_1[n-1|0]\ \textbf{\textit{else}}\ s_1[A(n)+1|0]$$

En esta especificación, el stream denotado por s_1 tiene, en cada posición de la traza, el valor de τ_1 en la próxima posición de salida de un subprocedimiento invocado desde el contexto actual, o 0 si no hay siguiente retorno. El grafo de dependencia explícito de esta especificación para una traza dada se muestra en la Figura 2.4.

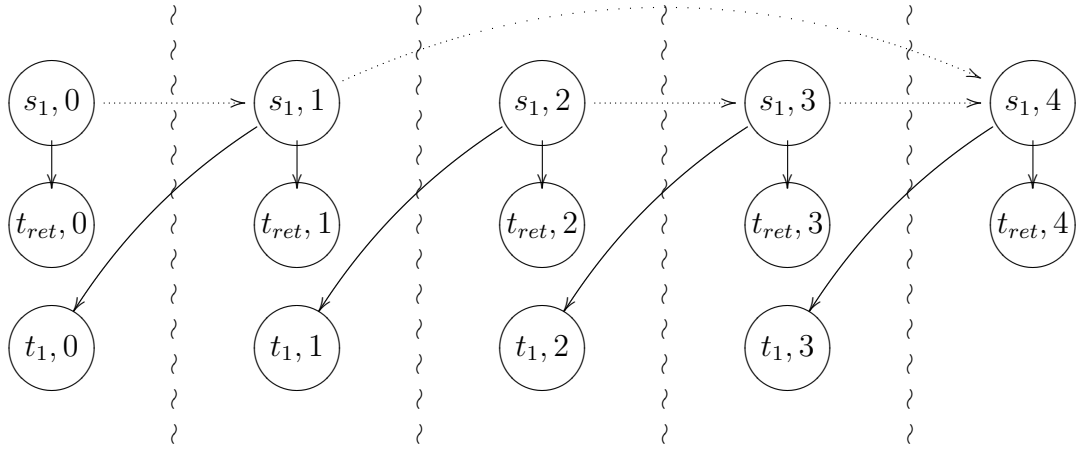

Figura 2.4: Gráfico explícito de la especificación del Ejemplo 2.16, para una traza con longitud 5 y modelo de ejecución inducido $\eta = \{(1, 4)\}$

Lema 2.18. *Sea φ una especificación NSRV con grafo de dependencia $\mathcal{G}_\varphi$. Sea $\tau[N]$ una traza complatible con φ, η el modelo de ejecución de ancho N inducido por τ y $\mathcal{G}_{(\varphi,N,\eta)}$ el correspondiente grafo de dependencia explícito. Si $\mathcal{G}_{(\varphi,N,\eta)}$ tiene un ciclo concreto entonces $\mathcal{G}_\varphi$ tiene un camino concreto cerrado de peso total 0.*

Demostración. Asuma que $\mathcal{G}_{(\varphi,N+1,\eta)}$ tiene un ciclo concreto:

$(s_0, j_0) \to (s_1, j_1) \to ... \to (s_k, j_k) \to (s_0, j_0)$

El correspondiente camino concreto cerrado en $\mathcal{G}_\varphi$ es:

$s_0 \xrightarrow{j_1 - j_0} s_1 \to ... \to s_k \xrightarrow{j_0 - j_k} s_1$

Con un peso total $\sum_{i=0}^{k}(j_{i\oplus 1} - j_i) = 0$, donde $\oplus$ es la suma módulo $k + 1$. □

Lema 2.19. *Sea φ una especificación NSRV unitaria en su forma canónica con grafo de dependencia $\mathcal{G}_\varphi$. Sea $\tau[N]$ una traza compatible con φ, η el modelo de ejecución de ancho N inducido por τ y $\mathcal{G}_{(\varphi,N,\eta)}$ el correspondiente grafo de dependencia explícito. Si $\mathcal{G}_{(\varphi,N,\eta)}$ tiene un ciclo abstracto entonces al menos una de las siguientes proposiciones es verdadera:*

1. *$\mathcal{G}_\varphi$ tiene un camino abstracto cerrado conteniendo arcos abstractos $e_i : \langle s_i, s_{i+1}, k_i\rangle$ y $e_j : \langle s_j, s_{j+1}, k_j\rangle$ tales que $k_i \cdot k_j < 0$.*

2. *$\mathcal{G}_\varphi$ tiene un camino abstracto cerrado con peso concreto K y peso abstracto K' tal que $K \cdot K' < 0$ y $|K'| \leq |K|$.*

Demostración. Sea $\mathcal{G}_{(\varphi,N+1,\eta)} = \langle V, E_c \cup E_a\rangle$ el grafo de dependencia explícito. Asuma que $\mathcal{G}_{(\varphi,N+1,\eta)}$ tiene un ciclo abstracto:

$(s_0, j_0) \xrightarrow{e_0} (s_1, j_1) \xrightarrow{e_1} ... \xrightarrow{e_{k-1}} (s_k, j_k) \xrightarrow{e_k} (s_0, j_0)$

donde $\sum_{i=0}^{k}(j_{i\oplus 1} - j_i) = 0$. Sean $I_c = \{i \mid e_i \in E_c,\ i \leq k\}$ y $I_a = \{i \mid e_i \in E_a,\ i \leq k\}$, los conjuntos de índices de los arcos concretos y abstractos, respectivamente. Entonces $\mathcal{G}_\varphi$ tiene un camino cerrado abstracto γ dado por:

$s_0 \xrightarrow{\overline{j_1 - j_0}} s_1 \to ... \to s_k \xrightarrow{\overline{j_0 - j_k}} s_0$, donde

$$\overline{j_{i\oplus 1} - j_i} = \begin{cases} j_{i\oplus 1} - j_i & \text{si } e_i \text{ es concreto} \\ \frac{j_{i\oplus 1} - j_i}{|j_{i\oplus 1} - j_i|} & \text{si } e_i \text{ es abstracto y } j_{i\oplus 1} - j_i \neq 0 \\ 0 & \text{si } e_i \text{ es abstracto y } j_{i\oplus 1} - j_i = 0 \end{cases}$$

Note que dado que φ es unitaria, $\overline{j_{i\oplus 1} - j_i} \in \{-1, 0, 1\}$, para todo $i \leq k$. Luego, γ tiene peso concreto $K = \sum_{i\in I_c} j_{i\oplus 1} - j_i$ y peso abstracto $K' = \sum_{i\in I_a}(\overline{j_{i\oplus 1} - j_i})$. Si el ciclo contiene arcos abstractos e_i, $e_{i'}$ tales que $(j_{i\oplus 1} - j_i) \cdot (j_{i'\oplus 1} - j_{i'}) < 0$, entonces se satisface $(\overline{j_{i\oplus 1} - j_i}) \cdot (\overline{j_{i'\oplus 1} - j_{i'}}) < 0$ y *(1)* es verdadera. En caso contrario, todos los arcos abstractos de $\mathcal{G}_\varphi$ tienen peso del mismo signo y por hipótesis

$$\sum_{i=1}^{k}(j_{i\oplus 1} - j_i) = \sum_{i\in I_c}(j_{i\oplus 1} - j_i) + \sum_{i\in I_a}(\overline{j_{i\oplus 1} - j_i}) = 0$$

se tiene $|\sum_{i\in I_c}(j_{i\oplus 1} - j_i)| = |\sum_{i\in I_c}(j_{i\oplus 1} - j_i)|$, lo cual implica *(2)*:

$|K'| = |\sum_{i\in I_a}(\overline{j_{i\oplus 1} - j_i})| \leq |\sum_{i\in I_a} j_{i\oplus 1} - j_i| = |\sum_{i\in I_c} j_{i\oplus 1} - j_i| = |K|$

La igualdad se da sólo si $j_{i\oplus 1} - j_i \in \{-1, 0, 1\}$ para todo $i = 0..k$ en el ciclo de $\mathcal{G}_{(\varphi,N+1,\eta)}$. □

Lema 2.20. *Sea φ una especificación NSRV con grafo de dependencia $\mathcal{G}_\varphi$. Sea $\tau[N]$ una traza compatible con φ, η el modelo de ejecución de ancho N*

inducido por τ y $\mathcal{G}_{(\varphi,N,\eta)}$ el correspondiente grafo de dependencia explícito. Si $\mathcal{G}_{(\varphi,N,\eta)}$ no tiene ciclos, entonces φ tiene un único modelo de evaluación para τ.

Demostración. Asuma que $\mathcal{G}_{(\varphi,N,\eta)}$ no tiene ciclos, entonces es un DAG (Grafo Acíclico Dirigido). Definimos un orden topológico $>$ sobre $\mathcal{G}_{(\varphi,N,\eta)}$ tal que $(v_1, j_1) > (v_2, j_2)$ si hay un arco $\langle (v_1, j_1), (v_2, j_2) \rangle$ en $\mathcal{G}_{(\varphi,N,\eta)}$.

Probamos por inducción en este orden que el valor de cada vértice es unívocamente determinado, ya sea porque es obtenido directamente de un stream de entrada o una constante en la especificación, o porque puede ser computado a partir de otros valores conocidos.

Para el caso base, el valor del vértice (v_1, j_1) sin arcos de salida no depende de otros streams: o bien es el valor de un stream de entrada en la posición j, o es un valor constante obtenido de una expresión igual a c, $v_2[n + k|\ c]$ y $j_1 + k \notin\ [0..N]$, o $v_2[A(n) + k|\ c]$ y $j_1 \hat{+}_\eta k \notin\ [0..N]$. En todos los casos el valor es unívocamente determinado.

Para el caso inductivo, el valor de (v_1, j_1) puede ser obtenido unívocamente a partir de los vértices adyacentes. Más aún, por definición de grafo de dependencia explícito, si el valor de (v_1, j_1) depende del valor de (v_2, j_2) existe un arco $\langle (v_1, j_1), (v_2, j_2) \rangle$ en $\mathcal{G}_{(\varphi,N,\eta)}$, y entonces $(v_1, j_1) > (v_2.j_2)$ y por hipótesis inductiva el valor para (v_2, j_2) es unívocamente determinado. □

Teorema 2.21. *Toda especificación φ bien formada está bien definida.*

Demostración. Asuma que φ es una especificación bien formada. Entonces por Lema 2.18 y 2.19, su grafo de dependencia explícito no tiene ciclos para ninguna traza. Luego, por Lema 2.20, para cualquier traza compatible existe un único modelo de evaluación de φ. En consecuencia, φ está bien definida. □

2.4.1. Control de especificaciones bien formadas

Una especificación NSRV φ está bien formada si su grafo de dependencia $\mathcal{G}_\varphi$ satisface las condiciones de la definición 2.14 sobre sus caminos cerrados.

Cada una de las condiciones sobre caminos cerrados puede ser reducida a comprobar propiedades sobre los ciclos y los Subgrafos Fuertementente Conexos Maximales (SFCM) de $\mathcal{G}_\varphi$.

La primera condición es la misma que garantiza que una especificación LOLA está bien formada, y nos basamos en la prueba de ello en [8].

Lema 2.22. *Sea $\mathcal{G} = \langle V, E\rangle$ un grafo. Todo camino $W = v_0...v_n$ de $\mathcal{G}$ puede puede ser descompuesto en ciclos $C_1, ..., C_k$ tales que:*

1. *Cada vértice v_i, $i = 0..n$ y cada arco $\langle v_i, v_{i+1}\rangle$, $i = 0..n-1$, ocurre exactamente en un solo ciclo C_j, $j = 1..k$.*

2. *Para cada par de ciclos C_i, C_j existe una secuencia de ciclos $C_{k_1}, ..., C_{k_m}$ tal que $C_i = C_{k_1}$, $C_j = C_{k_m}$, y C_{k_ℓ} y $C_{k_{\ell+1}}$ comparten exactamente un vértice.*

Demostración. Probamos por inducción completa en la longitud de W que la descomposición siempre puede ser realizada. Si W es un ciclo no hay nada que probar. Caso contrario, sea b el primer vértice de W que ocurre dos veces:

$$W = v_0, ..., v_{i-1}, b, v_{i+1}, ..., v_{j-1}, b, v_{j+1}, ..., v_n$$

Los caminos cerrados $W_1 = v_0$, ..., v_{i-1}, b, $v_{j+1}, ..., v_n$ y $W_2 = b$, $v_{i+1}, ...,$ v_{j-1}, b son estrictamente más cortos que W y por lo tanto la hipótesis inductiva aplica. Sus descomposiciones pueden ser combinadas para obtener una descomposición de W porque son disjuntas y comparten el vértice b. □

Lema 2.23. *Sea $\mathcal{G}_\varphi$ el grafo de dependencia de una especificación φ dada. $\mathcal{G}_\varphi$ tiene un camino concreto cerrado de peso total 0 si y sólo si tiene un vértice v que forma parte de un ciclo concreto con peso total menor o igual que 0 y de un ciclo concreto con peso total mayor o igual que 0 al mismo tiempo.*

Demostración. ($\Rightarrow$) Asuma que v es parte de un ciclo concreto C_1 con peso total $k_1 \geq 0$ y de un ciclo C_2 con peso total $k_2 \leq 0$. Si $k_1 = 0$ o $k_2 = 0$, entonces C_1 o C_2, respectivamente, es un camino concreto cerrado de peso

total 0. Caso contrario, el camino cerrado que consiste en k_1 caminatas de C_2 y $|k_2|$ caminatas de C_1 tiene peso total $k_1k_2 + |k_2|k_1 = 0$. Como ambos ciclos C_1 y C_2 son concretos, el camino también lo es.

($\Leftarrow$) Asuma que $\mathcal{G}_\varphi$ tiene un camino cerrado concreto con peso total 0. Si $\mathcal{G}_\varphi$ tiene un ciclo concreto C de peso total 0 no hay nada que probar, pues C es un ciclo concreto con pesto total $k \geq 0$ y $k \leq 0$ al mismo tiempo, y cualquier vértice en C cumple la condición.

Caso contrario, $\mathcal{G}_\varphi$ no tiene ciclos concretos de peso total 0, y por Lema 2.22 todo camino cerrado concreto $W = v_0, v_1, ..., v_n$ puede ser descompuesto en ciclos $C_1, ..., C_k$ tales que:

1. Cada vértice v_i, $i = 0..n$ y cada arco $\langle v_i, v_{i+1}\rangle$, $i = 0..n-1$, ocurre exactamente en un solo ciclo C_j, $j = 1..k$.

2. Para cada par de ciclos C_i, C_j existe una secuencia de ciclos $C_{k_1}, ..., C_{k_m}$ tal que $C_i = C_{k_1}$, $C_j = C_{k_m}$, y C_{k_ℓ} y $C_{k_{\ell+1}}$ comparten exactamente un vértice.

Para probar que esta descomposición lleva a los resultados esperados supongamos que $\mathcal{G}_\varphi$ no tiene un vértice que forma parte de un ciclo concreto de peso total $k_1 \geq 0$ y de un ciclo concreto con peso total $k_2 \leq 0$ al mismo tiempo. Entonces, por (2) $C_1, ..., C_k$ deben ser todos ciclos con peso mayor o igual que 0 o todos ciclos con peso menor o igual que 0. Sin embargo, por (1), la suma de los pesos de $C_1, ..., C_k$ es igual al peso de un camino cerrado concreto de peso total 0, lo cual es contradice la suposición inicial. □

Teorema 2.24. *Sea φ una especificación NSRV y $\mathcal{G}_\varphi$ su grafo de dependencia. $\mathcal{G}_\varphi$ no tiene caminos cerrados concretos de peso total 0 si y sólo si cada Subgrafo Fuertemente Conexo Maximal (SFCM) de $\mathcal{G}_\varphi$ sólo ciclos concretos de peso total menor o igual que 0, o sólo ciclos concretos de pesto total mayor o igual que 0.*

Demostración. La demostración sigue el Lema 2.23.

($\Rightarrow$) Supongamos que un SFCM G tiene un ciclo concreto C_1 de peso total $k_1 \leq 0$ y un ciclo concreto C_2 con peso total $k_2 \geq 0$ y veamos que entonces $\mathcal{G}_\varphi$ tiene un camino cerrado concreto de peso total 0. Si C_1 y C_2 comparten un vértice, entonces las condiciones del Lema 2.23 aplican directamente. Si no comparten vértice alguno, considere cualquier camino que una C_1 y C_2 y extiéndalo a un ciclo C' uniendo los extremos del mismo. El ciclo C' comparte al menos un vértice con C_1 y al menos un vértice con C_2. Si el peso total de C' es $k' \geq 0$, el Lema 2.23 aplica para C_1 y C'; si $k' \leq 0$, el Lema 2.23 aplica para C_2 y C'.

($\Leftarrow$) Supongamos que $\mathcal{G}_\varphi$ tiene un camino cerrado concreto de peso total 0, lo cual por Lema 2.23 implica que hay un vértice v contenido en un ciclo concreto C_1 con peso total $k_1 \geq 0$ y en un ciclo concreto C_2 con peso total $k_2 \leq 0$ al mismo tiempo. Tanto v como todos los vértices de ambos ciclos pertenecen a un mismo SFCM ya que todos ellos pueden alcanzar todos los otros, por ejemplo siguiendo los ciclos C_1 y C_2. □

Teorema 2.25. *Sea φ una especificación NSRV y $\mathcal{G}_\varphi$ su grafo de dependencia. Las siguientes son equivalentes*

1. *$\mathcal{G}_\varphi$ tiene un camino asbtracto cerrado conteniendo arcos abstractos $e_i = \langle v_i, v_{i'}, k_i\rangle$ y $e_j = \langle v_j, v_{j'}, k_j\rangle$ tales que $k_i.k_j < 0$.*

2. *$\mathcal{G}_\varphi$ tiene arcos abstractos $e_i = \langle v_i, v_{i'}, k_i\rangle$ y $e_j = \langle v_j, v_{j'}, k_j\rangle$ tales que $v_i, v_{i'}, v_j, v_{j'}$ pertencen a un mismo SFCM y $k_i \cdot k_j < 0$.*

Demostración. ($\Rightarrow$) Sea $W = v_0...v_iv_{i'}...v_jv_{j'}...v_n$ el camino abstracto cerrado conteniendo e_i y e_j. Luego, $v_i, v_{i'}, v_j$ y $v_{j'}$ pertencen al mismo SFCM de $\mathcal{G}_\varphi$ pues pueden alcanzarse los unos a los otros, por ejemplo siguiendo W.

($\Leftarrow$) Asumamos que $v_i, v_{i'}, v_j$ y $v_{j'}$ pertencen al mismo SFCM de $\mathcal{G}_\varphi$ y que $\mathcal{G}_\varphi$ contiene los arcos abstractos $e_i = \langle v_i, v_{i'}, k_i\rangle$ y $e_j = \langle v_j, v_{j'}, k_j\rangle$ con $k_i \cdot k_j < 0$. Por pertencer a un mismo SFCM, existe un camino $v_{i'}...v_j$ y un camino $v_{j'}...v_i$, y por lo tanto existe un camino abtracto cerrado $v_{i'}...v_jv_{j'}...v_iv_{i'}$ que contiene arcos e_i y e_j. □

Teorema 2.26. *Sea φ una especificación NSRV y $\mathcal{G}_\varphi$ su grafo de dependencia. Si cada SFCM de $\mathcal{G}_\varphi$ contiene ciclos abstractos con peso abstracto de igual signo, entonces las siguientes son equivalentes:*

1. *$\mathcal{G}_\varphi$ contiene un camino abstracto cerrado de peso concreto K y peso abstracto K' tal que $K.K' < 0$ y $|K'| \leq |K|$.*

2. *$\mathcal{G}_\varphi$ contiene un ciclo abstracto con peso concreto K y peso abstracto K' tal que $K.K' < 0$ y $|K'| \leq |K|$.*

Demostración. ($\Rightarrow$) Asumamos que $W = v_0...v_n$ es un camino abstracto cerrado de peso concreto K y peso abstracto K' tal que $K.K' < 0$ y $|K'| \leq |K|$. Sea $C_1, ..., C_r$ la descomposición de W según el Lema 2.22, y sean K_i y K'_i los pesos concretos y abstractos, respectivamente, del ciclo C_i para $i \leq r$. Luego, $K = K_0 + ... + K_r$ y $K' = K'_0 + ... + K'_r$.

Ahora suponga $K < 0$ y $K' > 0$. Entonces $K'_j > 0$ para todo $j \leq r$ por hipótesis. Además, existe i tal que $K_i < 0$ y $|K'_i| \leq |K_i|$, pues sino $|K| = |\sum_{j \leq r} K_j| \leq |\sum_{j \leq r} K'_j| = |K'|$, contradiciendo la hipótesis sobre el camino W. El razonamiento para $K > 0$ y $K' < 0$ es análogo. Luego, el ciclo C_i satisface la implicación.

($\Leftarrow$) Asuma que C es un ciclo con peso conreto K y peso abstracto K' tal que $K \cdot K' < 0$ y $|K'| \leq |K|$. Entonces C forma un camino abstracto cerrado que satisface las condiciones. □

Luego, dada una especificación φ con grafo de dependencia $\mathcal{G}_\varphi$, el problema de decidir si φ está bien formada se reduce a:

1. Controlar que en cada SFCM de $\mathcal{G}_\varphi$ todos los ciclos concretos tienen peso total mayor o igual que 0 o todos tienen peso total menor o igual que 0.

2. Controlar que ningún SFCM de $\mathcal{G}_\varphi$ contiene arcos abstractos con pesos de distinto signo.

3. Controlar que cada ciclo abstracto en $\mathcal{G}_\varphi$ con peso concreto K y peso abstracto K' satisface que $K \cdot K' \geq 0$ o $|K| < |K'|$.

Es importante destacar que el orden en que se realizan los controles 2 y 3 se debe respetar ya que el Teorema 2.26 aplica sólo si cada SFCM de $\mathcal{G}_\varphi$ contiene ciclos abstractos con peso abstracto de igual signo, lo cual es una implicación directa si $\mathcal{G}_\varphi$ satisface 2.

2.5. Simulación y equivalencia

La simulación de especificaciones captura la idea de que los valores de los streams de salida descriptos por una especificación pueden ser obtenidos por otra, posiblemente utilizando diferentes contrucciones y cantidad de variables dependientes.

Definición 2.27 (Simulación de especificaciones)**.** *Sea φ una especificación NSRV con variables dependientes $s_1, ..., s_r$ y φ' una especificación NSRV con variables dependientes $s'_1, ..., s'_{r'}$. La especificación φ' simula φ si para toda traza $\tau[N+1]$ compatible con φ se cumple que:*

i. Para cada modelo de evaluación $\sigma = \langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ de φ de longitud $N+1$, φ' tiene un modelo de evaluación $\sigma' = \langle \tau_1, ..., \tau_m, \sigma'_1, ..., \sigma'_{r'} \rangle$ de igual longitud.

ii. Existe un mapa $\mu : \{s_1, ..., s_r\} \to \{s'_1, ..., s'_{r'}\}$ tal que para todo modelo de evaluación σ de φ, $j \leq N$ y $i \leq r$:

$$[\![s_i[n]]\!]_\sigma(j) = [\![\mu(s_i)[n]]\!]_{\sigma'}(j)$$

Informalmente, el mapa μ indica qué variable dependiente de φ' simula a una variable dependiente de φ dada.

Definición 2.28 (Equivalencia de especificaciones)**.** *Dos specificaciones NSRV φ y φ' son equivalentes si y sólo si φ simula φ' y viceversa.*

En particular, es sencillo mostrar que si una especificación está bien formada entonces todas sus especificaciones equivalentes también lo están. Además, la equivalencia de especificaciones es una relación de equivalencia.

Una vez definida la noción de simulación de especificaciones se puede ver que toda especificación tiene una forma canónica que la simula. No es cierto, sin embargo, que toda especificación tenga una forma canónica equivalente, puesto que se requieren variables extras para contener las subexpresiones no llanas.

Lema 2.29. *Sea φ una especificación NSRV. Entonces existe una especificación φ' que simula a φ y está en forma canónica.*

Demostración. Si φ está en forma canónica no hay nada que probar. Caso contrario, para cada ecuación $s_i[n] = e_i$ en φ agregamos $s'_i[n] = e_i$ a φ' y asignamos s_i a s'_i. Luego, iterativamente hacemos lo siguiente:

- Si existe una ecuación $s'_i[n] = e_i$ en φ' tal que e_i no es una expressión llana entonces si:
  - $e_i = f(e_1, ..., e_k)$: para cada expresión no llana e_j, $j \leq k$, agregamos $s'_{i_{new}}[n] = e_j$ a φ' y reemplazamos e_j por $s'_{i_{new}}[n]$ en e_i.
  - $e_i =$ **if** e_1 **then** e_2 **else** e_3: para cada expresión no llana e_j, $j \leq 3$, agregamos $s'_{i_{new}}[n] = e_j$ a φ' y reemplazamos e_j por $s'_{i_{new}}[n]$ en e_i.

Se asume que i_{new} es un nuevo índice para variables independientes en φ. Es claro que la iteración termina porque en cada paso al menos una expresión no llana es eliminada. Además, en ambos casos es claro que dada cualquier traza compatible $\tau[N]$, $[\![s_i[n]]\!](j) = [\![s'_i[n]]\!](j)$ para todo $j < N$. □

Definición 2.30 (Equivalencia de expresiones)**.** *Sea φ una especificación NSRV con variables independientes $t_1, ..., t_m$ y variables dependientes $s_1, ..., s_r$. Sea $\tau[N+1]$ una traza compatible con φ. Dos expresiones de streams e y e' sobre las variables $t_1, ..., t_m, s_1, ..., s_r$ son equivalentes en φ, denotado por $e \equiv_\varphi e'$, si para todo modelo de evaluación $\sigma = \langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ de φ*

para τ, $\sigma' = \langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r, \sigma_{r+1}, \sigma_{r+2} \rangle$ *es un modelo de evaluación de* $\varphi \cup \{s_{r+1}[n] = e, s_{r+2}[n] = e'\}$ *para* τ, *y*

$$[\![s_{r+1}[n]]\!](j) = [\![s_{r+2}[n]]\!](j) \text{ para todo } j \leq N.$$

Lema 2.31. *Sea* φ *una especificación definida sobre variables independientes* $t_1, ..., t_m$ *y variables dependientes* $s_1, ..., s_r$, *y sean* e, e' *expresiones de streams sobre las misma variables. Sea* $\tau[N+1]$ *una traza compatible con* φ *y sea* $\sigma = \langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ *un modelo de evaluación de* φ *para* $\tau[N+1]$. *Entonces* $[\![e]\!]_\sigma(j) = [\![e']\!]_\sigma(j)$ *para todo* $j \leq N$ *si y sólo si* $e \equiv_\varphi e'$.

Demostración. Asuma que $[\![e]\!]_\sigma(j) = [\![e']\!]_\sigma(j)$ para todo $j \leq N$. Sea $\varphi' = \varphi \cup \{s_{r+1}[n] = e,\ s_{r+2}[n] = e'\}$. Es claro que si σ es modelo de evaluacón de φ entonces $\sigma' = \langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r, \sigma_{r+1}, \sigma_{r+2} \rangle$ es modelo de evaluación de φ' pues s_{r+1} y s_{r+2} son variables nuevas, ninguna expresión en φ' las utiliza, y por lo tanto no pueden formar ningún camino cerrado en el grafo de dependencia de φ'. Luego, $[\![s_{r+1}[n]]\!](j) = \sigma_{r+1}(j) = [\![e]\!](j) = [\![e']\!](j) = \sigma_{r+2}(j) = [\![s_{r+2}[n]]\!](j)$ para toda $j \leq N$, y entonces e y e' son equivalentes en φ.

Ahora, asuma que $e \equiv_\varphi e'$. Por definición, e y e' son expresiones de streams definidas sobre $n, t_1, ..., t_m, s_1, ..., s_r$, así que ninguna de ellas utiliza s_{r+1} o s_{r+2}. Luego, $\langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r, \sigma_{r+1}, \sigma_{r+2} \rangle$ es un modelo de evaluación de $\varphi \cup \{s_{r+1}[n] = e, s_{r+2}[n] = e'\}$ tal que $\sigma_{r+1}(j) = \sigma_{r+2}(j)$ para toda $j \leq N$. Por definición de modelo de evaluación, $\sigma_{r+1}(j) = [\![e]\!](j)$ y $\sigma_{r+2}(j) = [\![e']\!](j)$. □

Teorema 2.32 (Substitución de Expresiones Equivalentes)**.** *Sea* φ *una especificación NSRV y sean* e, e' *dos expresiones de streams tales que* $e \equiv_\varphi e'$. *Sea* φ' *una especificación idéntica a* φ *salvo que algunas ocurrencias de* e *ha sido remplazadas por* e'. *Entonces* φ *y* φ' *son especificaciones equivalentes.*

Demostración. Sea $\sigma = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ un modelo de evaluación de φ. Luego, por Lema 2.31 tenemos que $[\![e]\!]_\sigma(j) = [\![e']\!]_\sigma(j)$ para toda $j \leq N$. Sea φ' la especificación definida como φ, pero con una ocurrencia de e reemplazada por e'. Luego, como $[\![e]\!]_\sigma(j) = [\![e']\!](j)$, se tiene que cualquier

modelo de evaluación $\langle \tau_1, ..., \tau_m, \sigma_1, ..., \sigma_r \rangle$ para φ es también un modelo de evaluación φ'. Con el mismo razonamiento se puede reemplazar cualquier cantidad de ocurrencias de e en φ por e'. □

Con el concepto de equivalencia de expresiones definido, se muestra a continuación que para toda especificación existe otra que la simula y que además es unitaria.

Lema 2.33. *Si φ es una especificación NSRV, entonces existe una especificación unitaria φ' que simula a φ.*

Demostración. Sea τ una traza estructurada cualquiera. Si φ no tiene modelo de evaluación para τ entonces cualquier especificación simula a φ por definición.

Sea σ un modelo de evaluación de φ para τ y v una variable cualquiera. Primero probamos que la expresión $v[n+k|c]$ con $k > 1$ es equivalente a $s_{(v,k-1)}[n+1|c]$ en $\varphi \cup \{s_{(v,k-1)}[n] = v[n+(k-1)|c]\}$.

Si $j + k > N$, entonces:

- $[\![v[n+k|c]]\!](j) = c$, y

- $$\begin{aligned}[\![s_{(v,k-1)}[n+1|c]]\!](j) &= [\![s_{(v,k-1)}[n]]\!](j+1) \\ &= \sigma_{(v,k-1)}(j+1) \\ &= [\![v[n+(k-1)|c]]\!](j+1) = c,\end{aligned}$$
  pues $(j+1)+(k-1) > N$.

Ahora, si $j + k \leq N$ se tiene que:

- $[\![v[n+k|c]]\!](j) = [\![v]\!](j+k)$

- $$\begin{aligned}[\![s_{(v,k-1)}[n+1|c]]\!](j) &= [\![s_{(v,k-1)}[n+1|c]]\!](j+1) \\ &= \sigma_{(v,k-1)}(j+1) \\ &= [\![v[n+(k-1)|c]]\!](j+1) \\ &= [\![v[n]]\!](j+1+k-1) = [\![v[n]]\!](j+k).\end{aligned}$$

Entonces por el Lema 2.31, $v[n+k|c]$ y $s_{(v,k-1)}[n+1|c]$ son equivalentes en $\varphi \cup \{s_{(v,k-1)}[n] = v[n+(k-1)|c]\}$. La prueba es análoga para el caso $k < 0$ y para los casos de desplazamientos abstractos, donde se utiliza el operador $\hat{+}_\eta$ para el modelo de ejecución inducido por τ.

Si $\varphi = \varphi_k$ una especificación en la cual el mayor desplazamiento usado es k, con $|k| > 1$, por la prueba anterior y el Teorema de Substitución de Expresiones, podemos obterner especificaciones $\varphi_{k-1}, \varphi_{k-2}, ..., \varphi_1$ tales que φ_i simula φ_{i+1} para $1 \leq i < k$. Luego, por transitividad de la simulación de especificaciones, $\varphi' = \varphi_1$ simula a φ y todo desplazamiento en φ' se construye con $k = 1$ o $k = -1$, y por lo tanto es unitaria. □

2.6. Monitoreo en tiempo de ejecución

En esta sección describimos el algoritmo de evaluación de especificaciones NSRV. Sea φ la especificación a ser monitoreada. Como se mencionó anteriormente, asumimos que φ está en su forma canónica. Además, asumimos que φ es una especificación unitaria, por lo que en el resto de la sección todo desplazamiento se asume construído con $k = 1$ o $k = -1$.

El algoritmo de evaluación mantiene tres almacenes:

- Un almacén R de ecuaciones *resueltas*, de la forma $s_j[i] = c$, para una constante c.
- Un almacén P de ecuaciones *pendientes* de la forma $s_j[i] = e_j$, con expresiones que usan posiciones absolutas, algunas posiblemente desconocidas e identificadas con el símbolo ret^ℓ.
- Una secuencia L de números enteros que denotará las posiciones de llamada sin su correspondiente retorno. Trataremos esta secuencia como una pila, pero asumiremos que podemos saber su longitud. Se pueden aplicar sobre L los operadores comunes de pila: $push(L, a)$ para agregar un elemento al final de la secuencia, $top(L)$ para saber el último

elemento de L, si existe, y $pop(L)$ para eliminar el último elemento de L cuando la secuencia no es vacía.

Las ecuaciones en R relacionan variables en una posición específica de la traza con un valor constante que corresponden con valores de variables de entrada que han sido recibido o valores de variables dependientes que han sido computados. Las ecuaciones en P relacionan variables dependientes en una posición específica de la traza con expresiones sobre variables en posiciones posiblemente aún no determinadas, cuyos valores no están diponibles todavía.

Si v_j es una variable, $v_j[ret^\ell]$ denotará el valor de v_j en la posición ret^ℓ, donde ret^ℓ es una referencia que será luego reemplazada por la próxima posición en la cual ocurre un retorno y la longitud de L es ℓ.

Las ecuaciones resueltas han de permanecer en R hasta que no sean necesarias para el cálculo de una variable en alguna posición de la traza. Para cada variable de una especificación definimos dos distancias que representan la cantidad de posiciones, respecto del modelo de ejecución inducido por la traza, que un valor debe permanecer en R.

Definición 2.34 (Distancia de referencia anterior)**.** *Sea φ una especificación NSRV y $\mathcal{G}_\varphi : \langle V, E_c \cup E_a \rangle$ su grafo de dependencia. Para cualquier vértice $v \in V$ la distancia de referencia anterior concreta de v, denotada por $\nabla^c v$, y la distancia de referencia anterior abstracta de v, denotada por $\nabla^a v$, se definen como:*

- $\nabla^c v = max(0, \{k | \langle s, v, -k \rangle\} \in E_c)$
- $\nabla^a v = max(0, \{k | \langle s, v, -k \rangle\} \in E_a)$

Nótese que si una especificación es unitaria, como asumido para φ en esta sección, ninguna distancia de referencia anterior concreta o abstracta es mayor a 1.

Al comienzo, R, P y L están vacíos, y la llegada de los valores de los streams de entrada siete pasos son llevados a cabo en cada instante i, $0 \leq i \leq N$:

Algoritmo 1

1. Si i es una llamada ($t_{call}[i] = \textbf{true}$), i es apilado en L, $push(L, i)$.

2. Las ecuaciones $t_j[i] = c_j$ para $j \in \{call, en, ex, ret, 1, ..., m\}$ son agregadas a R.

3. Para cada ecuación $s_j[n] = e_j$ en φ, una ecuación es agregada a P o R dependiendo de e_j como sigue:

   - $s_j[n] = c$: se agrega $s_j[i] = c$ a R;
   - $s_j[n] = v[n]$: se agrega $s_j[i] = v[i]$ a P;
   - $s_j[n] = f(v_1[n], ..., v_k[n])$: se agrega $s_j[i] = f(v_1[i], ..., v_k[i])$ a P;
   - $s_j[n] = \textbf{if } v_1[n] \textbf{ then } v_2[n] \textbf{ else } v_3[n]$: se agrega $s_j[i] = \textbf{if } v_1[i] \textbf{ then } v_2[i] \textbf{ else } v_3[i]$ a P;
   - $s_j[n] = v[n + k|c]$:
     - Si $i + k < 0$, entonces $s_j[i] = c$ es agregada a R
     - Caso contrario, se agrega $s_j[i] = v[i + k|c]$ a P.
   - $s_j[n] = v[A(n) + k|c]$:
     - $k > 0$:
       (a) si i, es una llamada $s_j[i] = v[ret^{\ell}|c]$ se agrega a P,
       (b) caso contrario, $s_j[i] = v[i + k|c]$ se incluye en P.
     - $k < 0$:
       (a) Si $i + k < 0$, $s_j[i] = c$ es agregada a R,
       (b) si $i + k \geq 0$ y i es un retorno entonces $s_j[i] = v[top(L)]$ se agrega a P. En caso de que i no sea un retorno, $s_j[i] = v[i + k|c]$ es agregada a P.

4. Si ocurre un retorno en i ($\tau_{ret}[i] = \textbf{true}$), entonces toda ocurrencia de ret^{ℓ} es reemplazada por i en las ecuaciones de P. Si i es el último punto

de la traza, toda ecuación $s_j[i'] = v[i''|c]$, con $i < i''$, o $s_j[i'] = v[ret^{\ell}|c]$ es eliminada de P y $s_j[i'] = c$ es agregada a R.

Las ecuaciones en P son simplificadas lo más posible usando las ecuaciones en R. Si una ecuación llega a la forma $s_j[i] = c$, es removida de P y agregada a R. Este paso se realiza hasta que ningún cambio sea realizado en las ecuaciones.

5. La expresiones de disparadores son evaluadas; si alguna de las expresiones fue evaluada a **true**, la violación es reportada.

6. Si i es un retorno L es desapilada, $pop(L)$.

7. Las ecuaciones en R que ya no son necesarias se eliminan de R. Una ecuación $v[i'] = c$, con $i' \leq i$, es eliminada si $i' + \nabla^c v \leq i$ (ya no es necesaria por un desplzamiento negativo sobre el camino concreto de la traza) y:

   - $\nabla^a v = 0$, o
   - $\nabla^a v = 1$, $i' + \nabla^a v \leq i$ y $i' \notin L$

Note que de acuerdo con el paso (7) del Algoritmo 1, si η es el modelo de ejecución inducido por la traza, el segundo caso es equivalente a requerir que $i' \hat{+}_{\eta} \nabla^a v \leq i$ sea verdadero en cada posición i (es decir, el valor de la variable v es usada por un desplazamiento negativo sobre el camino abstracto pero ya no es necesario).

Ejemplo 2.35. *Considere la siguiente especificación NSRV:*

$s_1[n] = t_{call}[n] \rightarrow s_2[A(n) + 1|true]$

$s_2[n] = t_1[A(n) - 1|0] == t_1[n]$

$\boldsymbol{trigger}(\neg s_1[n])$

la cual controla que el valor del stream de entrada τ_1 es exactamente el mismo en la posición de llamada y de retorno de cualquier módulo. Si τ_1 representa el valor de un semáforo, por ejemplo, esta especificación controlaría que cada función o procedimiento deja el semáforo en un estado consistente. Para el stream de entrada $\tau_1 : \langle 0, 1, 1, 2, 1\rangle$ y modelo de ejecución $\eta = \{(1, 4)\}$, las ecuaciones de los almacenes R y P, y el estado de la pila representada por L al completarse el paso (7) del algoritmo se muestran en la Figura 2.5. Además, se incluye el estado del almacen R al final del paso (4) para hacer más explícito la forma en la que se resuelven la ecuaciones. Note que la referencia pasada abstracta de t_1 es 1, y entonces la ecuación resuelta en la llamada de la posición 1 no puede ser eliminada hasta que el correspondiente retorno suceda.

Teorema 2.36 (Correción del Algoritmo de Evaluación)**.** *Sea φ una especificación NSRV unitaria que está en su forma canónica, $\tau[N] = \langle \tau_1, ..., \tau_m\rangle$ una traza compatible con φ y η el modelo de ejecución inducido por τ. Si φ está bien definida entonces el Algoritmo 1 computa el único modelo de evaluación de φ para τ. Al final de la traza el valor apropiado para cada stream de salida ha sido calculado y los almacenes P y L están vacíos.*

Demostración. Asuma que φ está bien definido. Sea $\mathcal{G}_{(\varphi,N,\eta)} = \langle V, E_c \cup E_a\rangle$ el grafo de dependencia para φ, cualquier traza estructurada τ de longitud N y modelo de ejecución inducido η. Aún cuando la especificación no está bien formada, $\mathcal{G}_{(\varphi,N,\eta)}$ no tiene ciclos y entonces por el Lema 2.20, cada vértice de $\mathcal{G}_{(\varphi,N,\eta)}$ puede ser asignado al valor correspondiente en el único modelo de evaluación. Ahora probamos, por inducción en $\mathcal{G}_{(\varphi,N,\eta)}$, que al final de la traza cada valor ha estado disponible en alguna posición $j \leq N$ durante la ejecución del algoritmo.

Para el caso base, todas los vértices que son hojas (v, j) corresponden o bien a un valor de un stream de entrada o bien a valores de ecuaciones de la forma $v[n] = c$, o $v[n] = v'[n + k|c]$ y $j + k \notin [0..N]$, o $v[n] = v'[A(n) + k|c]$ y $j \hat{+}_\tau k \notin [0..N]$. En cualquier caso, el valor es unívocamente determinado y agregado a las ecuaciones de R.

j	$R^{(4)}$	$U^{(7)}$	$R^{(7)}$	$L^{(7)}$
0	$t_{call}[0] = false$ $t_{en}[0] = false$ $t_{ex}[0] = false$ $t_{ret}[0] = false$ $t_1[0] = 0$ $s_1[0] = true$ $s_2[0] = true$	$\emptyset$	$t_1[0] = 0$	$\langle\rangle$
1	$t_{call}[1] = true$ $t_{en}[1] = false$ $t_{ex}[1] = false$ $t_{ret}[1] = false$ $t_1[1] = 1$ $s_2[1] = false$	$s_1[1] = s_2[ret^0]$	$t_1[1] = 1$	$\langle 1\rangle$
2	$t_{call}[2] = false$ $t_{en}[2] = true$ $t_{ex}[2] = false$ $t_{ret}[2] = false$ $t_1[2] = 1$ $s_1[2] = true$ $s_2[2] = true$	$s_1[1] = s_2[ret^0]$	$t_1[1] = 1$ $t_1[2] = 1$	$\langle 1\rangle$
3	$t_{call}[3] = false$ $t_{en}[3] = false$ $t_{ex}[3] = true$ $t_{ret}[3] = false$ $t_1[3] = 2$ $s_1[3] = true$ $s_2[3] = false$	$s_1[1] = s_2[ret^0]$	$t_1[1] = 1$	$\langle 1\rangle$
4	$t_{call}[4] = false$ $t_{en}[4] = false$ $t_{ex}[4] = false$ $t_{ret}[4] = true$ $t_1[4] = 1$ $s_1[4] = true$ $s_2[4] = true$ $s_1[1] = true$	$\emptyset$	$t_1[4] = 1$	$\langle\rangle$

Figura 2.5: Estado del Algoritmo 1 en cada posición en diferentes pasos para la especificación del Ejemplo 2.35

Para el caso inductivo, el valor del vértice (v_1, j_1) es unívocamente computado de valores de vértices (v_2, j_2) tal que $\langle (v_1, j_1), (v_2, j_2) \rangle \in E_c \cup E_a$ y, entonces, por hipótesis inductiva, el valor de (v_2, j_2) fue unívocamente computado u obtenido y estuvo en alguna posición disponible en R. Ahora probamos que estos valores estuvieron disponibles en R para ser subtituídos mientras $v_1[j_1] = e_1$ está en P. Distinguimos 3 casos:

1. Si $j_1 = j_2$ entonces $v_1[j_1] = e_1$ y $v_2[j_2] = e_2$ son agregados a P (o R) al mismo tiempo en el paso 2 o 3. Si $v_2[j_2] = e_2$ fue agregado a R, el valor de $v_2[j_2]$ es reemplazado en e_1 en el paso 4. Si $v_2[j_2] = e_2$ fue agregado a P, por hipótesis inductiva, está disponible en alguna posición luego en la computación. Entonces debe ser movido a R en el paso 4, y luego es reemplazado en e_1 en el mismo paso.

2. Si $j_1 < j_2$ entonces, si fue agregado a U, $v_1[j_1] = e_1$ es agregado antes que $v_2[j_2] = e_2$ sea agregado a P (o R). Nuevamente, por hipótesis inductiva, el valor de $v_2[j_2]$ está disponible en alguna posición luego en la computación. Por lo tanto, es movido a R en el paso 4, y reemplazado en e_1 en el mismo paso.

3. En el último caso, $j_1 > j_2$. Por hipótesis inductiva, $v_2[j_2]$ es resuelto en alguna posición $j_{res} \leq N$. Si $j_{res} \geq j_1$, $v_2[j_2]$ es reemplazado en e_1 en la misma posición que es resuelto, en el paso 4 ya que $v_1[j_1] = e_1$ todavía está en P en esa posición. Si $j_{res} < j_1$ tenemos que asegurar que $v_2[j_2]$ estará en R en la posición j_1. Sea j_{rem} la posición en la que $v_2[j_2]$ es removido de R, el cual es:

$$j_{rem} = \begin{cases} max(j_2 + \nabla^c v_2,\, j_2 \hat{+}_\tau \nabla^a v_2) & \text{si } max(j_2 + \nabla^c v_2,\, j_2 \hat{+}_\tau \nabla^a v_2) > j_{res} \\ j_{res} & \text{caso contrario} \end{cases}$$

Luego $j_{rem} \geq j_{res}$ y probamos que $j_{rem} \geq j_1$. Sea $e_1 = v_2[n + k|c]$ o $e_1 = v_2[A(n) + k|c]$, para $k < 0$. Dado que φ es unitaria, tenemos que $\nabla^c v_2, \nabla^a v_2 \leq 1$. Si $j_{rem} = j_{res}$ entonces v_2 no es requerido por ningu-

na variable mediante un desplazamiento negativo en la especificación, incluyendo v, y por lo tanto puede ser removido en la misma iteración en la que su valor es resuelto. En cualquier otro caso, tenemos dos posibilidades:

- Si $j_{rem} = j_2 + \nabla^c v_2$, entonces $j_1 = j_2 + 1$ y $j_{rem} = j_2 + 1 = j_1$. Por lo tanto $j_{rem} = j_1$.
- Si $j_{rem} = j_2 \hat{+}_\tau \nabla^a v_2$, entonces $j_1 = j_2 \hat{+}_\tau 1$:

$$j_{rem} = j_2 \hat{+}_\tau 1 = j_1$$

Note que en este caso $j_2 \hat{+}_\tau 1$ puede ser mucho mayor que $j_2 + 1$.

Entonces el valor para $v_2[j_2]$ todavía se encuentra en R cuando $v_1[j_1] = e_1$ es agregado a P en el paso 3, y es reemplazado en e_1 en el paso 4.

Ahora, como la posición N es alcanzada en algún momento y por definición de traza estructurada cada retorno tuvo su llamada que le corresponde y viceversa al alcanzarse N, la pila L está vacía cuando el algoritmo termina y, por el razonamiento previo, P está vacío. □

2.6.1. Especificaciones eficientemente monitoreables

La monitorización en línea debe hacerse de manera eficiente para no afectar la ejecución del programa observado en el caso de recursos compartidos, para poder responder a tiempo a posibles violaciones de la especificación. Además, es importante poder garantizar que los recursos necesarios para la monitorización estarán disponibles a lo largo de la ejecución.

En el Algoritmo 1 para la evaluación de especificaciones NSRV, las ecuaciones con dependencias de valores futuros deben ser almacenadas hasta que los correspondientes valores son resueltos. En algunos casos, tales ecuaciones deben ser almacenadas una cantidad de posiciones indeterminada y por lo tanto resulta imposible dar una cota de los requerimientos de memoria del algoritmo en principio. Presentamos en esta sección la clase de especificaciones para las cuales es posible definir tal cota.

Definición 2.37 (Especificación Eficientemente Monitorable)**.** *Un especificación NSRV es eficientemente monitoreable si el peor caso de requerimiento de memoria es lineal en el máximo nivel de anidamiento de llamadas en la traza e independiente de su longitud.*

Definición 2.38 (Especificación Acotada en el Futuro)**.** *Una especificación NSRV φ es acotada en el futuro si y sólo si su grafo de dependencia $\mathcal{G}_\varphi$:*

- *No tiene ciclos de peso abstracto o concreto positivo.*
- *No tiene ciclos de peso total negativo tal que alguno de los vértices del ciclo sea origen de un arco abstracto y positivo.*

Ejemplo 2.39. *Considere la siguiente especificación φ sobre las variables dependientes de control y t_1 :*

$$s_1[n] = s_2[n-1 \mid 0]$$

$$s_2[n] = s_1[n] + t_1[A(n)+1 \mid 1]$$

Para la traza estructurada $\tau = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1 \rangle$ con modelo de ejecución inducido $\eta = \{(0,5)\}$ y tal que $\tau_1 = \langle 1,1,1,1,1,1 \rangle$, el único modelo evaluación de φ para τ es:

	0	1	2	3	4	5
τ_{call}	$true$	$false$	$false$	$false$	$false$	$false$
τ_{en}	$false$	$true$	$false$	$false$	$false$	$false$
τ_{ex}	$false$	$false$	$false$	$false$	$true$	$false$
τ_{ret}	$false$	$false$	$false$	$false$	$false$	$true$
τ_1	1	1	1	1	1	1
σ_1	0	1	2	3	4	5
σ_2	1	2	3	4	5	6

Dado el Algoritmo 1, la ecuación $s_2[0] = 0 + t_1[ret^0|1]$ *se mantiene en* P *sin ser resuelta hasta que el correspondiente retorno sucede, en la posición* 5. *De esta forma, todas las ecuaciones* $s_1[j] = s_2[j-1]$ *para* $j \in [1,5]$ *se mantienen sin resolver hasta que la posición* 5. *En consecuencia, las ecuaciones* $s_2[j] = s_1[j] + t_1[j+1|1]$ *para* $j \in [1,4]$ *también deben mantenerse en memoria a la espera de que el valor de* $t_1[ret^0|1]$ *sea resuelto, suficiente para resolver el resto de las ecuaciones pendientes. Por lo tanto, los requerimientos de memoria del Algoritmo 1 para esta especificación dependen del modelo de ejecución y la longitud de la traza, y por ende no es eficientemente monitoreable.*

Definición 2.40 (Visibilidad y Alcance)**.** *Sea* φ *una especificación NSRV bien formada y* $\mathcal{G}_{(N,\varphi,\eta)} = \langle V, E_c \cup E_a \rangle$ *su grafo de dependencia explícito para una longitud de traza* N *y modelo de ejecución inducido* η. *La* visibilidad concreta *de un vértice* (s,j), *denotado por* $\mathcal{V}_c(s,j)$, *es el conjunto de nodos alcanzables desde* (s,j) *a través de arcos concretos. La* visibilidad abstracta *de* (s,j), *denotado por* $\mathcal{V}_a(s,j)$, *es el conjunto de nodos alcanzables desde* (s,j) *a través de, al menos, un arco abstracto:*

- $\mathcal{V}_c(s,j) = \{(v,j') \mid \langle (s,j),(v,j') \rangle \in E_c^*\}$
- $\mathcal{V}_a(s,j) = \{(v,j') \mid \langle (s,j),(v,j') \rangle \in (E_c \cup E_a)^*\ E_a\ (E_c \cup E_a)^*\}$

donde E^* *es la clausura transitiva del conjunto* E.
Sean $Max, Min : \mathcal{P}(V) \to [0..N]$ *los operadores definidos como:*

$$Min(X) = \begin{cases} min\{k|\ (v,k) \in X\} & si\ min\{k|\ (v,k) \in X\} \geq 0 \\ 0 & caso\ contrario. \end{cases}$$

$$Max(X) = \begin{cases} max\{k|\ (v,k) \in X\} & si\ max\{k|\ (v,k) \in X\} \leq N \\ N & caso\ contrario. \end{cases}$$

El alcance concreto *de un vértice* (s,j), *denotado* $\mathcal{A}_c(s,j)$, *y el* alcance abstracto, *denotado* $\mathcal{A}_a(s,j)$, *denotan la posición más alejada de un vértice en la visibilidad concreta y abstracta de* (s,j), *respectivamente:*

- $\mathcal{A}_c(s,j) = max(Min(\mathcal{V}_c(s,j)), Max(\mathcal{V}_c(s,j)))$
- $\mathcal{A}_a(s,j) = max(Min(\mathcal{V}_a(s,j)), Max(\mathcal{V}_a(s,j)))$

Definición 2.41 (Distancia de Anticipación)**.** *Sea φ una especificación NSRV acotada en el futuro con grafo de dependencia $\mathcal{G}_\varphi = \langle V, E_c \cup E_a \rangle$. La distancia de anticipación concreta de un vértice v, denotada $\Delta^c v$, es el máximo peso de un camino concreto en $\mathcal{G}_\varphi$ que empieza en v, o cero si el peso máximo es negativo. La distancia de anticipación abstracta, denotada $\Delta^a v$, es el máximo peso de un camino abstracto en $\mathcal{G}_\varphi$ que empieza en v, o cero si el máximo peso es negativo.*

Note que las distancias de anticipación están bien definidas únicamente en ausencia de ciclos de peso positivo. La distancia de anticipación concreta de un vértice puede ser determinada usando una versión modificada del algoritmo Bellman-Ford [6] para obtener el camino más corto en un grafo dirigido ponderado invirtiendo los signos de los pesos de todos los arcos en el grafo de dependencia e ignorando los arcos en E_a. De igual manera, eliminando primero todos los arcos de ciclos concretos para un vértice v y los arcos de caminos maximales concretos a partir de v, se puede usar el algoritmo para encontrar la distancia de anticipación abstracta.

Lema 2.42. *Sea φ una especificación NSRV acotada en el futuro con grafo de dependencia $\mathcal{G}_\varphi = \langle V, E_c \cup E_a \rangle$ y grafo de dependencia explícito $\mathcal{G}_{(N,\varphi,\eta)} = \langle V', E'_c \cup E'_a \rangle$ para una longitud de traza N y modelo de ejecución η de ancho N. Entonces para cada vértice $(s,j) \in V'$ el alcance concreto (abstracto) es menor o igual que j mas la distancia de anticipación concreta (abstracta) de su vértice correspondiente en $\mathcal{G}_\varphi$:*

$$\mathcal{A}_c(s,j) \leq j + \Delta^c s$$
$$\mathcal{A}_a(s,j) \leq j \hat{+}_\eta \Delta^a s$$

Demostración. Primero considere el vértice arbitrario $(s,j) \in V'$ con rango concreto $\mathcal{A}_c(s,j) = r$. Entonces existe una secuencia de vértices del grafo $(s_1,j_1),...,(s_n,j_n)$ tal que $(s,j) = (s_1,j_1)$, $\langle (s_i,j_i),(s_{i+1},j_{i+1}) \rangle \in E'_c$ y

$j_n = r$. El camino de los correspondientes vértices $s_1, ..., s_n$ en $\langle V, E_c \rangle$, con $\langle s_i, s_{i+1}, j_{i+1} - j_i \rangle \in E_c$, tiene peso

$$\sum_{i=1}^{n} j_{i+1} - j_i = j_n - j_1$$

y entonces. por definición de distancia de anticipación concreta,

$$\mathcal{A}_c(s, j) = j_n \leq j_1 + (j_n - j_1) = j + \Delta^c s.$$

Ahora considere un vértice arbitrario $(s, j) \in V'$ con rango abstracto $\mathcal{A}_a(s, j) = r$. Entonces existe una secuencia de vértices $(s_1, j_1), ..., (s_n, j_n)$ tal que $(s, j) = (s_1, j_1)$, $\langle (s_i, j_i), (s_{i+1}, j_{i+1}) \rangle \in (E'_c \cup E'_a)$ y $j_n = r$. Luego, existen $k_1, ..., k_n$ tales que $j_i \hat{+}_\eta k_i = j_{i+1}$ si $\langle (s_i, j_i), (s_{i+1}, j_{i+1}) \rangle \in E_a$ y $j_i + k_i = j_{i+1}$ si $\langle (s_i, j_i), (s_{i+1}, j_{i+1}) \rangle \in E_c$. El camino correspondiente a los vértices $s_1, ..., s_n$ en $\mathcal{G}_\varphi$ está dada por $\langle s_i, s_{i+1}, k_i \rangle \in (E_c \cup E_a)$ y tiene peso abstracto igual a $\sum_{i=1}^{n} k_i = k$. Por lo tanto, por definición de distancia de anticipación abstracta y $\hat{+}_\eta$,

$$\mathcal{A}_c(s, j) = j_n \leq j_1 \hat{+}_\eta k = j \hat{+}_\eta \Delta^a s.$$

□

Teorema 2.43 (Requerimientos de Memoria)**.** *Sea $\varphi(t_1, ..., t_m, s_1, ..., s_n)$ una especificación NSRV acotada en el futuro. Cuando se ejecuta el Algoritmo 1 sobre φ para una traza cuyo máximo nivel de anidamiento de llamadas es L, el máximo número de ecuaciones almacenadas en P y R juntos en cualquier instante está acotado por:*

$$M_{max} = \sum_{i=1}^{m} (\nabla^c t_i \; + \; L \cdot \nabla^a t_i) \; + \sum_{i=1}^{n} [\Delta^c s_i \; + \nabla^c s_i \; + \; L \cdot (\Delta^a s_i + \nabla^a s_i)] \\ + \; 3n \; + \; m$$

Demostración. De la descripción del Algoritmo 1 y el Lema 2.42 puede deducirse que el máximo número de ecuaciones en P es menor o igual a

$$\sum_{i=1}^{n} (\Delta^c s_i \; + \; L \cdot \Delta^a s_i) \; + \; n$$

donde el último término refleja el hecho de que todas las ecuaciones para las variables dependientes son primero almacenadas en P en el paso (3) y sólo se mueven a R luego de la simplificación del paso (4). El factor de multiplicación L refleja el hecho de que las ecuaciones para variables dependientes, en una posición de llamada, con distancia de anticipación no nula serán no serán resueltas hasta que el retorno correspondiente a la llamada ocurra.

El máximo número de ecuaciones almacenadas en R en cualquier instante de tiempo es menor o igual a

$$\sum_{i=1}^{m}(\nabla^c t_i + L \cdot \nabla^a t_i) + \sum_{i=1}^{n}(\nabla^c s_i + L \cdot \nabla^a s_i) + n + \sum_{i=1}^{n}(\Delta^c s_i + L \cdot \Delta^a s_i) + n + m$$

donde nuevamente los últimos dos términos reflejan que todas las ecuaciones son agregadas a R en el paso (4) antes de ser eliminadas en el paso (6) (posiblemente en la misma ejecución del bucle). El tercer y cuarto término son incluídos para cubrir la posibilidad de que todas las ecuaciones en P se simplifiquen por completo en el mismo instante y entonces sean todas agregadas a R. El factor de multiplicación L en la primera y segunda sumatoria tiene una justificación similar a la anterior: los valores de las variables en posiciones de llamadas y referenciadas con desplazamientos negativos sobre el camino abstracto deberán ser alacenadas hasta que el correspondiente retorno ocurra y se utilicen en las ecuaciones que los requieren. El cuarto y quinto término son incluídos para cubrir la posibilidad de que todas las ecuaciones en P se simplifiquen por completo en el mismo instante y entonces sean todas agregadas a R. □

Capítulo 3

Control de especificaciones mal formadas

Como se ha visto previamente en la Sección 2.4, toda especificación bien formada tiene exactamente un modelo de evaluación para cualquier traza dada, es decir, está bien definida. Pero, ¿puede una especificación que no está bien formada estar bien definida de todas maneras?. Llamamos a este problema de decisión el *problema de la buena definición* y en esta sección presentamos diversos resultados sobre el mismo. Primero, probamos que el problema es indecidible para especificaciones que utilizan variables enteras sin restricción alguna. Luego mostramos que el problema se torna decidible cuando es acotado a especificaciones booleanas, y finalmente definimos ciertas restricciones sobre especificiaciones que utilizan variables enteras y booleanas, de tal manera que cierta dependencia entre variables de diferentes tipos existe y el problema se mantiene decidible.

3.1. Especificaciones sin restricciones

El Post Correspondence Problem [16] [20] es un problema de decición famoso e indecidible más conveniente que el *problema de parada* [18] para

algunas pruebas de indecibilidad. La entrada del problema PCP consiste en dos listas finitas $\alpha_0, ..., \alpha_{k-1}$ y $\beta_0, ..., \beta_{k-1}$ de palabras sobre un alfabeto A con al menos 2 símbolos. Una solución al problema es una secuencia de índices $\{i_j\}_{1 \leq j \leq l}$ con $l \geq 1$ y $0 \leq i_j \leq k-1$ para toda l, tales que $\alpha_{i_0}...\alpha_{i_l} = \beta_{i_0}...\beta_{i_l}$. El problema consiste en decidir si tal secuencia de índices existe o no.

Proposición 3.1. *Dada una especificacón NSRV φ que no está bien formada, el problema de decidir si φ está bien definida es indecidible.*

Para ver que la proposición es cierta construiremos una especificación a partir de una instancia del *Post Correspondence Problem*. Sean $\alpha_0, ..., \alpha_{k-1}$ y $\beta_0, ..., \beta_{k-1}$ palabras sobre un alfabeto $A = \{0, 1\}$ de una instancia arbitraria. Consideraremos a las palabras sobre este alfabeto como la representación binaria de un número entero. Dado que en la comparación de números enteros los ceros a la izquierda no son tenidos en cuenta a la hora de comparar números enteros, necesitaremos recordar también la longitud de una palabra sobre este alfabeto. Sea $\Sigma = A^*$ nuestro lenguaje y $|\omega|$ la longitud de una palabra $\omega \in \Sigma$. Usaremos los α_i y β_i como enteros en una especificación y a su vez $|\alpha_i|$ y $|\beta_i|$ denotará la longitud de α_i o β_i como cadena para simplificar la construcción. Definiremos dos variables dependientes s_α y s_β para representar las palabras formadas por la concatenación de los α_i y los β_i, y a su vez usaremos otras dos variables dependientes $s_{|\alpha|}$ y $s_{|\beta|}$ para representar la longitud de tales concatenaciones.

La especificación a considerar tendrá una sola variable dependiente entera y elige, a partir de ella, un índice entre 0 y $k-1$ de los elementos α_i, β_i a concatenar en s_α y s_β. De esta manera, si hay una secuencia de índices que cumplen la condición del PCP para las familias dadas, esa secuencia corresponde con una entrada que hace $s_{|\alpha|}[n] = s_{|\beta|}[n]$ y $s_{|\alpha|}[n] = s_{|\beta|}[n]$. En caso de que tal secuencia no exista, ninguna entrada hará que las igualdades se den.

Para una explicación más sencilla eliminamos los modificadores en expresiones de desplazamientos sobre streams y asumimos por convención que son todos desplazamientos concretos. Además, utilizaremos las siguientes macros:

- $s_{idx}[n] = t_1[n] \texttt{ mod } k$
- $Concat(v, w)[n] = v[n-1|0] * 2^{|w|} + w$

Nótese que el valor de $s_{idx}[n]$ en cualquier es un valor entero en el conjunto $\{0..k-1\}$. Definimos las variables dependientes:

$$
\begin{aligned}
s_\alpha[n] = \; & \textbf{if } s_{idx}[n] = 0 \textbf{ then } Concat(s_\alpha, \alpha_0)[n] \textbf{ else} \\
& \textbf{if } s_{idx}[n] = 1 \textbf{ then } Concat(s_\alpha, \alpha_1)[n] \textbf{ else} \\
& \quad . \\
& \quad . \\
& \quad . \\
& \textbf{if } s_{idx}[n] = k-1 \textbf{ then } Concat(s_\alpha, \alpha_{k-1})[n] \textbf{ else} \\
& s_\alpha[n-1|0]
\end{aligned}
$$

$$
\begin{aligned}
s_{|\alpha|}[n] = \; & s_{|\alpha|}[n-1|0] + \\
& \textbf{if } s_{idx}[n] = 0 \textbf{ then } |\alpha_0| \textbf{ else} \\
& \textbf{if } s_{idx}[n] = 1 \textbf{ then } |\alpha_1| \textbf{ else} \\
& \quad . \\
& \quad . \\
& \quad . \\
& \textbf{if } s_{idx}[n] = k-1 \textbf{ then } |\alpha_{k-1}| \textbf{ else } 0
\end{aligned}
$$

La variable dependiente s_α codifica exactamente la concatenación de las palabras $\alpha_{s_{idx}[0]}...\alpha_{s_{idx}[j]}$ según los índices vistos hasta el momento. Dado que los prefijos de 0s que las palabras α_i puedan contener son omitidos es necesario también saber la longitud de la concatención, valor que es acumulado en $s_{|\alpha|}[j]$.

Las ecuaciones para s_β y $s_{|\beta|}$ son análogas a las anteriores, usando las palabras $\beta_0, ..., \beta_{k-1}$. Finalmente, definimos

$$
\begin{aligned}
s_{bad}[n] = \; & \textbf{if } (s_\alpha[n] = s_\beta[n] \wedge s_{|\alpha|}[n] = s_{|\beta|}[n]) \\
& \textbf{then } s_{bad}[n] \\
& \textbf{else } s_{bad}[n-1|true]
\end{aligned}
$$

Luego, la cláusula $(s_\alpha[n] = s_\beta[n] \wedge s_{|\alpha|}[n] = s_{|\beta|}[n])$ es verdadera para alguna posición arbitraria de una traza si y sólo si la especificación no está bien definida.

A partir de esta construcción se puede demostrar el siguiente Lema para la especificación φ previamente descrita.

Lema 3.2. *Sea $\tau = \langle \tau_1 \rangle$ una traza estructurada de longitud $N+1$ y ancho 1 tal que $\tau_1(j)$ `mod` $k = i_j$, para todo $0 \leq j \leq N$, y sea $\tau = \langle \tau_1, \sigma_{|\alpha|}, \sigma_\alpha, \sigma_{|\beta|}, \sigma_\beta, \sigma_{idx}, \sigma_{bd} \rangle$ un modelo de evaluación de φ para τ. Entonces lo siguiente es verdadero para todo $j \leq N$:*

1. $\sigma_{|\alpha|}(j) = |\alpha_{i_0}...\alpha_{i_j}|$
2. $\sigma_\alpha(j) = \mathit{StringToInt}(\alpha_{i_0}...\alpha_{i_j})$

donde $\mathit{StringToInt}(\alpha_{i_1}...\alpha_{i_j})$ es la palabra $\alpha_{i_1}...\alpha_{i_j}$ convertida a un número entero, y por ende el máximo prefijo de 0's eliminado.

Demostración. De la especificación descripta no es difícil ver que:

$\sigma_{|\alpha|}(j) = \sum_{l=0}^{j} |\alpha_{i_l}| = |\alpha_{i_0}| + ... + |\alpha_{i_j}| = |\alpha_{i_1}...\alpha_{i_j}|$

Luego, (*1*) se cumple. Para probar (*2*) utilizamos inducción en j. Para $j = 0$ tenemos que $\sigma_\alpha(j) = 0 * 2^{|\alpha_{i_0}|} + \alpha_{i_0} = \alpha_{i_0}$. Como σ_α es un stream entero, su valor tiene una *expansión mínima* de su valor númerico. En otras palabras, es la representación binaria de su valor. Ahora, para $j \geq 1$ tenemos que $\sigma_\alpha(j) = \sigma_\alpha(j-1) * 2^{|\alpha_{i_j}|} + \alpha_{i_j}$. Dado que la multiplicación de un entero m por 2^r es exactamente un desplazamiento a la izquierda de r bits, estamos seguros de que los $|\alpha_{i_j}|$ bits menos significativos de $\sigma_\alpha(j)$ son 0. Más aún, el número α_{i_j} y el resultado de tal multiplicación no tienen bits en 1 que ocurran en las misma posiciones, y por lo tanto la suma de ambos tiene exactamente a α_{i_j} ocupando los $|\alpha_{i_j}|$ bits menos significativos. Por hipótesis inductiva, $\sigma_\alpha(j-1) = \mathit{StringToInt}(\alpha_{i_0}...\alpha_{i_{j-1}})$. Luego, $\sigma_\alpha(j) = \mathit{StringToInt}(\alpha_{i_0}...\alpha_{i_j})$ para todo $0 \leq j \leq N$. □

El Lema es análogo para σ_β y $s_{|\beta|}$.

Un vez demostrado el Lema, probamos la equivalencia entre el problema de decidir si la especificación φ construída de tal manera está bien definida con la instancia del PCP dada.

Primero asumamos que la secuencia $i_0, ..., i_N$ es una solución a la instancia del problema y que ningún prefijo estricto de la secuencia es solución también. Entonces existen números $j_0, ..., j_N$ tales que $j_l \texttt{ mod } k = i_l$. Si la especificación está bien definida, tiene exactamente un modelo de evaluación para cualquier traza estructurada $\tau = \langle \tau_1 \rangle$ de longitud N que satisface $\tau_1(l) = j_l$, para $l \leq N$. Sabemos que si $i_0, ..., i_N$ es una solución entonces $\alpha_{i_0}...\alpha_{i_l}$ es un prefijo de $\beta_{i_0}...\beta_{i_l}$ o viceversa, para todo $l \leq N$, y además $\alpha_{i_0}...\alpha_{i_N} = \beta_{i_0}...\beta_{i_N}$. Caso contrario, algún prefijo de las palabras sería diferente y llegaríamos a una contradicción. Por el Lema probado, se tiene en la especificación que:

- si $\sigma_{|\alpha|}(l) > \sigma_{|\beta|}(l)$, entonces $\sigma_\alpha(l)$ es un prefijo de $\sigma_\beta(l)$, sino es una contradicción puesto que un prefijo de ambas concatenaciones no coincide e $i_0, ..., i_N$ no sería solución.

- si $\sigma_{|\alpha|}(l) < \sigma_{|\beta|}(l)$, entonces $\sigma_\beta(l)$ es un prefijo de $\sigma_\alpha(l)$, por el mismo argumento.

- si $\sigma_{|\alpha|}(l) = \sigma_{|\beta|}(l)$ existen tres posibles casos:

  a. si $l = N$ entonces $\sigma_\alpha(l)$ es un prefijo de $\sigma_\beta(l)$ y viceversa, y la especificación no está bien definida porque $[\![s_{bd}[n]]\!](N) = [\![s_{bd}[n]]\!](N)$ y entonces cualquier valor para $\sigma_{bad}(N)$ satisface la ecuación, generando más de un modelo de evaluación.

  b. si $l < N$, y $s_\alpha(l)$ es un prefijo de $s_\beta(l)$ o viceversa, entonces $i_0, ..., i_l$ con $l < N$ sería una solución a la instancia del problema, lo cual no puede ser porque $i_0...i_N$ era el prefijo más corto.

  c. si $l < N$, y $\sigma_\alpha(l)$ no es prefijo de $\sigma_\beta(l)$ ni viceversa, entonces $i_0, ..., i_N$ no sería una solución porque $\alpha_{i_0}...\alpha_{i_N}$ y $\beta_{i_0}...\beta_{i_N}$ no coincidirían en algún prefijo no coincide, llegando a un absurdo.

Por lo tanto, si la instancia del problema para PCP tiene solución $i_0, ..., i_N$ entonces la especificación tiene más de un modelo de evaluación para alguna traza, y por lo tanto no está bien definida.

Asumamos ahora que la especificación φ no está bien definida. Entonces por construcción de φ, existe una traza estructurada τ de longitud $N > 0$ tal que existe más de un modelo de evaluación de φ para τ. Por lo tanto, si $\langle \tau_1, \sigma_{|\alpha|}, \sigma_\alpha, \sigma_{|\beta|}, \sigma_\beta, \sigma_{idx}, \sigma_{bd} \rangle$ es un modelo de evaluación, existe un $\ell \leq N$ tal que $\sigma_{|\alpha|}(\ell) = \sigma_{|\beta|}(\ell)$ y $\sigma_\alpha(\ell) = \sigma_\beta(\ell)$, y por Lema 3.2 se tiene:

- $|\alpha_{i_0}| + ... + |\alpha_{i_\ell}| = |\alpha_{i_0}...\alpha_{i_\ell}| = |\beta_{i_0}...\beta_{i_\ell}| = |\beta_{i_0}| + ... + |\beta_{i_\ell}|$
- $\mathit{StringToInt}(\alpha_{i_0}...\alpha_{i_\ell}) = \mathit{StringToInt}(\beta_{i_0}...\beta_{i_\ell})$

donde $i_j = \sigma_{idx}(j) = \tau_1(j) \texttt{ mod } k$ para $j \leq \ell$. Sea k la longitud del máximo prefijo de ceros que $\alpha_{i_0}...\alpha_{i_\ell}$ y $\beta_{i_0}...\beta_{i_\ell}$ comparten y sea $\gamma = 0^k$. Dado que $\alpha_{i_0}...\alpha_{i_\ell} = \beta_{i_0}...\beta_{i_\ell}$ si y sólo si $\gamma\alpha_{i_0}...\alpha_{i_\ell} = \gamma\beta_{i_0}...\beta_{i_\ell}$, podemos concluir que $i_0, ..., i_\ell$ es una solución a la instacia del PCP.

Luego, la Proposición 3.1 es verdadera y el problema de la buena definición para especificaciones que utilizan enteros es indecidible.

3.2. Especificaciones booleanas puras

El problema de decidir si una especificación NSRV está bien definida se torna decidible si se restringe el problema a especificaciones booleanas. Presentamos en esta sección una solución al problema que divide al mismo en dos. Por un lado, se controla que la especificación no esté subdefinida, y por otro, que no esté sobredefinida. Si la espeficificación resulta no estar subdefinida ni sobredefinida, entonces está bien definida: tiene un único modelo de evaluación para toda traza posible. Para realizar tales controles construimos *Recursive Generalized Büchi Automatas* (RGBA), que son *Recursive State Machines* (RSM) con condiciones de aceptación. Las RSM han sido introducidas en [2] y [7] bajo diferentes nombres para modelar el flujo de control de

programas recursivos y son equivalentes a los *pushdown systems* en expresibilidad [2], pero es más sencillo razonar sobre ellas debido a la representación del control de flujo que poseen. Los RGBA fueron introducidos Alur et al. para definir la semántica del lenguaje de especificación CaRet[1].

Sintáxis. Un *Atómata Generalizado Recursivo de Büchi* (RGBA, por su definición en inglés *Recursive Generalized Büchi Automata*) $\mathcal{A}$ sobre un alfabeto Σ es una tupla $\mathcal{A} = (M, \{S_m\}_{m \in M}, inicio, \mathcal{F})$, donde M es un conjunto finito de nombres de módulos, para cada $m \in M$, S_m es un *módulo* $S_m = (N_m, B_m, Y_m, En_m, Ex_m, \delta_m, \eta_m)$ y $inicio \subseteq \bigcup_{m \in M} N_m$ es un conjunto de nodos de inicio. Cada módulo S_m consiste en siguientes componentes:

- Un conjunto finito no vacío de *nodos* N_m y un conjunto finito de *cajas* B_m.
- Una función de etiquetado $Y_m : B_m \to M$ que asigna a cada caja un nombre de módulo.
- Un conjunto no vacío de *entradas* $En_m \subseteq N_m$ y un conjunto no vacío de *salidas* $Ex_m \subseteq N_m$.
- Sea $Llamadas_m = \{(b, e) |\ b \in B_m, e \in En_{Y_m(b)}\}$ el conjunto de *llamadas* de S_m y sea $Retornos_m = \{(b, x) | b\ \in B_m, x \in Ex_{Y_m(b)}\}$ el conjunto de *retornos* en S_m.
  Entonces, $\delta_m : (N_m \cup Retornos_m) \to 2^{N_m \cup Llamadas_m}$ es una *función de transición*.
- Sea $V_m = (N_m \cup Llamadas_m \cup Retornos_m)$. Nos referimos a V_m como el conjunto de *vértices* de S_m. η_m es una función de etiquetado $\eta_m : V_m \to \Sigma$ que asocia un valor del conjunto Σ a cada vértice, es decir, a nodos, llamadas y retornos.

Denotamos con $V = \bigcup_{m \in M} V_m$ el conjunto de todos los vértices de $\mathcal{A}$ y definimos $\eta : V \to \Sigma$ como la extensión de todas las η_m, $m \in M$. El conjunto

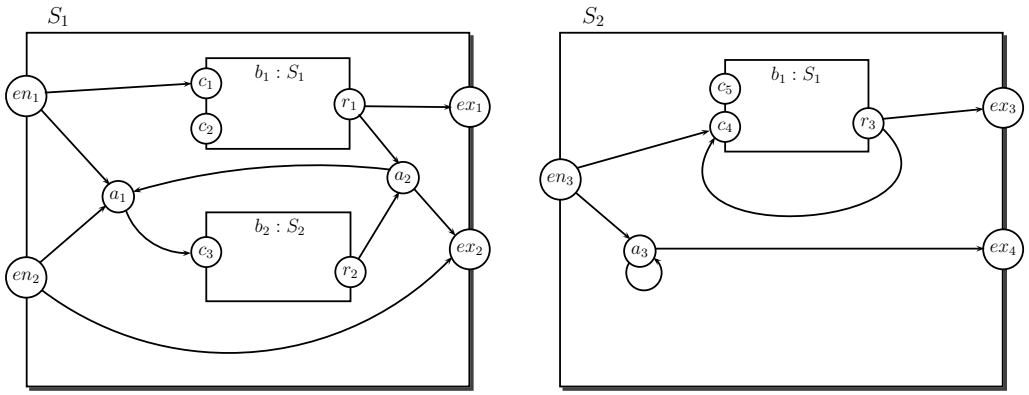

Figura 3.1: A simple RSM

$\mathcal{F} = \{F_1, ..., F_r\}$ es una familia de conjuntos de vértices de aceptación en $\mathcal{A}$, donde $F_j \subseteq V$, para $j \in \{1, ..., r\}$. Cuando hay un solo conjunto de aceptación $\mathcal{F} = \{F\}$, tenemos simplemente un *Autómata Recursivo de Büchi* (RBA, por su definición en inglés *Recursive Büchi Automaton*).

La Figura 3.2 muestra una RGBA o RBA simple con dos módulos, donde omitimos los nodos de aceptación. Los módulos son S_1 y S_2. Las entradas y salidas han sido identificadas explícitamente etiquetándolos con en_i y ex_i respectivamente, las llamadas y retornos han sido identificados con símbolos c_i y r_i respectivamente, y el resto de los nodos fueron etiquetados con símbolos a_i. El módulo S_2, por ejemplo, tiene una entrada etiquetada con en_3, dos salidas etiquetadas con ex_3 y ex_4, otro nodo etiquetado con a_3 y una caja b_1 que es asignada al módulo S_1. Vale la pena destacar que las llamadas y retornos pueden estar etiquetadas con símbolos diferentes a aquellos usados para sus correspondientes entradas o salidas. Por ejemplo, la llamada a la caja b_2 en S_1 tiene la etiqueta c_3 y la entrada del módulo S_2 fue etiquetado con en_3. Si la entrada etiquetada con en_1 en S_1 fuera el único nodo de inicio, la siguiente sería un ejemplo de una computación de la RGBA: en_1, a_1, c_3, en_3, a_3, a_3, a_3, ex_4, r_2, a_2,a_1, ...

Un parámetro importante del tamaño de una RSM (y por lo tanto de un RGBA) $\mathcal{A}$ es la *accesibilidad*, y se define como $\theta_{\mathcal{A}} = max_{m \in M}(|En_m|, |Ex_m|)$ [2]. Es decir, cada módulo en $\mathcal{A}$ tiene a lo sumo $\theta_{\mathcal{A}}$ entradas y $\theta_{\mathcal{A}}$ salidas.

Semántica. De una RGBA o RBA $\mathcal{A} = (M, \{S_m\}_{m \in M}, inicio, \mathcal{F})$ definimos una estructura de Kripke global (infinita) $K_{\mathcal{A}} = (Q, Inicio, \kappa, \delta, \mathcal{F}^{\#})$. Los *estados* (globales), denotados con Q, son elementos $(\gamma, u) \in B^* \times V$ tales que:

- $\gamma = \epsilon$ y $u \in V$, o
- $\gamma = b_1 ... b_k$ (con $k \geq 1$) y $\forall i \in [1, k-1]$, $b_{i+1} \in B_{Y(b_i)}$ y $u \in V_{Y(b_k)}$.

Los estado iniciales son $Inicio = \{(\epsilon, u) | u \in inicio\}$, y la función de etiquetado es $\kappa((\gamma, u)) = \eta(u)$.

La relación global de transición $\delta : Q \to 2^Q$, se define como sigue. Para $s = (\gamma, u)$ y $s' = (\gamma', u')$, $s' \in \delta(s)$ si y sólo si una de las siguientes es verdadera:

- **Movimientos internos:** $u \in (N_m \cup Retornos_m) \setminus Ex_m$, $u' \in \delta_m(u)$ y $\gamma' = \gamma$
- **Llamada a un módulo:** $u = (b, e) \in Llamadas_m$, $u' = e$ y $\gamma' = \gamma.b$
- **Retorno de un módulo:** $u \in Ex_m$, $u' = (b, u)$ y $\gamma'.b = \gamma$

Para una palabra $\alpha = \alpha_0 \alpha_1 ... \in \Sigma^\omega$, una *ejecución* de $K_{\mathcal{A}}$ en α es una secuencia de estados $\pi = s_0, s_1, ...$ donde $\kappa(s_i) = \alpha_i$, para todo $i \in \mathbb{N}$, y tal que $s_0 \in Inicio$ y para cada $i \in \mathbb{N}$, $s_{i+1} \in \delta(s_i)$.

La condición de aceptación en $\mathcal{A}$ induce una condición de aceptación en la estructura de Kripke, $\mathcal{F}^{\#} = \{F_1^{\#}, ..., F_r^{\#}\}$, donde $F_i^{\#} = \{(\gamma, u) \in Q | u \in F_i\}$. Decimos que una *ejecución* π de $K_{\mathcal{A}}$ es una *ejecución de aceptación* si y sólo si para todo $F_j \in \mathcal{F}$, para infinitos $i \in \mathbb{N}$, $s_i \in F_j^{\#}$.

Dado un RGBA o RBA $\mathcal{A}$, definimos el lenguaje de $\mathcal{A}$ como:

$$\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega | \text{ hay una ejecución de } K_{\mathcal{A}} \text{ en } \alpha\}$$

Para facilitar la construcción de los RGBA o RBA de esta sección definimos una nueva clase de especificaciones.

Definición 3.3 (Especificaciones de entrada actual)**.** *Una especificación NSRV φ es de entrada actual si no contiene desplazamientos, ni concretos ni abstractos, de las variables independientes.*

Es decir, los valores de las variables independientes que se utilizan en una especificación de entrada actual se obtienen mediante expresiones del tipo $t_i[n]$, sin desplazamientos. Es fácil ver que para una especificación φ unitaria siempre existe una especificación φ' unitaria y de entrada actual que la simula, pues la expresión $t_j[Z(n)+k|c]$, con $k \in \{-1, 1\}$ y $Z \in \{C, A\}$, es equivalente a $s_{t_j}[Z(n)+k|c]$ en $\varphi' = \varphi \cup \{s_{t_j}[n] = t_j[n]\}$.

En el resto del capítulo asumiremos que φ es una especificación booleana unitaria en forma canónica y de entrada actual sobre variables independientes $t_0, ..., t_{m-1}$ y variables dependientes $\sigma_0, ..., s_{r-1}$.

Los vértices de los RGBA o RBA que se construirán en esta sección contendrán *predicciones de entrada* y *predicciones de salida*. Una *predicción de entrada* es una tupla en $\mathbb{B}^m$ que representa una predicción sobre los valores de los streams de entrada. Una *predicción de salida* es una 5-upla en $(\mathbb{B}^n)_\perp \times (\mathbb{B}^n)_\perp \times \mathbb{B}^n \times (\mathbb{B}^n)_\perp \times (\mathbb{B}^n)_\perp$. Informalmente, una predicción de entrada representará los valores de los streams de entrada en el vértice actual y una predicción de salida denotará los valores que son predichos o adivinados para los streams de salida: algunas predicciones serán buenas (en particuar aquellas que puedan formar un modelo de evaluación) y algunas predicciones serán malas.

Si $p = (p_1, p_2, p_3, p_4, p_5)$ es una predicción de salida entonces:

- $p.\texttt{prev}_{abs}$ denota p_1 y representa la predicción en la posición previa, sobre el camino abstracto de la traza.
- $p.\texttt{prev}_{con}$ denota p_2 y representa la predicción en la posición previa, sobre el camino concreto.
- $p.\texttt{actual}$ denota p_3 y representa la predicción en la posición actual.
- $p.\texttt{prox}_{con}$ denota p_4 y representa la predicción en la próxima posición, sobre el camino concreto de la traza.

- $p.\texttt{prox}_{abs}$ denota p_5 y representa la predicción en la próxima posición, sobre el camino abstracto.

Por convención, si $p_i = \bot$, $p_i(j) = \bot$ para cualquier j. Una predicción de salida p es:

- *inicial* si $p.\texttt{prev}_{abs} = \bot = p.\texttt{prev}_{con}$,
- *única* si $p.\texttt{prev}_{abs} = p.\texttt{prev}_{con} = \bot = p.\texttt{prox}_{abs} = p.\texttt{prox}_{con}$
- *intermedia* si $p.\texttt{prev}_{abs} \neq \bot \neq p.\texttt{prev}_{con}$, $p.\texttt{prox}_{abs} \neq \bot \neq p.\texttt{prox}_{con}$, y
- *final* si $p.\texttt{prev}_{abs} \neq \bot \neq p.\texttt{prev}_{con}$, y $p.\texttt{prox}_{abs} = \bot = p.\texttt{prox}_{con}$.

De aquí en adelante sólo consideraremos predicciones de salida de alguna de las clases anteriormente definidas: el conjunto de todas las predicciones de entrada será denotado por $\mathcal{I}^?$ y el conjunto de todas las predicciones de salida previamente clasificadas será denotado por $\mathcal{G}^? = \{p \mid p \text{ es una predicción inicial, única, intermedia o final}\}$.

Sea $e \triangleleft (e_1 \leftarrow e'_1, ..., e_k \leftarrow e'_k)$ la expresión resultante de reemplazar simultáneamente cada e_i por e'_i en e. Para cualquier predicción de salida $p \in \mathcal{G}^?$ y predicción de entrada $i \in \mathcal{I}^?$, diremos que i y p son *compatibles* si y sólo si $i \rightleftharpoons o$, donde:

$$i \rightleftharpoons p = \bigwedge_{j \leq n} [\ p.\texttt{actual}(j) \leftrightarrow [\![e_j]\!]^{(i,p)}_{pred}\ ], \text{ y}$$

$$[\![e_j]\!]^{(i,p)}_{pred} = [\![e_j \triangleleft (t_r[n] \leftarrow i(r),$$
$$s_k[n] \leftarrow p.\texttt{actual}(k),$$
$$s_k[C(n)-1|c] \leftarrow \begin{cases} p.\texttt{prev}_{con}(k) & \text{si } p.\texttt{prev}_{con}(k) \neq \bot \\ c & \text{caso contrario,} \end{cases}$$
$$s_k[A(n)-1|c] \leftarrow \begin{cases} p.\texttt{prev}_{abs}(k) & \text{si } p.\texttt{prev}_{abs}(k) \neq \bot \\ c & \text{caso contrario,} \end{cases}$$
$$s_k[C(n)+1|c] \leftarrow \begin{cases} p.\texttt{prox}_{con}(k) & \text{si } p.\texttt{prox}_{con}(k) \neq \bot \\ c & \text{caso contrario,} \end{cases}$$

$$s_k[A(n)+1|c] \leftarrow \begin{cases} p.\texttt{prox}_{abs}(k) & \text{si } p.\texttt{prox}_{abs}(k) \neq \perp \\ c & \text{caso contrario} \end{cases}$$

$)]\!]_{Bool}$, $\forall r \leq m, k \leq n$.

En palabras, $[\![e_j]\!]^{(i,p)}_{pred}$ es la expresión e_j que define la ecuación para la variable dependiente s_j, donde los desplazamientos futuros y pasados (concretos y abstractos) y los valores actuales de las variables dependientes e independientes son reemplazados por los valores adivinados en i y p.

Sin pérdida de generalidad en los RGBA o RBA asumimos que las salidas no tienen transiciones de salida y que no hay transiciones de retornos a salidas.

Definimos la función *alinear* para cualquier secuencia $r_0, ..., r_n$ de tuplas en $\mathcal{D}^k$ para algún dominio $\mathcal{D}$ y $k \in \mathbb{N}$ como $alinear(r_0, ..., r_n) = \langle \sigma_0, ..., \sigma_{k-1} \rangle$, donde:

$$\sigma_i(j) = r_j(i), \text{ para todo } i < k,\ j \leq n.$$

Por ejemplo, si $r_0 = \langle 0, 7 \rangle$, $r_1 = \langle 1, 0 \rangle$ y $r_2 = \langle 2, -1 \rangle$, entonces resulta que $alinear(r_0, r_1, r_2) = \langle \sigma_0, \sigma_1 \rangle$, y $\sigma_0 = \langle 0, 1, 2 \rangle$ y $\sigma_1 = \langle 7, 0, -1 \rangle$.

Especificaciones subdefinidas. Para controlar si una especificación φ está subdefinida primero construimos un RBA $\mathcal{A}_p$ que luego enriquecemos con arcos para obtener, de un módulo de $\mathcal{A}_p$, un autómata de Büchi $\mathcal{B}_{\mathcal{A}_p}$ tradicional. Luego, probamos que $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$ si y sólo si existe una traza estructurada tal que φ tiene más de un modelo de evaluación para ella.

Los nodos del RBA que construiremos tendrán una predicción de entrada y dos predicciones de salida, que pueden ser idénticas o diferentes. Además, los nodos estarán duplicados al agregar un bit en la codificación: el valor 1 en este bit será para aquellos nodos que tengan predicciones de salida diferentes o que tengan algún predecesor con esta propiedad, es decir, es un bit de historia. El RBA tendrá dos módulos: S_{ini} con los vértices iniciales y finales, nodos intermedios y todas sus cajas estarán asignadas a S_{rec}. El

módulo S_{rec} tendrá sólo nodos intermedios, pero tendrá sus cajas asignadas a si mismo, permitiendo de esta manera la recursión. Si bien ninguna llamada o retorno del módulo S_{ini} será efectivamente usado, realizamos la construcción de manera general y luego basamos la construcción de S_{rec} en ella.

Para definir los nodos de la RGA consideramos el dominio $\mathcal{D}$ que consta de una predicción de entrada, dos predicciones de salida compatibles con la predicción de entrada y el bit de historia mencionado:

$$\mathcal{D} = \{(i, p_1, p_2, h) |\ i \in \mathcal{I}^?, p_1, p_2 \in \mathcal{G}^?, h \in \mathbb{B}, i \rightleftharpoons p_1, i \rightleftharpoons p_2\}$$

Si $v = (i, p_1, p_2, pd)$ es un elemento de $\mathcal{D}$, la predicción de entrada i será denotado por $v.i$, las predicción de salida p_j será denotada por $v.p_j$ y $v.h$ denotará el bit de historia h, y su valor indica si el camino para llegar a v puede formar parte de una traza para la cual φ tenga más de un modelo de evaluación y por ende esté subdefinida.

El RBA de dos módulos sobre el alfabeto Σ es $\mathcal{A}_p = (\{ini, rec\}, \{S_{ini}, S_{rec}\}, inicio, \mathcal{F})$. El módulo $S_{ini} = (N_{ini}, B_{ini}, Y_{ini}, En_{ini}, Ex_{ini}, \delta_{ini}, \eta_{ini})$ está definido como sigue:

- $N_{ini} = \mathcal{D} \cup \mathcal{D} \times \mathcal{D}$, es el conjunto de nodos que contiene todas las posibles combinaciones entre predicciones de entrada y pares de predicciones salida compatibles, duplicados por el bit de historia. Además, contiene pares de tales combinaciones ($\mathcal{D} \times \mathcal{D}$) para incluir, en aquellos nodos que representen la salida de un módulo, todas las predicciones que deban hacerse en el retorno.

- $B_{ini} = \mathcal{D}$, y $Y_{ini}(b) = rec$, para toda $b \in B_{ini}$. Entonces S_{ini} contiene una caja por cada combinación de predicciones posibles que puede darse al momento de la invocación del módulo al que fue asignada, el cual es constante: S_{rec}.

- $En_{ini} = \mathcal{D}$

- $Ex_{ini} = (\mathcal{D} \times \mathcal{D})$

Las entradas y salidas de este módulo son en realidad irrelevantes porque, como se verá al terminar la construcción, no existirán cajas asignadas a S_{ini}.

- La función de etiquetado η_{ini} se define por casos:
  - Para nodos no de salida: $\eta_{ini}(v) = v.i$
  - Para nodos de salida: $\eta_{ini}(v_{ex}, v_r) = v_{ex}.i$, para todo $v_r \in \mathcal{D}$
  - Para llamadas: $\eta_{ini}(b, v_{en}) = b.i$, para todo $v_{en} \in En$
  - Para retornos: $\eta_{ini}(b, (v_{ex}, v_r)) = v_r.i$, para todo $b \in B_{ini}, v_{ex} \in \mathcal{D}$

Note que las llamadas son de la forma (b, v_{en}), donde $b.p_1$ y $b.p_2$ son las predicciones de los streams de salida hechas en el momento de la llamada, y $v_{en}.p_1$ y $v_{en}.p_2$ son predicciones de los streams de salida en la entrada al módulo $Y(b) = S_{rec}$. Como el nombre de la caja $b = (i_c, p_{c_1}, p_{c_2}, h_c)$ es puesto en la pila, los valores adivinados en la llamada son recordados a lo largo de la ejecución del módulo.

La salidas son de la forma (v_{ex}, v_r), y aquí $v_{ex}.p_1$ y $v_{ex}.p_2$ son predicciones hechas a la salida mientras que $v_r.p_1$ y $v_r.p_2$ son predicciones hechas en la posición en la cual el control vuelve al módulo que realizó la llamada. Además, en un retorno $(b, (v_{ex}, v_r))$, como $b.p_1$ y $b.p_2$ son predicciones realizadas en la posición en que se realizó la llamada y $v_r.p_1$ y $v_r.p_1$ son predicciones en el retorno, pediremos que ciertas condiciones sobre las predicciones hechas para el camino abstracto se cumplan.

Ahora, para las posibles transiciones entre vértices de la RBA requerimos que las predicciones de los nodos involucrados sean coherentes, y capturamos esta propiedad como sigue. Definimos los siguientes predicados sobre predicciones, que definirán la *coherencia* entre un par de predicciones:

- Para transiciones de un vértice que no es llamada ni salida, $concretoYabstracto(p, p')$ si y sólo si:
  - p es una predicción inicial o intermedia

- p' es una predicción intermedia o final
- $\bigwedge_{j \leq n}(p.\texttt{prox}_{abs}(j) = p'.\texttt{actual}(j) \wedge$
  $p.\texttt{prox}_{con}(j) = p'.\texttt{actual}(j) \wedge$
  $p.\texttt{actual}(j) = p'.\texttt{prev}_{abs}(j) \wedge$
  $p.\texttt{actual}(j) = p'.\texttt{prev}_{con}(j))$

- Para los requerimientos entre llamadas y entradas, $llamadaEntrada(p, p')$ si y sólo si:

  - p es una predicción inicial o intermedia
  - p' es una predicción intermedia
  - $\bigwedge_{j \leq n}(p.\texttt{prox}_{con}(j) = p'.\texttt{actual}(j) \wedge$
    $p.\texttt{actual}(j) = p'.\texttt{prev}_{con}(j) \wedge$
    $p.\texttt{actual}(j) = p'.\texttt{prev}_{abs}(j))$

- Para los requerimientos entre una salida y un retorno $salidaRetorno(p, p')$ si y sólo si:

  - p es una predicción intermedia
  - p' es una predicción intermedia o final
  - $\bigwedge_{j \leq n}(p.\texttt{prox}_{con}(j) = p'.\texttt{actual}(j) \wedge$
    $p.\texttt{prox}_{abs}(j) = p'.\texttt{actual}(j) \wedge$
    $p.\texttt{actual}(j) = p'.\texttt{prev}_{con}(j))$

- Y cuando un módulo retorna, requerimos que las predicciones hechas sobre el camino abstracto entre la llamada y el retorno sean coherentes, $llamadaRetorno(p, p')$ si y sólo si:

  - p es una predicción inicial o intermedia
  - p' es una predicción intermedia o final
  - $\bigwedge_{j \leq n}(p.\texttt{prox}_{abs}(j) = p'.\texttt{actual}(j) \wedge p.\texttt{actual}(j) = p'.\texttt{prev}_{abs}(j))$

Una vez definidos estos predicados, la función de transición del módulo, δ_{ini} es definida como:

- **De nodos no de aceptación a nodos no de salida**: $\delta_{ini}(v)$ contiene a v' para todos los v' que cumplen:
  - $\bigwedge_{j=1,2}$ $concretoYabstracto(v.p_j, v'.p_j)$
  - $v'.h = [v.h \vee (v'.p_1 \neq v'.p_2)]$
- **De nodos a llamadas**: $\delta_{ini}(v)$ contiene a (b, v_{en}) si y sólo si:
  - $\bigwedge_{j=1,2}$ $concretoYabstracto(v.p_j, b.p_j)$
  - $\bigwedge_{j=1,2}$ $llamadaEntrada(b.p_j, v_{en}.p_j)$
  - $b.h = [v.h \vee (b.p_1 \neq b.p_2)]$
  - $v_{en}.h = [b.h \vee (v_{en}.p_1 \neq v_{en}.p_2)]$
- **De nodos a salidas**: $\delta_{ini}(v)$ contiene a (v_{ex}, v_r) si y sólo si:
  - $\bigwedge_{j=1,2}$ $concretoYabstracto(v.p_j, v_{ex}.p_j)$
  - $\bigwedge_{j=1,2}$ $salidaRetorno(v_{ex}.p_j, v_r.p_j)$
  - $v_{ex}.h = [v.h \vee (v_{ex}.p_1 \neq v_{ex}.p_2)]$
  - $v_r.h = [v_{ex}.h \vee (v_r.p_1 \neq v_r.p_2)]$
- **De retornos a nodos**: $\delta_{ini}(b, (v_{ex}, v_r))$ contiene a v' si y sólo si:
  - $\bigwedge_{j=1,2}$ $salidaRetorno(v_{ex}.p_j, v_r.p_j)$
  - $\bigwedge_{j=1,2}$ $concretoYabstracto(v_r.p_j, v'.p_j)$
  - $\bigwedge_{j=1,2}$ $llamadaRetorno(b.p_j, v_r.p_j)$
  - $v_r.h = [v_{ex}.h \vee (v_r.p_1 \neq v_r.p_2) \vee b.h \vee (b.p_1 \neq b.p_2)]$
  - $v_{ex}.h = [v.h \vee (v_{ex}.p_1 \neq v_{ex}.p_2)]$
  - $v'.h = [v_r.h \vee (v'.p_1 \neq v'.p_2)]$

- **De retornos a llamadas**: $\delta_{ini}(b, (v_{ex}, v_r))$ contiene a (b', v_{en}) si y sólo si:
  - $\bigwedge_{j=1,2}$ $salidaRetorno(v_{ex}.p_j, v_r.p_j)$
  - $\bigwedge_{j=1,2}$ $llamadaRetorno(b.p_j, v_r.p_j)$
  - $\bigwedge_{j=1,2}$ $concretoYabstracto(v_r.p_j, b'.p_j)$
  - $\bigwedge_{j=1,2}$ $llamadaEntrada(b'.p_j, v_{en}.p_j)$
  - $v_r.h = [v_{ex}.h \vee (v_r.p_1 \neq v_r.p_2) \vee b.h \vee (b.p_1 \neq b.p_2)]$
  - $v_{ex}.h = [v.h \vee (v_{ex}.p_1 \neq v_{ex}.p_2)]$
  - $b'.h = [v_r.h \vee (b'.p_1 \neq b'.p_2)]$
  - $v_{en}.h = [b'.h \vee (v_{en}.p_1 \neq v_{en}.p_2)]$

El módulo S_{rec} es muy similar al anterior, salvo que las cajas son asignadas a si mismo y no tiene vértices con predicciones iniciales o finales. Se verá luego que esto último implica que ninguna ejecución de $\mathcal{A}_p$ puede empezar en vértices de S_{rec} o alcanzar infinitas veces uno de ellos. $S_{rec} = (N_{rec}, B_{rec}, Y_{rec}, En_{rec}, Ex_{rec}, \delta_{rec}, \eta_{rec})$ se define como sigue:

- $N_{rec} = \{v \in \mathcal{D} \mid v.p_1, v.p_2$ son predicciones intermedias$\} \cup$
  $\{(v, v') \in \mathcal{D}\times\mathcal{D} \mid v.p_1, v.p_2, v'.p_1, v'.p_1$ son predicciones intermedias$\}$
- $B_{rec} = \{v \in \mathcal{D} \mid v.p_1, v.p_2$ son predicciones intermedias$\}$
- $Y_{rec}(b) = rec$, para toda $b \in B_{rec}$.
- $En_{rec} = \{v \in \mathcal{D} \mid v.p_1, v.p_2$ son predicciones intermedias$\}$
- $Ex_{rec} = \{(v, v') \in \mathcal{D}\times\mathcal{D} \mid v.p_1, v.p_2, v'.p_1, v'.p_1$ son predicciones intermedias$\}$
- La función de etiquetado η_{rec} se define igual a η_{ini}:
  - Para nodos no de salida: $\eta_{rec}(v) = v.i$
  - Para nodos de salida: $\eta_{rec}(v_{ex}, v_r) = v_{ex}.i$, para todo $v_r \in \mathcal{D}$

- Para llamadas: $\eta_{rec}(b, v_{en}) = b.i$, para todo $v_{en} \in En$
- Para retornos: $\eta_{rec}(b, (v_{ex}, v_r)) = v_r.i$, para todo $b \in B_{ini}, v_{ex} \in \mathcal{D}$

La función de transición se define de manera similar, salvando que en este módulo todas las predicciones son intermedias, y que no hay vértices de aceptación. Luego, δ_{rec} está definida exactamente como δ_{ini}, pero restringida a vértices de S_{rec}:

$\delta_{rec}(v)$ contiene a v' si y sólo si $v, v' \in V_{rec}$ y $\delta_{ini}(v)$ contiene a v'

Luego $N = N_{ini} \cup N_{rec}$ será el conjunto de todos los nodos de $\mathcal{A}_p$, y el conjunto de nodos iniciales es definido como $inicio = \{v \in N \mid v.p_1$ y $v.p_2$ son predicciones iniciales o únicas$\}$. La condición de aceptación de $\mathcal{A}_p$ está dada por el conjunto

- $F_1 = \{v \in N |\ h = true$, y $v.p_1$ y $v.p_2$ son predicciones finales o únicas$\} \cup$
  $\{(b, (v_{ex}, v_r)) \in V \mid h = true$, y $v_r.p_1$ y $v_r.p_2$ son predicciones finales o únicas$\}$.

Note que los vértices iniciales y los vértices de aceptación se encuentran únicamente en el módulo S_{ini}.

Finalmente, para que $\mathcal{A}_p$ acepte cadenas infinitas enriquecemos la función de transición δ_{ini} de tal forma que cada vértice de aceptación se comporte como un vértice *terminal* añadiendo una transición a sí mismo sin restricciones y por lo tanto, una vez alcanzado, cualquier sufijo será aceptado:

- **De vértices de aceptación a ellos mismos**: Si $v \in F_1$, entonces $\delta_{ini}(v)$ contiene a v.

Para obtener el autómata de Büchi $\mathcal{B}_{\mathcal{A}_p}$ mencionado al principio de la sección nos referimos primero al algoritmo de decisión en [1] para vacuidad de lenguaje de RGBAs. Lo que hacemos para enriquecer $\mathcal{A}_p$ es, para cada entrada v_{en} y cada salida v_{ex} de S_{rec}, chequeamos si existe un camino desde

(ϵ, v_{en}) hasta (ϵ, v_{ex}) en la estructura de Kripke global inducida. Esto fue presentado en [2] y requiere chequear alcanzabilidad en un grafo $And-Or$, y toma tiempo $\mathcal{O}(|\mathcal{A}_p|\theta^2_{\mathcal{A}_p})$ y espacio $\mathcal{O}(|\mathcal{A}_p|\theta_{\mathcal{A}_p})$. Si el camino existe, entonces agregamos un arco desde cada llamada (b, v_{en}) a cada salida $(b, (v_{ex}, v_r))$ en S_{ini} precisamente cuando ambas satisfacen:

- $\bigwedge_{j=1,2} llamadaRetorno(b.p_i, v_r.p_i)$
- $v_r.h = [v_{ex}.h \vee (v_r.p_1 \neq v_r.p_2) \vee b.h]$

Es decir, cuando existe tal camino, las predicciones en la llamada son coherentes con las predicciones en el retorno y el bit de historia del retorno es consistente con sus predecesores. A partir del módulo S_{ini} enriquecido podemos definir $\mathcal{B}_{\mathcal{A}_p} = (Q, L, Q_0, \delta_{\mathcal{B}}, F)$, donde:

- $Q = V$
- $L(q) = \eta_{ini}(q)$, para todo $v \in Q$
- $Q_0 = inicio$
- $q' \in \delta_{\mathcal{B}}(q)$ si y sólo si $q' \in \delta_{ini}(q)$
- $F = F_1$

Probamos a continuación la equivalencia entre la vacuidad de $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p})$ y el problema de decidir si una especificación está subdefinida.

Lema 3.4. *Sea φ una especificación NSRV booleana. Si $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$ entonces la especificación está subdefinida.*

Demostración. Asuma que $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$, y sea $\alpha = i_0, i_1, ...$ una palabra de $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p})$. Luego, existe una secuencia de estados $q_0 q_1 ...$ de $\mathcal{B}_{\mathcal{A}_p}$ tal que $q_{j+1} \in \delta_{\mathcal{B}}(q_j)$, $q_j.i = i_j$ para todo $j \in \mathbb{N}$. Además, $q_j \in F$ para infinitos $j \in \mathbb{N}$, y dado que los estado de aceptación son terminales, existe un mínimo $\ell \in \mathbb{N}$ tal que $q_j = q_\ell$ para todo $j \geq \ell$.

Por otro lado, para cada par de estados q_j, q_{j+1} con $j < \ell$, por construcción de $\mathcal{A}_\varphi$, existe un camino $s_1^j ... s_{k_j}^j$ en $K_{\mathcal{A}_\varphi}$ tal que $s_1^j \in \delta((\epsilon, q_j))$, $q_{j+1} \in \delta(s_{k_j}^j)$, y s_1^j y $s_{k_j}^j$ son a entrada y la salida del módulo llamado en q_j. Sea $I = \langle j_1, ..., j_r \rangle$ la secuencia ordenada de índices tal que q_j es una llamada si y sólo si $j \in I$, y sea $s_j = (\epsilon, q_j)$ para $j \leq \ell$. Luego, la secuencia de $K_{\mathcal{A}_\varphi}$:

$$s_0 s_1 \ ... \ s_{j_1} s_1^{j_1} \ ... \ s_{k_{j_1}}^{j_1} s_{j_1+1} \ ..... \ s_{j_r} s_1^{j_r} \ ... \ s_{k_{j_r}}^{j_r} s_{j_r+1} \ ... \ s_\ell$$

es tal que $s_{j+1} \in \delta(s_j)$ para $j \in \{0, ..., \ell\}$, $s_0 \in Inicio$, $s_\ell \in F_1^\#$ y las predicciones de los vértices satisfacen las ecuaciones de φ. Luego, si $s_j = (z_j, q_j)$ para $j \leq \ell$, $\sigma_1 = alinear(q_0.p_1.\texttt{actual}, ..., q_\ell.p_1.\texttt{actual})$ y $\sigma_2 = alinear(q_0.p_2.\texttt{actual}, ..., q_\ell.p_2.\texttt{actual})$ son modelos de evaluación de φ para la traza estructurada $\tau = alinear(q_0.i, ..., q_\ell.i)$. Adicionalmente, $q_\ell.pd = true$ y por definición de δ existe entonces un mínimo $d \in \mathbb{N}$ tal que $q_d.p_1 \neq q_d.p_2$ y por lo tanto $\sigma_1 \neq \sigma_2$ y φ está subdefinida. □

Lema 3.5. *Sea φ una especificación NSRV booleana. Si φ está subdefinida entonces $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$.*

Demostración. Asuma que φ está subdefinida. Entonces existe una traza estructurada $\tau = alinear(i_0, ..., i_N)$ de longitud $N+1$ con modelo de ejecución inducido η tal que $\sigma_k = alinear(p_o^k.\texttt{actual}, ..., p_N^k.\texttt{actual})$ para $k = 1, 2$ son distintos modelos de evaluación de φ para σ, para algunas predicciones de salida $p_0^k, ..., p_N^k$. Luego, por construcción de $\mathcal{A}_\varphi$ y por definición de $K_{\mathcal{A}_\varphi}$, existen estados $s_0, ..., s_N$, $s_j = (z_j, v_j)$, de $K_{\mathcal{A}_\varphi}$ tales que, para $k = 1, 2$:

- $s_0 \in Inicio$ y:
  - $(v_0.p_k).\texttt{prev}_{con} = \perp = (v_0.p_k).\texttt{prev}_{abs}$
  - $(v_0.p_k).\texttt{actual} = p_0^k$
  - $(v_0.p_k).\texttt{prox}_{con} = (v_1.p_k).\texttt{actual}$
  - $(v_0.p_k).\texttt{prox}_{abs} = (v_{0\hat{+}_\eta 1}.p_k).\texttt{actual}$
- Para $j = 1..N-1$:

- $(v_j.p_k).\texttt{prev}_{con} = (v_{j-1}.p_k).\texttt{actual}$
- $(v_j.p_k).\texttt{prev}_{abs} = (v_{j\hat{+}_\eta -1}.p_k).\texttt{actual}$
- $(v_j.p_k).\texttt{actual} = p_j^k$
- $(v_j.p_k).\texttt{prox}_{con} = (v_{j+1}.p_k).\texttt{actual}$
- $(v_j.p_k).\texttt{prox}_{abs} = (v_{j\hat{+}_\eta 1}.p_k).\texttt{actual}$

▪ $s_N \in F_1^{\#}$ y:

- $(v_N.p_k).\texttt{prev}_{con} = (v_{N-1}.p_k).\texttt{actual}$
- $(v_N.p_k).\texttt{prev}_{abs} = (v_{N\hat{+}_\eta -1}.p_k).\texttt{actual}$
- $(v_N.p_k).\texttt{actual} = p_N^k$
- $(v_N.p_k).\texttt{prox}_{con} = \perp = (v_N.p_k).\texttt{prox}_{abs}$

Es fácil ver que $s_{j+1} \in \delta(s_j)$ para $j < N$. Sea $L = \{(i,j) \mid i \rightsquigarrow j, \nexists\, i', j'$ tales que $i' < i < j'$ y $i' \rightsquigarrow j'\}$ el conjunto de pares (i_r, j_r) tales que v_{i_r} es una llamada, $z_{i_r} = \epsilon$, v_{j_r} es el retorno correspondiente a v_{i_r} y $z_{j_r} = \epsilon$. Luego, para cada par $(i_s, j_s) \in L$ existe un camino desde (ϵ, v_{i_r}) hasta (ϵ, v_{j_r}) pasando por la correspondiente entrada y salida en $K_{\mathcal{A}_\varphi}$, y por lo tanto existe un arco entre q_{i_r} y q_{j_r} en $\mathcal{B}_{\mathcal{A}_\varphi}$. Sea $L_{ord} = \langle (i_1, j_1), ..., (i_r, j_r) \rangle$ la secuencia ordenada de pares de L, entonces $q_0, ..., q_{i_1} q_{j_1} ... q_{i_r} q_{j_r} ... q_N$ es una secuencia de estados de $\mathcal{B}_{\mathcal{A}_\varphi}$ tal que $q_0 \in Q_0$ y $q_{j+1} \in \delta(q_j)$ para todo $j < N$. Además, si $s_N \in F_1^{\#}$, entonces $q_N \in F$ y $q_N \in \delta(q_N)$. Por ende, $\alpha = q_0.i, ..., q_N.i, q_N.i, q_N.i, ...$ es una palabra en $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p})$ y $\mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$. □

Teorema 3.6. *Sea φ una especificación NSRV booleana:*

$$\varphi \text{ está subdefinida} \iff \mathcal{L}(\mathcal{B}_{\mathcal{A}_p}) \neq \emptyset$$

Demostración. La prueba es directa de los Lemas 3.4 y 3.5. □

Esto demuestra que la comprobación de si φ está subdefinida es decidible, ya que la vacuidad de autómatas de Büchi lo es y $\mathcal{B}_{\mathcal{A}_p}$ se construye efectivamente a partir de φ.

Especificaciones sobredefinidas. Un especificación φ está sobredefinida si existe una traza τ estructurada tal que no hay ningún modelo de evaluación de φ para τ. Para decidir si una especificación está sobredefinida construímos un RBA $\mathcal{A}_d$ similar a $\mathcal{A}_p$, usado en el control de especificaciones subdefinidas, sólo que en este caso los nodos de un módulo contendrán conjuntos de predicciones, en lugar de un par. Este conjunto tendrá todas las predicciones compatibles con la predicción de entrada del nodo y que a su vez sean coherentes con alguna predicción de salida de los nodos predecesores. Al recordar en un nodo todas y cada una de las predicciones compatibles con la entrada dada, podremos determinar si una entrada no tiene predicciones compatibles con ella, y por lo tanto ningún modelo de evaluación. Para este RBA probaremos luego que $\mathcal{L}(\mathcal{A}_d) \neq \emptyset$ si y sólo si existe una traza para la cual φ no tiene modelo de evaluación.

Definimos el dominio que consta de una predicción de entrada y un conjunto de predicciones de salida compatibles con ella como:

$$\mathcal{D} = \{(i, P) \mid i \in \mathcal{I}^?, P \in \mathcal{P}(\mathcal{G}^?),\ i \rightleftharpoons p \text{ para todo } p \in P\}$$

Si $v = (i, P)$ es un elemento de $\mathcal{D}$, entonces $v.i$ denotará a i y $v.P$ denotará a P.

El RBA de un solo módulo sobre el alfabeto Σ que construimos es $\mathcal{A}_d = (\{0\},\ \{S_0\},\ \mathit{inicio},\ F)$, y el módulo $S_0 = (N,\ B,\ Y,\ En,\ Ex,\ \delta,\ \eta)$ se define como:

- $N = \mathcal{D} \cup (\mathcal{D} \times \mathcal{D})$, es el conjunto de nodos que contiene todas las combinaciones de predicciones de entradas y conjuntos de predicciones de salida. Además, contiene pares de combinaciones $(\mathcal{D} \times \mathcal{D})$ para incluir, en aquellos nodos que representen la salida de un módulo, todas las predicciones que deban hacerse en el retorno. Luego se definirán las condiciones sobre estas predicciones para lograr la correspondencia entre salidas y retornos.

- $B = \mathcal{D}$ y $Y(b) = 0$ para toda $b \in B$. Es decir, S_0 contiene una caja por

cada combinación de predicciones posibles que puede darse al momento de la invocación de S_0.

- $En = \mathcal{D}$
- $Ex = \mathcal{D} \times \mathcal{D}$
- La función de etiquetación η se define por casos:
  - Para nodos no de salida: $\eta(v) = v.i$
  - Para nodos de salida: $\eta((v_{ex}, v_r)) = v_{ex}.i$
  - Para llamadas: $\eta(b, v_{en}) = v_{en}.i$, para todo $b \in B$
  - Para retornos: $\eta((b, (v_{ex}, v_r))) = v_r.i$, para todo $b \in B, v_{ex} \in Ex$

La función de transición δ se define usando los predicados sobre predicciones usados para especificaciones subdefinidas como sigue:

- **De nodos no de aceptación a nodos no de salida**: $\delta(v)$ contiene a v' si y sólo si:
  - Las predicciones en $v.P$ son todas iniciales o todas intermedias
  - Las predicciones en $v'.P$ son todas intermedias o todas finales
  - $v'.P = \bigcup_{p \in v.P} \{p' \in \mathcal{G} \mid \mathit{concretoYabstracto}(p, p')\}$
- **De nodos a llamadas**: $\delta(v)$ contiene a (b, v_{en}) si y sólo si:
  - Las predicciones en $v.P$ son todas iniciales o todas intermedias
  - Las predicciones en $b.P$ y $v_{en}.P$ son todas intermedias
  - $b.P = \bigcup_{p \in v.P} \{p' \in \mathcal{G} \mid \mathit{concretoYasbtracto}(p, p')\}$
  - $v_{en}.P = \bigcup_{p \in b.P} \{p' \in \mathcal{G} \mid \mathit{llamadaEntrada}(p, p')\}$
- **De nodos a salidas**: $\delta(v)$ contiene a (v_{ex}, v_r) si y sólo si:
  - Las predicciones en $v.P$ y $v_{ex}.P$ son todas intermedias

- Las predicciones en $v_r.P$ son todas intermedias o todas finales
- $v_{ex}.P = \bigcup_{p \in v.P} \{p' \in \mathcal{G} \mid concretoYasbtracto(p, p')\}$
- $v_r.P = \bigcup_{p \in v_{ex}.P} \{p' \in \mathcal{G} \mid salidaRetorno(p, p')\}$

▪ **De retornos a nodos**: $\delta(b, (v_{ex}, v_r))$ contiene a v' si y sólo si:

- Las predicciones en $b.P$, $v_{ex}.P$ y $v_r.P$ son todas intermedias
- Las predicciones en $v'.P$ son todas intermedias o todas finales
- $v_r.P = (\bigcup_{p \in v_{ex}.P} \{p' \in \mathcal{G} \mid salidaRetorno(p, p')\}) \bigcap$
  $(\bigcup_{p \in b.P} \{p' \in \mathcal{G} \mid llamadaRetorno(p, p')\})$
- $v'.P = \bigcup_{p \in v_r.P} \{p' \in \mathcal{G} \mid concretoYabstracto(p, p')\}$

▪ **De retornos a llamadas**: $\delta(b, (v_{ex}, v_r))$ contiene a (b', v_{en}) si y sólo si:

- Las predicciones en $b.P$, $v_{ex}.P$ y $v_r.P$ son todas intermedias
- Las predicciones en $v'.P$ son todas intermedias o todas finales
- $v_r.P = (\bigcup_{p \in v_{ex}.P} \{p' \in \mathcal{G} \mid salidaRetorno(p, p')\}) \bigcap$
  $(\bigcup_{p \in b.P} \{p' \in \mathcal{G} \mid llamadaRetorno(p, p')\})$
- $v'.P = \bigcup_{p \in v_r.P} \{p' \in \mathcal{G} \mid concretoYabstracto(p, p')\}$
- Las predicciones en $b'.P$ y $v_{en}.P$ son todas intermedias
- $b'.P = \bigcup_{p \in v_r.P} \{p' \in \mathcal{G} \mid concretoYasbtracto(p, p')\}$
- $v_{en}.P = \bigcup_{p \in b'.P} \{p' \in \mathcal{G} \mid llamadaEntrada(p, p')\}$

La condición de aceptación de $\mathcal{A}_d$ está dada por el único conjunto F_1, definido como:

▪ $F_1 = \{v \in (\mathcal{D} \cap V) \mid v.P = \emptyset\} \cup \{(b, (v_{ex}, v_r)) \in Retornos_0 \mid v_r.P = \emptyset\}$

Nuevamente, enriquecemos la función de transición para convertir cada vértice de aceptación en un vértice *terminal*, añadiendo una transición a sí mismo si restricción alguna.

- **De nodos de aceptación a ellos mismos**: Si $v \in F_1$, entonces $\delta(v)$ contiene a v.

Note que cada vértices sucesor en $\mathcal{A}_d$ contiene *todas* las predicciones de salida que son compatibles con su respectiva predicción de entrada y que son coherentes con alguna las predicciones de salida del vértice predecesor.

Probaremos primero un Lema auxiliar sobre predicciones y modelos de evaluación que será de utilidad en las demostraciones siguientes.

Lema 3.7. *Sea φ una especificación NSRV booleana unitaria y de entrada fija. Sea $\tau = alinear(i_0, ..., i_N)$ una traza estructurada de longitud $N+1$ con modelo de ejecución inducido η, para algunas predicciones de entrada $i_0, ..., i_N$. Entonces existe un modelo de evaluación $\langle \tau, \sigma \rangle$ de φ si y sólo si existen predicciones de salida $p_0, ..., p_N$ tales que: (a) $i_j \rightleftharpoons p_j$ para todo $j \leq N$, (b) p_0 es una predicción inicial, p_k es una predicción final y $p_1, ..., p_{N-1}$ son predicciones intermedias, (c) $\sigma = alinear(p_0.\texttt{actual}, ..., p_N.\texttt{actual})$, y (d) para todo $j < N$:*

- *Si i_j es una llamada, entonces*
  - *llamadaEntrada(p_j, p_{j+1})*
- *Si i_j es una salida, entonces salidaRetorno(p_j, p_{j+1})*
- *Si i_j es un retorno, entonces*
  - *concretoYabstracto(p_j, p_{j+1})*
  - *llamadaRetorno$(p_j, p_{j'})$, donde $j \rightsquigarrow j'$*
- *En otro caso, concretoYabstracto(p_j, p_{j+1})*

Si un conjunto de predicciones de salida $p_0, ..., p_N$ satisface las propiedades descritas diremos que el conjunto de predicciones es coherente con la entrada τ.

Demostración. ($\Leftarrow$) Asumamos que las predicciones de salida $p_0, ..., p_N$ satisfacen (a), (b), (c) y (d). Vale destacar que estas propiedades sobre $p_0, ..., p_N$ aseguran que las predicciones de valores pasados y futuros entre predicciones sucesivas son coherentes con respecto a τ. Basta con probar que $[\![e_\ell]\!](j) = [\![e_\ell]\!]^{(i_j,p_j)}_{pred}$, ya que por consistencia entre i_j y p_j se tiene que $[\![e_\ell]\!]^{(i_j,p_j)}_{pred} = p_j.\texttt{actual}(\ell)$ y entonces $\sigma_\ell(j) = p_j.\texttt{actual}(\ell)$, para todo $\ell \leq n, j \leq N$. Dado que no hay desplazamientos sobre las variables independientes y que los desplazamientos sobre variables dependientes son unitarios, tenemos los siguientes casos para e_j:

Los casos bases:

- c: trivial, pues no hay reemplazos.
- $t_r[n]$: es exactamente el valor de $i_j(r)$.
- $s_r[n]$: se tiene que $[\![e_\ell]\!](j) = \sigma_r(j)$, el cual es $p_j.\texttt{actual}(r)$, al igual que $[\![e_\ell]\!]^{(i_j,p_j)}_{pred}$.

Los casos inductivos:

- $f(v_1[n], ..., v_q[n])$: se da naturalmente pues los reeplazos en $[\![e_\ell]\!]^{(i_j,p_j)}_{pred}$ no modifican los operadores.
- $s_r[C(n)+1|c]$: tenemos que:

  $$[\![s_r[C(n)+1|c]]\!](j) = \begin{cases} \sigma_r(j+1) & \text{si } 0 \leq j+1 \leq N \\ c & \text{caso contrario} \end{cases}$$

  $$[\![e_\ell]\!]^{(i_j,p_j)}_{pred} = \begin{cases} p_j.\texttt{prox}_{con}(r) & \text{si } p_j.\texttt{prox}_{con} \neq \perp \\ c & \text{caso contrario} \end{cases}$$

  Dado que la única predicción final es p_N y que por hipótesis sobre los p_j se da $p_j.\texttt{prox}_{con}(r) = p_{j+1}.\texttt{actual}(r)$, los casos se corresponden.
- $s_r[A(n)+1|c]$: tenemos que:

$$[\![s_r[A(n)+1|c]]\!](j) = \begin{cases} \sigma_r(j \hat{+}_\eta 1) & \text{si } 0 \leq j \hat{+}_\eta 1 \leq N \\ c & \text{caso contrario} \end{cases}$$

$$[\![e_\ell]\!]^{(i_j,p_j)}_{pred} = \begin{cases} p_{j'}.\texttt{prox}_{abs}(r) & \text{si } p_j.\texttt{prox}_{abs} \neq \bot, \text{ donde } (j,j') \in \eta \\ c & \text{caso contrario} \end{cases}$$

Por hipótesis que $p_0, ..., p_N$ satisfacen (d), es fácil ver que $p_j.\texttt{prox}_{abs}(r) = p_{j'}.\texttt{actual}(r)$, donde $j' = j \hat{+}_\eta 1$. O equivalentemente, $p_j.\texttt{prox}_{abs}(r) = p_{(j \hat{+}_\eta 1)}.\texttt{actual}(r)$. Además, p_N es la única predicción final, y entonces los casos se corresponden.

- Los casos para desplazamientos negativos son anólogos y se cumplen dado que la única predicción inicial es p_0 y por las condiciones que satisfacen $p_0, ..., p_N$.

($\Rightarrow$) Asumamos que $\langle \tau, \sigma \rangle$ es un modelo de evaluación de φ y que $\tau = \mathit{alinear}(i_0, ..., i_N)$, $\sigma = \langle \sigma_1, ..., \sigma_n \rangle$, para algunas predicciones de entrada $i_0, ..., i_N$. Existen dos casos.

Si $N = 0$, definimos la predicción p_0 tal que:

- $p_0.\texttt{prev}_{con} = \bot$, $p_0.\texttt{prev}_{abs} = \bot$
- $p_0.\texttt{actual}(j) = \sigma_j(0)$, para $j \leq n$
- $p_0.\texttt{prox}_{con} = \bot$ y $p_0.\texttt{prox}_{abs} = \bot$

Es decir, los desplazamientos positivos y negativos, son reemplazados por sus valores por defecto y las referencias actuales son reemplazadas por los valores de $p_0.\texttt{actual}$ y i_0. Luego, $\sigma = \mathit{alinear}(p_0.\texttt{actual})$ y (c) se satisface. Además, como $\langle \tau, \sigma \rangle$ es un modelo de evaluación, p_0 satisface (d). Entonces, por equivalencia entre $[\![\cdot]\!]$ y $[\![\cdot]\!]_{pred}$, $p_0.\texttt{actual}(j) \leftrightarrow [\![e_\ell]\!]^{(i_0,p_0)}_{pred}$ para todo $j \leq n$, implicando $i_0 \rightleftharpoons p_0$ y satisfaciendo (a). Finalmente, p_0 es una predicción inicial y final, y (b) se cumple.

Si $N > 0$, entonces podemos definir p_j naturalmente como:

- Para j =0:
  - $p_0.\texttt{prev}_{con} = \bot = p_0.\texttt{prev}_{abs}$
  - $p_0.\texttt{actual}(j) = \sigma_j(0)$, para $j \leq n$
  - $p_0.\texttt{prox}_{con} = p_1.\texttt{actual}$, $p_0.\texttt{prox}_{abs} = p_{(0\hat{+}_\eta 1)}.\texttt{actual}$
- Para $j \in \{1, ..., N-1\}$:
  - $p_j.\texttt{prev}_{con} = p_{j-1}.\texttt{actual}$, $p_j.\texttt{prev}_{abs} = p_{(j\hat{+}_\eta -1)}.\texttt{actual}$
  - $p_0.\texttt{actual}(j) = \sigma_j(0)$, para $j \leq n$
  - $p_j.\texttt{prox}_{con} = p_{j+1}.\texttt{actual}$, $p_j.\texttt{prox}_{abs} = p_{(j\hat{+}_\eta 1)}.\texttt{actual}$
- Para $j = N$:
  - $p_N.\texttt{prev}_{con} = p_{N-1}.\texttt{actual}$, $p_N.\texttt{prev}_{abs} = p_{(N\hat{+}_\eta -1)}.\texttt{actual}$
  - $p_0.\texttt{actual}(j) = \sigma_j(0)$, para $j \leq n$
  - $p_N.\texttt{prox}_{con} = \bot = p_N.\texttt{prox}_{abs}$

Luego, p_0 es una predicción inicial, $p_1, ..., p_{N-1}$ son predicciones intermedias y p_N es una predicción final. Además, como fue probado anteriormente $[\![e_\ell]\!](j) = [\![e_\ell]\!]^{(i_j,p_j)}_{pred}$, para todo $\ell \leq n, j \ \leq N$, y entonces $i_j \rightleftharpoons p_j$ para todo $j \leq N$, y $\sigma = alinear(p_0.\texttt{actual}, ..., p_N.\texttt{actual})$. □

Lema 3.8. *Sea una especificación NSRV booleana φ. Si $\mathcal{L}(\mathcal{A}_d) \neq \emptyset$ entonces φ está sobredefinida.*

Demostración. Asuma que $\mathcal{L}(\mathcal{A}_d) \neq \emptyset$ y sea $\alpha = i_0, i_1, ...$ una palabra en $\mathcal{L}(\mathcal{A}_d)$. Entonces existe una secuencia $\pi = s_0, s_1, ...$ de estados de la estructura de Kripke inducida $K_{\mathcal{A}_d}$ tal que $\kappa(s_j) = i_j$, $s_{j+1} \ \in \delta(s_j)$ para todo $j \in \mathbb{N}$, $s_0 \in \mathit{Inicio}$ y para infinitos $j \ \in \mathbb{N}$, $s_j \in F_1^{\#}$. Si $s_j = (z_j, v_j)$ para todo $j \in \mathbb{N}$, $\nu = v_0, v_1, ...$ es una secuencia de vértices de $\mathcal{A}_d$ tal que $\eta(v_j) = i_j$, $v_{j+1} \in \delta(v_j)$ para todo $j \in \mathbb{N}$, $v_0 \in \mathit{inicio}$ y para infinitos $j \in \mathbb{N}$, $v_j \in F_1$.

Dado que los vértices de aceptación en $\mathcal{A}_p$ son vértices terminales, existe un mínimo k tal que $v_k \in F_1$ y $v_j = v_k$ para todo $j \geq k$. Supongamos que φ no

está sobredefinida, y sea $\tau = alinear(i_0, ..., i_k, \hat{i}_{k+1}, ..., \hat{i}_N)$ una traza estructurada de longitud $N + 1 > k$. Entonces existe un modelo de evaluación de φ para τ, y por Lema 3.7 existen predicciones de salida $p_0, ..., p_k, p_{k+1}, ..., p_N$ coherentes con la traza τ. Pero si p_k y p_{k+1} son coherentes entonces existe $v_{k+1} \in V$ tal que $v_{k+1} \in \delta(v_k)$ y $p_{k+1} \in v_{k+1}.O$, y entonces $v_k \notin F_1$ y k no es mínimo, lo cual es una contradicción. Entonces no existe modelo de evaluación de φ para la traza τ y φ está sobredefinda. □

Lema 3.9. *Sea φ una especificación NSRV booleana. Si φ está sobredefinida entonces $\mathcal{L}(\mathcal{A}_d) \neq \emptyset$.*

Demostración. Asuma que φ está sobredefinida. Entonces existe una traza estructurada τ de alguna longitud $N + 1$ tal que no existe modelo de evaluación φ para τ. Sean $i_0, ..., i_N$ las predicciones de entrada que forman τ, es decir $\tau = alinear(i_0, ..., i_N)$. Entonces por Lema 3.7 no existen predicciones de salida $p_0, ..., p_N$ coherentes con τ, porque en caso contrario existiría un modelo de evaluación de φ. Entonces, existe un k mínimo tal que algunos $p_0, ..., p_{k-1}$ satisfacen las condiciones del Lema 3.7 y no existe predicción p_k tal que $p_0, ..., p_{k-1}, p_k$ también las satisfacen.

Si $k = 0$, considere el vértice $v = (i_0, \emptyset)$ del conjunto $inicio \cap F_1$. Entonces $v \in \delta(v)$ y es un nodo inicial, por lo tanto $\alpha = \eta(v)\eta(v)\eta(v)...$ es una palabra en $\mathcal{L}(\mathcal{A}_d)$.

Si $k > 0$, por definición de la función de transición δ, del conjunto de vértices V y por Lema 3.7, existen vértices $v_0, ..., v_{k-1}$ tales que $p_j \in v_j.P$ y $v_{j+1} \in \delta(v_j)$ para todo $j < k - 1$. Ahora, dado que tal p_k no existe, si $v_k \in \delta(v_k)$ necesariamente $v_k.P = \emptyset$. Luego, el vértice $v_k = (i_k, \emptyset)$ es tal que $v_k \in \delta(v_{k-1})$ y $v_k \in F_1$, entonces existe una secuencia infinita de vértices $\nu = v_0, ..., v_{k-1}, v_k, v_k, ...$ y, por lo tanto, $\alpha = \eta(v_o)...\eta(v_{k-1})\eta(v_k)\eta(v_k)\eta(v_k)...$ es una palabra de $\mathcal{L}(\mathcal{A}_d)$. Luego, $\mathcal{L}(\mathcal{A}_d) \neq \emptyset$. □

Teorema 3.10. *Sea φ una especificación NSRV booleana. Entonces:*

$$\varphi \text{ está sobredefinida} \iff \mathcal{L}(\mathcal{A}_d) \neq \emptyset$$

Demostración. Consecuencia directa de los Lemas 3.8 y 3.9. □

3.3. Especificaciones mixtas

En las secciones previas se ha demostrado que el problema de buena definición para especificaciones con enteros, sin restricción alguna, es indecidible. Por otra parte, se ha demostró que el problema es decidible para especificaciones que sólo utilizan valores booleanos. En esta sección se caracteriza un conjunto que especificaciones que describen valores enteros y booleanos y que, bajo ciertas restricciones, el problema de buena definición es aún decidible.

Definición 3.11 (Especificaciones Dominadas por Enteros)**.** *Una especificación NSRV φ es dominada por enteros si en su grafo de dependencia $\mathcal{G}_\varphi$ no contiene arcos $v \xrightarrow{w} v'$ tales que v es una variable booleana y v' es una variable entera.*

Definición 3.12 (Subrafos de dependencia)**.** *Sea φ una especificación NSRV dominada por enteros con grafo de dependencia $\mathcal{G}_\varphi = \langle V, E\rangle$. Sea $\{V_{Int}, V_{Bool}\}$ la partición de V tal v es una variable entera si y sólo si $v \in V_{Int}$ y v es una variable booleana si y sólo si $v \in V_{Bool}$. El subgrafo entero de dependencia $\mathcal{G}_{Int}$ y el subgrafo booleano de dependencia $\mathcal{G}'_{Bool}$ se definen como:*

$$\mathcal{G}_\varphi^{Int} = \langle V_{Int}, E_{Int}\rangle, \text{ donde } E_{Int} = E \setminus \{v \xrightarrow{w} v' \mid v \in V_{Int}, v' \in V_{Bool}\}$$

$$\mathcal{G}_\varphi^{Bool} = \langle V_{Bool}, E_{Bool}\rangle, \text{ donde } E_{Bool} = \{v \xrightarrow{w} v' \mid v, v' \in V_{Bool}\}.$$

Es decir, el subgrafo entero de dependencia de una especificación φ dominada por enteros elimina aquellos arcos que tienen como destino variables booleanas. Sin embargo, es importante notar que si existe un camino cerrado en el grafo de dependencia $\mathcal{G}_\varphi$ también existe en $\mathcal{G}_\varphi^{Int}$ o $\mathcal{G}_\varphi^{Bool}$, pues todo camino que alcanza una variable booleana termina en una variable del mismo tipo, y entonces ningún camino cerrado existente en $\mathcal{G}_\varphi$ es omitido al quitar tales arcos.

Nótese además que al ser el grafo de dependencia un multigrafo, los subgrafos enteros y booleanos también lo son.

Lema 3.13. *Sea φ una especificación NSRV. Si φ es dominada por enteros y el subgrafo entero de dependencia está bien formado, entonces el problema de buena definición para φ es decidible.*

Demostración. Sea $\mathcal{G}_\varphi = \langle V, E\rangle$ el grafo de dependencia de φ y $\mathcal{G}_\varphi^{Int} = \langle V_{Int}, E_{Int}\rangle$ y $\mathcal{G}_\varphi^{Bool} = \langle V_{Bool}, E_{Bool}\rangle$ sus subgrafos entero y boolean respectivamente. El multigrafo $\mathcal{G}_\varphi^{Int}$ puede ser controlado para saber si la parte entera de la especificación está bien formada usando los resultados de los Teoremas 2.24, 2.25 y 2.26 como visto en la sección 2.4.1. Si no está bien definida, entonces claramente φ no está bien definida. Caso contrario, el multigrafo $\mathcal{G}_\varphi^{Bool}$ puede ser controlado de la misma manera. Note que a parte booleana puede ser considerada una especificación aparte puesto que no tiene dependencia de variables enteras. Luego, si el grafo de dependencia de la parte booleana, $\mathcal{G}_\varphi^{Bool}$, no satisface las propiedades de una especificación bien formada, entonces los algoritmos de decisión de la sección anterior para especificaciones booleanas pueden computarse para determinar si está bien definida. Si tanto la parte booleana como la parte entera resultan estar bien definidas, φ está bien definida. □

Note que el orden en el que se realicen los chequeos de ambos multigrafos es independiente, así como la parte booleana puede controlarse directamente con los algoritmos de la sección anterior, aunque pueda resultar más costoso computacionalmente.

Ejemplo 3.14. *Sea φ la siguiente especificación NSRV sobre las variables enteras $t_1, t_2, s_1, s_2, s_3, s_4$ y las variables booleanas t_3, s_5, s_6, s_7:*

$$s_1[n] = t_1[C(n) + 1|0] \ \texttt{mod}\ 5$$

$$s_2[n] = s_2[C(n) - 1|1] + \ (\textbf{\textit{if}}\ s_7[C(n) - 1|false]\ \textbf{\textit{then}}\ 5\ \textbf{\textit{else}}\ 0)$$

$$s_3[n] = (t_2[n] \neq 0) \ \vee \ s_4[A(n) + 3|\ true]$$

$$s_4[n] = s_1[n] + \ (\textbf{\textit{if}}\ s_5[n] \vee s_3[C(n) - 2|true]\ \textbf{\textit{then}}\ s_1[C(n) + 1|0]\ \textbf{\textit{else}}\ 0)$$

$$s_5[n] = \textbf{\textit{if}}\ t_{en}[n]\ \textbf{\textit{then}}\ true\ \textbf{\textit{else}}\ \neg s_5[A(n) - 1|true]$$

$$s_6[n] = \textbf{\textit{if}}\ t_{en}[n] \wedge \neg s_5[n]\ \textbf{\textit{then}}\ s_7[A(n)+3|true]\ \textbf{\textit{else}}\ s_6[C(n)-1|false]$$

$$s_7[n] = t_3[n] \vee s_6[A(n)-1|false]$$

cuyo grafo de dependencia es ilustrado por la Figura 3.2, donde se puede observar que φ es dominada por enteros. Además, φ no está bien formada ya que $s_6 \overset{+3}{\cdots\!\!>} s_7 \overset{-1}{\cdots\!\!>} s_6$ es un camino abstracto cerrado que contiene arcos abstractos con pesos de distinto signo. Los multigrafos que representan la parte entera y booleana de la especificación están separados con una linea a puntos. La parte entera está claramente bien definida ya que, obviando los arcos hacia variables booleanas, el grafo de la parte entera satisface las propiedades de la definición 2.14. La parte booleana, en cambio, contiene el camino cerrado mencionado. Sin embargo, uno puede notar que la dependencia de s_6 en s_7 es espuria puesto que nunca puede darse que $\tau_{en}(i)$ sea verdadero y $\sigma_5(i)$ sea falso en el mismo instante i, para cualquier traza y modelo de evaluación. Luego, los algoritmos de decisión de la Sección 2.6.2 indicarían que la especificación dada por la parte booleana de φ está bien definida. Es decir, φ en su totalidad está bien definida por el Lema 2.20.

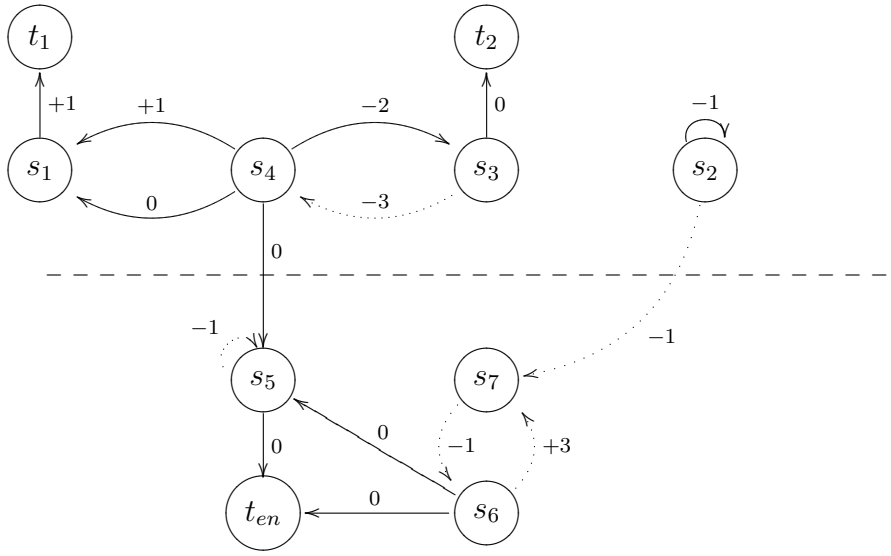

Figura 3.2: Grafo de dependencia de la especificación dominada por enteros del Ejemplo 2.39

Capítulo 4

Aplicaciones y ejemplos

"A good example has twice the value of good advice."

NSRV tiene sus raíces en el lenguaje de especificación LOLA, el cual fue diseñado para verificar sistemas síncronos. Sin embargo, es posible adaptar la estrategia de verificación de NSRV y LOLA para sistemas asíncronos, utilizando interrupciones, señales o mensajes como *tic* del reloj en reemplazo a aquel que estaría naturalmente dado en un sistema síncrono. Lo único que requiere el algoritmo de evaluación es obtener en cada *tic* del reloj (posición de la traza) los valores de los streams de entrada, claro que a la hora de escribir las especificaciones es menester que tales instantes estén bien definidos, pues de ello dependen los desplazamientos o constructores condicionales usados para definir las propiedades deseadas.

A continuación presentamos un conjunto de especificaciones ejemplo que muestran la expresividad del lenguaje y cómo contribuye a la descripción de propiedades de manera intuitiva, precisa y clara. Es importante destacar que todas las propiedades y operadores especificados aquí son eficientemente monitoreables y que aquellos requerimientos usualmente encontrados literatura

tienen una especificación eficientemente monitoreables que los describe.

Pre y post condiciones. En los formalismos clásicos usados para verificación como las lógica de Hoare, la corrección de procedimientos se expresa usando pre y post condiciones [12]. La corrección total de un procedimiento A establece que si la precondición p es válida cuando A es invocado, entonces el procedimiento debe terminar y la post condición q debe ser válida en su retorno. La corrección parcial simplemente requiere que *si* el procedimiento termina, entonces q se satisfaga en su retorno, lo cual debido a la semántica de NSRV sobre streams finitos no es expresable. Sin embargo, s_p y s_q son variables dependientes que describen la pre y post condición y t_{call_A} es una variable independiente booleana que denota la llamada al módulo A, la corrección total de A puede especificarse como sigue:

$$s_{total}[n] = (t_{call_A}[n] \wedge s_p[n]) \rightarrow s_q[A(n) + 1|false]$$

Bajo condiciones normales el módulo A terminará su ejecución y entonces $s_q[A(n) + 1|false]$ será cierto si y sólo si q se satisface en su retorno. Si la traza terminara abruptamente antes que A finalice su ejecución, la propiedad no debería ser válida, lo cual se establece mediante el valor por defecto $false$.

Una respuesta para cada pedido, versión mejorada. Una de las propiedades comúnmente deseadas en un sistema y usualmente encontrada en la literatura es aquella que establece *"cada pedido tendrá eventualmente una respuesta"*, que puede relacionarse de manera análoga a propiedades como *"cada alocación de memoria tiene eventualmente una liberación"* o *"cada envío tendrá su respuesta en algún momento"*. Suponiendo que *enviar* es el evento que denota el envío de un mensaje y *recibir* aquel que indica la recepción de un mensaje, en LTL y otras lógicas similares este requerimiento se escribiría sencillamente como $\Box(enviar \rightarrow \diamond recibir)$. Sin embargo, aquellos familiarizados con este formalismo sabrán que la correspondencia entre eventos descripta en la propiedad no tiene por qué ser 1 a 1, y mucho menos

que el lenguaje que denotan ambos eventos sea un lenguaje de paréntesis balanceados. Es decir, esta fórmula no implica que sucedan igual cantidad de eventos de cada tipo, o que nunca se reciben más respuestas que los envíos realizados hasta el momento. Dicho de otra manera, independientemente de la cantidad de envíos que se realicen, cualquier traza en la cual el último evento en suceder es *recibir* satisface esta especificación, lo cual claramente puede estar muy alejado del objetivo de la verificación.

En NSRV, la propiedad deseada puede ser especificada, e inclusive puede requerirse que la propiedad se cumpla para cada módulo. Considere la siguiente especificación NSRV, donde las variables independientes t_{E} y t_{R} denotan los eventos de envío y respuesta respectivamente:

$$s_{E}[n] = (\textbf{if } t_{en}[n] \textbf{ then } 0 \textbf{ else } s_{E}[A(n)-1|0]) + (\textbf{if } t_{E}[n] \textbf{ then } 1 \textbf{ else } 0)$$

$$s_{R}[n] = (\textbf{if } t_{en}[n] \textbf{ then } 0 \textbf{ else } s_{R}[A(n)-1|0]) + (\textbf{if } t_{R}[n] \textbf{ then } 1 \textbf{ else } 0)$$

$$s_{bal}[n] = (s_{E}[n] - s_{R}[n]) \geq 0$$

$$s_{prop}[n] = (t_{ex}[n] \rightarrow s_{E}[n] = s_{R}[n]) \wedge s_{bal}[n]$$

$$\textbf{trigger}(\neg\ s_{prop}[n])$$

Las variables s_{E} y s_{E} definen contadores para los eventos en observación para cada contexto: cada vez que comienza la ejecución de un procedimiento el valor acumulado es olvidado. Además, dado que el desaplazamiento es realizado sobre el camino abstracto de la traza, los valores en los subprocedimientos son abstraídos, y entonces cada contexto sigue la cuenta desde la última llamada en casa que un módulo sea ejecutado. La variable s_{bal} define un stream cuyo valor en cada posición es *true* si y sólo si la cantidad total de envíos realizados hasta ese momento es mayor o igual a la cantidad de respuestas obtenidas. Finalmente, el valor del stream denotado por s_{prop} en cada posición i es **true** si y sólo si $\sigma_{bal}(i) =$ **true** y en cada salida de un módulo, los valores de los contadores para cada evento tienen igual valor. Luego, al monitorear esta especificación, se generará una notificación sólo

cuando el lenguaje definido por los envíos y respuestas en el contexto de un módulo no pueda un lenguaje de paréntesis equilibrado, o cuando a la salida de un módulo la cantidad de envíos y respuestas sea distinta.

En ocasiones resulta de interés controlar propiedades sólo bajo cierto contextos de ejecución. Por ejemplo, podría querer especificarse que la propiedad descrita anteriormente se cumpla dentro del módulo A. Si t_{id} es una variable independiente entera que identifica el módulo en ejecución y id_A es el identificador del módulo A, entonces la anterior especificación puede hacerse *local* a A simplemente modificando s_{prop}:

$$s_{_E}[n] = (\textbf{if } t_{en}[n] \textbf{ then } 0 \textbf{ else } s_{_E}[A(n)-1|0]) + (\textbf{if } t_{_E}[n] \textbf{ then } 1 \textbf{ else } 0)$$

$$s_{_R}[n] = (\textbf{if } t_{en}[n] \textbf{ then } 0 \textbf{ else } s_{_R}[A(n)-1|0]) + (\textbf{if } t_{_R}[n] \textbf{ then } 1 \textbf{ else } 0)$$

$$s_{bal}[n] = (s_{_E}[n] - s_{_R}[n]) \geq 0$$

$$s_{prop}[n] = (t_{id}[n] = id_A) \rightarrow [(t_{ex}[n] \rightarrow s_{_E}[n] = s_{_R}[n]) \wedge s_{bal}[n]]$$

$$\textbf{trigger}(\neg\ s_{prop}[n])$$

Ejecución segura. Muchas veces resulta importante o hasta requerido garantizar que cierta acción ha sido realizada antes de ejecutar un módulo crítico, y no basta sólo con saber que ha sido realizada previamente sino que a veces es necesario que el módulo se ejecute bajo un contexto que asegure las condiciones adecuadas. Por ejemplo, podría requerirse que una función de transferencia sea ejecutada sólo en una comunicación establecida con encriptación o que el cerrojo de un fichero ha sido efectivamente obtenido antes de iniciar su escritura. En este sentido, resulta útil muchas veces poder asegurar que un módulo aún se encuentra en la pila de ejecución al momento de realizar la llamada al módulo crítico.

Sea t_{id} la variable independiente entera que denota el identificador de cada módulo que puede ser ejecutado y sea 0 el valor para aquellos módulos sin interés. Si el valor de t_{id} en el instante de una llamada es exactamente

el identificador del módulo invocado, se puede considerar la siguiente especificación NSRV que asegura la ejecución del módulo identificado por M_{crit} bajo el contexto del módulo seguro M_{seg}:

$$s_{pila}[n] = \textbf{if } t_{en}[n] \wedge (t_{id}[n-1|0] = M_{seg}) \textbf{ then } true \textbf{ else } s_{pila}[A(n)-1|\ false]$$

$$s_{crit}[n] = \textbf{if } t_{call}[n] \wedge (t_{id}[n] = M_{crit}) \textbf{ then } s_{pila}[n] \textbf{ else } true$$

$$\textbf{trigger}(\neg\ s_{crit}[n])$$

En esta especificación, la variable s_{pila} denota si el contexto actual se encuentra dentro del contexto de M_{seg}, que asegura las condiciones para la ejecución del módulo M_{crit}. En la entrada del módulo M_{seg} su valor es claramente $true$, en el resto de las posiciones su valor es exactamente el de la posición predecesora abstracta, o $false$ si es el comienzo de la traza. La variable s_{crit} y el disparador cercioran que siempre que el módulo M_{crit} sea llamado, M_{seg} se encuentra en la pila de ejecución, sin importar el nivel en la pila en que se encuentre.

Solo f llama a g. De manera similar al ejemplo anterior, se puede requerir que una función puede ser llamada únicamente y de manera directa desde un módulo o función determinada. Esto es especialmente útil para restringir el acceso a regiones críticas en concurrencia o para garantizar la abstracción en capas de un protocolo de comunicación.

Nuevamente, sea t_{id} la variable entera independiente que hace referencia al identificador del módulo llamado cada vez que ocurre una llamada. Sean f_{id} y g_{id} los valores enteros que identifican a las funciones f y g respectivamente, y 0 el valor por defecto que no identifica ninguna función, entonces la siguiente especificación NSRV provee una manera de controlar la propiedad deseada:

$$s_{pila}[n] = \textbf{if } t_{en}[n] \textbf{ then } t_{id}[n-1|0] = f_{id} \textbf{ else } s_{pila}[A(n)-1|\ false]$$

$$s_{f,g}[n] = \textbf{if } t_{call}[n] \wedge (t_{id}[n] = g_{id}) \textbf{ then } s_{pila}[n] \textbf{ else } true$$

$$\textbf{trigger}(\neg\ s_{f,g}[n])$$

El stream descrito por la ecuación de s_{pila} es $true$ en una posición si y sólo si la última llamada sin retorno es la de f: si se invoca un módulo y es f entonces es $true$; y si el módulo no es f entonces el valor del stream es $false$. En las posiciones que no son de entrada basta con tomar el valor en la posición previa asbtracta. La variable $s_{f,g}$ usada con el disparador controla que siempre que g sea llamada, f debe ser la primer función en la pila de ejecución o, equivalentemente, que el control del programa se encuentra en f al momento de realizar la llamada.

Recursión acotada. En aquellos lenguajes de programación que proveen recursión, un error de implementación puede llevar a una recursión no acotada y por ende al consumo exahustivo de la memoria en el peor escenario. Esto puede causar un error no sólo en el programa problemático sino además puede afectar a otros sistemas en el caso de recursos compartidos o servidores. Como en los últimos dos ejemplos, si la variable entera independiente t_{id} identifica al módulo llamado cada vez que ocurre una llamada, se puede acotar la recursión directa a K llamadas recursivas para cualquier función con la siguiente especificación:

$$s_{pila}[n] = \textbf{if } t_{en}[n] \textbf{ then } t_{id}[n-1|0] \textbf{ else } s_{pila}[A(n)-1|\ false]$$

$$\begin{aligned} s_{nivel}[n] = \textbf{if } & t_{call}[n] \\ & \textbf{then } (\textbf{if } t_{id}[n] = s_{pila}[n] \textbf{ then } s_{nivel}[n-1|0] + 1 \textbf{ else } 0) \\ & \textbf{else } s_{nivel}[n-1|0] \end{aligned}$$

$$s_{rec}[n] = s_{nivel}[n] \leq K$$

$$\textbf{trigger}(\neg\ s_{rec}[n])$$

La varible s_{pila} simplemente indica el identificador del módulo en el tope de la pila de ejecución. Luego s_{nivel} cuenta el nivel de recursión directa en el

cual se encuentra el programa. Note que s_{nivel} se reinicia cuando el módulo llamado es distinto de aquel en el tope de la pila. Finalmente, el disparador usa s_{rec} para generar un alerta si el nivel de recursión directa supera K alguna vez.

Aunque esta especificación no sirve para modelar la recursión mutua acotada, también es posible especificarla de manera similar:

$$s_{pila_1}[n] = \textbf{if } t_{en}[n] \textbf{ then } t_{id}[n-1|0] \textbf{ else } s_{pila_1}[A(n)-1|\ false]$$

$$s_{pila_2}[n] = \textbf{if} t_{en}[n] \textbf{ then } s_{pila_1}[A(n)-1|0] \textbf{ else } s_{pila_2}[A(n)-1|\ false]$$

$$\begin{aligned} s_{nivel}[n] = \textbf{if } & t_{call}[n] \\ & \textbf{then } (\textbf{if } t_{id}[n] = s_{pila_2}[n] \textbf{ then } s_{nivel}[n-1|0]+1 \textbf{ else } 0) \\ & \textbf{else } s_{nivel}[n-1|0] \end{aligned}$$

$$s_{rec}[n] = s_{nivel}[n] \leq K$$

$$\textbf{trigger}(\neg\ s_{rec}[n])$$

Las variables s_{pila_1} y s_{pila_2} denotan los identificadores de las últimas funciones en la pila, donde el 0 es usado como elemento nulo. Entonces la variable s_{nivel} cuenta la cantidad de veces que una función ha sido llamada por aquella dos niveles más arriba en el nivel de la pila. Note que el conteo se reinicia cada vez que otra invocación ocurre. La conjugación del disparador con s_{rec} simplemente genera una notificación cuando el nivel de recursión supera K.

Uso de información de subcontextos. A pesar de que los desplazamientos abstractos saltan de la invocación de un módulo a su retorno y viceversa, el hecho de poder combinar desplazamientos concretos con desplazamientos abstractos y valores actuales permite expresar propiedades que se satisfagan a la salida de un módulo pero que a su vez utilice la información recolectada en los subcontextos, o simplemente recolectar la información un procedimiento si alguna condición se cumple. Por ejemplo, supongamos que queremos saber *"cuantas veces se invoca a f en toda la traza, contando solo*

las llamadas que ocurren en procedimientos que cumplen su precondición pero no su post condición". Si bien suena complicado, presentamos la siguiente especificación, donde se asume que las variables s_{pre} y s_{post} definen la pre y post condición de un módulo:

$$s_f[n] = \textbf{if } t_{en}[n] \textbf{ then } 0 \textbf{ else } s_f[A(n) - 1|0] + (\textbf{if } t_{call}[n] \wedge (t_{id}[n] = f_{id}) \textbf{ then } 1 \textbf{ else } 0)$$

$$s_{cond}[n] = t_{ret}[n] \wedge s_{pre}[A(n) - 1|true] \wedge \neg s_{post}[n]$$

$$s_{ac}[n] = s_{ac}[A(n) - 1|0] + (\textbf{if } s_{cond}[n] \textbf{ then } s_f[n - 1|0] \textbf{ else } 0)$$

La variable s_f representa la cantidad de llamadas a f realizadas en el contexto actual. En cada entrada a un módulo el contador se reinicia y se suma 1 si ocurre una llamada y si la función invocada es f. Gracias al desplazameinto abstracto, la recolección de llamadas hechas en subcontextos es abstraída. El valor del stream representado por s_{cond} es *true* en una posición i si y sólo si i es un retorno y el módulo que retorna satisface su precondición pero no su post condición. Finalmente, s_{ac} combina los desplazamientos concretos y abstractos para acumular solo las llamadas invocaciones de f que suceden en los módulos de interés: primero toma el valor anterior sobre el camino abstracto para evitar acumular valores indeseados y luego, si es el retorno de un módulo de interés, acumula la información obtenida en los subcontexto haciendo uso de un desplazameinto concreto.

Otros operadores. Havelund y Roşu presentaron en 2004 algoritmos para verificar eficientemente propiedades LTL con operadores pasados [11] y expusieron operadores de igual expresividad que los operadores básicos pero más intuitivos y frecuentemente útiles en verificación de requerimientos en tiempo de ejecución. Damos a continuación la definición de tales operadores en NSRV.

- Dada una fórmula LTL pasada F, el operador $\uparrow F$ se lee como *"empieza F"*; dice que la fórmula F ha empezado a ser verdadera, es decir, fue falsa en el instante anterior y es verdadera ahora. De manera dual, el operador $\downarrow F$ se lee *"termina F"* y establece que F fue verdadera en el instante anterior y es falsa ahora. Si s_F es una variable dependiente que define la propiedad de F, estos operadores pueden definirse en NSRV simplemente como:

$$s_{\uparrow}[n] = \neg s_F[n-1|true] \wedge s_F[n]$$
$$s_{\downarrow}[n] = s_F[n-1|false] \wedge \neg s_F[n]$$

  Nótese que ninguno de los streams de salida denotados por $s_{\uparrow}$ o $s_{\downarrow}$ es *true* en el primer instante de tiempo, pues se requiere que la validez de F *cambie*. Si se quisiera relajar esta condición para que $\uparrow F$ se satisfaga también en la primera posición, bastaría con cambiar el valor por defecto:

$$s_{\uparrow}[n] = \neg s_F[n-1|false] \wedge s_F[n]$$

- Los operadores $[F_1, F_2)_s$ y $[F_1, F_2)_w$ se interpretan como *"intervalo F_1, F_2 estricto/laxo"* e intuitivamente establecen que F_1 fue verdadera en el pasado y que desde entonces F_2 no ha sido verdadera, incluyendo aquel momento. El operador estricto $[F_1, F_2)_s$ exige que F_1 haya sucedido alguna vez en el pasado, mientras que el operador laxo $[F_1, F_2)_w$ no. Asumiendo que las variables s_{F_1} y s_{F_2} describen las propiedades expresadas por F_1 y F_2 respectivamente, los operadores se pueden definir como a continuación:

$$s_{past_1}[n] = s_{F_1}[n] \vee s_{past_1}[n-1|false]$$
$$s_{1-2}[n] = \textbf{if } s_{past_1}[n] \textbf{ then } s_{F_2}[n] \textbf{ else } s_{1-2}[n-1|false]$$
$$s_{[)_s}[n] = s_{past_1}[n] \wedge \neg s_{1-2}[n]$$
$$s_{[)_w}[n] = s_{[)_s}[n] \vee \neg s_{past_1}[n]$$

Aquí, la variable s_{past_1} denota si F_1 ha valido alguna vez, incluyendo el momento actual, y s_{1-2} denota si F_2 ha sido satisfecha luego o en el mismo instante que F_1 lo haya sido.

Por otro lado, en 2008, Roşu, Cheng y Ball presentaron por otra parte ptCaRet [17], una extensión de LTL con llamadas y retornos, posiblemente el formalismo más cercano a NSRV expresivamente. En su trabajo se define un conjunto de operadores derivados que resultan interesantes. Damos a continuación la definición de tales operadores en NSRV.

- *Al comienzo* y *en la llamada*: Suponga que uno quiere que cierta propiedad, descrita por la variable s_ψ, sea verdadera al comienzo de la ejecución del contexto actual, es decir, a la entrada del contexto. Esta propiedad puede especificatse de manera sencilla utilizando un desplazamiento abstracto:

$$s_{@_{en}\psi}[n] = \textbf{if } t_{en}[n] \textbf{ then } s_\psi[n] \textbf{ else } s_{@_{en}\psi}[A(n) - 1|true]$$

Si la posición actual es una entrada a un módulo, entonces el valor del stream que corresponde a $s_{@_{en}\psi}[n]$ debe ser igual al valor de $s_\psi[n]$ en esa posición. En cambio, si la ejecución se encuentra en cualquier otra posición distinta a una entrada, el valor es igual al de la posición anterior sobre el camino abatracto de la traza, implicando que los subcontextos son abstraídos y asegurando que el valor de $s_{@_{en}\psi}[n]$ en cada instante se corresponde con el contexto actual.

De manera similar se puede definir el operador *en la llamada*, el cual describe que ψ es verdadera en la llamada del módulo actual:

$$s_{@_{call}\psi}[n] = \textbf{if } t_{en}[n] \textbf{ then } s_\psi[n-1|false] \textbf{ else } s_{@_{call}\psi}[A(n)-1|true]$$

En este caso no se utiliza el valor de s_ψ en la entrada, sino en el instante anterior: en la llamada al módulo.

- Propiedades sobre la pila: Existen casos en los cuales es deseable expresar propiedades que se refieren exclusivamente a la pila de ejecución de un programa, ignorando cualquier otro estado. Por ejemplo, uno podría querer expresar que una propiedad ψ fue verdadera en la pila desde que ψ' fue verdadera en ella. Como es usual, uno podría estar interesado en que ψ y ψ' sean verdaderas en la llamada del módulo o en el comienzo de su ejecución. Si s_ψ y $s_{\psi'}$ son las variables dependientes que describen las propiedades ψ y ψ' respectivamente, estos operadores pueden definirse en NSRV como sigue.

$$
\begin{aligned}
s_{\psi'_{past}}[n] &= (t_{en}[n] \wedge s_{\psi'}[n]) \vee s_{\psi'_{past}}[A(n)-1|false] \\
s_{\psi S_{en} \psi'}[n] &= \textbf{if } t_{en}[n] \wedge s_{\psi'_{past}}[n] \\
&\quad \textbf{then } s_{\psi'}[n] \\
&\quad \textbf{else } s_{\psi S_{en} \psi'}[A(n)-1|\ true]
\end{aligned}
$$

La variable $s_{\psi'_{past}}[n]$ describe si ψ' ha sucedido alguna vez en el pasado en la entrada de un contexto anterior en la pila de ejecución. Luego, la ecuación para $s_{\psi S_{en} \psi'}$ se define usando esa información: si ψ' ha sucedido en el pasado y es una entrada a un nuevo contexto, entonces ψ debe cumplirse; caso contrario, el valor anterior (omitiendo subcontextos mediante un desplazamiento abstracto) es usado. Nuevamente, el operador que controla las propiedades en la llamada se define de manera similar:

$$
\begin{aligned}
s_{\psi'_{past}}[n] &= (t_{en}[n] \wedge s_{\psi'}[n-1|false]) \vee s_{\psi'_{past}}[A(n)-1|false] \\
s_{\psi S_{call} \psi'}[n] &= \textbf{if } t_{en}[n] \wedge s_{\psi'_{past}}[n] \\
&\quad \textbf{then } s_{\psi'}[n-1 \mid false] \\
&\quad \textbf{else } s_{\psi S_{call} \psi'}[A(n)-1|\ true]
\end{aligned}
$$

En este caso se usan las expresiones $s_{\psi'}[n-1|false])$ y $s_{\psi'}[n-1 \mid false]$ para obtener el valor de $s_{\psi'}$ y s_ψ en el momento de la llamada. En caso de que se trate del primer instante de tiempo, las propiedades se asumen no válidas.

Capítulo 5

Inevitabilidad

En este capítulo se presenta el concepto de *inevitabilidad* sobre especificaciones NSRV booleanas. Este concepto captura el hecho de que dada una traza estructurada parcial inicial τ de un sistema, una variable dependiente s_j de una especificación booleana puede denotar el mismo valor b para toda posible continuación de τ. Dicho de otra manera, el valor denotado por s_j habiendo observado el prefijo τ será inevitablemente b, no cambiará en el futuro. Siendo capaces de computar tal propiedad sobre variables dependientes permite a una especificación activar un disparador de manera anticipada, y por lo tanto reaccionar a eventualidades tan pronto como es posible.

5.1. Introducción

Un problema en verificación en tiempo de ejecución con fórmulas de lógica temporal lineal (LTL) es que los modelos de la lógica son trazas infinitas, y durante la ejecución de un sistema, uno cuenta sólo con un prefijo finito del comportamiento del mismo. Surge entonces la pregunta *"¿cuándo una fórmula que hace referencia al futuro es necesariamente cierta, independientemente de aquellas entradas de las que aparentemente depende?"*. Si bien LTL con su sintaxis y semántica sobre trazas infinitas es muy aceptada en

la literatura, no existe una opinión homogénea sobre una definición de LTL para trazas finitas.

Diferentes versiones sobre semánticas de *dos* valores para LTL han sido propuestas, pero tres situaciones pueden ocurrir al monitorear una propiedad en tiempo de ejecución: (a) la propiedad ya es satisfecha luego de algún prefijo finito de la traza, (b) la propiedad es falsa para toda posible continuación, o (c) el prefijo observado tiene tanto continuaciones que hacer la propiedad verdadera como continuaciones que hacen la propiedad falsa. Bajo esta premisa, ninguna semántica con dos valores es suficiente pues el tercer caso no puede ser expresado apropiadamente. Para superar este obstáculo, Vorbehalten et al. proponen una semántica de 3 valores para LTL [19] (verdadero, falso e inconcluso), donde la semántica de un prefijo u y una fórmula φ es verdadera si también lo es para uw' y φ, para todo posible sufijo infinito w'. La semántica es *inconclusa* si el prefijo u no es suficiente para determinar como evaluará φ en el futuro. Detrás de este enfoque se encuentra una noción de *anticipación*: la semántica evalúa a un valor de verdad tan pronto como se tiene certeza acerca del futuro. La inevitavilidad de variables dependientes en NSRV sigue esta idea y nace como un primer paso hacia la adaptación de semánticas de 3 valores [19] [5] o semánticas de 4 valores [5] para especificaciones NSRV booleanas. La posibilidad de reconocer anticipadamente cuándo una variable dependiente denotará el mismo valor para cualquier extensión de la traza parcial observada hasta el momento permite a una especificación utilizar esta información para generar una señal mediante un disparador y, por ejemplo, iniciar la ejecución un módulo de corrección del sistema monitoreado.

5.2. Definición

La inevitabilidad de variables dependientes en NSRV puede ser definida como sigue.

Definición 5.1 (Inevitabilidad)**.** *Sea φ una especificación NSRV booleana y bien definida sobre las variables dependientes $s_0, ..., s_r$ y τ una traza*

parcial inicial de longitud $M+1$ *y ancho* $m+1$*. Dado un valor constante* c *del mismo tipo que* s_j*, diremos que* s_j *es* inevitablemente c *a partir de la posición* M *si y sólo si para toda traza parcial final* τ' *de longitud* M' *tal que* $\tau\tau'$ *es una traza estructurada, el único modelo de evaluación* $\sigma = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_0, ..., \tau_m, \sigma_0, ..., \sigma_n \rangle$ *de* φ *para* $\tau\tau'$ *satisface que* $\sigma_j(i) = c$ *para todo* $M \leq j \leq M + M'$*.*

Ejemplo 5.2. *Considere la siguiente especificación NSRV booleana:*

$$s_1[n] = s_2[n] \wedge t_1[n]$$

$$s_2[n] = s_2[n+1|false]$$

Esta especificación no es eficientemente monitoreable y si uno evaluara esta especificación con el Algoritmo 1, el resultado de los streams de salida en la posición 0 serían resueltos únicamente al finalizar la traza. Sin embargo, es fácil darse cuenta de que el stream de salida denotado por s_2 *será falso en todas sus posiciones y por lo tanto el stream denotado por* s_1 *también lo será: son* inevitablemente *falsos.*

5.3. Problema de decisión y solución

Dada una especificación NSRV booleana φ sobre variables dependientes $s_0, ..., s_{r-1}$, definimos la especificación $\varphi_{s_j,b}$ sobre variable dependientes $s_0, ..., s_{r-1}, s_r$ como $\varphi_{s_j,b} = \varphi \cup \{s_r[n] = (s_j[n] = \neg b) \vee s_r[n+1|false]\}$. Es fácil ver que si φ está bien definida, entonces $\varphi_{s_j,b}$ también lo está, pues s_r es una variable que no existe en φ y por lo tanto no puede formar ciclos. Además, la variable s_r tiene un único valor bien definido en cada posición si s_j lo tiene.

Lema 5.3. *Sea* φ *una especificación NSRV booleana y bien definida sobre las variables dependientes* $s_0, ..., s_{r-1}$*,* τ *una traza parcial inicial de longitud* $M+1$ *y ancho* $m+1$*, y* b *una constante booleana. La variable* σ_j

es inevitablemente b a partir de M si y sólo si para toda traza parcial final τ' tal que $\tau\tau'$ es una traza estructurada, el único modelo de evaluación $\sigma = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_0, ..., \tau_m, \sigma_0, ..., \sigma_{r-1}, \sigma_r \rangle$ de $\varphi_{s_j,b}$ para $\tau\tau'$ satisface que $\sigma_r(M) = false$.

Demostración. Asumamos que s_j es inevitablemente b en φ para la traza parcial inicial τ de longitud $M+1$ y ancho $m+1$. Entonces para toda traza parcial final τ' de longitud M' tal que $\tau\tau'$ es una traza estructurada , el único modelo de evaluación $\sigma = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_0, ..., \tau_m, \sigma_0, ..., \sigma_{r-1} \rangle$ de φ para $\tau\tau'$ satisface que $\sigma_j(i) = b$ para todo $M \leq i \leq M + M'$. Luego, como $\varphi_{s_j,b}$ está bien definida y mantiene las ecuaciones para $s_0, ..., s_{r-1}$, el único modelo de evaluación $\sigma' = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_0, ..., \tau_m, \sigma_0, ..., \sigma_{r-1}, s_r \rangle$ de $\varphi_{s_j,b}$ para $\tau\tau'$ satisface que $\sigma_j(i) = b$ para todo $M \leq i \leq M + M'$ también. Entonces $[\![s_j[n] = \neg b]\!]_{\sigma'}(i) = false$ y luego $\sigma_r(i) = [\![s_r[n+1|false]]\!]_{\sigma'}(i) = \sigma_r(i+1)$ para todo $M \leq i < M + M'$. Debido a que $s_{r+1}(M + M') = false$ y que $\sigma_r(M) = \sigma_r(M+1) = ... = \sigma_r(M+M')$, se tiene que $s_r(M) = false$.

Ahora asumamos que para toda traza parcial final τ' tal que $\tau\tau'$ es una traza estructurada, el único modelo de evaluación $\sigma = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_0, ..., \tau_m, \sigma_0, ..., \sigma_{r-1}, \sigma_r \rangle$ de $\varphi_{s_j,b}$ para $\tau\tau'$ satisface que $\sigma_r(M) = false$. Luego, no existe posición $k \geq M$ tal que $\sigma_j(k) \neq b$, porque si existiera se tendría que $\sigma_r(k) = true$, y puesto que $\sigma_r(M) = \sigma_r(M+1) = ... = \sigma_r(k)$, obtendríamos $\sigma_r(M) = true$, lo cual es absurdo por hipótesis. Luego, como φ está bien definida y tiene las mismas ecuaciones que $\varphi_{s_j,b}$ para $s_0, ..., s_{r-1}$, s_j es inevitablemente b a partir de la posición M. □

A continuación construiremos, a partir de una especificación, un *RBA extendido*, similar a los RBA de la Sección 3.2, pero cada nodo tendrá sólo una predicción y, además, consideraremos módulos de una sola entrada y una sola salida. Es decir, el RBA extendido tendrá tantos módulos como posibles combinaciones de entradas y salidas existan, y cada módulo tendrá una caja por combinación de: un módulo del RBA, una posible llamada a ese módulo y un posible retorno del mismo. La extensión resulta de etiquetar los vértices de cada módulo. Un vértice del *RBA extendido* será etiquetado

como *finalizable* si puede alcanzar un vértice de aceptación del módulo al que pertenece y, por otro lado, será etiquetado como *extendible* si puede alcanzar la única salida del módulo al que pertenece.

Sea $\mathcal{D} = \{(i,p) \mid i \in \mathcal{I}^?, p \in \mathcal{G}^?, i \rightleftharpoons p\}$ el dominio de pares que contienen una predicción de entrada y una predicción de salida que son compatibles. Si $v = (i,p)$ es un elemento de $\mathcal{D}$, $v.i$ denotará a la predicción de entrada i, y $v.p$ denotará la predicción de salida p. Sean π_i las funciones de proyección usuales.

El RBA extendido sobre el alfabeto Σ es $\hat{\mathcal{A}}_\varphi = (S_\varphi,\ fin,\ ext)$, donde S_φ es un RBA, y $fin : V \to \mathbb{B}$ y $ext : V \to \mathbb{B}$ son las funciones que etiquetan los vértices de S como *finalizables* o *extendibles* respectivamente. El RBA S está definido como $S = (M, \{S_m\}_{m \in M},\ inicio,\ \mathcal{F})$, donde el conjunto de nombres de módulos codifica una entrada y un retorno:

$$M = \{(v_{en}, (v_{ex}, v_r)) \mid v_{en}, v_{ex}, v_r \in \mathcal{D},\ v_{en}.i \text{ es una entrada},\\ v_{ex}.i \text{ es una salida},\ v_r.i \text{ es un retorno}\}.$$

Cada módulo $S_m = (N_m,\ B_m,\ Y_m,\ En_m,\ Ex_m,\ \delta_m,\ \eta_m)$ se define como sigue:

- $N_m = \mathcal{D} \cup \{\pi_2(m)\}$

  El conjunto de nodos tiene todas las predicciones compatibles con sus entradas y una única salida, codificada en m.

- $B_m = \{(m', v_c, v_r) \mid$
  $m' = (v_{en}, (v_{ex}, v_r)) \in M,\ v_c,\ v_r \in \mathcal{D},$
  $llamadaRetorno(v_c.p, v_r.p),\ llamadaEntrada(v_c.p, v_{en}.p),$
  $salidaRetorno(v_{ex}.p, v_r.p),$
  $v_c.i$ es una llamada, $v_r.i$ es un retorno$\}$

  Cada módulo tiene tantas cajas como combinaciones entre módulos y formas de llamar y retornar de cada uno, y los componentes de cada caja tienen sus predicciones coherentes con respecto a la llamada, entrada, salida y retorno. Además, todos tienen sus predicciones compatibles.

- $Y_m((m', v_c, v_r)) = m'$
- $En_m = \{\pi_1(m)\}$
- $Ex_m = \{\pi_2(m)\}$

  Informalmente, la entrada y la salida de un módulo son aquellos que identifican al nombre del módulo.

- La función de transición δ_m está definida, para cada caso, como:

  - **De nodos de no aceptación a nodos de no salida**: $\delta_m(v)$ contiene a v' si y sólo si:
    - $concretoYabstracto(v.p, v'.p)$
  - **De nodos a llamadas**: $\delta_m(v)$ contiene a $((m'', v_c, v_r), v_{en})$, con $m'' = (v'_{en}, (v'_{ex}, v'_r))$ si y sólo si:
    - $v_{en}.i$ es una entrada
    - $v_{en} = v'_{en}$ (la entrada es la del módulo invocado)
    - $v_r = v'_r$ (el retorno de la caja forma parte de la única salida)
    - $concretoYabstracto(v.p,\ v_c.p)$
  - **De nodos a salidas**: $\delta_m(v)$ contiene a (v_{ex}, v_r) si y sólo si:
    - $v_{ex}.i$ es una salida y $v_r.i$ es un retorno
    - $concretoYabstracto(v.p,\ v_{ex}.p)$
    - $salidaRetorno(v_{ex}.p,\ v_r.p)$
  - **De retornos a nodos**: $\delta_m((m', v''_c, v''_r), (v_{ex}, v_r))$, con el nombre de módulo $m' = (v'_{en}, (v'_{ex}, v'_r))$ contiene a v si y sólo si:
    - $v_{ex}.i$ es una salida y $v_r.i$ es un retorno
    - $(v_{ex}, v_r) = (v'_{ex}, v'_r)$, o equivalentemente $(v_{ex}, v_r) = \pi_2(m')$ (se retorna exactamente como indica la caja)
    - $concretoYabstracto(v_r.p, v.p)$

- **De retornos a llamadas**: $\delta_m((m_r, v'_c, v'_r), (v_{ex}, v_r))$, con $m_r = (v'_{en}, (v'_{ex}, v'_r))$ contiene a $(((m^*, v^*_{en}, (v^*_{ex}, v^*_r)), v^*_c, v''_r), v''_{en})$ si y sólo si:
  - $v_{ex}.i$ es una salida, $v^*_{en}.i$ es una entrada
  - $v^*_{en} = v''_{en}$
  - $v^*_r = v''_r$
  - $(v_{ex}, v_r) = (v'_{ex}, v'_r)$, equivalentemente $(v_{ex}, v_r) = \pi_2(m')$
  - $concreto Y abstracto(v_r.p,\ v^*_c.p)$

- La función η está definida como:
  - Para nodos no de salida: $\eta_m(v) = v.i$
  - Para nodos de salida: $\eta_m(v_{ex}, v_r) = v_{ex}.i$
  - Para llamadas: $\eta_m(b, v_{en}) = v_{en}.i$
  - Para retornos: $\eta_m(b, (v_{ex}, v_r)) = v_r.i$

El conjunto de vértices iniciales es $inicio = \{v \in V \mid v.p$ es una predicción inicial o única$\}$ y el conjunto de vértices de aceptación de este RBA está dado por el único conjunto:

$$F_1 = \{v \in \bigcup_{m \in M} N_m \mid v.p \text{ es una predicción única o final}\} \cup \{v \in \bigcup_{m \in M} Retornos_m \mid v.p \text{ es una predicción final}\}.$$

Además, extendemos la función de transición δ_m para que los vértices de aceptación tengan una transición a ellos mismos, sin restricciones, convirtiéndolos así en vértices *terminales*. Por lo tanto, una vez alcanzado un vértice de aceptación, cualquier sufijo será aceptado por $\hat{\mathcal{A}}_\varphi$.

- **De nodos de aceptación a ellos mismos**: si $v \in F_1$, entonces $\delta_m(v)$ contiene a v.

Ahora bien, queda definir las funciones de etiquetación *fin* y *ext*. Para hacerlo, nos referimos a la prueba de vacuidad de RGBA dada en [1]. Primero, para cada componente S_m de la RSM construída, computamos para cada caja $b = (m', v_c, v_r)$ en B_m, si existe un camino desde $(\epsilon, \pi_1(m'))$ hasta $(\epsilon, \pi_2(m'))$ en la estructura de Kripke global K_S, y si existe, agregamos al RBA un *arco de atajo* entre la llamada y el retorno de la caja b en S_m. Note que $\pi_1(m')$ y $\pi_2(m')$ representan la (única) entrada y la (única) salida del módulo m'. Como se mencionó anteriormente, determinar si tal camino existe requiere resolver alcanzabilidad en un grafo $And - Or$, y toma tiempo $\mathcal{O}(|S_\varphi|\theta^2_{S_\varphi})$ y espacio $\mathcal{O}(|S_\varphi|\theta_{S_\varphi})$. Los módulos del RBA resultante tiene arcos entre las llamadas y los retornos en sus cajas cuando es posible alcanzar el retorno desde la llamada.

Dado un vértice $v \in V$ del RBA extendido, definimos $\mathcal{B}usq(v)$ como el conjunto de vértices del RBA que pueden ser alcanzados desde v. Este conjunto puede computarse con una búsqueda hacia atrás sobre la función de transición δ. Y luego definimos las funciones, para todo $v \in V_m$:

$$\begin{array}{ll} \mathit{fin}(v) = \mathit{true} & \text{si y sólo si} \quad v \in \bigcup_{v' \in F_1} \mathcal{B}usq(v') \\ \mathit{ext}(v) = \mathit{true} & \text{si y sólo si} \quad v \in \mathcal{B}usq(\pi_2(m)) \end{array}$$

Note que gracias a los *arcos de atajo*, las llamadas y sus predecesores también pueden ser alcanzados por la búsqueda.

Dado que la especificación φ está bien definida y que cada transición en δ satisface las ecuaciones de φ, es sencillo probar que si $s_0, ..., s_k$, $s_i = (z_i, v_i)$, son estados de la estructura de Kripke $K_{\hat{\mathcal{A}}_\varphi}$ y $z_k = \langle\rangle$, la palabra $(v_0.i)...(v_k.i)(v_k.i)(v_k.i)...$ está en $\mathcal{L}(\hat{\mathcal{A}}_\varphi)$ si y sólo si

$$\sigma = \mathit{alinear}((v_0.p).\texttt{actual}, ..., (v_{k-1}.p).\texttt{actual}, (v_k.p).\texttt{actual})$$

es el (único) modelo de evaluación para la traza estructurada τ definida como $\tau = \mathit{alinear}(v_0.i, ..., v_{k-1}.i, v_k.i)$.

Lema 5.4. *Sea $s_0, ..., s_k$, $s_j = (z_j, v_j)$, una secuencia de estados de la estructura de Kripke $K_{\hat{\mathcal{A}}_\varphi}$ tal que $s_0 \in Inicio$, $s_{j+1} \in \delta(s_j)$ para todo $j < k$, $z_k = \epsilon$*

y $s_k \in F_1^{\#}$. *Entonces la palabra* $\alpha = (v_0.i)...(v_k.i)(v_k.i)...$ *está en* $\mathcal{L}(K_{\hat{\mathcal{A}}_\varphi})$ *si y sólo si* $\sigma = alinear((v_0.p).\texttt{actual}, ..., (v_k.p).\texttt{actual})$ *es el único modelo de evaluación para la traza estructurada* $\tau = alinear(v_0.i, ..., v_k.i)$.

Demostración. ($\Rightarrow$) Por definición de la función de transición de $K_{\hat{\mathcal{A}}_\varphi}$, las predicciones $v_0.p, ..., v_k.p$ son coherentes a lo largo del camino y por lo tanto satisfacen las ecuaciones de φ para los valores de entrada $v_0.i, ..., v_k.i$. Además, como $z_0 = \epsilon = z_k$, $\tau = alinear(v_0.i, ..., v_k.i)$ define una traza estructurada. Luego, como φ está bien definida, σ es el único modelo de evaluación para τ.

($\Leftarrow$) Sea $\tau = alinear(i_0, ..., i_k)$ una traza estructurada y σ un modelo de evaluación de φ para τ definido como $\sigma = alinear(p_0.\texttt{actual}, ..., p_k.\texttt{actual})$, para algunas predicciones $p_0, ..., p_k$. Por construcción, existen vértices $v_0, ..., v_k$ de $\hat{\mathcal{A}}_\varphi$ tales que:

- $v_j.i = i_j$ y $(v_j.p).\texttt{actual} = p_j$, para todo $j \leq k$
- $v_0.p$ es una predicción inicial
- $v_1.p, ..., v_{k-1}.p$ son predicciones intermedias
- $v_k.p$ es una predicción final

Note que en $\hat{\mathcal{A}}_\varphi$ existen muchos conjuntos $v_0, ..., v_k$ que satisfacen esto porque no están especificadas las predicciones de valores futuros o pasados. Luego, para alguno de esos $v_0, ..., v_k$, y dado que σ es un modelo de evaluación, existen en $K_{\hat{\mathcal{A}}_\varphi}$ estados $s_0, ..., s_k$ tales que $s_j = (z_j, v_j)$ para $j \leq k$, y $s_{j+1} \in \delta(s_j)$ para todo $j < k$. Además, si $v_k \in F_1$ y $v_0 \in inicio$ entonces $s_k \in F_1^{\#}$ y $s_0 \in Inicio$. Luego, $s_k \in \delta(s_k)$, y $s_0, ..., s_k, s_k, ...$ es una ejecución de aceptación de $K_{\hat{\mathcal{A}}_\varphi}$ en $\alpha = v_0.i, ..., v_k.i, v_k.i, ...$, y por lo tanto $\alpha \in \mathcal{L}(K_{\hat{\mathcal{A}}_\varphi})$. □

Ahora supongamos que se consta de dos oráculos. Uno de los oráculos puede responder, dado un vértice $v \in V_m$, si v puede alcanzar la salida de S_m. El otro, responder si v puede alcanzar un vértice de aceptación dentro del

módulo al que pertenece, posiblemente llamando a otros módulos de $\hat{\mathcal{A}}_\varphi$. Sean $\mathcal{O}_{salida}, \mathcal{O}_{acept} : V \to \mathbb{B}$ los oráculos. Bajo esta suposición, y considerando a cada caja c_i definida como $c_i = ((v_{en_i}, (v_{ex_i}, v_{r_i})), v_{c_i}, v_{r_i})$, describimos el siguiente algoritmo:

funcion $finaliza(\langle c_1, ..., c_\ell \rangle, v)$:
if $\ell = 0$ **then**
  {Pila vacía}
  return $\mathcal{O}_{acept}(v)$
else
  {Si el vértice no es el retorno de la caja, debe poder alcanzar la salida}
  if $v \neq (c_\ell, (v_{ex_\ell}, v_{r_\ell}))$ **then**
    if $\mathcal{O}_{salida}(v)$ **then**
      $v = (c_\ell, (v_{ex_\ell}, v_{r_\ell}))$
      {Llamada recursiva desapilando la pila}
      return $finaliza(\langle c_1, ..., c_{\ell-1} \rangle, v)$
    end if
  end if
  return $False$
end if

Lema 5.5. *Sea $s_k = (\langle c_1, ..., c_\ell \rangle, v_k)$ un estado de la estructura de Kripke $K_{\hat{\mathcal{A}}_\varphi}$. El resultado de $finaliza(s_k)$ es true si y solo si existe una secuencia de estados $s_{k+1}, ..., s_N$ de $K_{\hat{\mathcal{A}}_\varphi}$, $s_j = (z_j, v_j)$, tal que $s_{j+1} \in \delta(s_j)$ para todo $k \leq j < N$, y $(\epsilon, v_N) \in F_1^{\#}$.*

Demostración. ($\Rightarrow$) Hacemos inducción en ℓ. Para el caso base $\ell = 0$, se tiene $\mathcal{O}_{acept}(v_k) = true$ y por lo tanto v_k puede alcanzar un estado de aceptación v_N del módulo al que pertenece, posiblemente usando los arcos de atajo que representan llamadas a otros módulos. Cada uno de estos arcos implica la existencia de un camino $(\epsilon, v_{en})...(\epsilon, v_{ex})$ en $K_{\hat{\mathcal{A}}_\varphi}$ que une la entrada v_{en}

del módulo que se invoca con la salida v_{ex}. Luego, independientemente de los arcos de atajo usados por v_k para alcanzar un estado de aceptación, siempre existe una secuencia $s_{k+1}...s_N$, $s_j = (z_j, v_j)$, de estados de $K_{\hat{\mathcal{A}}_\varphi}$ tal que $s_{j+1} \in \delta(s_j)$ para todo $k \leq j < N$ y $(\epsilon, v_N) \in F_1^{\#}$.

Para el caso inductivo $\ell > 0$ se tiene que $\mathcal{O}_{salida}(v_k)$ es verdadero y por lo tanto, por el mismo razonamiento anterior sobre los arcos de atajo, existe una secuencia de estados $s_{k+1}...s_{k'}$, $s_j = (z_j, v_j)$, tal que $s_{j+1} \in \delta(s_j)$ para todo $k \leq j < k'$ y $v_{k'}$ es la (única) salida de la caja de v_k. Luego, por hipótesis inductiva, existe una secuencia $s_{k'+1}...s_N$ tal que $s_{j+1} \in \delta(s_j)$ para todo $k' \leq j < N$ y $(\epsilon, v_N) \in F_1^{\#}$. Del algoritmo se deduce que $v_{k'+1}$ es un retorno que utiliza la salida $v_{k'}$, y la existencia de tal camino y la definición de δ garantizan que las predicciones de $v_{k'}$ y $v_{k'+1}$ son coherentes. Luego, $s_{k+1}...s_{k'}s_{k'+1}...s_N$ es la secuencia tal que $s_{j+1} \in \delta(s_j)$ para todo $k \leq j < N$ y $(\epsilon, v_N) \in F_1^{\#}$.

($\Leftarrow$) Existen dos casos. Si $N = k$, se tiene que $s_k = (\epsilon, v_k)$ y $(\epsilon, v_k) \in F_1^{\#}$, y entonces $s_k \in \delta(s_k)$. Luego, $\mathcal{O}_{acept}(v_k) = True$ y $finaliza(s_k) = True$.

Si $N > k$, tenemos un camino $s_{k+1}, ..., s_N$, $s_j = (z_j, v_j)$, tal que $s_{j+1} \in \delta(s_j)$ para todo $k \leq k < N$, y $(\epsilon, v_N) \in F_1^{\#}$. Supongamos $z_k = \langle c_1, ..., c_\ell \rangle$. Por definición de δ sabemos que $z_{j+1} = z_j.c_r$ si y sólo si v_j es una llamada, y que $z_{j+1}.c_r = z_j$ si y sólo i v_{j+1} es un retorno. Luego, como $z_N = \epsilon$, tenemos que los retornos de todas las cajas en z_k han ocurrido al final del camino y por ende cada retorno de una caja puede alcanzar la salida de su módulo: $\mathcal{O}_{salida}(v_{r_j}) = True$, para $1 \leq j \leq \ell$, donde v_{r_j} es el retorno de la caja c_j. Finalmente, el camino $(\epsilon, v_{r_1}), (\epsilon, v_{r_1+1}), ..., (\epsilon, v_N)$ en $K_{\hat{\mathcal{A}}_\varphi}$ es el testigo de que $\mathcal{O}_{acept}(v_{r_1}) = True$. Por lo tanto, $finaliza(\langle c_1, ..., c_\ell \rangle, v_k) = finaliza(s_k) = True$. □

Teorema 5.6. *Sea φ una especificación NSRV booleana y bien definida. Dada una traza parcial inicial τ de longitud M y una constante $b \in \mathbb{B}$ el problema de decidir si alguna variable dependiente s_j de φ es inevitablemente b a partir de la posición M es decidible.*

Demostración. Por Lema 5.3 asumimos que φ es $\varphi_{s_j,b}$, una especificación

sobre las variables dependientes $s_0, ..., s_n$.

Sea $\hat{\mathcal{A}}_\varphi = (S_\varphi, fin, ext)$ el RBA para φ, con $S = (M, \{S_m\}_{m \in M}, inicio, \mathcal{F})$. Sea $\tau = \langle \tau_{call}, \tau_{en}, \tau_{ex}, \tau_{ret}, \tau_1, ..., \tau_m \rangle$ la traza inicial parcial de longitud M y $Inicio_\tau$ el conjunto de estados $(\epsilon, v) \in Inicio$ tales que $v \in inicio$ y $v.i(j) = \tau_j(0)$. Sea Q_s^τ, para cada $s \in Inicio_\tau$, el conjunto de estados s_M tales que existe un camino $s_0, ..., s_M$, $s_i = (z_i, v_i)$ tales que $s = s_0$ y $v_k.e(j) = \tau_j(k)$ para todo $k \leq m, j \leq M$. Cada conjunto Q_s^τ puede calcularse recorriendo $\hat{\mathcal{A}}_\varphi$ y almacenandos las cajas b_i a las cuales se entra y se sale mediante llamadas y retornos, lo cual se puede hacer en tiempo $\mathcal{O}(|M|.|\hat{\mathcal{A}}_\varphi|)$ y espacio $\mathcal{O}(|\hat{\mathcal{A}}_\varphi|)$. Luego $Q_\tau = \bigcup_{s \in Inicio_\tau} Q_s^\tau$ es el conjunto de todos los estados alcanzables desde $Inicio_\tau$ usando vértices de $\hat{\mathcal{A}}_\varphi$ cuya predicciones de entrada forman exactamente τ.

Sea $s_M = (z_M, v_M)$ un estado de Q_τ y sea $s_0, ..., s_M$ unos de los caminos en $K_{\hat{\mathcal{A}}_\varphi}$ tal que $s_0 \in Inicio_\tau$. Luego, por Lema 5.5, $finaliza((z_M, v_M)) = True$ si y sólo si existe un camino $s_{M+1}...s_N$ en $K_{\hat{\mathcal{A}}_\varphi}$ tal que $s_N = (\epsilon, v_N)$ y $s_N \in F_1^\#$, satisfaciendo todas las ecuaciones de φ con predicciones en los vértices coherentes a lo largo del camino. Es decir, si y sólo si $s = alinear((v_0.p).\texttt{actual}, ..., (v_N.p).\texttt{actual})$ es el único modelo de evaluación para la traza estructurada $\tau = alinear(v_0.i, ..., v_N.i)$. Luego, si se cumple $(s_M.p).\texttt{actual}(n) = true$ para algún $s_M \in Q_\tau$, existe al menos un modelo de evaluación de φ para alguna extensión de τ tal que s_j es $true$ en alguna posición, actual o futura, s_j no es inevitablemente b. Si no existe $s_M \in Q_\tau$ tal que $(s_M.p).\texttt{actual}(n) = true$, estamos seguros de que para toda traza parcial final τ' tal que $\tau\ \tau'$ es una traza estructurada, el modelo de evaluación de φ para $\tau\ \tau'$ satisface $\sigma_j(k) = b$ para todo $k \geq M$, y por lo tanto s_j es inevitablemente b. □

Capítulo 6

Conclusiones y trabajo futuro

> *"We can only see a short distance ahead, but we can see plenty there that needs to be done."*,
>
> Alan Turing

Se ha presentado un lenguaje de especificación sobre *streams* finitos que es simple y, en nuestro conocimiento, más expresivo que todos los otros formalismos existentes aplicables a la verificación en tiempo de ejecución. El conjunto de especificaciones bien formadas fue definido, se probaron los resultados necesarios para su identificación de manera estática y luego se demostró que las especificaciones de este conjunto resultan efectivamente bien definidas, es decir, que existe una y sólo una solución para ellas dada una traza. Para las especificaciones *mal formadas* se probó que el problema es indecidible en el caso general con valores numéricos mediante la codificación del Post Correspondence Problem, y que el problema es decidible si la especificación utiliza sólo valores de verdad. Además, se presentó un algoritmo que sigue una estrategia de evaluación parcial y una familia de especificaciones que pueden ser eficientemente monitoreables con tal algoritmo y que pueden ser sintácticamente identificables. Finalmente, se introdujo el concepto de *inevitabilidad* sobre la semántica del lenguaje y se demostró que el problema de

decisión asociado es decidible, abriendo una puerta hacia la extensión de la semántica del lenguaje a semánticas de 3 o 4 valores.

De todas maneras, quedan varios interrogantes abiertos acerca del lenguaje y divisamos diferentes problemas que resultarían interesantes investigar e incluso posibles tareas a realizar a nivel implementación.

Por un lado, el algoritmo de evaluación presentado mantiene en memoria, entre otras, las ecuaciones que todavía no han podido ser resueltas por su dependencia en otras ecuaciones. Una vez que todas las dependencias son resueltas, el algoritmo aplica los operadores sobre valores *ground* cada vez que los aplica. Esto nos hace pensar que es posible generalizar la técnica de verificación de NSRV para incluir tipos de streams arbitrarios, cada uno con un sistema de reestritura convergente y que para cada operador, dados todos valores *ground*, reescriba al valor *ground* que representa el resultado de la operación. Creemos que de esta manera se podrían incluir nuevos tipos de streams como cadenas, pilas, árboles binarios, entre otros. Estudiar la mejor forma de generalizar la estrategia y cómo cambian las nociones y algoritmos asociadas a las especificaciones es una línea a seguir en el futuro.

Además, se ha presentando en este trabajo una manera de controlar si especificaciones cumpliendo ciertas propiedades, aún cuando están mal formadas, están bien definidas. Las condiciones establecen que no existe dependencia de variables dependientes booleanas en variables enteras y divide el grafo de dependencia de la especificación en dos sub multigrafos: uno para la parte entera, que debe estar bien formada, y otro para la parte booleana que aún cuando no esté bien formado puede ser chequeado con los algoritmos sobre RBA (Autómatas de Büchi Recursivos). Sin embargo, en una especificación en la cual las ecuaciones para variables booleanas sólo utilizan variables enteras para comparar, se puede generar el conjunto de ecuaciones que se inducen para cada valor entero usado en la parte booleana. Conjugar estas ecuaciones junto con las ecucaciones definidas en la especificación para las variables enteras creemos que puede resultar en una forma menos restrictiva de controlar la buena definición de especificaciones mal formadas

que merece atención.

Como otra posible materia de estudio aparece la inclusión de *inevitabilidad* en la semántica del lenguaje, y el estudio de la adaptación de semánticas de 3 y 4 valores de verdad a especificaciones booleanas del lenguaje. De manera adicional, estudiar para cuales especificaciones existe otra especificación eficientemente monitoreable que las simule es interesante desde el punto de vista práctico.

Por último, la implementación de una herramienta para el lenguaje, que compruebe sintácticamente especificaciones, realice los controles para identificar si están bien definidas e implemente el algoritmo de evaluación a la llegada de nuevos valores de entrada es sin dudas la tarea más importante a realizar en el campo práctico.

Bibliografía

[1] Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Analysis of recursive state machines. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 207–220, London, UK, 2001. Springer-Verlag.

[3] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. In *In Developments in Language Theory, LNCS 4036*, pages 1–13. Springer, 2006.

[4] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Program monitoring with LTL in EAGLE. *Parallel and Distributed Processing Symposium, International*, 17:264b, 2004.

[5] Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *RV '07*, pages 126–138, 2007.

[6] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[7] Michael Benedikt, Patrice Godefroid, and Thomas W. Reps. Model checking of unrestricted hierarchical state machines. In *ICALP '01: Pro-*

ceedings of the 28th International Colloquium on Automata, Languages and Programming,, pages 652–666, London, UK, 2001. Springer-Verlag.

[8] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: Runtime monitoring of synchronous systems. In *Proceedings of the 12th International Symposium of Temporal Representation and Reasoning (TIME 2005)*, pages 166–174. IEEE Computer Society Press, 2005.

[9] Bernd Finkbeiner, Sriram Sankaranarayanan, and Henny B. Sipma. Collecting statistics over runtime executions. In Klaus Havelund and Grigore Roşu, editors, *In Proceedings of Runtime Verification (RV'02) Volume 1*, pages 36–55. Elsevier, 2002.

[10] Klaus Havelund and Grigore Roşu. Synthesizing monitors for safety properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer-Verlag, 2002.

[11] Klaus Havelund and Grigore Roşu. Efficient monitoring of safety properties. *STTT*, 6(2):158–173, 2004.

[12] C. A. R. Hoare. An axiomatic basis for computer programming. *COMMUNICATIONS OF THE ACM*, 12(10):576–580, 1969.

[13] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, May 2009.

[14] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[15] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–67, 1977.

[16] Emil Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[17] Grigore Roşu, Feng Chen, and Thomas Ball. Synthesizing monitors for safety properties – this time with calls and returns –. Technical Report UIUCDCS-R-2007-2908, University of Illinois at Urbana-Champaign, Department of Computer Science, 2007.

[18] Alan Madison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[19] Alle Rechte Vorbehalten, Technischen Universität München, Oliver Arafat, Oliver Arafat, Andreas Bauer, Andreas Bauer, Martin Leucker, Martin Leucker, and Christian Schallhart. Runtime verification revisited. Technical report, 2005.

[20] Wikipedia. Post correspondence problem — Wikipedia, the free encyclopedia, 2004. [Online; accedido 29-Mayo-2010].